Intel® Integrated Performance Primitives (Intel® IPP) Developer Guide and Reference

Contents

Chapter 1: Developer Guide and Reference for Intel® Integrated Performance Primitives

Introducing the Intel® Integrated Performance Primitives for Intel® Architecture 8
Getting Started with Intel® Integrated Performance Primitives 9
Finding Intel® IPP on Your System 9
Setting Environment Variables 11
Compiler Integration 11
Building Intel® IPP Applications 12
Using Intel® IPP Examples 15
Intel® IPP Examples Directory Structure 15
Building Intel® IPP Examples 16
Finding the Intel® IPP Documentation 16
Programming Considerations 16
Core and Support Functions 17
Channel and Planar Image Data Layouts 19
Regions of Interest 20
Managing Memory Allocations 21
Cache Optimizations 22
Using Intel® Integrated Performance Primitives Platform-Aware Functions 23
Using Integration Wrappers for Intel® Integrated Performance Primitives 23
Notational Conventions 23
Linking Your Application with Intel® Integrated Performance Primitives 24
Linking Options 24
Automatically Linking Your Microsoft* Visual Studio* Project with Intel IPP 26
Using Intel® Integrated Performance Primitives Threading Layer (TL) Functions 26
Finding Intel® IPP TL Source Code Files 27
Building Intel® IPP TL Libraries from Source Code 27
Automatically Getting Performance Benefits with Intel® IPP Threading Layer 28
Using Custom Library Tool for Intel® Integrated Performance Primitives 28
System Requirements for Custom Library Tool 29
Operation Modes 29
Using Console Version of Custom Library Tool 29
Programming with Intel® Integrated Performance Primitives in the Microsoft* Visual Studio* IDE 30
Configure Microsoft Visual Studio* to Link with Intel® Integrated Performance Primitives 31
Using the IntelliSense* Features 31
Appendix: Intel® IPP Threading and OpenMP* Support 33
Using a Shared L2 Cache 33
Avoiding Nested Parallelization 34
Intel® Integrated Performance Primitives Theory of Operation 34
Dispatching 34

Function Naming Conventions 35
Intel® Integrated Performance Primitives Domain Details 38
Performance Test Tool (perfsys) Command Line Options 38
Notices and Disclaimers 40
Developer Reference: Signal and Data Processing 41
Intel® Integrated Performance Primitives Concepts 42
Function Naming 42
Structures and Enumerators 47
Data Ranges 51
Data Alignment 52
Rounding Mode 52
Integer Scaling 52
Error Reporting 53
Platform-Aware Functions in Signal and Data Processing 55
Code Examples 56
Support Functions 57
Version Information Functions 57
Memory Allocation Functions 59
Common Functions 61
Dispatcher Control Functions 76
Vector Initialization Functions 77
Vector Initialization Functions 77
Sample-Generating Functions 84
Essential Functions 98
Logical and Shift Functions 98
Arithmetic Functions 108
Conversion Functions 163
Windowing Functions 228
Statistical Functions 238
Sampling Functions 267
AI Inference Functions 271
Filtering Functions 273
Convolution and Correlation Functions 273
Filtering Functions 287
Transform Functions 368
Fourier Transform Functions 368
Discrete Cosine Transform Functions 409
Hilbert Transform Functions 417
Wavelet Transform Functions 421
Fixed-Accuracy Arithmetic Functions 441
Arithmetic Functions 442
Power and Root Functions 455
Exponential and Logarithmic Functions 475
Trigonometric Functions 483
Hyperbolic Functions 500
Special Functions 511
Rounding Functions 523
Long Term Evolution (LTE) Wireless Support Functions 532
CRC_8u 533
ippsGenCRCOptPoly_8u 535
CRC16 537
CRC24a, CRC24b, CRC24c 538
Data Compression Functions 540
Application Notes 540
Dictionary-Based Compression Functions 540

BWT-Based Compression Functions 609
ZFP Compression Functions 638
Appendix A: Handling of Special Cases 650
Appendix B: Removed Functions for Signal Processing 655
Bibliography for Signal Processing 751
Glossary 756
Developer Reference: Image Processing and Computer Vision 758
Intel® Integrated Performance Primitives Concepts 759
Function Naming 760
Function Prototypes in Intel IPP 763
Rounding Mode 764
Integer Result Scaling 765
Error Reporting 765
Platform-Aware Functions for Image Processing 770
Threading Layer Functions 771
Structures and Enumerators 771
Image Data Types and Ranges 778
Major Operation Models 779
Support Functions 783
Version Information Function 783
Status Information Function 784
Memory Allocation Functions 785
Threading Layer Functions 788
Image Data Exchange and Initialization Functions 793
Convert 793
BinToGray, GrayToBin 798
Scale 799
ScaleC 802
Set 805
Copy 808
CopyManaged 812
CopyConstBorder 813
CopyMirrorBorder 817
CopyReplicateBorder 821
CopyWrapBorder 824
CopySubpix 827
CopySubpixIntersect 829
Dup 832
Transpose 833
SwapChannels 835
AddRandUniform 838
AddRandGauss 839
ImageJaehne 840
ImageRamp 842
SampleLine 844
ZigzagFwd8x8 846
ZigzagInv8x8 848
Image Arithmetic and Logical Operations 849
Arithmetic Operations 849
Logical Operations 923
Alpha Composition 940
Image Color Conversion 950
Gamma Correction 951
CIE Chromaticity Diagram and Color Gamut 951
Color Models 952

Image Downsampling 966
RGB Image Formats 967
Pixel and Planar Image Formats 967
Color Model Conversion 972
Color - Gray Scale Conversions 1058
Format Conversion 1067
Color Twist 1110
Color Keying 1114
Gamma Correction 1117
Intensity Transformation 1123
Image Bit Conversion 1140
Threshold and Compare Operations 1142
Thresholding 1142
Compare Operations 1159
Morphological Operations 1166
Dilate3x3 1173
Dilate 1174
DilateBorder 1176
DilateGetBufferSize 1179
DilateGetSpecSize 1180
DilateInit 1181
Erode3x3 1182
Erode 1183
ErodeBorder 1185
ErodeGetBufferSize 1187
ErodeGetSpecSize 1188
ErodeInit 1189
GrayDilateBorder 1190
GrayErodeBorder 1191
MorphAdvInit 1192
MorphAdvGetSize 1194
MorphGetBufferSize 1195
MorphGetSpecSize 1196
MorphInit 1197
MorphologyBorderGetSize 1198
MorphologyBorderInit 1199
MorphBlackhat 1200
MorphBlackhatBorder 1202
MorphClose 1205
MorphCloseBorder 1207
MorphGradient 1210
MorphGradientBorder 1213
MorphOpen 1215
MorphOpenBorder 1217
MorphTophat 1220
MorphTophatBorder 1222
MorphGrayInit 1224
MorphGrayGetSize 1226
MorphReconstructGetBufferSize 1227
MorphReconstructDilate 1227
MorphReconstructErode 1231
MorphSetMode 1233
Filtering Functions 1233
Borders in Neighborhood Operations 1234
User-defined Border Types 1235

Filters with Borders 1241
Median Filters....1278
General Linear Filters 1290
Separable Filters....1302
Smoothing Filters....1320
Wiener Filters....1324
Convolution....1329
Deconvolution 1335
Fixed Filters 1342
Deinterlacing Filters 1428
Image Linear Transforms....1431
Fourier Transforms 1431
Windowing Functions....1469
Discrete Cosine Transforms 1475
Image Statistics Functions....1489
Sum....1490
Integral 1492
SqrIntegral 1495
TiltedIntegral 1496
TiltedSqrIntegral....1498
Mean....1500
Mean_StdDev....1502
RectStdDev....1504
TiltedRectStdDev 1506
HistogramGetBufferSize....1508
HistogramGetLevels 1509
HistogramInit, HistogramUniformInit 1510
Histogram....1511
CountInRange 1515
BlockMinMax 1516
Min 1518
MinIndx....1519
Max 1521
MaxIndx 1522
MinMax....1524
MinMaxIndx 1525
MaxEvery....1527
MinEvery 1529
FindPeaks3x3GetBufferSize 1531
FindPeaks3x3 1532
Image Moments....1535
Image Norms 1545
Image Quality Index....1569
Image Proximity Measures 1574
Image Geometry Transforms 1589
ROI Processing in Geometric Transforms 1590
Geometric Transform Functions....1591
Miscellaneous Image Transforms 1771
ThresholdAdaptiveBoxGetBufferSize 1772
ThresholdAdaptiveBox 1773
ThresholdAdaptiveGaussGetBufferSize 1775
ThresholdAdaptiveGaussInit 1776
ThresholdAdaptiveGauss 1778
Wavelet Transforms....1781
WTFwdGetSize 1785

WTFwdInit 1786
WTFwd 1787
WTFwdGetSize 1792
WTInvInit 1793
WTInv 1795
Computer Vision 1800
Using ippiAdd for Background Differencing 1800
Feature Detection Functions 1801
Distance Transform Functions 1865
Image Gradients 1875
Flood Fill Functions 1893
Motion Analysis and Object Tracking 1904
Universal Pyramids 1923
Image Inpainting 1941
Image Segmentation 1948
Pattern Recognition 1964
Camera Calibration and 3D Reconstruction 1990
3D Data Processing Functions 1995
CopyConstBorder 1996
CopyReplicateBorder 1998
Filter 2001
FilterGetBufSize 2003
FilterBorder 2004
FilterBorderInit 2009
FilterBorderGetSize 2010
FilterMedian 2012
FilterMedianInit 2015
FilterMedianGetSize 2016
ResizeGetBufSize 2017
GetResizeCuboid 2018
Resize 2019
Remap 2022
WarpAffineGetBufSize 2025
WarpAffine 2026
Appendix A: Handling of Special Cases 2029
Appendix B: Interpolation in Image Geometric Transform Functions 2030
Overview of Interpolation Modes 2030
Mathematical Notation 2032
Nearest Neighbor Interpolation 2033
Linear Interpolation 2033
Cubic Interpolation 2034
Super Sampling 2035
Lanczos Interpolation 2036
Tricubic Interpolation 2038
Trilinear Interpolation 2040
Interpolation with Two-Parameter Cubic Filters 2042
Appendix C: Removed Functions for Image and Video Processing 2043
Bibliography for Image Processing 2148
Glossary 2153
Getting Help and Support 2155

Developer Guide and Reference for Intel® Integrated Performance Primitives

1

NOTE The Intel® Integrated Performance Primitives Developer Guide for Intel® oneAPI Base Toolkit and Intel® Integrated Performance Primitives Developer Reference have been combined in to this single Developer Guide and Reference. To find prior versions of the individual Developer Guide and Developer Reference, go to the Downloadable Documentation page.

Introducing Intel® Integrated Performance Primitives

The Intel® Integrated Performance Primitives (Intel® IPP) is a software library that provides a comprehensive set of application domain-specific highly optimized functions for data, signal, and image processing.

Signal and Data Processing

The Intel IPP signal and data processing software is a collection of low-overhead, high-performance operations performed on one-dimensional (1D) data arrays. Examples of such operations are linear transforms, filtering, string processing, and vector math.

See Intel IPP Developer Reference. Volume 1: Signal and Data Processing.

Image Processing

The Intel IPP image processing software is a collection of low-overhead, high-performance operations performed on two-dimensional (2D) arrays of pixels. Examples of such operations are linear transforms, filtering, and arithmetic on image data.

See Intel IPP Developer Reference. Volume 2: Image Processing.

Product and Performance Information
Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex. Notice revision #20201201

- Introducing the Intel® Integrated Performance Primitives for Intel® Architecture
- Getting Started with Intel® Integrated Performance Primitives
- Programming Considerations
- Performance Test Tool (perfsys) Command Line Options
- Notices and Disclaimers
- Developer Reference: Signal and Data Processing
- Developer Reference: Image Processing and Computer Vision
- Getting Help and Support

Introducing the Intel® Integrated Performance Primitives for Intel® Architecture

The Intel IPP software enables taking advantage of the parallelism of single-instruction, multiple data (SIMD) instructions, which make the core of the MMX technology and Intel® Streaming SIMD Extensions. These technologies improve the performance of computation-intensive signal, image, and video processing applications. Plenty of the Intel IPP functions are tuned and threaded for multi-core systems.

Intel IPP supports application development for various Intel® architectures. By providing a single cross-architecture application programmer interface, Intel IPP permits software application repurposing and enables developers to port to unique features across Intel® processor-based desktop, server, mobile, and handheld platforms. Use of the Intel IPP primitive functions can help drastically reduce development costs and accelerate time-to-market by eliminating the need of writing processor-specific code for computation intensive routines.

You can find more details of the product usage in the Intel IPP Developer Guide for your operating system.

Getting Started with Intel® Integrated Performance Primitives

This section helps you start using Intel® Integrated Performance Primitives (Intel® IPP) by giving a quick overview of some fundamental concepts and showing how to build an Intel® IPP program.

- Finding Intel® IPP on Your System
- Setting Environment Variables
- Compiler Integration
- Building Intel® IPP Applications
- Using Intel® IPP Examples
- Finding the Intel® IPP Documentation

Finding Intel® IPP on Your System

The Unified Directory Layout was implemented in 2021.10 (Intel® oneAPI Toolkit release 2024.0). If you have multiple toolkit versions installed, the Unified layout adds the ability to ensure your development environment contains the component versions that were released as part of that specific toolkit version. The Unified Directory Layout is available **only** when installing the Intel® oneAPI Base Toolkit. It is not available if you installed Intel® IPP as a standalone component.

Environment variables are set up with a script called setvars or oneapi-vars, depending on which layout you are using.

To understand more about the Unified Directory Layout, including how the environment is initialized and the advantages of using the layout, see Use the setvars and oneapi-vars Scripts with Linux * or Use the setvars and oneapi-vars Scripts with Windows* .

Intel® Integrated Performance Primitives (Intel® IPP) installs in the directory referred to as `<ipp directory>` . By default, the `<ipp directory>` is :

- On Windows* OS Component Directory Layout: `C:\Program Files (x86)\Intel\oneAPI\ipp\<version>`
- On Windows* OS Unified Directory Layout: `C:\Program Files (x86)\Intel\oneAPI\<toolkit-version>`
- On Linux* OS Component Directory Layout : `/opt/intel/oneapi/ipp/<version>`
- On Linux* OS Unified Directory Layout: `/opt/intel/oneapi/<toolkit-version>`

> **NOTE** If you installed Intel® IPP as a standalone component, you will only find Intel® IPP in the Component Directory Layout.

The tables below describe the structure of the high-level directories on:

- Windows* OS
- Linux* OS

Windows* OS:

Directory	Contents
`env`	Batch files to set environmental variables in the user shell
`include`	Header files for the library functions
`lib`	Single-threaded static libraries for the Intel® 64 architecture A static library of Integration Wrappers for the Intel® 64 architecture Threading Layer static libraries for the Intel® 64 architecture
`bin`	Single-threaded DLLs for applications running on processors with the Intel® 64 architecture Threading layer dynamic libraries for the Intel® 64 architecture
`sharedocipp`	Intel IPP interfaces and example files
`optipptoolscustom_library_tool_python`	Command-line and GUI tool for building custom dynamic libraries
`optipptools<arch>staticlib`	Header files for disabling the Intel IPP static dispatcher (linking with one CPU-optimized variant) For Intel® 64, `<arch> = intel64`
`sharedocipplicensing`	License files and a third party programs file

Linux* OS:

Directory	Contents
`env`	Bash files to set environmental variables in the user shell
`include`	Header files for the library functions
`lib`	Single-threaded static and dynamic libraries for the Intel® 64 architecture A static library of Integration Wrappers for the Intel® 64 architecture Threading Layer static and dynamic libraries for the Intel® 64 architecture
Intel® 64: `/opt/intel/oneapi/redist/lib`	Redistributable dynamically linked shared object libraries
Intel® 64: `lib/nonpic`	Non-PIC single-threaded static libraries
`share/doc/ipp`	Intel IPP interfaces and example files
`opt/ipp/tools/custom_library_tool_python`	Command-line and GUI tool for building custom dynamic libraries
`opt/ipp/tools/<arch>/staticlib`	Header files for disabling the Intel IPP static dispatcher (linking with one CPU-optimized variant) For Intel® 64, `<arch> = intel64`
`etc/modulefiles/intel_ipp_<arch>`	Tcl modulefiles For Intel® 64, `<arch>` = `intel64`

share/doc/ipp/licensing	License files and a third party programs file

See Also

Notational Conventions

Setting Environment Variables

When the installation of Intel IPP is complete, set the environment variables in the command shell using one of the script files in the env subdirectory of the Intel IPP installation directory:

On Windows* OS:

vars.bat for the Intel® 64 architecture.

On Linux* OS:

vars.sh for the Intel® 64 architecture.

When using the vars script, you need to specify the architecture as a parameter. For example:

- vars.bat intel64

  sets the environment for Intel IPP to use the Intel® 64 architecture on Windows* OS.
- . vars.sh intel64

  sets the environment for Intel IPP to use the Intel® 64 architecture on Linux* OS.

Alternatively, you can use the oneapi-vars script in the Unified Directory Layout. To learn more, see Finding Intel® IPP on Your System .

The scripts set the following environment variables:

Windows* OS	Linux* OS	Purpose
IPPROOT	IPPROOT	Point to the Intel IPP installation directory
LIB	n/a	Add the search path for the Intel IPP single-threaded libraries
PATH	LD_LIBRARY_PATH	Add the search path for the Intel IPP single-threaded DLLs
INCLUDE	n/a	Add the search path for the Intel IPP header files

Compiler Integration

Intel® C++ Compiler and Microsoft Visual Studio* compilers simplify developing with Intel® IPP.

On Windows* OS, a default installation of Intel® IPP installs integration plug-ins. These enable the option to configure your Microsoft Visual Studio* project for automatic linking with Intel IPP.

Intel® C++ Compiler also provides command-line parameters to set the link/include directories:

- On Windows* OS:

  /Qipp-link:{dynamic|static} and / Qipp
- On Linux* OS:

  -ipp-link={dynamic|static}

See Also

Automatically Linking Your Microsoft* Visual Studio* Project with Intel IPP
Linking Your Application with Intel® IPP

Building Intel® IPP Applications

The code example below represents a short application to help you get started with Intel® IPP:

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

#include "ipp.h"
#include <stdio.h>

int main(int argc, char *argv[])
{
    const IppLibraryVersion *lib;
    IppStatus status;
    Ipp64u mask, emask;

    /* Init Intel® Integrated Performance Primitives (Intel® IPP) library */
    ippInit();
    /* Get Intel IPP library version info */
    lib = ippGetLibVersion();
    printf("%s %s\n", lib->Name, lib->Version);

    /* Get CPU features and features enabled with selected library level */
    status = ippGetCpuFeatures(&mask, 0);
    if (ippStsNoErr == status) {
        emask = ippGetEnabledCpuFeatures();
        printf("Features supported by CPU\tby IPP\n");
        printf("----------------------------------------\n");
        printf("  ippCPUID_MMX        = ");
        printf("%c\t%c\t", (mask & ippCPUID_MMX) ? 'Y' : 'N', (emask & ippCPUID_MMX) ? 'Y' :
'N');
        printf("Intel® Architecture MMX technology supported\n");
        printf("  ippCPUID_SSE        = ");
        printf("%c\t%c\t", (mask & ippCPUID_SSE) ? 'Y' : 'N', (emask & ippCPUID_SSE) ? 'Y' :
'N');
        printf("Intel® Streaming SIMD Extensions\n");
        printf("  ippCPUID_SSE2       = ");
        printf("%c\t%c\t", (mask & ippCPUID_SSE2) ? 'Y' : 'N', (emask & ippCPUID_SSE2) ? 'Y' :
'N');
        printf("Intel® Streaming SIMD Extensions 2\n");
        printf("  ippCPUID_SSE3       = ");
```

```
        printf("%c\t%c\t", (mask & ippCPUID_SSE3) ? 'Y' : 'N', (emask & ippCPUID_SSE3) ? 'Y' :
'N');
        printf("Intel® Streaming SIMD Extensions 3\n");
        printf("  ippCPUID_SSSE3      = ");
        printf("%c\t%c\t", (mask & ippCPUID_SSSE3) ? 'Y' : 'N', (emask & ippCPUID_SSSE3) ? 'Y' :
'N');
        printf("Supplemental Streaming SIMD Extensions 3\n");
        printf("  ippCPUID_MOVBE      = ");
        printf("%c\t%c\t", (mask & ippCPUID_MOVBE) ? 'Y' : 'N', (emask & ippCPUID_MOVBE) ? 'Y' :
'N');
        printf("The processor supports MOVBE instruction\n");
        printf("  ippCPUID_SSE41      = ");
        printf("%c\t%c\t", (mask & ippCPUID_SSE41) ? 'Y' : 'N', (emask & ippCPUID_SSE41) ? 'Y' :
'N');
        printf("Intel® Streaming SIMD Extensions 4.1\n");
        printf("  ippCPUID_SSE42      = ");
        printf("%c\t%c\t", (mask & ippCPUID_SSE42) ? 'Y' : 'N', (emask & ippCPUID_SSE42) ? 'Y' :
'N');
        printf("Intel® Streaming SIMD Extensions 4.2\n");
        printf("  ippCPUID_AVX        = ");
        printf("%c\t%c\t", (mask & ippCPUID_AVX) ? 'Y' : 'N', (emask & ippCPUID_AVX) ? 'Y' :
'N');
        printf("Intel® Advanced Vector Extensions (Intel® AVX) instruction set\n");
        printf("  ippAVX_ENABLEDBYOS  = ");
        printf("%c\t%c\t", (mask & ippAVX_ENABLEDBYOS) ? 'Y' : 'N', (emask &
ippAVX_ENABLEDBYOS) ? 'Y' : 'N');
        printf("The operating system supports Intel® AVX\n");
        printf("  ippCPUID_AES        = ");
        printf("%c\t%c\t", (mask & ippCPUID_AES) ? 'Y' : 'N', (emask & ippCPUID_AES) ? 'Y' :
'N');
        printf("Intel® AES instruction\n");
        printf("  ippCPUID_SHA        = ");
        printf("%c\t%c\t", (mask & ippCPUID_SHA) ? 'Y' : 'N', (emask & ippCPUID_SHA) ? 'Y' :
'N');
        printf("Intel® SHA new instructions\n");
        printf("  ippCPUID_CLMUL      = ");
        printf("%c\t%c\t", (mask & ippCPUID_CLMUL) ? 'Y' : 'N', (emask & ippCPUID_CLMUL) ? 'Y' :
'N');
        printf("PCLMULQDQ instruction\n");
        printf("  ippCPUID_RDRAND     = ");
        printf("%c\t%c\t", (mask & ippCPUID_RDRAND) ? 'Y' : 'N', (emask & ippCPUID_RDRAND) ?
'Y' : 'N');
        printf("Read Random Number instructions\n");
        printf("  ippCPUID_F16C       = ");
        printf("%c\t%c\t", (mask & ippCPUID_F16C) ? 'Y' : 'N', (emask & ippCPUID_F16C) ? 'Y' :
'N');
        printf("Float16 instructions\n");
        printf("  ippCPUID_AVX2       = ");
        printf("%c\t%c\t", (mask & ippCPUID_AVX2) ? 'Y' : 'N', (emask & ippCPUID_AVX2) ? 'Y' :
'N');
        printf("Intel® Advanced Vector Extensions 2 instruction set\n");
        printf("  ippCPUID_AVX512F    = ");
        printf("%c\t%c\t", (mask & ippCPUID_AVX512F) ? 'Y' : 'N', (emask & ippCPUID_AVX512F) ?
'Y' : 'N');
        printf("Intel® Advanced Vector Extensions 3.1 instruction set\n");
        printf("  ippCPUID_AVX512CD   = ");
        printf("%c\t%c\t", (mask & ippCPUID_AVX512CD) ? 'Y' : 'N', (emask & ippCPUID_AVX512CD) ?
'Y' : 'N');
```

```
        printf("Intel® Advanced Vector Extensions CD (Conflict Detection) instruction set\n");
        printf("  ippCPUID_AVX512ER   = ");
        printf("%c\t%c\t", (mask & ippCPUID_AVX512ER) ? 'Y' : 'N', (emask & ippCPUID_AVX512ER) ?
'Y' : 'N');
        printf("Intel® Advanced Vector Extensions ER instruction set\n");
        printf("  ippCPUID_ADCOX      = ");
        printf("%c\t%c\t", (mask & ippCPUID_ADCOX) ? 'Y' : 'N', (emask & ippCPUID_ADCOX) ? 'Y' :
'N');
        printf("ADCX and ADOX instructions\n");
        printf("  ippCPUID_RDSEED     = ");
        printf("%c\t%c\t", (mask & ippCPUID_RDSEED) ? 'Y' : 'N', (emask & ippCPUID_RDSEED) ?
'Y' : 'N');
        printf("The RDSEED instruction\n");
        printf("  ippCPUID_PREFETCHW  = ");
        printf("%c\t%c\t", (mask & ippCPUID_PREFETCHW) ? 'Y' : 'N', (emask &
ippCPUID_PREFETCHW) ? 'Y' : 'N');
        printf("The PREFETCHW instruction\n");
        printf("  ippCPUID_KNC        = ");
        printf("%c\t%c\t", (mask & ippCPUID_KNC) ? 'Y' : 'N', (emask & ippCPUID_KNC) ? 'Y' :
'N');
        printf("Intel® Xeon Phi™ Coprocessor instruction set\n");
    }
    return 0;
}
```

This application consists of three sections:

1. Initialize the Intel IPP library. This stage is required to take advantage of full Intel IPP optimization. The `ippInit()` function detects the processor type and sets the dispatcher to use the processor-specific code of the Intel® IPP library corresponding to the instruction set capabilities available. If your application runs without `ippInit()`, the Intel IPP library is auto-initialized with the first call of the Intel IPP function from any domain that is different from ippCore.

   In certain debugging scenarios, it is helpful to force a specific implementation layer using `ippSetCpuFeatures()`, instead of the best as chosen by the dispatcher.
2. Get the library layer name and version. You can also get the version information using the `ippversion.h` file located in the `/include` directory.
3. Show the hardware optimizations used by the selected library layer and supported by CPU.

NOTE Support for Intel ® Advanced Vector Extensions (Intel ® AVX) and Supplemental Streaming SIMD Extensions 3 (SSSE3) has been deprecated and will be removed in a future release.

Building the First Example with Microsoft Visual Studio* Integration on Windows* OS

On Windows* OS, Intel IPP applications are significantly easier to build with Microsoft* Visual Studio*. To build the code example above, follow the steps:

1. Start Microsoft Visual Studio* and create an empty C++ project.
2. Add a new c file and paste the code into it.
3. Set the include directories and the linking model as described in Automatically Linking Your Microsoft* Visual Studio* Project with Intel IPP.
4. Compile and run the application.

If you did not install the integration plug-in, configure your Microsoft* Visual Studio* IDE to build Intel IPP applications following the instructions provided in Configuring the Microsoft Visual Studio* IDE to Link with Intel® IPP.

Building the First Example on Linux* OS

To build the code example above on Linux* OS, follow the steps:

1. Paste the code into the editor of your choice.
2. Make sure the compiler and Intel IPP variables are set in your shell. For information on how to set environment variables see Setting Environment Variables.
3. Compile with the following command:

   Intel® 64:

```
icpx ipptest.cpp -o ipptest -I $IPPROOT/include -L $IPPROOT/lib -lippi -lipps -lippcore
```

   For more information about which Intel IPP libraries you need to link to, see Library Dependencies by Domain and Linking Options.

   For more information about the Component Directory Layout and the Unified Directory Layout, see Finding Intel IPP on Your System.
4. Run the application.

See Also

Automatically Linking Your Microsoft* Visual Studio* Project with Intel IPP

Configuring the Microsoft Visual Studio* IDE to Link with Intel® IPP

Setting Environment Variables

Library Dependencies by Domain

Linking Options

Dispatching

Intel® IPP Examples Directory Structure

Using Intel® IPP Examples

This section provides information on Intel IPP examples directory structure and examples build system.

- Intel® IPP Examples Directory Structure
- Building Intel® IPP Examples

Intel® IPP Examples Directory Structure

The Intel IPP package includes code examples, located in the `components_and_examples_<os>.zip` archive at the `<ipp directory>/``components/ subdirectory. The ``examples_core` subdirectory inside the archive contains the following files and directories:

Directory	Contents
`common`	Common code files for all examples
`documentation`	Documentation for the Intel IPP examples (`ipp-examples.html`)
`ipp_custom_dispatcher`	Custom dispatcher usage example
`ipp_fft`	Fast Fourier transformation example
`ipp_morphology`	Morphological reconstruction example

ipp_resize_mt	Image resizing example
ipp_thread	Example of external threading of Intel IPP functions

See Also

Finding Intel® IPP on Your System

Building Intel® IPP Examples

For building instructions refer to examples_core/documentation/ipp-examples.html provided with the /<ipp directory>/components/components_and_examples_<os>.zip | .tgz archive.

See Also

Intel® IPP Examples Directory Structure

Finding the Intel® IPP Documentation

You can find getting started instructions and a listing of all the available online documents with links in the get_started.htm file available in the following directory:

- On Windows* OS Component Directory Layout: C:\Program Files (x86)\Intel\oneAPI\ipp\<version>\shared\doc\ipp
- On Windows* OS Unified Directory Layout: C:\Program Files (x86)\Intel\oneAPI\<toolkit-version>\shared\doc\ipp
- On Linux* OS Component Directory Layout : /opt/intel/oneapi/ipp/<version>/share/doc/ipp
- On Linux* OS Unified Directory Layout: /opt/intel/oneapi/<toolkit-version>/share/doc/ipp

For more information on the differences between Component Directory Layout and Unified Directory layout, see Finding Intel® IPP on Your System .

Additional documentation on the Intel IPP examples (examples_core/documentation/ipp-examples.html) is available here:

- Windows: <ipp directory>/share/doc/ipp/components_and_examples_win.zip
- Linux: <ipp directory>/share/doc/ipp/components_and_examples_lin.tgz

The Intel® IPP Forum and knowledge base can be useful locations to search for questions not answered by the documents above.

See Also

Finding Intel® IPP on Your System

Programming Considerations

- Core and Support Functions
- Channel and Planar Image Data Layouts
- Regions of Interest
- Managing Memory Allocations
- Cache Optimizations
- Using Intel® Integrated Performance Primitives Platform-Aware Functions
- Using Integration Wrappers for Intel® Integrated Performance Primitives
- Notational Conventions
- Linking Your Application with Intel® Integrated Performance Primitives
- Using Intel® Integrated Performance Primitives Threading Layer (TL) Functions

- Using Custom Library Tool for Intel® Integrated Performance Primitives
- Programming with Intel® Integrated Performance Primitives in the Microsoft* Visual Studio* IDE
- Appendix: Intel® IPP Threading and OpenMP* Support
- Intel® Integrated Performance Primitives Theory of Operation

Page References

- Automatically Getting Performance Benefits with Intel® IPP Threading Layer
- Automatically Linking Your Microsoft* Visual Studio* Project with Intel IPP
- Avoiding Nested Parallelization
- Building Intel® IPP TL Libraries from Source Code
- Cache Optimizations
- Channel and Planar Image Data Layouts
- Configure Microsoft Visual Studio* to Link with Intel® Integrated Performance Primitives
- Core and Support Functions
- Data-domain
- Data Types
- Descriptor
- Dispatching
- Finding Intel® IPP TL Source Code Files
- Library Dependencies by Domain
- Linking Options
- Managing Memory Allocations
- Notational Conventions
- Operation Modes
- Parameters
- Primitive vs. Variant Name
- Regions of Interest
- System Requirements for Custom Library Tool
- Using Console Version of Custom Library Tool
- Using Integration Wrappers for Intel® Integrated Performance Primitives
- Using Intel® Integrated Performance Primitives Platform-Aware Functions
- Using a Shared L2 Cache
- Using the IntelliSense* Features
- Appendix: Intel® IPP Threading and OpenMP* Support
- Function Naming Conventions
- Intel® Integrated Performance Primitives Domain Details
- Intel® Integrated Performance Primitives Theory of Operation
- Linking Your Application with Intel® Integrated Performance Primitives
- Programming with Intel® Integrated Performance Primitives in the Microsoft* Visual Studio* IDE
- Using Custom Library Tool for Intel® Integrated Performance Primitives
- Using Intel® Integrated Performance Primitives Threading Layer (TL) Functions

Core and Support Functions

There are several general purpose functions that simplify using the library and report information on how it is working:

- `Init/GetCpuFeatures/ SetCpuFeatures/GetEnabledCpuFeatures`
- `GetStatusString`
- `GetLibVersion`
- `Malloc/Free`

Init/GetCpuFeatures/ SetCpuFeatures/GetEnabledCpuFeatures

The `ippInit` function detects the processor type and sets the dispatcher to use the processor-specific code of the Intel® IPP library corresponding to the instruction set capabilities available. If your application does not call the `ippInit` function, initialization of the library to the available instruction set capabilities is performed automatically with the first call of any Intel IPP function from the domain different from `ippCore`.

In some cases like debugging and performance analysis, you may want to get the data on the difference between various processor-specific codes on the same machine. Use the `ippSetCpuFeatures` function for this. This function sets the dispatcher to use the processor-specific code according to the specified set of CPU features. You can obtain features supported by CPU using `ippGetCpuFeatures` and obtain features supported by the currently dispatched Intel IPP code using `ippGetEnabledCpuFeatures`. If you need to enable support of some CPU features without querying the system (without CPUID instruction call), you must set the `ippCPUID_NOCHECK` bit for `ippSetCpuFeatures`, otherwise, only supported by the current CPU features are set.

The `ippInit`, `ippGetCpuFeatures`, `ippGetEnabledCpuFeatures`, and `ippSetCpuFeatures` functions are a part of the `ippCore` library.

GetStatusString

The `ippGetStatusString` function decodes the numeric status return value of Intel® IPP functions and converts them to a human readable text:

```
status= ippInit();
if( status != ippStsNoErr ) {
        printf("IppInit() Error:\n");
        printf("%s\n", ippGetStatusString(status) );
        return -1;
}
```

The `ippGetStatusString` function is a part of the `ippCore` library.

GetLibVersion

Each domain has its own `GetLibVersion` function that returns information about the library layer in use from the dispatcher. The code snippet below demonstrates the usage of the `ippiGetLibVersion` from the image processing domain:

```
const IppLibraryVersion* lib = ippiGetLibVersion();
printf("%s %s %d.%d.%d.%d\n", lib->Name, lib->Version,
lib->major, lib->minor, lib->majorBuild, lib->build);
```

Use this function in combination with `ippInitCpu` to compare the output of different implementations on the same machine.

Malloc/Free

Intel IPP functions provide better performance if they process data with aligned pointers. Intel IPP provides the following functions to ensure that data is aligned appropriately - 16-byte for CPU that does not support Intel® Advanced Vector Extensions (Intel® AVX) instruction set, 32-byte for Intel AVX and Intel® Advanced Vector Extensions 2 (Intel® AVX2), and 64-byte for Intel® Many Integrated Core Instructions.

```
void* ippMalloc(int length)
void ippFree(void* ptr)
```

> **NOTE** Support for Intel ® Advanced Vector Extensions (Intel ® AVX) has been deprecated and will be removed in a future release.

The `ippMalloc` function provides appropriately aligned buffer, and the `ippFree` function frees it.

The signal and image processing libraries provide `ippsMalloc` and `ippiMalloc` functions, respectively, to allocate appropriately aligned buffer that can be freed by the `ippsFree` and `ippiFree` functions.

NOTE

- When using buffers allocated with routines different from Intel IPP, you may get better performance if the starting address is aligned. If the buffer is created without alignment, use the `ippAlignPtr` function.

For more information about the Intel IPP functions see the *Intel® Integrated Performance Primitives for Intel® Architecture Developer Reference* available in Intel® Software Documentation Library.

See Also

Cache Optimizations

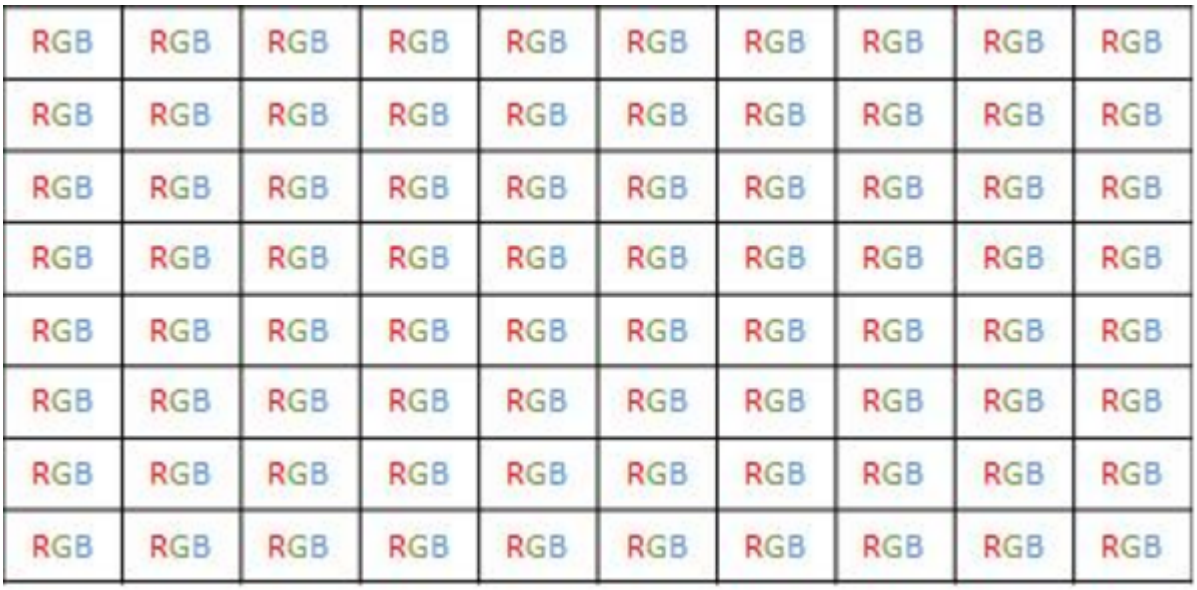
Intel® Software Documentation Library

Channel and Planar Image Data Layouts

Intel® IPP functions operate on two fundamental data layouts: channel and planar.

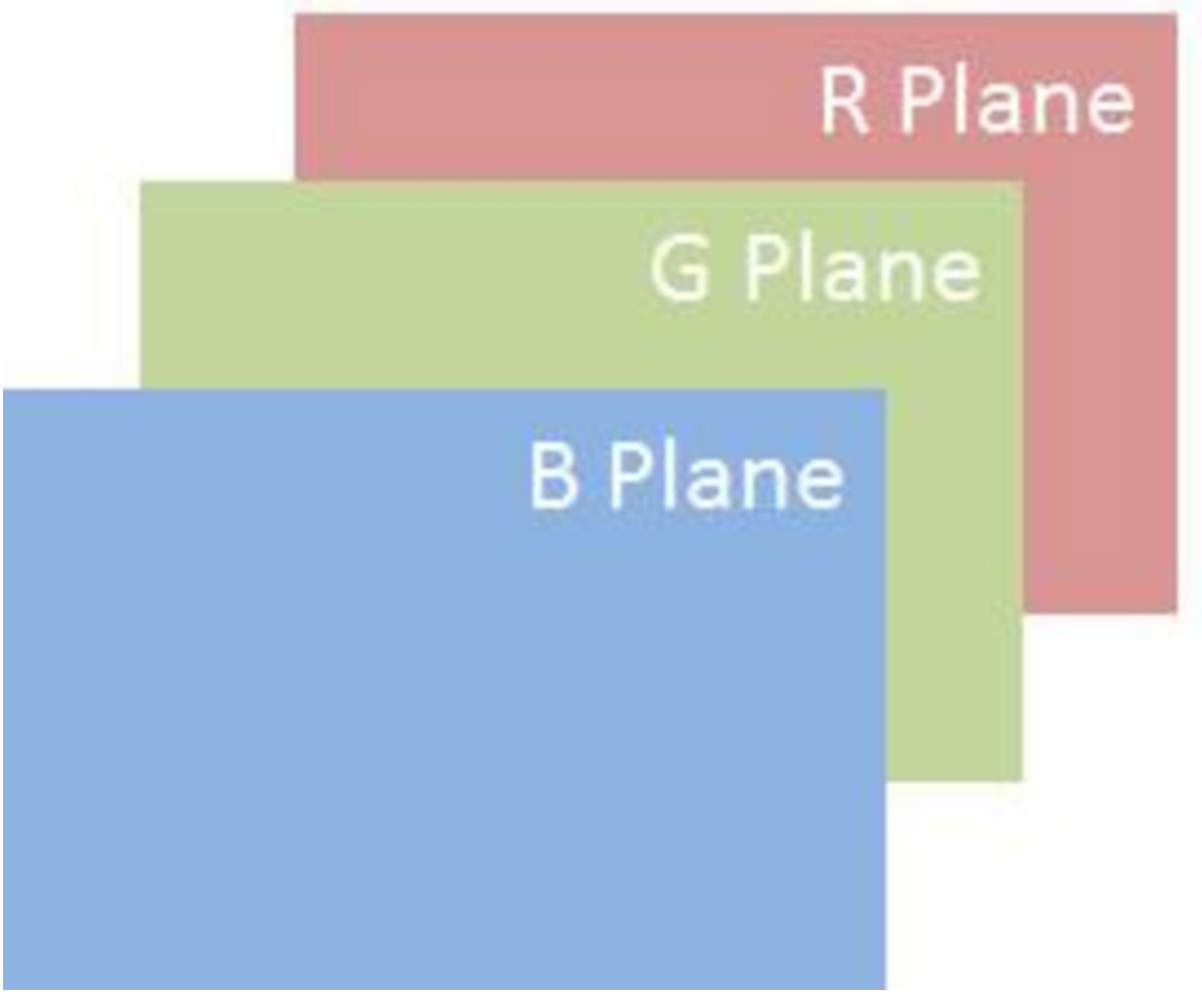
In channel format, all values share the same buffer and all values for the same pixel position are interleaved together. Functions working with channel data have a `_Cn` descriptor, where `n` can take one of the following values: 1, 2, 3, or 4. The figure below shows 24 bit per pixel RGB data, which is represented as `_C3`.

RGB data in `_C3` layout



For planar format, there is one value per pixel but potentially several related planes. Functions working with planar data have a `_Pn` descriptor, where `n` can take one of the following values: 1, 2, 3, or 4. The figure below shows 24 bit per pixel RGB data represented as `_P3`.

RGB data in `_P3` layout



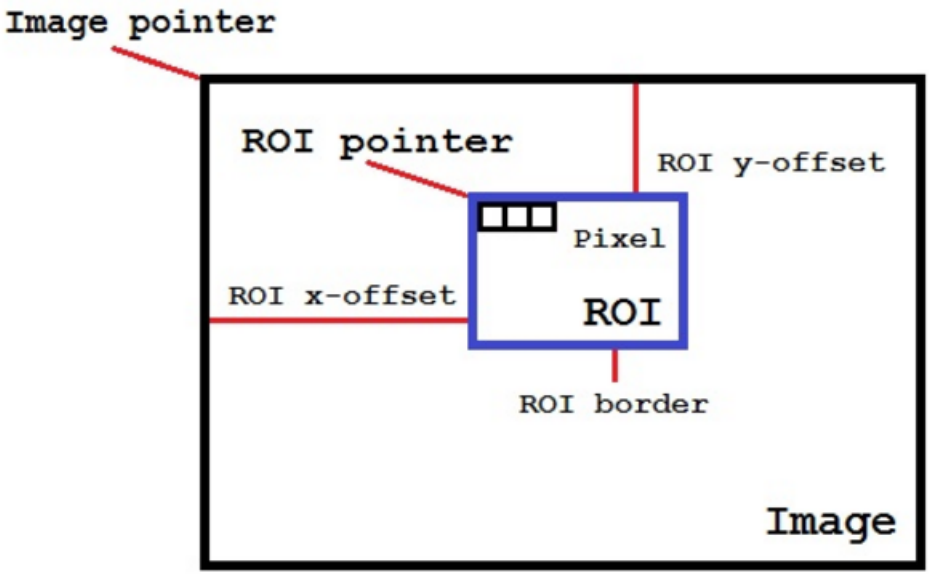
NOTE For many video and image processing formats planes may have different sizes.

Regions of Interest

Many Intel® IPP image processing functions operate with a region of interest (ROI). These functions include an `R` descriptor in their names.

A ROI can be the full image or a subset. This can simplify thread or cache blocking.

Many functions sample a neighborhood and cannot provide values for an entire image. In this case a ROI must be defined for the subset of the destination image that can be computed.



Managing Memory Allocations

In Intel® Integrated Performance Primitives (Intel® IPP) functions, the areas in memory allocated for the source and destination data must not overlap, except for functions that have the descriptor `I` in their name. Only the functions that have the descriptor `I` (see Descriptors) in their name can have the same area in memory allocated for both the source and destination data. Intel IPP does not guarantee correct behavior and results for not-in-place functions that are used in in-place mode.

Depending on the implementation layer and the specific operation parameters, some Intel IPP functions need varying amounts of memory for internal structures and working buffers. To address this, follow the steps below:

1. Compute the size of the required buffer using the `<function base name>GetSize` function (some functions have `GetBufSize` or `GetBufferSize` in their name instead of `GetSize`).
2. Set up any buffers needed for initialization. For more information, see the section Setting up Buffers below.
3. Initialize the specification or state structure for the operation using `<function base name>Init` function. For more information about the specification and state structures, see the section Specification and State Structures below.
4. Free the buffers need for initialization only (the ones you set up in step 2).
5. Set up working buffers for the main operation. For more information, see the section Setting up Buffers below.
6. Do the main operation.
7. Free the specification or state buffers that you set up in step 3 and the working buffers that you set up in step 5.

If you use several Intel IPP functions with the `pBuffer` parameter (external memory buffer), for better efficiency and performance it is recommended to call all `<function base name>GetSize` functions in one single location within your application and allocate only one buffer that has the largest size. This approach ensures optimal use of system memory and all cache levels.

Setting up Buffers

In this document, "setting up a buffer" refers to allocating the required amount of memory and providing a pointer to this memory to the Intel IPP function you are calling. For better performance, you should allocate aligned memory buffers, where the alignment factor depends on the architecture and should be at least 16 bytes for Intel® Streaming SIMD Extensions, 32 bytes for Intel® Advanced Vector Extensions, and 64 bytes for Intel® Advanced Vector Extensions 512 Foundation instruction sets.

To set up aligned memory buffers, it is recommended to use the `ipp<domain letter>Malloc_<IPP data type>` functions; these functions always provide memory buffers with the required alignment.

NOTE Intel IPP functions do not allocate any memory internally. You must manually allocate and free previously allocated memory, that is required for your Intel IPP functions at the application level. `ipp<domain letter>Malloc_<IPP data type>` and `ipp<domain letter>Free` functions allocate and free a memory block aligned to 64-byte boundary for elements of different data types. Not aligned memory allocation could cause not reproducible performance and precision results.

Specification and State Structures

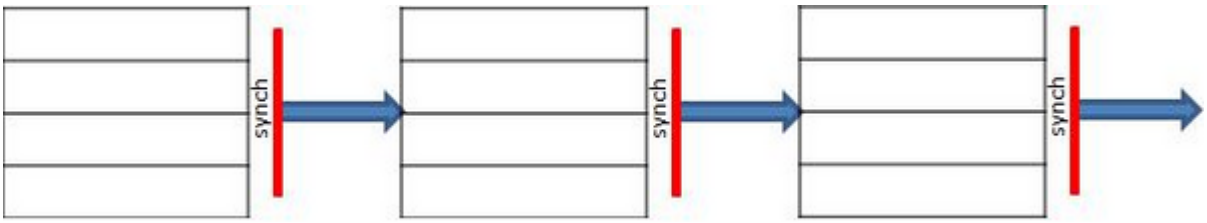
Specification, or spec, structures are `const`; an instance of a specification structure does not change between Intel IPP function calls. Therefore, you can use one instance of a specification structure simultaneously in different application threads for the same operation.

State structures are not `const`; they always contain the state of an intermediate computation stage of an Intel IPP function. Therefore, you can use a single instance of a state structure only for consecutive operations. In the case of a threaded application, each thread must have its own instance of the state structure.

Cache Optimizations

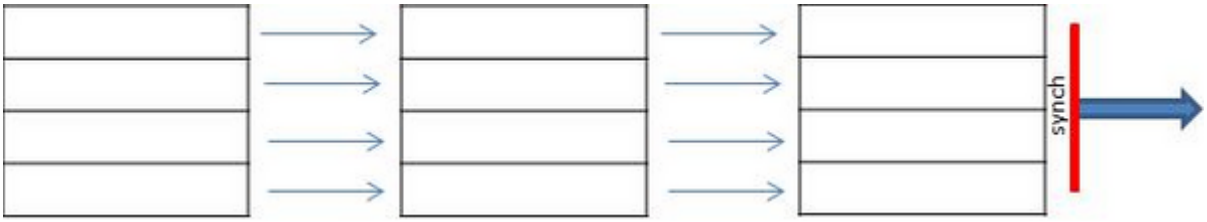
To get better performance, work should be grouped to take advantage of locality in the lowest/fastest level of cache possible. This is the same for threading or cache blocking optimizations.

For example, when operations on each pixels in an image processing pipeline are independent, the entire image is processed before moving to the next step. This may cause many inefficiencies, as shown in a figure below.



In this case cache may contain wrong data, requiring re-reading from memory. If threading is used, the number of synchronization point/barriers is more than the algorithm requires.

You can get better performance after combining steps on local data, as shown in a figure below. In this case each thread or cache-blocking iteration operates with ROIs, not full image.



NOTE It is recommended to subdivide work into smaller regions considering cache sizes, especially for very large images/buffers.

Using Intel® Integrated Performance Primitives Platform-Aware Functions

Intel® Integrated Performance Primitives (Intel® IPP) library provides so-called platform-aware functions for signal and image processing. While the rest of Intel IPP functions support only signals or images of 32-bit integer size, Intel IPP platform-aware functions work with 64-bit object sizes if it is supported by the target platform.

The API of platform-aware functions is similar to the API of other Intel IPP functions and has only slight differences. You can distinguish Intel IPP platform-aware functions by the `L` suffix in the function name, for example, `ippiAdd_8u_C1RSfs_L`. With Intel IPP platform-aware functions you can overcome 32-bit size limitations.

Intel IPP platform-aware functions are declared in separate header files with the `_l` suffix, for example, `ippi_l.h`. However, you do not have to additionally include these headers in your application because they are already included in standard Intel IPP headers (without the `_l` suffix). Platform-aware functions cover only the functionality that is implemented in standard Intel IPP functions, and can be considered as additional flavors to the existing functions declared in standard Intel IPP headers.

Using Integration Wrappers for Intel® Integrated Performance Primitives

Intel® Integrated Performance Primitives (Intel® IPP) Integration Wrappers aggregate Intel IPP functionality in easy-to-use functions and help to reduce effort required to integrate Intel IPP into your code.

Integration Wrappers consist of C and C++ interfaces:

- **C interface** aggregates Intel IPP functions of similar functionality with various data types and channels into one function. Initialization steps required by several Intel IPP functions are implemented in one initialization function for each functionality. To reduce the size of your code and save time required for integration, the wrappers handle all memory management and Intel IPP function selection routines.
- **C++ interface** wraps around the C interface to provide default parameters, easily initialized objects as parameters, exception handling, and objects for complex Intel IPP functions with automatic memory management for specification structures.

In general, Integration Wrappers are designed to improve user experience with threading of Intel IPP functions and tiling.

Integration Wrappers are provided as a separate download. For more information about the main concepts, usage, and implementation details, refer to the *Developer Guide and Reference for Intel IPP Integration Wrappers* document available with the Integration Wrappers package.

Notational Conventions

The code and syntax used in this document for function and variable declarations are written in the ANSI C style. However, versions of Intel IPP for different processors or operating systems may, of necessity, vary slightly.

This document uses the following notational conventions:

Convention	Explanation	Example
`THIS TYPE STYLE`	Used in the text for the Intel IPP constant identifiers.	`IPPI_MAX_64S`
`This type style`	Mixed with the uppercase in structure names; also used in function names, code examples and call statements.	`IppLibraryVersion`, `void ippsFree()`
``*This type style*``	Parameters in function prototypes and parameters description.	`value`, `srcStep`
`x(n)` and `x[n]`	Used to represent a discrete 1D signal. The notation `x(n)` refers to a conceptual signal, while the notation `x[n]` refers to an actual vector. Both of these are annotated to indicate a specific finite range of values.	`x[n], 0 ≤ n < len` Typically, the number of elements in vectors is denoted by `len`. Vector names contain square brackets as distinct from vector elements with current index `n`. The expression `pDst[n] = pSrc[n] + val` implies that each element `pDst[n]` of the vector `pDst` is computed for each `n` in the range from 0 to `len - 1`. Special cases are regarded and described separately.
Ipp<data-domain> and `Ipp` prefixes	All structures and enumerators, specific for a particular data-domain have the `Ipp<data-domain>` prefix, while those common for entire Intel IPP software have the `Ipp` prefix.	`IppsROI`, `IppLibraryVersion`

See Also

Data-Domain

Linking Your Application with Intel® Integrated Performance Primitives

This section discusses linking options available in Intel® Integrated Performance Primitives (Intel® IPP).

The Intel IPP library supports the following linking options:

- Single-threaded dynamic
- Single-threaded static
- Threading Layer static
- Threading Layer dynamic

See the following topics for more information:

- Linking Options
- Automatically Linking Your Microsoft* Visual Studio* Project with Intel IPP

Linking Options

Intel® Integrated Performance Primitives (Intel® IPP) is distributed as:

- **Static library:** static linking results in a standalone executable
- **Dynamic/shared library:** dynamic linking defers function resolution until runtime and requires that you bundle the redistributable libraries with your application

The following table provides description of libraries available for linking.

	Single-threaded (non-threaded)	**Threading Layer** (externally threaded)
Description	Suitable for application-level threading	Implementation of application-level threading depends on single-threaded libraries
Static linking	Windows* OS: `mt` suffix in a library name (`ipp<domain>mt.lib`) Linux* OS: no suffix in a library name (`libipp<domain>.a`)	Windows* OS: `_mt_tl_<threading_sfx>` suffix in a library name (`ipp<domain>mt_tl_<threading_sfx>.lib`) Linux* OS: _tl_<threading_sfx> suffix in a library name (`libipp<domain>_tl_<threading_sfx>.a`) + single-threaded libraries dependency, where <threading_sfx> is one of {tbb, omp}
Dynamic Linking	Default (no suffix) Windows* OS: `ipp<domain>.dll` Linux* OS: `libipp<domain>.so`	`_tl_<threading_sfx>` suffix Windows* OS: `ipp<domain>_tl_<threading_sfx>.dll` Linux* OS: `libipp<domain>_tl_<threading_sfx>.so` + single-threaded library dependency, where <threading_sfx> is one of {tbb, omp}

The binaries founf in main package after installation: - `<ipp directory>/lib` (static - **Windows* & Linux* OS**, dynamic - **Linux* OS** only) - `<ipp directory>\bin` (dynamic - - **Windows* OS** only)

To switch between Intel IPP libraries, set the path to the preferred library in system variables or in your project, for example:

- **Windows* OS**:

  Intel® 64: `SET LIB=<ipp directory>\lib`
- **Linux* OS**:

  Single-threaded Intel® 64: `gcc<options>-L<ipp directory>/lib`

> **NOTE** On Linux* OS, Intel IPP library depends on the following Intel® C++ Compiler runtime libraries: `libirc.a`, `libsvml.a`, and `libimf.a`. You should add a link to these libraries into your project. You can find these libraries in `<intel compiler directory>/lib` folders.
> Threading Layer depends on the OpenMP* or Intel® Threading Building Blocks (Intel® TBB) library according to the selected threading type. You can find these libraries in `<intel compiler directory>/lib` or `<tbb directory>/lib` folders.

See Also

Automatically Linking Your Microsoft* Visual Studio* Project with Intel IPP

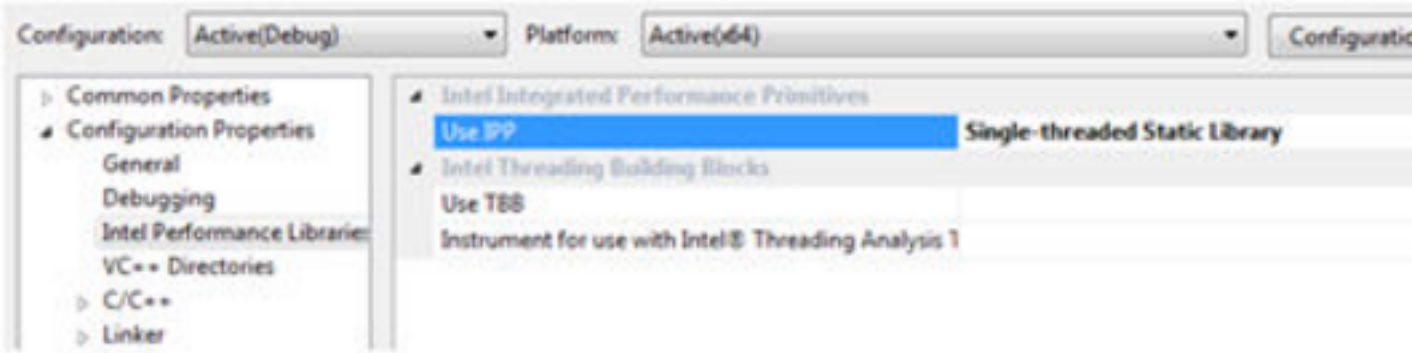
Configuring the Microsoft Visual Studio* IDE to Link with Intel® IPP

Library Dependencies by Domain

Automatically Linking Your Microsoft* Visual Studio* Project with Intel IPP

After a default installation of the Intel® IPP, you can easily configure your project to automatically link with Intel IPP. Configure your Microsoft* Visual Studio* project for automatic linking with Intel IPP as follows:

1. Go to **Project**> **<project_name> Properties**> **Configuration Properties**> **Intel Libraries for oneAPI**.
2. Change the **Use Intel® IPP** property setting by selecting one of the options to set the include directories and the linking model, as shown on the screenshot below. See linking options for more information.



Using Intel® Integrated Performance Primitives Threading Layer (TL) Functions

Intel® Integrated Performance Primitives (Intel® IPP) library provides threading layer (TL) functions for image processing. Intel IPP TL functions are visual examples of external threading for Intel IPP functions. Taking advantage of multithreaded execution and tile processing, Intel IPP TL functions enable you to overcome 32-bit size limitations.

TL functions are provided as:

- **Pre-built binaries:**
  - Header files have the `_tl` suffix and can be found in: `<ipp directory>/include`
  - Library files for use with Intel® OpenMP have the `_tl_omp` suffix and can be found in: `<ipp directory>/lib/<arch>/tl/openmp`
  - Library files for use with Intel® oneTBB have the `_tl_tbb` suffix and can be found in: `<ipp directory>/lib/<arch>/tl/tbb`
- **Source code samples:** the source code and corresponding header files are available in the `components_and_examples_<os>.zip` archive inside the `<ipp directory>/components` subdirectory. For more information about the archive contents and source code building instructions, refer to Finding Intel® IPP TL Source Code Files and Building Intel® IPP TL Libraries from Source Code, respectively.

The API of TL functions is similar to the API of other Intel IPP functions and has only slight differences. You can distinguish Intel IPP TL functions by the `_LT` or `_T` suffix in the function name, for example, `ippiAdd_8u_C1RSfs_LT`. Intel IPP TL functions are implemented as wrappers over Intel IPP functions by using tiling and multithreading with OpenMP* or the Intel® Threading Building Blocks. For implementation details, please see the corresponding source code files.

See Also

Finding Intel® IPP TL Source Code Files

Building Intel® IPP TL Libraries from Source Code

Using Intel® Integrated Performance Primitives Platform-Aware Functions

- Finding Intel® IPP TL Source Code Files
- Building Intel® IPP TL Libraries from Source Code
- Automatically Getting Performance Benefits with Intel® IPP Threading Layer

Finding Intel® IPP TL Source Code Files

You can find the Intel IPP TL source code files in the `components_and_examples_<os>.zip` archive available in the `<ipp directory>/components` subdirectory. The library source code and header files are located in the `interfaces/tl` subdirectory.

Building Intel® IPP TL Libraries from Source Code

The threading layer (TL) libraries' source code and the `tl_resize` example are in the following file path: `<oneapi>/share/doc/ipp/components_and_examples_<os>/components/interfaces/tl`. Before building an application that uses TL, ensure that the `IPPROOT` environment variable is set correctly and points to the Intel® Integrated Performance Primitives (Intel® IPP) library location. For more information, see Setting Environment Variables.

The steps to build Intel IPP TL libraries and the `tl_resize` example for Windows* and Linux* are outlined below.

Windows

The `tl_resize` example uses OpenGL* rendering to display results. To disable the rendering part of `tl_resize`, remove the `ENABLE_RENDERING` macro from the preprocessors definitions.

To build Intel IPP TL libraries:

1. Open the `tl.sln` file in Microsoft Visual Studio*.
2. Choose the required configuration in the solution and build the solution using the **Build** command. The example will be linked with the newly built TL libraries from the same solution.

> **NOTE** To build TL libraries on the Intel® Threading Building Blocks (Intel® TBB) library, you need to install the Intel TBB library (for the Intel IPP standalone package).

Linux

The `tl_resize` example uses OpenGL rendering to display results. This requires the following packages to be installed:

- `libx11-dev`
- `libgl1-mesa-dev`

Execute the following commands using GCC 4.0 or higher:

- To build TL libraries:

```
make libs [ARCH=intel64] [CONF=release|debug] [TBBROOT=]
```

- To build the `tl_resize` example and TL libraries:

```
make all [ARCH=intel64] [CONF=release|debug] [RENDERER=0|1] [TBBROOT=]
```

NOTE If `TBBROOT` is set to the Intel® TBB installation root, TL libraries will be built with the TBB support. In this case, you need to install Intel TBB library (for the Intel IPP standalone package). If `TBBROOT` is set to nothing, the OpenMP* support will be used.

See Also

Setting Environment Variables

Automatically Getting Performance Benefits with Intel® IPP Threading Layer

If you already have an application that uses Intel® IPP Single-Threaded libraries, you can easily get performance benefits using the IPP Threading Layer (OpenMP* or TBB* version) without modifying your source code.

To enable IPP Threading Layer support in your application:

1. Enable the preprocessor option `IPP_ENABLED_THREADING_LAYER_REDEFINITIONS`.
2. Add Threading Layer libraries and dependencies (TBB* or OpenMP* library) into the link line. For details about Threading Layer libraries linking and location, refer to Linking Options
3. Rebuild the application.

As a result, IPP APIs used in your application are redefined with IPP Threading Layer APIs when implemented and you receive performance benefits from external threading used in the IPP Threading Layer.

See Also

Linking Options

Using Custom Library Tool for Intel® Integrated Performance Primitives

With the Intel® Integrated Performance Primitives (Intel® IPP) Custom Library Tool, you can build your own dynamic library containing only the Intel IPP functionality that is necessary for your application.

The use of custom libraries built with the Custom Library Tool provides the following advantages:

- **Package size**. Your package may be much smaller if linked with a custom library because standard dynamic libraries contain all optimized versions of Intel IPP functions and a dispatcher. The following table compares the contents and size of packages for an end-user application linked with a custom dynamic library and an application linked with the standard Intel IPP dynamic libraries:

OS	Application linked with custom DLL	Application linked with Intel IPP dynamic libraries
Windows*	`ipp_test_app.exe` `ipp_custom_lib.dll` **Package size: 0.1 Mb**	`ipp_test_app.exe` `ippcore.dll` `ippi.dll` `ippid1.dll` `ippik0.dll` `ippil9.dll` `ippiy8.dll` `ipps.dll` `ippsd1.dll` `ippsk0.dll` `ippsl9.dll` `ippsy8.dll` **Package size: 100.1 Mb**
Linux* OS	`ipp_test_app` `ippi_custom_lib.so` **Package size: 0.1 Mb**	`ipp_test_app` `libippcore.so` `libippi.so` `libippid1.so` `libippik0.so` `libippil9.so` `libippiy8.so` `libipps.so` `libippsd1.so` `libippsk0.so` `libippsl9.so` `libippsy8.so` **Package size: 106.1 Mb**

- **Smooth transition to a higher version of Intel IPP**. You can build the same custom dynamic library from a higher version of Intel IPP and substitute the libraries in your application without relinking.

> **NOTE** The current Python* version of the Intel IPP Custom Library Tool supports only the host-host configuration. It does not currently support the host-target configuration.

- System Requirements for Custom Library Tool
- Operation Modes
- Using Console Version of Custom Library Tool

System Requirements for Custom Library Tool

Recommended hardware:

- System based on the 2nd Generation Intel® Core processor or newer.

Software requirements:

- Visual Studio* 2015 (or higher).

> **NOTE** If a custom library must support Intel® Control-Flow Enforcement Technology (Intel® CET), use Visual Studio 2019 (or higher).

- Python* 3.7.

Operation Modes

The Custom Library Tool (CLT) has two operation modes:

- **Build mode** (`--generate=none` or no `--generate` option): The tool automatically sets the environment and builds a dynamic library.
- **Generation mode**: The tool generates and saves a custom build script using one of two dispatchers:
  - **Dynamic dispatcher** (`--generate`, `--generate=dynamic`): Library initialization happens before a call of any custom dispatcher function:
    - Manually when generating a custom dispatcher script without building a custom library.
    - In dynamic/shared library initialization function (generated automatically by CLT) when building a custom library.
  - **Static dispatcher** (`--generate=static`): Enables custom dispatcher functions to initialize a library (if it hasn't been initialized yet) before the call of the dispatched function.

Using Console Version of Custom Library Tool

Follow the steps below to build a custom dynamic library using console version of the Custom Library Tool:

1. Define a list of Intel IPP functions that the Intel IPP Custom Library Tools should export to your custom dynamic library. See the example text file `my_functions.txt` below:

```
ippsGetLibVersion
ippsAdd_16s
ippsAdd_32f
```

2. Run `python main.py` with the following parameters:

`-c--console`	Does nothing. The option is left for compatibility with previous versions of Custom Library Tool.

`-g--generate-g={none|dynamic|static}--generate={none|dynamic|static}`	Enables the script generation mode (the build mode is used by default). For more infomation, see Operation Modes. If no arguments are provided to the options `-g` / `--generate`, dynamic dispatcher generation mode is used.
`-n <name>--name <name>`	Output library name.
`-p <path>--path <path>`	Path to the output directory.
`-root <root_path>`	Path to Intel IPP package root directory
`-f <function>--function <function>`	Name of a function to be included into your custom dynamic library.
`-ff <functions_file>--functions_file <functions_file>`	Path to a file with a list of functions to be included into your custom dynamic library (the `-f` or `--function` flag can be used to add functions on the command line).
`-arch={intel64}--architecture={intel64}`	Enables all actions for the Intel® 64 architecture. This is enabled by default. It is left for compatibility with previous versions of Custom Library Tool.
`-tl={tbb|openmp}--threading_layer_type={tbb|openmp}`	Sets Intel TBB or OpenMP* as the threading layer.
`-d <cpu_set>--custom_dispatcher <cpu_set>`	Sets the exact list of CPUs that must be supported by custom dynamic library and generates a C-file with the custom dispatcher.
`--prefix <prefix>`	Renames selected functions with specified prefix in the custom dispatcher files.
`-h--help`	Prints command help.

For example:

```
# Generate build scripts
# with the output shared library name "my_custom_lib"
# with functions defined in the "my_functions.txt" file
# optimized only for processors with Intel® Advanced Vector Extensions 512 (Intel® AVX-512)
# using Intel IPP libraries

python3 clt.py \
  -g \
  -n my_custom_lib \
  -p "~/my_project" \
  -ff "~/my_project/my_functions.txt" \
  -d avx512bw
```

Programming with Intel® Integrated Performance Primitives in the Microsoft* Visual Studio* IDE

This section provides instructions on how to configure your Microsoft* Visual Studio* IDE to link with the Intel® IPP, explains how to access Intel IPP documentation and use IntelliSense* Sense features.

- Configure Microsoft Visual Studio* to Link with Intel® Integrated Performance Primitives
- Using the IntelliSense* Features

Configure Microsoft Visual Studio* to Link with Intel® Integrated Performance Primitives

The steps for configuring Microsoft Visual C/C++* to link with Intel® Integrated Performance Primitives (Intel® IPP) depend on an installation of the C++ Integration(s) in Microsoft Visual Studio*. If you installed the integration component, see Automatically Linking Your Microsoft* Visual Studio* Project with Intel IPP. If you did not install the integration component, or need more control over linking Intel IPP libraries, you can configure Microsoft Visual Studio* with the following steps.

> **NOTE** The menu selections of different versions of Microsoft Visual Studio* can vary, but the steps should apply to all versions.

1. In Solution Explorer, right-click your project and click **Properties**.
2. Select **Configuration Properties**> **VC++ Directories** and set the following from the **Select directories for** drop-down menu:
   - Select **Include Files** and add the directory for the Intel IPP include files (the default is `<ipp directory>include`)
   - Select **Library Files** and add the directory for the Intel IPP library files (the default is `<ipp directory>lib`)
   - Select **Executable Files** and add the directory for the Intel IPP executable files (the default is `<install_dir>redist<arch>\`)

Using the IntelliSense* Features

Intel IPP supports two Microsoft* Visual Studio IntelliSense* features that support language references: Complete Word and Parameter Info.

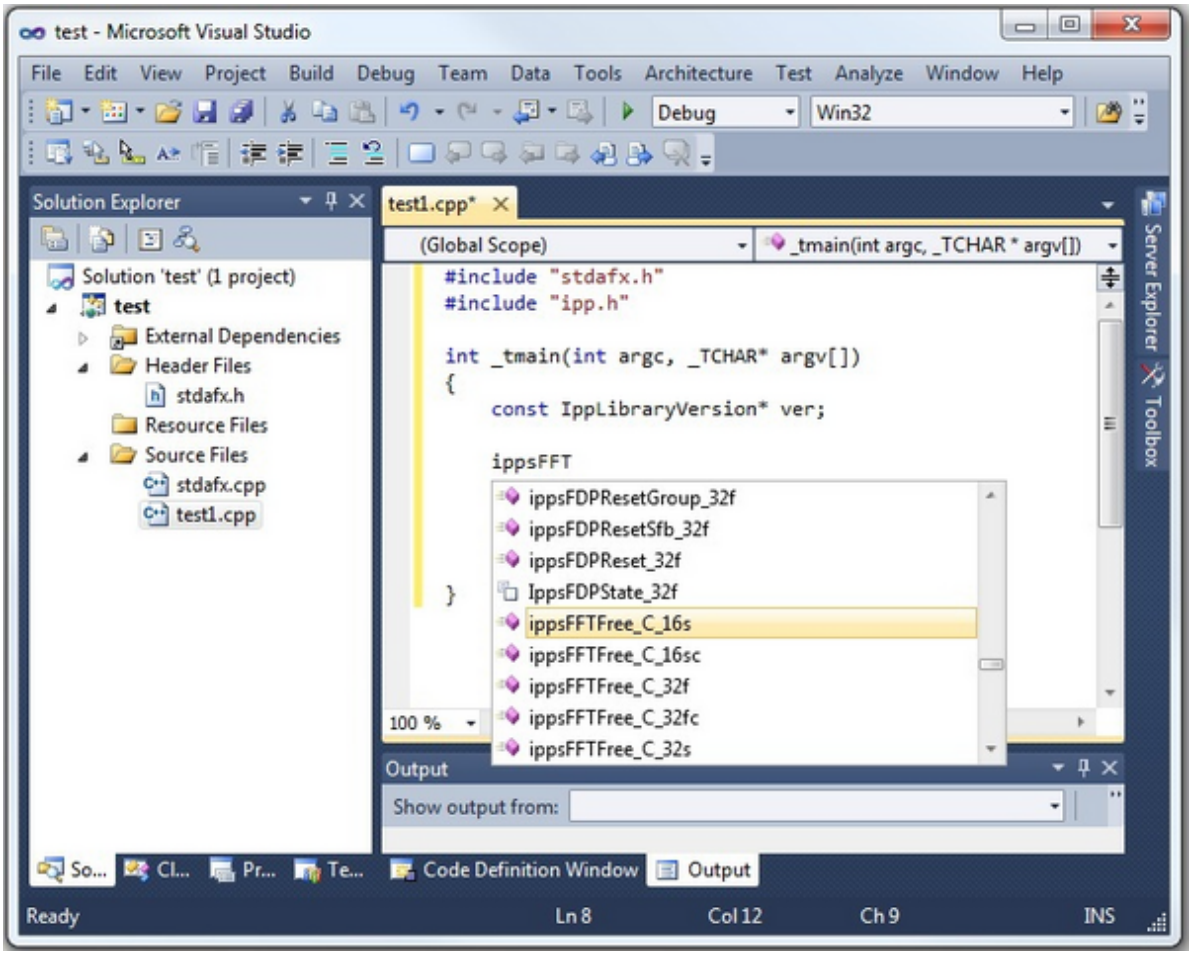
> **NOTE** Both features require header files. Therefore, to benefit from IntelliSense, make sure the path to the include files is specified in the Visual Studio solution settings. On how to do this, see Configuring the Microsoft Visual Studio* IDE to Link with Intel® IPP.

Complete Word

For a software library, the *Complete Word* feature types or prompts for the rest of the name defined in the header file once you type the first few characters of the name in your code.

Provided your C/C++ code contains the include statement with the appropriate Intel IPP header file, to complete the name of the function or named constant specified in the header file, follow these steps:

1. Type the first few characters of the name (for example, `ippsFFT`).
2. Press **Alt** + **RIGHT ARROW** or **Ctrl** + **SPACEBAR** If you have typed enough characters to eliminate ambiguity in the name, the rest of the name is typed automatically. Otherwise, the pop-up list of the names specified in the header file opens - see the figure below.



3. Select the name from the list, if needed.

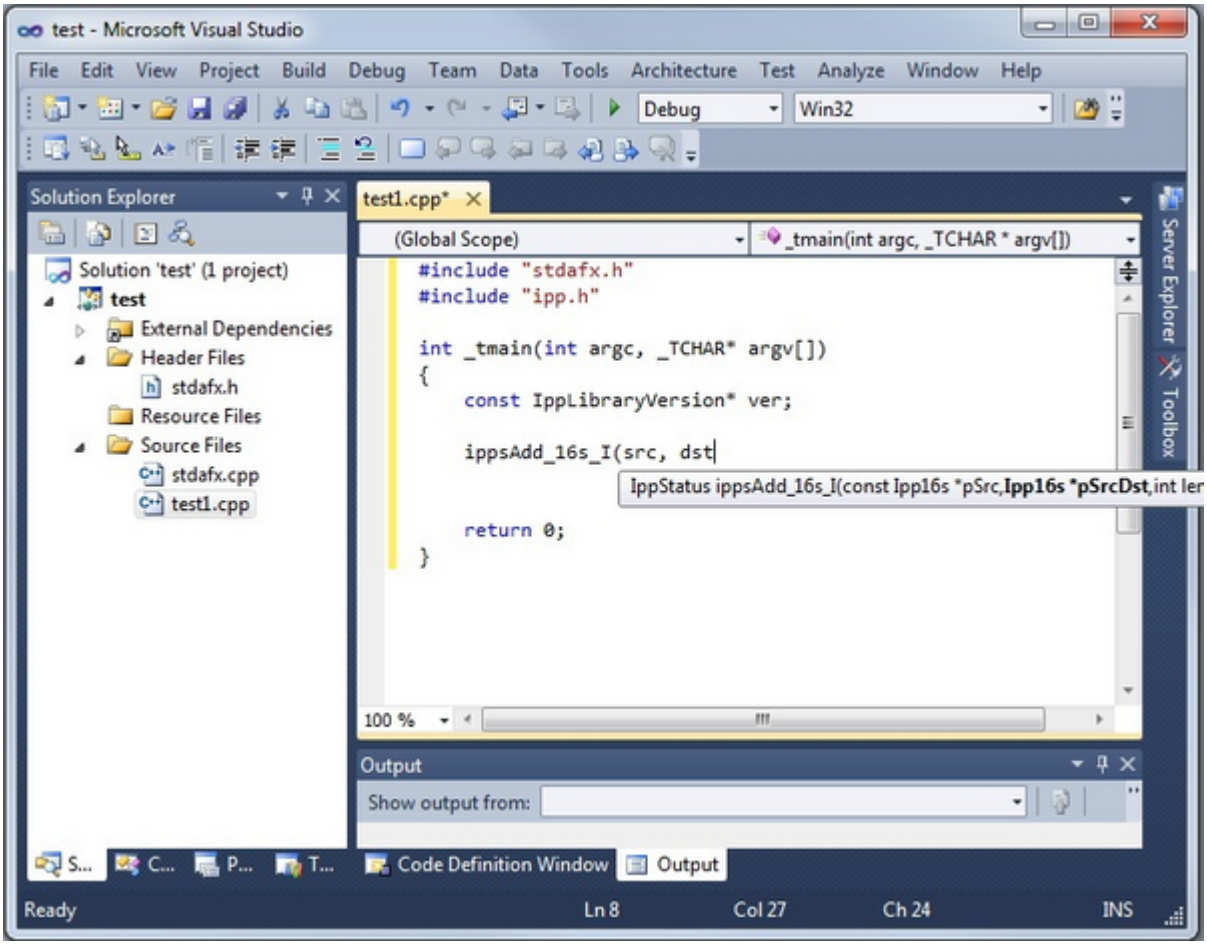
Parameter Info

The *Parameter Info* feature displays the parameter list for a function to give information on the number and types of parameters.

To get the list of parameters of a function specified in the header file, follow these steps:

1. Type the function name
2. Type the opening parenthesis

A tooltip appears with the function API prototype, and the current parameter in the API prototype is highlighted - see the figure below.



See Also

Configuring the Microsoft Visual Studio* IDE to Link with Intel® IPP

Appendix: Intel® IPP Threading and OpenMP* Support

All Intel® Integrated Performance Primitives functions are thread-safe. They support multithreading in both dynamic and static libraries and can be used in multi-threaded applications. However, if an application has its own threading model or if other threaded applications are expected to run at the same time on the system, it is strongly recommended to use non-threaded/single-threaded libraries.

- Using a Shared L2 Cache
- Avoiding Nested Parallelization

Using a Shared L2 Cache

Several functions in the signal processing domain are threaded on two threads intended for the Intel® Core™ 2 processor family, and make use of the merged L2 cache. These functions (single and double precision `FFT`, `Div, and Sqrt`) achieve the maximum performance if both two threads are executed on the same die. In this case, the threads work on the same shared L2 cache. For processors with two cores on the die, this condition is satisfied automatically. For processors with more than two cores, set the following OpenMP* environmental variable to avoid performance degradation:

```
KMP_AFFINITY=compact
```

Avoiding Nested Parallelization

Nested parallelization may occur if you use a threaded Intel IPP function in a multithreaded application. Nested parallelization may cause performance degradation because of thread oversubscription.

For applications that use OpenMP threading, nested threading is disabled by default, so this is not an issue.

However, if your application uses threading created by a tool other than OpenMP*, you must disable multi-threading in the threaded Intel IPP function to avoid this issue.

Disabling Multi-threading (Recommended)

The best option to disable multi-threading is to link your application with the Intel® IPP single-threaded (non-threaded) libraries included in the default package and discontinue use of the separately downloaded multi-threaded versions.

You may also call the `ippSetNumThreads_T` function with parameter `1`, but this method may still incur some OpenMP* overhead.

Intel® Integrated Performance Primitives Theory of Operation

This section discusses dispatching of the Intel® Integrated Performance Primitives (Intel® IPP) libraries to specific processors, provides functions and parameters naming conventions, and explains the data types on which Intel IPP performs operations. This section also provides Intel IPP domain details, including existing library dependencies by domain.

- Dispatching
- Function Naming Conventions
- Intel® Integrated Performance Primitives Domain Details

Dispatching

Intel® IPP uses multiple function implementations optimized for various CPUs. Dispatching refers to detection of your CPU and selecting the corresponding Intel IPP binary path. For example, the `ippie9` library in the `/bin` directory contains the image processing libraries optimized for 64-bit applications on processors with Intel® Advanced Vector Extensions (Intel® AVX) enabled such as the 2 nd Generation Intel® Core™ processor family.

A single Intel IPP function, for example `ippsCopy_8u()`, may have many versions, each one optimized to run on a specific Intel® processor with specific architecture, for example, the 64-bit version of this function optimized for the 2 nd Generation Intel® Core™ processor is `e9_ippsCopy_8u()`, and version optimized for 64-bit applications on processors with Intel® Streaming SIMD Extensions 4.2 (Intel® SSE 4.2) is `y8_ippsCopy_8u()`. This means that a prefix before the function name determines CPU model. However, during normal operation the dispatcher determines the best version and you can call a generic function (`ippsCopy_8u` in this example).

Intel® IPP is designed to support application development on various Intel® architectures. This means that the API definition is common for all processors, while the underlying function implementation takes into account the strengths of each hardware generation.

By providing a single cross-architecture API, Intel IPP enables you to port features across Intel® processor-based desktop, server, and mobile platforms. You can use your code developed for one processor architecture for many processor generations.

The following table shows processor-specific codes that Intel IPP uses:

Intel® 64 architecture	Windows*	Linux* OS	Description

m7	•	•	Optimized for processors with Intel SSE3
n8	•	•	Optimized for processors with Supplemental Streaming SIMD Extensions 3 (SSSE3)
y8	•	•	Optimized for processors with Intel SSE4.2
e9	•	•	Optimized for processors with Intel® Advanced Vector Extensions (Intel® AVX) and Intel® Advanced Encryption Standard New Instructions (Intel® AES-NI)
l9	•	•	Optimized for processors with Intel® Advanced Vector Extensions 2 (Intel® AVX2)
k0	•	•	Optimized for processors with Intel® Advanced Vector Extensions 512 (Intel® AVX-512)

Function Naming Conventions

Intel IPP functions have the same naming conventions for all domains.

Function names in Intel IPP have the following general format:

```
ipp<data-domain><name>_<datatype>[_<descriptor>](<parameters>)
```

> **NOTE** The core functions in Intel IPP do not need an input data type. These functions have `ipp` as a prefix without the data-domain field. For example, `ippGetStatusString`.

See Also

Core and Support Functions

- Data-domain
- Primitive vs. Variant Name
- Data Types
- Descriptor
- Parameters

Data-domain

The `data-domain` element is a single character indicating type of input data. Intel IPP supports the following data-domains:

`s`	one-dimensional operations on signals, vectors, buffers
`i`	two-dimensional operations on images, video frames

Primitive vs Variant Name

The `name` element identifies the algorithm or operation of the function. The low-level algorithm that function implements is a *primitive*. This algorithm often has several *variants* for different data types and implementation variations.

For example, the `CToC` modifier in the `ippsFFTInv_CToC_32fc` function signifies that the inverse fast Fourier transform operates on complex floating point data, performing the complex-to-complex (CToC) transform.

Data Types

The `datatype` element indicates data types used by the function, in the following format:

```
<bit depth><bit interpretation>,
```

where

```
bit depth = <1|8|16|32|64>
```

and

```
bit interpretation<u|s|f>[c]
```

Here `u` indicates “unsigned integer”, `s` indicates “signed integer”, `f` indicates “floating point”, and `c` indicates “complex”.

For functions that operate on a single data type, the `datatype` element contains only one value.

If a function operates on source and destination signals that have different data types, the respective data type identifiers are listed in the function name in order of source and destination as follows:

```
<datatype> = <src1Datatype>[src2Datatype][dstDatatype]
```

Descriptor

The optional `descriptor` element describes the data associated with the operation. Descriptors are individual characters that indicate additional details of the operation.

The Intel IPP functions use the following descriptors:

Descriptor	Description	Example
`A`	Image data contains an alpha channel as the last channel, requires `C4`, alpha-channel is not processed.	`ippiFilterMax_8u_AC4R`
`A xx`	Advanced arithmetic operations with `xx` bits of accuracy.	`ippsPowx_32f_A11`

C	The function operates on a specified channel of interest (COI) for each source image.	ippiSet_8u_C3CR
C n	Image data consists of n channels. Possible values for n: 1, 2, 3, 4.	ippiFilterBorder_32f_C1R
D x D	Signal is x-dimensional (default is D1).	ippsConcat_8u_D2
I	Operation is in-place (default is not-in-place).	ippsAdd_16s_I
L	Platform-aware functions (see Using Intel IPP Platform-Aware Functions for details).	ippiAdd_8u_C1RSfs_L
TL	Threading layer functions (see Using Intel IPP Threading Layer Functions for details).	ippiAdd_8u_C1RSfs_LT
M	Operation uses a mask to determine pixels to be processed.	ippiCopy_8u_C1MR
P n	Image data consists of n discrete planar (not-interleaved) channels with a separate pointer to each plane. Possible values for n: 1, 2, 3, 4.	ippiAlphaPremul_8u_AP4R
R	Function operates on a defined region of interest (ROI) for each source image.	ippiMean_8u_C4R
s	Saturation and no scaling (default).	ippiConvert_16s16u_C1Rs
Sfs	Saturation and fixed scaling mode (default is saturation and no scaling).	ippsConvert_16s8s_Sfs

The descriptors in function names are presented in the function name in alphabetical order.

Some data descriptors are default for certain operations and not added to the function names. For example, the image processing functions always operate on a two-dimensional image and saturate the results without scaling them. In these cases, the implied descriptors D2 (two-dimensional signal) and s (saturation and no scaling) are not included in the function name.

Parameters

The parameters element specifies the function parameters (arguments).

The order of parameters is as follows:

- All source operands. Constants follow vectors.
- All destination operands. Constants follow vectors.
- Other, operation-specific parameters.

A parameter name has the following conventions:

- All parameters defined as pointers start with p, for example, pPhase, pSrc; parameters defined as double pointers start with pp, for example, ppState. All parameters defined as values start with a lowercase letter, for example, val, src, srcLen.

- Each new part of a parameter name starts with an uppercase character, without underscore; for example, `pSrc`, `lenSrc`, `pDlyLine`.
- Each parameter name specifies its functionality. Source parameters are named `pSrc` or `src`, in some cases followed by names or numbers, for example, `pSrc2`, `srcLen`. Output parameters are named `pDst` or `dst` followed by names or numbers, for example, `pDst2`, `dstLen`. For in-place operations, the input/output parameter contains the name `pSrcDst` or `srcDst`.

Intel® Integrated Performance Primitives Domain Details

Intel IPP is divided into groups of related functions. Each subdivision is called *domain*, and has its own header file, static libraries, dynamic libraries, and tests. The table below lists each domain's code, header and functional area.

The file `ipp.h` includes Intel IPP header files with the exception of cryptography and generated functions. If you do not use cryptography and generated functions, include `ipp.h` in your application for forward compatibility. If you want to use cryptography functions, you must directly include `ippcp.h` in your application.

* available only within the Intel® System Studio suite

- Library Dependencies by Domain

Library Dependencies by Domain

When you link to a certain Intel® IPP domain library, you must also link to the libraries on which it depends. The following table lists library dependencies by domain.

Domain	Domain Code	Depends on
Color Conversion	`CC`	`Core`, `VM`, `S`, `I`
Computer Vision	`CV`	`Core`, `VM`, `S`, `I`
Data Compression	`DC`	`Core`, `VM`, `S`
Image Processing	`I`	`Core`, `VM`, `S`
Signal Processing	`S`	`Core`, `VM`
Vector Math	`VM`	`Core`

Performance Test Tool (perfsys) Command Line Options

Intel® Integrated Performance Primitives (Intel® IPP) installation includes command-line tools for performance testing in the `<install_dir>/tools/perfsys` directory. There is one perfsys tool for each domain. For example, `ps_ipps` executable measures performance for all Intel IPP signal processing domain functions.

Many factors may affect Intel IPP performance. One of the best way to understand them is to run multiple tests in the specific environment you are targeting for optimization. The purpose of the perfsys tools is to simplify performance experiments and empower developers with useful information to get the best performance from Intel IPP functions.

With the command-line options you can:

- Create a list of functions to test
- Set parameters for each function
- Set image/buffer sizes

To simplify re-running specific tests, you can define the functions and parameters in the initialization file, or enter them directly from the console.

The command-line format is:

```
ps_ipp*.exe [option_1] [option_2] ... [option_n]
```

To invoke the short reference for the command-line options, use `-?` or `-h` commands:

```
ps_ipp*.exe -h
```

The command-line options are divided into several groups by functionality. You can enter options in arbitrary order with at least one space between each option name. Some options (like `-r`, - R, `-o`, - O) may be entered several times with different file names, and option `-f` may be entered several times with different function patterns. For detailed descriptions of the perfsys command-line options see the following table:

Option	Description
`-T[cpu-features]`	Call `ippSetCpuFeatures`
`-A<Timing|Params|Misalign|All>`	Prompt for the parameters before every test from console
`-o[ <file-name> ]`	Create `<file-name>.txt` file and write console output to it
`-O[ <file-name> ]`	Add console output to the file `<file-name>.txt`
`-L <ERR|WARN|PARM|INFO|TRACE>`	Set detail level of the console output
`-r[ <file-name> ]`	Create `<file-name>.csv` file and write perfsys results to it
`-R[ <file-name> ]`	Add test results to the file `<file-name>.csv`
`-q[<file-name>]`	Create `<file-name>.csv` and write function parameter name lines to it
`-q+`	Add function parameter name lines to perfsys results table file
`-Q`	Exit after creation of the function parameter name table
`-u[ <file-name> ]`	Create `<file-name>.csv` file and write summary table (’`_sum`’ is added to default file name)
`-U[ <file-name> ]`	Add summary table to the file `<file-name>.csv` (’`_sum`’ is added to default file name)
`-g[ <file-name> ]`	Create signal file at the end of the whole testing
`-l <dir-name>`	Set default directory for output files
`-k<and|or>`	Compose different keys (-f, -t, -m) by logical operation
`-F <func-name>`	Start testing from function with `func-name` full name
`-Y <HIGH/NORMAL>`	Set high or normal process priority (normal is default)
`-H[ONLY]`	Add ‘Interest’ column to `.csv` file [and run only hot tests]

`-N <num-threads>`	Call `ippSetNumThreads_LT(<num-treads>)`
`-s[-]`	Sort or do not sort functions (sort mode is default)
`-e`	Enumerate tests and exit
`-v`	Display the version number of the perfsys and exit
`-@<file-name>`	Read command-line options for the specified file
`-d <name> = <value>`	Set perfsys parameter value
`-i[ <file-name> ]`	Read perfsys parameters from the file `<file-name>.ini`
`-I[ <file-name> ]`	Write perfsys parameters to the file `<file-name>.ini` and exit
`-P`	Read tested function names from the `.ini` file
`-n <title-name>`	Set default title name for `.ini` file and output files
`-p <dir-name>`	Set default directory for `.ini` file and input test data files
`-f <or-pattern>`	Run tests for functions with `pattern` in their names, case sensitive
`-f- <not-pattern>`	Do not test functions with `pattern` in their names, case sensitive
`-f+ <and-pattern>`	Run tests only for functions with `pattern` in their names, case sensitive
`-f= <eq-pattern>`	Run tests for functions with `pattern` full name
`-t[-|+|=] <pattern>`	Run (do not run) tests with `pattern` in test name
`-m[-|+|=] <pattern>`	Run (do not run) tests registered in file with `pattern` in file name
`-h`	Display short help and exit
`-hh`	Display extended help and exit
`-h<key>`	Display extended help for the key and exit

Notices and Disclaimers

Intel technologies may require enabled hardware, software or service activation.

No product or component can be absolutely secure.

Your costs and results may vary.

© Intel Corporation. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Other names and brands may be claimed as the property of others.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

MPEG-1, MPEG-2, MPEG-4, H.261, H.263, H.264, MP3, DV, VC-1, MJPEG, AC3, AAC, G.711, G.722, G.722.1, G.722.2, AMRWB, Extended AMRWB (AMRWB+), G.167, G.168, G.169, G.723.1, G.726, G.728, G.729, G.729.1, GSM AMR, GSM FR are international standards promoted by ISO, IEC, ITU, ETSI, 3GPP and other organizations. Implementations of these standards, or the standard enabled platforms may require licenses from various entities, including Intel Corporation.

Java is a registered trademark of Oracle and/or its affiliates.

Microsoft, Windows, and the Windows logo are trademarks, or registered trademarks of Microsoft Corporation in the United States and/or other countries.

Third Party

Intel® Integrated Performance Primitives (Intel® IPP) includes content from several 3rd party sources that was originally governed by the licenses referenced below:

- zlib library:

  zlib.h – interface of the 'zlib' general purpose compression library version 1.2.8, April 28th, 2013

  Copyright (C) 1995-2013 Jean-loup Gailly and Mark Adler

  This software is provided 'as-is', without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this software.

  Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

  1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
  2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
  3. This notice may not be removed or altered from any source distribution.

  Jean-loup Gailly Mark Adler

  jloup@gzip.orgmadler@alumni.caltech.edu
- bzip2:

  Copyright © 1996 - 2015 **julian@bzip.org**

Developer Reference: Signal and Data Processing

The following are some important features of the signal and data processing part of the Intel® Integrated Performance Primitives (Intel® IPP) library:

Essential Functions

Essential Functions provide arithmetic, statistical, logical and shift operations, windowing, Viterbi decoding, sampling, and conversion.

Filtering Functions

Filtering Functions perform convolution and correlation operations, as well as various types of filtering.

Transform Functions

Transform Functions implement Fourier, Hartley, Walsh-Hadamard, discrete cosine, Hilbert, and Wavelet transforms.

Vector Initialization Functions

Vector Initialization Functions initialize vectors containing either constants, the contents of other vectors, or the generated signals. This group also includes functions for generating samples of various types: tone, triangle, pseudo-random with uniform distribution, and pseudo-random with Gaussian distribution, as well as special test samples.

Fixed-Accuracy Arithmetic Functions

Fixed-Accuracy Arithmetic Functions provide power and root, exponential and logarithmic, trigonometric, hyperbolic, and rounding operations.

Data Compression Functions

Data Compression Functions perform VLC and Huffman coding, dictionary- and BWT-based compression.

- Intel® Integrated Performance Primitives Concepts
- Support Functions
- Vector Initialization Functions
- Essential Functions
- Filtering Functions
- Transform Functions
- Fixed-Accuracy Arithmetic Functions
- Long Term Evolution (LTE) Wireless Support Functions
- Data Compression Functions
- Appendix A: Handling of Special Cases
- Appendix B: Removed Functions for Signal Processing
- Bibliography for Signal Processing
- Glossary

Intel® Integrated Performance Primitives Concepts

This chapter explains the structure of the Intel® Integrated Performance Primitives (Intel® IPP) software and some of the basic concepts used in the signal and data processing part of Intel IPP. It also defines function naming conventions in the document, describes the supported data formats and operation modes.

- Function Naming
- Structures and Enumerators
- Data Ranges
- Data Alignment
- Rounding Mode
- Integer Scaling
- Error Reporting
- Platform-Aware Functions in Signal and Data Processing
- Code Examples

Function Naming

Naming conventions for the Intel IPP functions are similar for all covered domains.

Function names in Intel IPP have the following general format:

```
ipp<data-domain><name>_<datatype>[_<descriptor>][<_extension>](<parameters>)
```

The elements of this format are explained in the sections that follow.

NOTE In this document, each function is introduced by its short name (without the `ipps` prefix and modifiers) and a brief description of its purpose.
The `ipps` prefix in function names is always used in the code examples. In the text, this prefix is usually omitted when referring to the function group.

- Data-Domain
- Name
- Data Types
- Descriptors
- Parameters
- Extensions

Data-Domain

The `data-domain` element is a single character that denotes the group of functionality to which a given function belongs. The main distinction among these groups is the type of input data. Intel IPP supports the following data-domains:

`s`	signal processing (input data is a 1D signal)
`i`	images and video processing (input data is a 2D image)
`m`	small matrix operations (input data is a matrix)
`r`	realistic rendering functionality and 3D data processing (type of input data type depends on supported rendering techniques)
`g`	operations on signals of the fixed length

For example, function names that begin with `ipps` signify that respective functions are used for signal processing.

Name

The `name` element identifies what function does and has the following format:

`<name> = <operation>[_ modifier]`

The `operation` component is one or more words, acronyms, and abbreviations that describe the core operation.

The `modifier` component, if present, is a word or abbreviation that denotes a slight modification or variation of the given function.

For example, names without modifiers: `Add`, `Threshold`, `FirGenLowPass`; with modifiers: `ippsFFTInv_CToC`, `Threshold_LT`.

Data Types

The `datatype` field indicates data types used by the function, in the following format:

`<bit depth><bit interpretation>,`

where

`bit depth = <1|8|16|32|64>`

and

`bit interpretation<u|s|f>[c]`

Here `u` indicates "unsigned integer", `s` indicates "signed integer", `f` indicates "floating point", and `c` indicates "complex".

Intel IPP supports the data types of the source and destination for signal processing functions listed in the table below.

NOTE In the lists of function parameters, the `Ipp` prefix is added to the data type. For example, 8-bit signed data is denoted as `Ipp8s` type. These Intel IPP-specific data types are defined in the respective library header files.

Type	Usual C Type	Intel IPP Type
`8u`	`unsigned char`	`Ipp8u`
`8s`	`signed char`	`Ipp8s`
`16u`	`unsigned short`	`Ipp16u`
`16s`	`signed short`	`Ipp16s`
`16sc`	`complex short`	`Ipp16sc`
`32u`	`unsigned int`	`Ipp32u`
`32s`	`signed int`	`Ipp32s`
`32f`	`float`	`Ipp32f`
`32fc`	`complex float`	`Ipp32fc`
`64s`	`__int64` (Windows*) or `long long` (Linux*)	`Ipp64s`
`64f`	`double`	`Ipp64f`
`64fc`	`complex double`	`Ipp64fc`

For functions that operate on a single data type, the `datatype` field contains only one of the values listed above.

If a function operates on source and destination signals that have different data types, the respective data type identifiers are listed in the function name in order of source and destination as follows:

`<datatype> = <src1Datatype>[src2Datatype][dstDatatype]`

For example, the function `ippsDotProd_16s16sc_Sfs` computes the dot product of 16-bit short and 16-bit complex short source vectors and stores the result in a 16-bit complex short destination vector. The `dstDatatype` modifier is not present in the name because the second operand and the result are of the same type. The result is scaled and saturated.

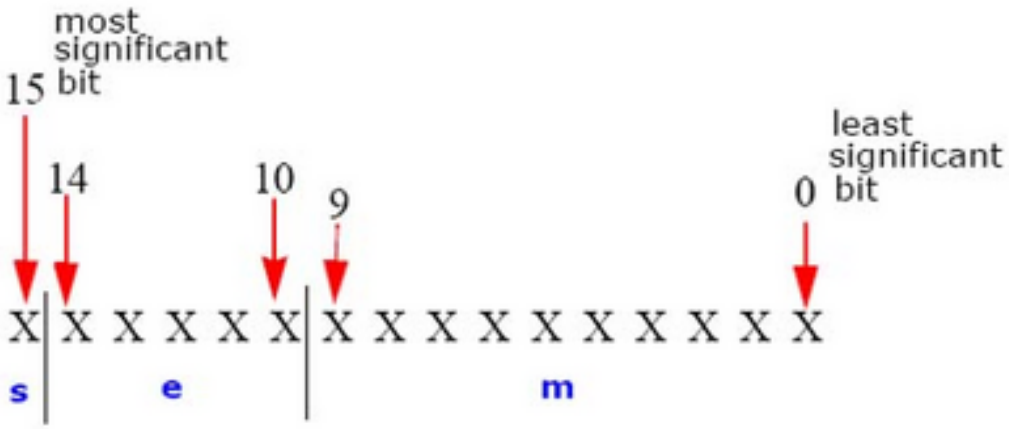
There are several data types, namely `24u`, `24s` and `16f` that are not supported by Intel IPP, but can be readily converted to the supported data types for further processing by the library functions.

For the unsigned `24u` data, each vector element consists of three consecutive bytes represented as `Ipp8u` data types. It has a little-endian byte order when a lower order byte is at the lower address. These data may be converted to and from `32u` or `32f` data types by using the appropriate flavors of the Intel IPP function `ippsConvert`.

For the signed `24s` data, each vector element consists of three consecutive bytes represented as `Ipp8s` data types. It has a little-endian byte order when a lower order byte is at the lower address. The sign is represented by the most significant bit of the highest order byte. These data may be converted to and from `32s` or `32f` data types by using the appropriate flavors of the Intel IPP function `ippsConvert`.

For the `16f` format, 16-bit floating point data (*half type*) can represent positive and negative numbers, whose magnitude is between roughly $6.1e^{-5}$ and $6.5e^{4}$, with a relative error of $9.8e^{-4}$; numbers smaller than $6.1e^{-5}$ can be represented with an absolute error of $6.0e^{-8}$. All integers from `-2048` to + 2048 can be represented exactly.

The figure below illustrates the bit-layout for a half number:



``s`` is the sign-bit, **``e``** is the exponent, and **``m``** is the significand.

These data may be converted to and from `16s` and `32f` data types by using the appropriate flavors of the Intel IPP function `ippsConvert`.

- IppHintAlgorithm enum Type

IppHintAlgorithm enum Type

The `IppHintAlgorithm` type is used by many Intel Integrated Performance Primitives (Intel® IPP) functions as an argument to select the algorithmic implementation of the function. The following values are supported:

Value	Description
`ippAlgHintAccurate`	Accuracy takes precedence over performance. This value provides better or exact bit precision (depending on the function).
`ippAlgHintFast`	Function performance takes precedence over accuracy. Possible deviation is bigger than in the accurate case.
`ippAlgHintNone`	The hint is absent. This value is the equivalent of `ippAlgHintFast`.

NOTE For some functions, this parameter is deprecated. Read the function description carefully.

Descriptors

The `descriptors` element further describes the operation. Descriptors are individual characters that indicate additional details of the operation.

The following descriptors are used in signal processing functions:

Descriptor	Description	Example
`I`	Operation is in-place (default is not-in-place). An in-place operation is guaranteed to be correct if and only if the Intel IPP function performing the operation has the descriptor `I` in its name. In addition to the descriptor `I`, all function declarations for in-place Intel IPP functions contain a pointer to an *in-place buffer* (for example, `pSrcDst`) in the parameter list. Before the function call, the in-place buffer contains the source data. After the function call, the in-place buffer contains the result of the operation.	`ippsAdd_16s_I`
`Sfs`	Saturation and fixed scaling mode (default is saturation and no scaling).	`ippsConvert_16s8s_Sfs`
`P`	Operation is performed for the specified number of vectors.	`ippsIIR_32f _P`

If the function has more than one descriptor, they are presented in the function name in alphabetical order.

Many functions have no descriptors listed above. Such functions operate with the default behavior.

Parameters

The `parameters` element specifies the function parameters (arguments).

The order of parameters is as follows:

- All source operands. Constants follow vectors.
- All destination operands. Constants follow vectors.
- Other, operation-specific parameters.

A parameter name has the following conventions:

- All parameters defined as pointers start with `p`, defined as double pointers start with `pp`, for example, `pPhase`, `pSrc`, `ppState`. All parameters defined as values start with a lowercase letter, for example, `val`, `src`, `srcLen`.
- Each new part of a parameter name starts with an uppercase character, without underscore; for example, `pSrc`, `lenSrc`, `pDlyLine`.
- Each parameter name specifies its functionality. Source parameters are named `pSrc` or `src`, in some cases followed by names or numbers, for example, `pSrc2`, `srcLen`. Output parameters are named `pDst` or `dst` followed by names or numbers, for example, `pDst2`, `dstLen`. For in-place operations, the input/output parameter contains the name `pSrcDst` or `srcDst`.

Extensions

The `extension` field denotes an Intel IPP extension to which the function belongs. The following extensions are supported in Intel IPP Signal Processing functions:

Extension	Description	Example
L	Intel IPP platform-aware functions	ippsMalloc_16u_L

See Also

Platform-Aware Functions for Signal Processing

Structures and Enumerators

This section describes the structures and enumerators used by Intel IPP for signal and data processing.

- Library Version Structure
- Complex Data Structures
- Function Context Structures
- Enumerators

Library Version Structure

The IppLibraryVersion structure describes the current Intel IPP software version. The main fields of this structure are:

- integer fields major and minor, containing version numbers;
- integer field majorBuild, containing update number;
- integer field build, containing build revision number;
- string field Name, containing the Intel IPP version name, for example, " ippSB SSE4.1 ";
- string field Version, containing the version description, for example, " 7.1.0 (r93873) ".
- string field BuildDate, containing the build date.

Complex Data Structures

Complex numbers in Intel IPP are described by the structures that contain two numbers of the respective data type. They are real and imaginary parts of the complex number. For example, a single precision complex number is described by the Ipp32fc structure as follows:

```
typedef struct {
     Ipp32f   re;
     Ipp32f  im;
 } Ipp32fc;
```

The following complex data types are defined: Ipp16sc, Ipp32fc, Ipp64fc.

Function Context Structures

Some Intel IPP functions use special structures to store function-specific (context) information. For example, the IppsFFTSpec structure stores twiddle factors and bit reverse indexes needed in the fast Fourier transform.

Two different kinds of structures are used:

- specification structures that are not modified during function operation; they have the suffix Spec in their names
- state structures that are modified during operation; they have the suffix State in their names.

The function context interpretation is processor dependent. Therefore, these context-related structures are not defined in the public headers, and their fields are not accessible. Intel IPP provides no option of modifying these structures or creating a function context as an automatic variable.

Enumerators

The `IppStatus` constant enumerates the status values returned by the Intel IPP functions, indicating whether the operation is error-free. See section Error Reporting in this chapter for more information on the set of valid status values and corresponding error messages for signal processing functions.

The `IppCmpOp` enumeration defines the type of relational operator to be used by threshold functions:

```
typedef enum {
    ippCmpLess,
    ippCmpLessEq,
    ippCmpEq,
    ippCmpGreaterEq,
    ippCmpGreater
} IppCmpOp;
```

The `IppRoundMode` enumeration defines the rounding mode to be used by conversion functions:

```
typedef enum {
    ippRndZero,
    ippRndNear,
    ippRndFinancial
} IppRoundMode;
```

The `IppHintAlgorithm` enumeration defines the type of code to be used in some operations: faster but less accurate, or vice-versa, more accurate but slower. For more information on using this enumeration, see IppHintAlgorithm enum Type.

```
typedef enum {
    ippAlgHintNone,
    ippAlgHintFast,
    ippAlgHintAccurate
} IppHintAlgorithm;
```

The `IppCpuType` enumerates processor types returned by the `ippGetCpuType` function:

```
typedef enum {
 /* Enumeration:           Processor:                                */
    ippCpuUnknown = 0x0,
    ippCpuPP,           /* Intel® Pentium® processor             */
    ippCpuPMX,          /* Pentium® processor
                           with MMX™ technology                    */
    ippCpuPPR,          /* Pentium® Pro processor                  */
    ippCpuPII,          /* Pentium® II  processor                  */
    ippCpuPIII,         /* Pentium® III processor
                           and Pentium®III Xeon® processor       */
    ippCpuP4,           /* Pentium® 4 processor
                           and Intel® Xeon® processor            */
    ippCpuP4HT,         /* Pentium® 4 processor with HT Technology */
    ippCpuP4HT2,        /* Pentium® 4 processor with Intel®
                                  Streaming SIMD Extensions 3     */
    ippCpuCentrino,     /* Intel® Centrino® processor technology   */
    ippCpuCoreSolo,     /* Intel® Core™ Solo processor           */
    ippCpuCoreDuo,      /* Intel® Core™ Duo processor            */
    ippCpuITP   = 0x10, /* Intel® Itanium® processor              */
    ippCpuITP2          /* Intel® Itanium® 2 processor           */
    ippCpuEM64T = 0x20, /* Intel® 64 Instruction Set
```

```
                                        Architecture(ISA)           */
    ippCpuC2D,          /* Intel® Core™ 2 Duo processor         */
    ippCpuC2Q,          /* Intel® Core™ 2 Quad processor        */
    ippCpuPenryn,     /* Intel® Core™ 2 processor with
                                 Intel® SSE4.1                 */
    ippCpuBonnell,                /* Intel Atom®  processor */
    ippCpuNehalem,      /* Intel® Core™ i7 processor
    ippCpuNext,
    ippCpuSSE   = 0x40, /* Processor supports Pentium® III
                           processor instruction set                */
    ippCpuSSE2,         /* Processor supports Intel® Streaming SIMD
                           Extensions 2 instruction set             */
    ippCpuSSE3,         /* Processor supports Intel® Streaming SIMD
                            Extensions 3 instruction set            */
    ippCpuSSSE3,        /* Processor supports Supplemental Streaming
                           SIMD Extensions 3 instruction set        */
    ippCpuSSE41,        /* Processor supports Intel® Streaming SIMD
                            Extensions 4.1 instruction set          */
    ippCpuSSE42,        /* Processor supports Intel® Streaming SIMD
                            Extensions 4.2 instruction set          */
    ippCpuAVX,        /* Processor supports Intel® Advanced Vector
                            Extensions instruction set              */
    ippCpuAES,        /* Processor supports Intel® AES
                            new instructions                        */
ippCpuX8664 = 0x60,   /* Processor supports 64 bit extension       */
} IppCpuType;
```

The `IppWinType` enumeration defines the type of window to be used by the FIR filter coefficient generating functions:

```
typedef enum {
    ippWinBartlett,
    ippWinBlackman,
    ippWinHamming,
    ippWinHann,
    ippWinRect
} IppWinType;
```

The `IppLZ77ComprLevel` enumeration defines the compression level to be used by the ZLIB data compression functions:

```
typedef enum {
    IppLZ77FastCompr,
    IppLZ77AverageCompr,
    IppLZ77BestCompr
} IppLZ77ComprLevel;
```

The `IppLZ77Chcksm` enumeration defines what algorithm is used to compute the checksum by the ZLIB data compression functions:

```
typedef enum {
    IppLZ77NoChcksm,
    IppLZ77Adler32,
    IppLZ77CRC32
} IppLZ77Chcksm;
```

The IppLZ77Flush enumeration defines what encoding mode is used by the ZLIB data compression functions:

```
typedef enum {
    IppLZ77NoFlush,
    IppLZ77SyncFlush,
    IppLZ77FullFlush,
    IppLZ77FinishFlush
} IppLZ77Flush;
```

The IppLZ77DeflateStatus enumeration defines the encoding status that is used by the ZLIB data compression functions:

```
typedef enum {
    IppLZ77StatusInit,
    IppLZ77StatusLZ77Process,
    IppLZ77StatusHuffProcess,
    IppLZ77StatusFinal
} IppLZ77DeflateStatus;
```

The IppLZ77InflateStatus enumeration defines the decoding status that is used by the ZLIB data compression functions:

```
typedef enum {
    IppLZ77InflateStatusInit,
    IppLZ77InflateStatusHuffProcess
    IppLZ77InflateStatusLZ77Process,
    IppLZ77InflateStatusFinal
} IppLZ77InflateStatus;
```

The IppLZ77HuffMode enumeration defines the encoding mode that is used by the ZLIB data compression functions:

```
typedef enum {
    IppLZ77UseFixed,
    IppLZ77UseDynamic,
    IppLZ77UsedStored
} IppLZ77HuffMode;
```

The IppInflateState enumeration defines the decoding parameters that are used by the ZLIB data compression functions:

```
typedef struct IppInflateState {
    const Ipp8u* pWindow;          // pointer to the sliding window
                                   // (the dictionary for the LZ77 algorithm)
    unsigned int winSize;          // size of the sliding window
    unsigned int tableType;        // type of Huffman code tables
                                   // (for example, 0 - tables for Fixed
                                   //  Huffman codes)
    unsigned int tableBufferSize;  //(ENOUGH = 2048) * (sizeof(code) = 4)
                                   // - sizeof(IppInflateState)
} IppInflateState;
```

The IppInflateMode enumeration defines the decode mode that is used by the ZLIB data compression functions:

```
typedef enum {
    ippTYPE,
    ippLEN,
    ippLENEXT
} IppInflateMode;
```

The `IppGITStrategyHint` enumeration defines which strategy of encoding is used in some operations by the GIT data compression functions:

```
typedef enum {
    ippGITNoStrategy,
    ippGITLeftReorder,
    ippGITRightReorder,
    ippGITFixedOrder
} IppGITStrategyHint;
```

The `IppEnum` enumeration defines the configuration of the algorithm for some functions:

```
typedef int IppEnum;
```

The `IppAlgType` enumeration defines the type of the algorithm implementation:

```
typedef enum {
    ippAlgAuto     = 0x00000000, // default
    ippAlgDirect   = 0x00000001,
    ippAlgFFT      = 0x00000002,
    ippAlgMask    = 0x000000FF,
} IppAlgType;
```

The `IppsNormOp` enumeration defines the type of normalization that should be applied to the output data:

```
typedef enum {
    ippsNormNone  = 0x00000000,// default
    ippsNormA     = 0x00000100,// biased normalization
    ippsNormB     = 0x00000200,// unbiased normalization
    ippsNormMask  = 0x0000FF00,
} IppsNormOp;
```

The `IppFourSymb` structure used in Long Term Evolution (LTE) Wireless Support Functions stores the destination data grouped by four symbols:

```
typedef struct {
     Ipp16sc symb[4];
} IppFourSymb;
```

Data Ranges

The range of values that can be represented by each data type lies between the lower and upper bounds. The following table lists data ranges and constant identifiers used in Intel IPP to denote the respective range bounds:

Data Type	Lower Bound Identifier	Lower Bound Value	Upper Bound Identifier	Upper Bound Value
8s	IPP_MIN_8S	-128	IPP_MAX_8S	127
8u		0	IPP_MAX_8U	255
16s	IPP_MIN_16S	-32768	IPP_MAX_16S	32767
16u		0	IPP_MAX_16U	65535
32s	IPP_MIN_32S	-2^{31}	IPP_MAX_32S	$2^{31}-1$
32u		0	IPP_MAX_32U	$2^{32}-1$
32f†	IPP_MINABS_32F	$1.175494351e^{-38}$	IPP_MAXABS_32F	$3.402823466e^{38}$

† The range for absolute values

Data Alignment

Intel IPP is built using the compiler option `/Zp16`, which aligns the structure fields on the field size or 16 bytes if the size is greater than 16.

You can also use the `ippsMalloc` function to align the allocated memory pointer on 64 bytes.

Rounding Mode

General signal processing functions use rounding. The default rounding mode is *nearest even*, that is the fixed point number $x = N + \alpha$, $0 \le \alpha < 1$, where N is an integer number, is rounded as given by:

$$\lceil x \rceil = \begin{cases} N, & 0 \le \alpha < 0.5 \\ N+1, & 0.5 < \alpha < 1 \\ N, & \alpha = 0.5, \; N-even \\ N+1, & \alpha = 0.5, \; N-odd \end{cases}$$

For example, 1.5 will be rounded to 2 and 2.5 to 2.

Some functions have additional rounding modes, which are set by the parameter `roundMode`.

NOTE
Important

- Functions for data compression, data integrity, string processing and fixed-accuracy arithmetic do not perform rounding.

Integer Scaling

Some signal processing functions operating on integer data use scaling of the internally computed output results by the integer `scaleFactor`, which is specified as one of the function parameters. These functions have the `Sfs` descriptor in their names.

The scale factor can be negative, positive, or zero. Scaling is applied because internal computations are generally performed with a higher precision than the data types used for input and output signals.

NOTE The result of integer operations is always saturated to the destination data type range.

Scaling of an integer result is done by multiplying the output vector values by $2^{-scaleFactor}$ before the function returns. This helps retain either the output data range or its precision. Usually the scaling with a positive factor is performed by the shift operation. The result is rounded off to the nearest even integer number (see "Rounding Mode").

For example, the integer `Ipp16s` result of the square operation `ippsSqr` for the input value 200 is equal to 32767 instead of 40000, that is, the result is saturated and the exact value can not be restored.

The scaling of the output value with the factor `scaleFactor = 1` yields the result 20000, which is not saturated, and the exact value can be restored as 20000*2. Thus, the output data range is retained.

The following example shows how the precision can be partially retained by means of scaling.

The integer square root operation `ippsSqrt` (without scaling) for the input value 2 gives the result equal to 1 instead of 1.414. Scaling of the internally computed output value with the factor `scaleFactor` = -3 gives the result 11, and permits to restore the more precise value as $11*2^{-3} = 1.375$.

See Also

Rounding Mode

Error Reporting

The Intel IPP functions return the status of the performed operation to report errors and warnings to the calling program. The last value of the error status is not stored, and you need to decide whether to check it or not as the function returns. The status values are of the `IppStatus` type and are global constant integers.

The following table lists status codes and corresponding messages reported by Intel IPP for signal processing.

Status	Message
`ippStsCpuNotSupportedErr`	The target cpu is not supported.
`ippStsUnknownStatusCodeErr`	Unknown status code.
`ippStsLzoBrokenStreamErr`	LZO safe decompression function cannot decode LZO stream.
`ippStsRoundModeNotSupportedErr`	Rounding mode is not supported.
`ippStsRegExpOptionsErr`	RegExp: Options for the pattern are incorrect.
`ippStsRegExpErr`	RegExp: The structure `pRegExpState` contains wrong data.
`ippStsRegExpMatchLimitErr`	RegExp: The match limit has been exhausted.
`ippStsRegExpQuantifierErr`	RegExp: Incorrect quantifier.
`ippStsRegExpGroupingErr`	RegExp: Incorrect grouping.
`ippStsRegExpBackRefErr`	RegExp: Incorrect back reference.
`ippStsRegExpChClassErr`	RegExp: Incorrect character class.
`ippStsRegExpMetaChErr`	RegExp: Icnorrect metacharacter.
`ippStsLengthErr`	Incorrect value for string length.
`ippStsToneMagnErr`	Tone magnitude is less than or equal to zero.
`ippStsToneFreqErr`	Tone frequency is negative, or greater than or equal to 0.5.
`ippStsTonePhaseErr`	Tone phase is negative, or greater than or equal to 2*PI.
`ippStsTrnglMagnErr`	Triangle magnitude is less than or equal to zero.
`ippStsTrnglFreqErr`	Triangle frequency is negative, or greater than or equal to 0.5.
`ippStsTrnglPhaseErr`	Triangle phase is negative, or greater than or equal to 2*PI.
`ippStsTrnglAsymErr`	Triangle asymmetry is less than -PI, or greater than or equal to PI.

ippStsHugeWinErr	The Kaiser window is too big.
ippStsJaehneErr	Magnitude value is negative.
ippStsStepErr	Step value is not valid.
ippStsStrideErr	Stride value is less than length of the row.
ippStsEpsValErr	Negative epsilon value.
ippStsScaleRangeErr	Scale bounds are out of range.
ippStsThresholdErr	Invalid threshold bounds.
ippStsWtOffsetErr	Invalid offset value for wavelet filter.
ippStsAnchorErr	Anchor point is outside the mask.
ippStsMaskSizeErr	Invalid mask size.
ippStsShiftErr	Shift value is less than zero.
ippStsSampleFactorErr	Sampling factor is less than or equal to zero.
ippStsSamplePhaseErr	Phase value is out of range, $0 \le phase < factor$.
ippStsFIRMRFactorErr	MR FIR sampling factor is less than or equal to zero.
ippStsFIRMRPhaseErr	MR FIR sampling phase parameter is negative, or greater than or equal to the sampling factor.
ippStsRelFreqErr	Relative frequency value is out of range.
ippStsFIRLenErr	Length of the FIR filter is less than or equal to zero.
ippStsIIROrderErr	Order of the IIR filter is not valid.
ippStsResizeFactorErr	Resize factor(s) is less than or equal to zero.
ippStsDivByZeroErr	An attempt to divide by zero.
ippStsInterpolationErr	Invalid interpolation mode.
ippStsMirrorFlipErr	Invalid flip mode.
ippStsMoment00ZeroErr	Moment value M(0,0) is too small to continue calculations.
ippStsThreshNegLevelErr	Negative value of the level in the threshold operation.
ippStsContextMatchErr	Context parameter does not match the operation.
ippStsFftFlagErr	Invalid value for the FFT flag parameter.
ippStsFftOrderErr	Invalid value for the FFT order parameter.
ippStsMemAllocErr	Not enough memory for the operation.
ippStsNullPtrErr	Null pointer error.
ippStsSizeErr	Incorrect value for data size.
ippStsBadArgErr	Incorrect argument/parameter of the function.

ippStsErr	Unknown/unspecified error.
ippStsNoErr	No errors.
ippStsNoOperation	No operation has been executed.
ippStsSqrtNegArg	Negative value(s) of the argument in the function Sqrt.
ippStsEvenMedianMaskSize	Even size of the Median Filter mask was replaced by the odd one.
ippStsDivByZero	Zero value(s) of the divisor in the function Div.
ippStsLnZeroArg	Zero value(s) of the argument in the function Ln.
ippStsLnNegArg	Negative value(s) of the argument in the function Ln.
ippStsNanArg	Argument value is not a number.
ippStsOverflow	Overflow in the operation.
ippStsUnderflow	Underflow in the operation.
ippStsSingularity	Singularity in the operation.
ippStsDomain	Argument is out of the function domain.
ippStsCpuMismatch	Cannot set the library for the given cpu.
ippStsOvermuchStrings	Number of destination strings is more than expected.
ippStsOverlongString	Length of one of the destination strings is more than expected.
ippStsSrcSizeLessExpected	DC: The size of source buffer is less than the expected one.
ippStsDstSizeLessExpected	DC: The size of destination buffer is less than the expected one.
ippStsNotSupportedCpu	The CPU is not supported.
ippStsAlgTypeErr	The algorithm type is not supported.

*)

The status codes ending with Err (except for the ippStsNoErr status) indicate an error; the integer values of these codes are negative. When an error occurs, the function execution is interrupted. All other status codes indicate warnings. When a specific case is encountered, the function execution is completed and the corresponding warning status is returned.

For example, if the integer function ippsDiv_8u meets an attempt to divide a positive value by zero, the function execution is not interrupted. The result of the operation is set to the maximum value that can be represented by the source data type, and the function returns the warning status ippStsDivByZero. This is the case for the vector-vector operation ippsDiv. For the vector-scalar division operation ippsDivC, the function behavior is different: if the constant divisor is zero, then the function stops execution and returns immediately with the error status ippStsDivByZeroErr.

Platform-Aware Functions in Signal and Data Processing

Intel® Integrated Performance Primitives (Intel® IPP) library provides so-called platform-aware functions. These functions use the special data type `IppSizeL` for object sizes. The `IppSizeL` data type represents memory-related quantities: it can be 32- or 64-bit wide depending on the target architecture.

While the rest of Intel IPP functions support only objects of 32-bit integer size, platform-aware functions can work with 64-bit object sizes if it is supported by the platform. The API of platform-aware functions is similar to the API of other Intel IPP functions and has only slight differences. You can distinguish platform-aware functions by the `L` suffix in the function name, for example, `ippsMalloc_16u_L`.

Currently, the following signal processing functions have platform-aware APIs:

Function Group	Header	Function Name
Support Functions	`ipps_l.h`	`Malloc`
Arithmetic Functions	`ipps_l.h`	`MulC`
Sort Functions	`ipps_l.h`	`SordRadix{Ascend|Descend}`, `SordRadixGetBufferSize`, `SordRadixIndex{Ascend|Descend}`

Intel IPP platform-aware functions are documented as additional flavors to the existing functions declared in standard Intel IPP headers (without the `l` suffix). The `ipps_l.h` header is included into `ipps.h`.

Code Examples

The document contains a number of code examples that use the Intel IPP functions. These examples show both some particular features of the primitives and how the primitives can be called. Many of these code examples output result data together with the status code and associated messages in case of an error or a warning condition.

To keep the example code simpler, special definitions of print statements are used that get output strings look exactly the way it is needed for better representation of results of different format, as well as print status codes and messages.

The code definitions given below make it possible to build the examples contained in the document by straightforward copying and pasting the example code fragments.

```
#define genPRINT(TYPE,FMT)     void printf_##TYPE(const char* msg, Ipp##TYPE* buf, int len,
IppStatus st ) {        int n;        if( st > ippStsNoErr )           printf( "\n-- warning %d,
%s", st, ippGetStatusString( st ));        else if( st < ippStsNoErr )           printf( "\n--
error %d, %s", st, ippGetStatusString( st ));        printf("\n %s \n", msg );        for( n=0;
n<len; ++n ) printf( FMT, buf[n] );        printf("\n" );     }
genPRINT( 64f, " %f" )
genPRINT( 32f, " %f" )
genPRINT( 32u, " %u" )
genPRINT( 16s, " %d" )
genPRINT( 8u, " %u" )


#define genPRINTcplx(TYPE,FMT)     void printf_##TYPE(const char* msg, Ipp##TYPE* buf, int len,
IppStatus st ) {        int n;        if( st > ippStsNoErr )           printf( "\n-- warning %d,
%s", st, ippGetStatusString( st ));        else if( st < ippStsNoErr )           printf( "\n--
error %d, %s", st, ippGetStatusString( st ));        printf(" %s ", msg );        for( n=0;
n<len; ++n ) printf( FMT, buf[n].re, buf[n].im );        printf("\n" );     }
genPRINTcplx( 64fc, " {%f,%f}" )
genPRINTcplx( 32fc, " {%f,%f}" )
genPRINTcplx( 16sc, " {%d,%d}" )


#define genPRINT_2D(TYPE,FMT)     void printf_##TYPE##_2D(const char* msg, Ipp##TYPE* buf,
```

```
IppiSize roi, int step, IppStatus st ) {        int i, j;        if ( st > ippStsNoErr )
{           printf( "\n-- warning %d, %s", st, ippGetStatusString( st ));        } else if ( st
< ippStsNoErr ) {           printf( "\n-- error %d, %s", st,
ippGetStatusString( st ));        }        printf("\n %s \n", msg );        for ( i=0;
i<roi.height; i++ ) {           for ( j=0; j<roi.width; j++ ) {              printf( FMT,
((Ipp##TYPE*)(((Ipp8u*)buf) + i*step))[j] );           }           printf("\n");        }
printf("\n" );     }
genPRINT_2D( 8u, " %u" )
genPRINT_2D( 32f, " %.1f" )
```

Support Functions

This chapter describes Intel ® IPP support functions. Use these functions to:

- Retrieve information about the current Intel IPP software version
- Allocate and free memory that is needed for the operation of other Intel IPP functions
- Retrieve information about the processor and perform specific auxiliary operations
- Perform internationalization

- Version Information Functions
- Memory Allocation Functions
- Common Functions
- Dispatcher Control Functions

Version Information Functions

These functions return the version number and other information about the active Intel IPP software.

- GetLibVersion

GetLibVersion

Returns information about the active version of the Intel IPP signal processing software.

Syntax

```
const IppLibraryVersion* ippsGetLibVersion(void);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Description

This function returns a pointer to a static data structure `IppLibraryVersion` that contains information about the current version of the Intel IPP software for signal processing. There is no need for you to release memory referenced by the returned pointer, as it points to a static variable. The following fields of the `IppLibraryVersion` structure are available:

`major`	Major number of the current library version.
`minor`	Minor number of the current library version.
`majorBuild`	Update number.
`build`	Build revision number.
`targetCpu[4]`	Intel® processor.
`Name`	Name of the current library version.
`Version`	Library version string.
`BuildDate`	Library version actual build date.

For example, if the library version is “9.0 ”, build revision number is “49671”, library name is “ippSP AVX2”, target CPU is processor with Intel® Advanced Vector Extensions 2 (Intel® AVX2) and build date is “Dec 7 2015”, then the fields in this structure are set as:

`major = 9`, `minor = 0`, `Name` = “ippSP AVX2”, `Version` = “9.0.1 (r49671)”, `targetCpu[4]` =“h9”, `BuildDate` = “Dec 7 2015”

> **NOTE** Each sub-library in the signal processing domain has its own similar function to retrieve information about the active library version. Version information functions for sub-libraries have the same interface as `ippsGetLibVersion`.

The following table provides the list of version information functions and respective header files where these functions are declared:

Function Name	Header File
`ippGetLibVersion`	`ippcore.h`
`ippccGetLibVersion`	`ippcc.h`
`ippcvGetLibVersion`	`ippcv.h`
`ippdcGetLibVersion`	`ippdc.h`
`ippvmGetLibVersion`	`ippvm.h`
`ippeGetLibVersion`	`ippe.h`

Example

The following example shows how to use the `ippsGetLibVersion` function :

```
const IppLibraryVersion* lib;
    lib = ippsGetLibVersion();


printf("major = %d\n",lib->major);
printf("minor = %d\n",lib->minor);
printf("majorBuild = %d\n",lib->majorBuild);
printf("build = %d\n",lib->build);
printf("targetCpu = %c%c%c%c\n",lib->targetCpu[0],lib->targetCpu[1],lib->targetCpu[2],lib-
>targetCpu[3]);
```

```
printf("Name = %s\n", lib->Name);
printf("Version = %s\n", lib->Version);
printf("BuildDate = %s\n", lib->BuildDate);
```

Memory Allocation Functions

This section describes the Intel IPP signal processing functions that allocate aligned memory blocks for data of required type or free the previously allocated memory. The size of allocated memory is specified by the number of allocated elements `len`.

> **NOTE** Use the `ippsFree()` to free memory allocated by `ippsMalloc()`. Use `free` to free memory allocated by `malloc` or `calloc`.

- Malloc
- Free

Malloc

Allocates memory aligned to 64-byte boundary.

Syntax

Case 1: Memory allocation for blocks of 32-bit length

```
Ipp8u* ippsMalloc_8u(int len);

Ipp16u* ippsMalloc_16u(int len);

Ipp32u* ippsMalloc_32u(int len);

Ipp8s* ippsMalloc_8s(int len);

Ipp16s* ippsMalloc_16s(int len);

Ipp32s* ippsMalloc_32s(int len);

Ipp64s* ippsMalloc_64s(int len);

Ipp32f* ippsMalloc_32f(int len);

Ipp64f* ippsMalloc_64f(int len);

Ipp8sc* ippsMalloc_8sc(int len);

Ipp16sc* ippsMalloc_16sc(int len);

Ipp32sc* ippsMalloc_32sc(int len);

Ipp64sc* ippsMalloc_64sc(int len);

Ipp32fc* ippsMalloc_32fc(int len);

Ipp64fc* ippsMalloc_64fc(int len);
```

Case 2: Memory allocation for platform-aware functions

```
Ipp8u* ippsMalloc_8u_L(IppSizeL len);

Ipp16u* ippsMalloc_16u_L(IppSizeL len);

Ipp32u* ippsMalloc_32u_L(IppSizeL len);

Ipp8s* ippsMalloc_8s_L(IppSizeL len);
```

```
Ipp16s* ippsMalloc_16s_L(IppSizeL len);
Ipp32s* ippsMalloc_32s_L(IppSizeL len);
Ipp64s* ippsMalloc_64s_L(IppSizeL len);
Ipp32f* ippsMalloc_32f_L(IppSizeL len);
Ipp64f* ippsMalloc_64f_L(IppSizeL len);
Ipp8sc* ippsMalloc_8sc_L(IppSizeL len);
Ipp16sc* ippsMalloc_16sc_L(IppSizeL len);
Ipp32sc* ippsMalloc_32sc_L(IppSizeL len);
Ipp64sc* ippsMalloc_64sc_L(IppSizeL len);
Ipp32fc* ippsMalloc_32fc_L(IppSizeL len);
Ipp64fc* ippsMalloc_64fc_L(IppSizeL len);
```

Include Files

`ipps.h`

Flavors with the `_L` suffix: `ipps_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`len`	Number of elements to allocate.

Description

This function allocates memory block aligned to 64-byte boundary for elements of different data types.

The function `ippsMalloc_<dataType>` uses a 32-bit length, allowing it to allocate memory up to approximately 2GB. If a larger memory size is requested, the function will return a `NULL` pointer. To allocate memory exceeding this limit, use the platform-aware `Malloc` function (`ippsMalloc_<dataType>_L`).

NOTE Not aligned memory allocation could cause not reproducibleperformance and precision results.

Example

The following example shows how to use the `ippsMalloc_8u` function:

```
void func_malloc(void)
{
    Ipp8u* pBuf = ippsMalloc_8u(8*sizeof(Ipp8u));
    if(NULL == pBuf)
      // not enough memory


    ippsFree(pBuf);
}
```

Return Values

The return value of `ippsMalloc` is a pointer to an aligned memory block. If no memory is available in the system, then the `NULL` value is returned. To free this block, use the `ippsFree` function.

See Also

Free Frees memory allocated by the function ippsMalloc.

Free

Frees memory allocated by the function `ippsMalloc`.

Syntax

```
void ippsFree(void* ptr);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`ptr`	Pointer to a memory block to be freed. The memory block pointed to with `ptr` is allocated by the function `ippsMalloc`.

Description

This function frees the aligned memory block allocated by the function `ippsMalloc`.

NOTE Use the `ippsFree()` to free memory allocated by `ippsMalloc()`. Use `free` to free memory allocated by `malloc` or `calloc`.

Common Functions

This section describes the Intel IPP functions that perform special operations common for all domains. All these functions are grouped in the separate sub-library called `ippcore`.

- GetStatusString
- GetL2CacheSize
- GetCacheParams
- GetCpuClocks
- GetCpuFreqMhz
- GetCpuFeatures
- GetEnabledCpuFeatures
- GetMaxCacheSizeB
- SetCpuFeatures
- SetFlushToZero

- SetDenormAreZeros
- AlignPtr
- SetNumThreads
- GetNumThreads
- Malloc
- Free

GetStatusString

Translates a status code into a message.

Syntax

```
const char* ippGetStatusString(IppStatus stsCode);
```

Include Files

`ippcore.h`

Parameters

`stsCode`	Code that indicates the status type (see Error Status Values and Messages).

Description

This function returns a pointer to the text string associated with a status code of `IppStatus` type . Use this function to produce error and warning messages for users. The returned pointer is a pointer to an internal static buffer and does not need to be released.

Example

The following code example shows how to use the function `ippGetStatusString`. If you call an Intel IPP function `ippsAddC_16s_I` with a `NULL` pointer, it returns an error code `-8`. The status information function translates this code into the corresponding message `"Null Pointer Error"`.

```
void statusinfo(void) {
       IppStatus st = ippsAddC_16s_I (3, 0, 0);
       printf("%d : %s\n", st, ippGetStatusString(st));
}
```

Output:

```
-8, Null Pointer Error
```

GetL2CacheSize

Retrieves L2 cache size, in bytes.

Syntax

```
IppStatus ippGetL2CacheSize(int* pSize);
```

Include Files

`ippcore.h`

Parameters

pSize	Pointer to an integer number to store the cache size.

Description

The ippGetL2CacheSize function retrieves L2 cache size for the CPU on which it is executed. This function is based on function #4 of the CPUID instruction, and therefore works only for the CPUs that support this function. For old and non-Intel CPUs that do not support this CPUID extension, the function returns the ippStsCpuNotSupportedErr status. It means that L2 cache size cannot be obtained with the ippGetL2CacheSize function and you should use other methods based on a particular CPU specification.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition when the pSize pointer is NULL.
ippStsNotSupportedCpu	Indicates that the processor is not supported.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

#include <stdio.h>
#include "ipp.h"

static char *cacheType[] = {"Data Cache", "Instruction Cache", "Unified Cache"};

int main()
{
    IppCache *pCacheInfo;
    int i;
    IppStatus sts;

    sts = ippGetCacheParams(&pCacheInfo);
    if (sts != ippStsNoErr) {
        printf("Intel(R) Integrated Performance Primitives (Intel(R) IPP) function returned
error %s\n", ippGetStatusString(sts));
        return 0;
    }
    i = 0;
```

```
    do {
        printf("cache type  = %s\n", cacheType[pCacheInfo[i].type - 1]);
        printf("cache level = %d\n", pCacheInfo[i].level);
        printf("cache size  = %d\n", pCacheInfo[i].size);
        printf("+--------------------------------------+\n");
    } while (pCacheInfo[++i].type > 0);

    sts = ippGetL2CacheSize(&i);
    printf("\nCache L2 size = %d\n", i);

    return 0;
}
```

GetCacheParams

Retrieves cache type, level, and size.

Syntax

```
IppStatus ippGetCacheParams(IppCache** ppCacheInfo);
```

Include Files

ippcore.h

Parameters

ppCacheInfo	Pointer to an array of structures describing CPU cache, which are defined in ipptypes.h: • 0 • NULL - no more caches • 1 • Data cache • 2 • Instruction cache • 3 • Unified cache

Description

The ippGetCacheParams function retrieves the following cache parameters for the CPU on which it is executed: type of cache (instruction, data, unified), cache level in cache hierarchy, and cache size, in bytes. The function is based on function #4 of the CPUID instruction, and therefore works only for the CPUs that support this function. For old and non-Intel CPUs that do not support this CPUID extension, the function returns the ippStsCpuNotSupportedErr status. It means that cache parameters cannot be obtained with the ippGetCacheParams function and you should use other methods based on a particular CPU specification.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition when the ppCacheInfo pointer is NULL.

ippStsNotSupportedCpu	Indicates that the processor is not supported.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

#include <stdio.h>
#include "ipp.h"

static char *cacheType[] = {"Data Cache", "Instruction Cache", "Unified Cache"};

int main()
{
    IppCache *pCacheInfo;
    int i;
    IppStatus sts;

    sts = ippGetCacheParams(&pCacheInfo);
    if (sts != ippStsNoErr) {
        printf("Intel(R) Integrated Performance Primitives (Intel(R) IPP) function returned
error %s\n", ippGetStatusString(sts));
        return 0;
    }
    i = 0;
    do {
        printf("cache type  = %s\n", cacheType[pCacheInfo[i].type - 1]);
        printf("cache level = %d\n", pCacheInfo[i].level);
        printf("cache size  = %d\n", pCacheInfo[i].size);
        printf("+--------------------------------------+\n");
    } while (pCacheInfo[++i].type > 0);

    sts = ippGetL2CacheSize(&i);
    printf("\nCache L2 size = %d\n", i);

    return 0;
}
```

GetCpuClocks

Returns a current value of the time stamp counter (TSC) register.

Syntax

```
Ipp64u ippGetCpuClocks (void);
```

Include Files

`ippcore.h`

Description

This function reads the current state of the TSC register and returns its value.

GetCpuFreqMhz

Estimates the processor operating frequency.

Syntax

```
IppStatus ippGetCpuFreqMhz(int* pMhz);
```

Include Files

`ippcore.h`

Parameters

`pMhz`	Pointer to the result.

Description

This function estimates the processor operating frequency and returns its value, in MHz as an integer stored in `pMhz`. The estimated value can vary depending on the processor workload.

> **NOTE** To improve precision of the return value, this function accumulates CPU clocks. This operation takes several seconds and may result in long execution time.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition when the `pMhz` pointer is `NULL`.

GetCpuFeatures

Retrieves the processor features.

Syntax

```
IppStatus ippGetCpuFeatures(Ipp64u* pFeaturesMask, Ipp32u pCpuidInfoRegs[4]);
```

Include Files

`ippcore.h`

Parameters

pFeaturesMask	Pointer to the features mask. Possible value is ippCPUID_GETINFO_A.
pCpuidInfoRegs	Pointer to the vector with four elements to store the data from the registers eax, ebx, ecx, edx of the function CPUID.1.

Description

This function retrieves some of the CPU features returned by the function CPUID.1 and stores them consecutively in the mask pFeaturesMask. The following table lists the features stored in the mask.

If pFeaturesMask does not have any input value, then the function retrieves the features in accordance with eax=1 and ecx=0. If pFeaturesMask is set to ippCPUID_GETINFO_A, then the function retrieves the features in accordance with the input values of the registers eax and ecx that are specified in this case by the pCpuidInfoRegs [0] and pCpuidInfoRegs [2] respectively.

Mask Value	Bit Name	Feature	Mask Bit Number
0x00000001	ippCPUID_MMX	MMX™ technology	0
0x00000002	ippCPUID_SSE	Intel® Streaming SIMD Extensions	1
0x00000004	ippCPUID_SSE2	Intel® Streaming SIMD Extensions 2	2
0x00000008	ippCPUID_SSE3	Intel® Streaming SIMD Extensions 3	3
0x00000010	ippCPUID_SSSE3	Supplemental Streaming SIMD Extensions	4
0x00000020	ippCPUID_MOVBE	MOVBE instruction is supported	5
0x00000040	ippCPUID_SSE41	Intel® Streaming SIMD Extensions 4.1	6
0x00000080	ippCPUID_SSE42	Intel® Streaming SIMD Extensions 4.2	7
0x00000100	ippCPUID_AVX	The processor supports Intel® Advanced Vector Extensions (Intel® AVX) instruction set	8
0x00000200	ippAVX_ENABLEDBYOS	The operating system supports Intel® AVX	9
0x00000400	ippCPUID_AES	Advanced Encryption Standard (AES) instructions are supported	10
0x00000800	ippCPUID_CLMUL	PCLMULQDQ instruction is supported	11

0x00002000	ippCPUID_RDRAND	Read Random Number instructions are supported	13
0x00004000	ippCPUID_F16C	16-bit floating point conversion instructions are supported	14
0x00008000	ippCPUID_AVX2	Intel® Advanced Vector Extensions2 (Intel® AVX2)instruction set is supported	15
0x00010000	ippCPUID_ADCOX	ADCX and ADOX instructions are supported	16
0x00020000	ippCPUID_RDSEED	Read Random SEED instruction is supported.	17
0x00040000	ippCPUID_PREFETCHW	PREFETCHW instruction is supported	18
0x00080000	ippCPUID_SHA	Intel® Secure Hash AlgorithmExtensions (Intel® SHA Extensions)are supported	19
0x00100000	ippCPUID_AVX512F	Intel® Advanced Vector Extensions512 (Intel® AVX-512) foundation instructions are supported	20
0x00200000	ippCPUID_AVX512CD	Intel® AVX-512 conflict detection instructions are supported	21
0x00400000	ippCPUID_AVX512ER	Intel® AVX-512 exponential and reciprocal instructions are supported	22
0x80000000	ippCPUID_KNC	Intel® Xeon Phi™ is supported	23

All features returned by the `CPUID.1` function can be stored in the vector with four elements `pCpuidInfoRegs` where each element contains data from one of the registers `eax`, `ebx`, `ecx`, `edx` respectively. If these data are not required, the pointer `pCpuidInfoRegs` must be set to `NULL`.

NOTE Intel ® Itanium ® processors are not supported.
Support for Intel ® Advanced Vector Extensions (Intel ® AVX) and Supplemental Streaming SIMD Extensions 3 (SSSE3) has been deprecated and will be removed in a future release.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition when the pFeaturesMask pointer is NULL.
ippStsNotSupportedCpu	Indicates that the processor is not supported.

GetEnabledCpuFeatures

Returns a features mask for enabled processor features.

Syntax

```
Ipp64u ippGetEnabledCpuFeatures (void);
```

Include Files

ippcore.h

Description

This function detects the enabled CPU features for the currently loaded libraries and returns the corresponding features mask.

See Also

GetCpuFeatures Retrieves the processor features.

GetMaxCacheSizeB

Returns maximum size of the L2 and L3 caches of the processor.

Syntax

```
IppStatus ippGetMaxCacheSizeB(int* pSizeByte);
```

Include Files

ippcore.h

Parameters

pSizeByte	Pointer to the output result.

Description

This function finds the maximum size (in bytes) of the L2 and L3 caches of the processor used on your computer system. The result is stored in the pSizeByte.

NOTE Intel® Itanium® processors are not supported.

If the processor is not supported, or size of cache is unknown, the result is 0, and the function returns corresponding warning message.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition when the pSizeByte pointer is NULL.
ippStsNotSupportedCpu	Indicates that the processor is not supported.
ippStsUnknownCacheSize	Indicates that the size of the cache is unknown.

SetCpuFeatures

Sets the processor-specific library code for the specified processor features.

Syntax

```
IppStatus ippSetCpuFeatures(Ipp64u cpuFeatures);
```

Include Files

ippcore.h

Parameters

cpuFeatures	Features to be supported by the library. Refer to ippdefs.h for ippCPUID_xx definition.

Description

This function sets the processor-specific code of the Intel IPP library according to the processor features specified in cpuFeatures. You can use the following predefined sets of features (the FM suffix below means *feature mask*):

64-bit code:

```
#define M7_FM ( PX_FM | ippCPUID_SSE3 )
#define Y8_FM ( U8_FM | ippCPUID_SSE41 | ippCPUID_SSE42 )
#define L9_FM ( E9_FM | ippCPUID_MOVBE | ippCPUID_AVX2 | ippCPUID_PREFETCHW )
#define K0_FM ( L9_FM | ippCPUID_AVX512F | ippCPUID_AVX512CD | ippCPUID_AVX512VL |
ippCPUID_AVX512BW | ippCPUID_AVX512DQ | ippAVX512_ENABLEDBYOS )
```

NOTE Do not use any other Intel IPP function while ippSetCpuFeatures is executing. Otherwise, your application behavior is undefined.
Support for Intel® Advanced Vector Extensions (Intel® AVX) and Supplemental Streaming SIMD Extensions 3 (SSSE3) has been deprecated and will be removed in a future release.

To avoid initialization of internal structures for one Intel® architecture and then call of the processing function that is optimized for another architecture, do not use the ippSetCpuFeatures function in chains of Intel IPP connected calls like <processing functionGetSize + <processing functionInit + <processing function>. Otherwise, Intel IPP functionality behavior is undefined.

The Intel IPP library supports two internal sets of CPU features:

- *Real CPU features*: the features that are supported by the CPU at which the library is executed. These features are read-only and can be obtained with the ippGetCpuFeatures function.

- *Enabled features*: the features that are enabled externally to Intel IPP by the application. These features are read-write and can be obtained with ippGetEnabledCpuFeatures

  and set with `ippSetCpuFeatures`.

The `ippSetCpuFeatures` function provides additional flexibility in measuring performance improvements reached by using specific CPU features. For example, the call of the ippInit() function (or the first call of any Intel IPP function) in an application running on an Intel® processor with 64-bit OS installed dispatches the L9 code version optimized for Intel® Advanced Vector Extensions 2 (Intel® AVX2) with several other features like fast 16-bit floating point support. To check performance improvement for all Intel IPP functionality reached by using Intel® AVX2, you can run a benchmark for the currently dispatched version of code and then compare performance with the Intel® Streaming SIMD Extensions 4.2 (Intel® SSE 4.2) version of code with Intel® AVX2 disabled. To disable Intel AVX2, call `ippSetCpuFeatures(Y8_FM)`. To enable Intel AVX2 back, call `ippSetCpuFeatures( L9_FM )`. Thus, you can use the `ippSetCpuFeatures` function to dispatch any version of Intel IPP code and enable/disable specific CPU features. If you are not well familiar with the features of your CPU, use the ippInit() function (or auto-initialization mechanism available starting with Intel IPP 9.0) for the default library behavior.

Return Values

`ippStsNoErr`	Indicates that the required processor-specific code is successfully set.
`ippStsCpuMismatch`	Indicates that the specified processor features are not valid. Previously set code is used. If the requested feature is below the minimal supported by the **w7** library - that is Intel® SSE3 for Intel® 64 architecture, **w7** code is dispatched respectively.
`ippStsFeatureNotSupported`	Indicates that the current CPU does not support at least one of the requested features. If the `ippCPUID_NOCHECK` bit of the `cpuFeatures` parameter is set to 1, these not supported features are enabled, otherwise - disabled.
`ippStsUnknownFeature`	Indicates that at least one of the requested features is unknown. It means that the feature is not defined in the `ippdefs.h` file. Further behavior of the library depends on known features passed to `cpuFeatures`. Unknown features are ignored.
`ippStsFeaturesCombination`	Indicates that the combination of features is not correct. For example, `ippSetCpuFeatures(ippCPUID_AVX2);` will generate this warning that means that `ippCPUID_AVX2` bit is set to 1 in `cpuFeatures`, but at least one of the `ippCPUID_MMX`, `ippCPUID_SSE`, …, `ippCPUID_AVX` bits or all of these bits are not set. Use `ippSetCpuFeatures(L9_FM)` instead to avoid this warning. All the missing bits, if supported by CPU, are set to 1. This means that if the library supports the Intel® AVX2 code, it also internally uses all known MMX™, Intel® SSE, and Intel® AVX extensions, which are below Intel® AVX2.

See Also

Init Automatically initializes the library code that is most appropriate for the current processor type.

GetCpuFeatures Retrieves the processor features.

GetEnabledCpuFeatures Returns a features mask for enabled processor features.

SetFlushToZero

Enables or disables flush-to-zero (FTZ) mode.

Syntax

```
IppStatus ippSetFlushToZero(int value, unsigned int* pUMask);
```

Include Files

`ippcore.h`

Parameters

`value`	Switch to set or clear the corresponding bit of the MXCSR register.
`pUMask`	Pointer to the current underflow exception mask; may be set to `NULL`.

Description

This function enables FTZ mode for processors that support Intel® Streaming SIMD Extensions [xx] instructions. The FTZ mode controls the masked response to a SIMD floating-point underflow condition. Use this function to improve performance of applications where underflows are common and rounding the underflow result to zero is acceptable.

FTZ mode is possible only when the mask register is in a certain state. The `ippSetFlushToZero` function checks and changes this state if necessary. After disabling the FTZ mode, you can restore the initial mask register state. To do this, declare a variable of `unsigned integer` type in your application and point to it the parameter `pUMask` of the `ippSetFlushToZero` function. The initial state of mask register is saved in this location and can be restored later. If you do not need to restore the initial mask state, then the pointer `pUMask` may be set to `NULL`.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsCpuNotSupportedErr`	Indicates an error condition when the FTZ mode is not supported by the processor.

SetDenormAreZeros

Enables or disables denormals-are-zero (DAZ) mode.

Syntax

```
IppStatus ippSetDenormAreZeros(int value);
```

Include Files

ippcore.h

Parameters

value	Switch to set or clear the corresponding bit of the MXCSR register.

Description

This function enables the DAZ mode for processors that support Intel® Streaming SIMD Extensions instructions. The DAZ mode controls the processor response to a SIMD floating-point denormal operand condition. When the DAZ flag is set, the processor converts all denormal source operands to zero with the sign of the original operand before performing any computations on source data. Use this function to improve processor performance of applications such as streaming media processing, where rounding a denormal operand to zero does not noticeably affect the quality of the processed data.

Return Values

ippStsNoErr	Indicates no error.
ippStsCpuNotSupportedErr	Indicates an error condition when the DAZ mode is not supported by the processor.

AlignPtr

Aligns a pointer to the specified number of bytes.

Syntax

```
void* ippAlignPtr(void* ptr, int alignBytes);
```

Include Files

ippcore.h

Parameters

ptr	Aligned pointer.
alignBytes	Number of bytes to align. Possible values are the powers of 2, that is, 2, 4, 8, 16 and so on.

Description

This function returns a pointer `ptr` aligned to the specified number of bytes `alignBytes`. Possible values of `alignBytes` are powers of two. The function does not check the validity of this parameter.

NOTE Do not free the pointer returned by the function, but free the original pointer.

SetNumThreads

Sets the number of threads in the multithreading environment.

Syntax

Case 1: Setting number of threads for operations with TL functions based on the Platform Aware API

```
IppStatus ippSetNumThreads_LT(int numThr);
```

Case 2: Setting number of threads for operations with TL functions based on the Classic API

```
IppStatus ippSetNumThreads_T(int numThr);
```

Include Files

`ippcore.h`

Parameters

`numThr`	Number of threads, should be more than zero.

Description

This function sets the number of OpenMP* threads. A number of established threads may be less than specified `numThr`. Functions are not thread-safe and shall be called outside of the parallel region of the program.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsSizeErr`	Indicates an error when `numThr` is less than, or equal to zero.
`ippStsNoOperation`	Indicates that the function is called from the application linked to the single-threaded version of the library. No operation is performed.
`ippStsOperationNotSupported`	Indicates that function trying to set the number of TBB threads.

GetNumThreads

Returns the number of existing threads in the multithreading environment.

Syntax

Case 1: Getting number of threads for operations with TL functions based on the Platform Aware API

```
IppStatus ippGetNumThreads_LT(int* pNumThr);
```

Case 2: Getting number of threads for operations with TL functions based on the Classic API

```
IppStatus ippGetNumThreads_T(int* pNumThr);
```

Include Files

`ippcore.h`

ippcore_tl.h

Parameters

pNumThr	Pointer to the number of threads.

Description

This function returns the number of OpenMP* threads specified by the user previously. If it is not specified, the function returns the initial number of threads that depends on the number of logical processors. Functions are not thread-safe, first call shall be outside of the parallel region of the program.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition when the pNumThr pointer is NULL.
ippStsNoOperation	Indicates that the function is called from the application linked to the single-threaded version of the library. No operation is performed and return value is always == 1.

Malloc

Allocates memory aligned to 64-byte boundary.

Syntax

```
void* ippMalloc(int length);
```

Memory allocation for platform-aware functions

```
void* ippMalloc_L(IppSizeL length);
```

Include Files

ippcore.h

Flavors with the _L suffix: ippcore_l.h

Parameters

length	Size (in bytes) of the allocated block.

Description

This function allocates a memory block aligned to a 64-byte boundary.

Return Values

The return value of ippMalloc is a pointer to an aligned memory block. To free this block, use the ippFree function.

See Also

Free Frees memory allocated by the function ippMalloc.

Free

Frees memory allocated by the function `ippMalloc`.

Syntax

```
void ippFree(void* ptr);
```

Include Files

`ippcore.h`

Parameters

`ptr`	Pointer to a memory block to be freed.

Description

This function frees an aligned memory block previously allocated by the function `ippMalloc`.

> **NOTE** Use the `ippFree()` to free memory allocated by `ippMalloc()`. Use `free` to free memory allocated by `malloc` or `calloc`.

Dispatcher Control Functions

This section describes Intel IPP functions that control the dispatchers of the merged static libraries.

- Init

Init

Automatically initializes the library code that is most appropriate for the current processor type.

Syntax

```
IppStatus ippInit(void));
```

Include Files

`ippcore.h`

Description

This function detects the processor type used in the user computer system and sets the processor-specific code of the Intel IPP library most appropriate for the current processor type.

> **NOTE** You can not use any other Intel IPP function while the function `ippInit` continues execution.

Return Values

ippStsNoErr	Indicates that the required processor-specific code is successfully set.
ippStsNotSupportedCpu	Indicates that the CPU is not supported.
ippStsNonIntelCpu	Indicates that the target CPU is not Genuine Intel.

Vector Initialization Functions

This chapter describes the Intel® IPP functions that initialize vectors with either constants, the contents of other vectors, or the generated signals.

- Vector Initialization Functions
- Sample-Generating Functions

Vector Initialization Functions

This section describes functions that initialize the values of vector elements. All vector elements can be initialized to a common zero or another specified value. They can also be initialized to respective values of a second vector elements.

- Copy
- CopyLE, CopyBE
- Move
- Set
- Zero

Copy

Copies the contents of one vector into another.

Syntax

```
IppStatus ippsCopy_8u(const Ipp8u* pSrc, Ipp8u* pDst, int len);
IppStatus ippsCopy_16s(const Ipp16s* pSrc, Ipp16s* pDst, int len);
IppStatus ippsCopy_32s(const Ipp32s* pSrc, Ipp32s* pDst, int len);
IppStatus ippsCopy_32f(const Ipp32f* pSrc, Ipp32f* pDst, int len);
IppStatus ippsCopy_64s(const Ipp64s* pSrc, Ipp64s* pDst, int len);
IppStatus ippsCopy_64f(const Ipp64f* pSrc, Ipp64f* pDst, int len);
IppStatus ippsCopy_16sc(const Ipp16sc* pSrc, Ipp16sc* pDst, int len);
IppStatus ippsCopy_32sc(const Ipp32sc* pSrc, Ipp32sc* pDst, int len);
IppStatus ippsCopy_32fc(const Ipp32fc* pSrc, Ipp32fc* pDst, int len);
IppStatus ippsCopy_64sc(const Ipp64sc* pSrc, Ipp64sc* pDst, int len);
IppStatus ippsCopy_64fc(const Ipp64fc* pSrc, Ipp64fc* pDst, int len);
```

Platform-aware functions:

```
IppStatus ippsCopy_32f_L(const Ipp32f* pSrc, Ipp32f* pDst, IppSizeL len);
```

```
IppStatus ippsCopy_64f_L(const Ipp64f* pSrc, Ipp64f* pDst, IppSizeL len);
IppStatus ippsCopy_32fc_L(const Ipp32fc* pSrc, Ipp32fc* pDst, IppSizeL len);
IppStatus ippsCopy_64fc_L(const Ipp64fc* pSrc, Ipp64fc* pDst, IppSizeL len);
```

Include Files

`ipps.h`

Flavors with the `_L` suffix: `ipps_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

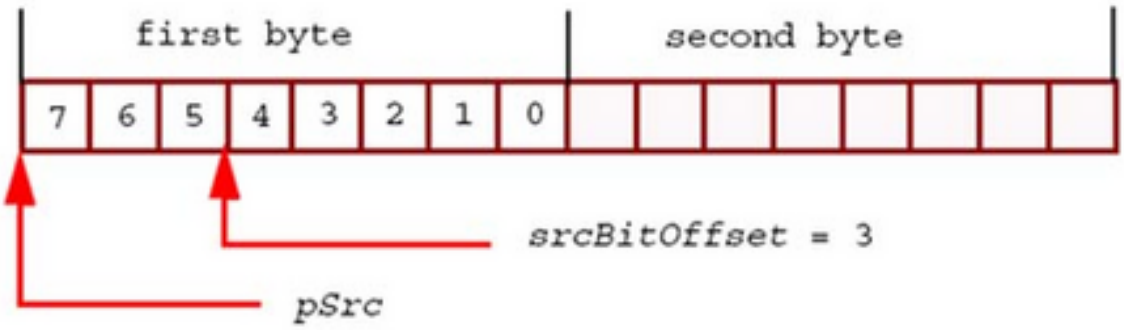
`pSrc`	Pointer to the source vector.
`pDst`	Pointer to the destination vector.
`len`	Number of elements to copy.

Description

This function copies the first `len` elements from a source vector `pSrc` into a destination vector `pDst`.

``ippsCopy_1u``. This function flavor copies elements of a vector that has a `8u` data type. It means that each byte consists of eight consecutive elements of the vector (1 bit per element). You need to specify the start position of the source and destination vectors in the `srcBitOffset` and `dstBitOffset` parameters, respectively. The bit order of each byte is inverse to the element order. It means that the first element in a vector represents the last (seventh) bit of the first byte in a vector, as shown in the figure below.

Bit Layout for the Function `ippsCopy_1u`.



NOTE These functions perform only copying operations described above and are not intended to move data. Their behavior is unpredictable if source and destination buffers are overlapping. To move data, use `ippsMove`.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when the pSrc or pDst pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to zero.

Example

The example below shows how to use the ippsCopy function.

```
IppStatus copy(void) {
       char src[] = "to be copied\0";
       char dst[256];
       return ippsCopy_8u(src, dst, strlen(src)+1);
  }
```

See Also

Move Moves the contents of one vector to another vector.

CopyLE, CopyBE

Copies the contents of one bit vector into another.

Syntax

```
IppStatus ippsCopyLE_1u(const Ipp8u* pSrc, int srcBitOffset, Ipp8u* pDst, int dstBitOffset, int len);
```

```
IppStatus ippsCopyBE_1u(const Ipp8u* pSrc, int srcBitOffset, Ipp8u* pDst, int dstBitOffset, int len);
```

Include Files

ipps.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

pSrc	Pointer to the source vector.
pDst	Pointer to the destination vector.
len	Number of elements to copy.
srcBitOffset	Offset, in bits, from the first byte of the source vector.

dstBitOffset	Offset, in bits, from the first byte of the destination vector.

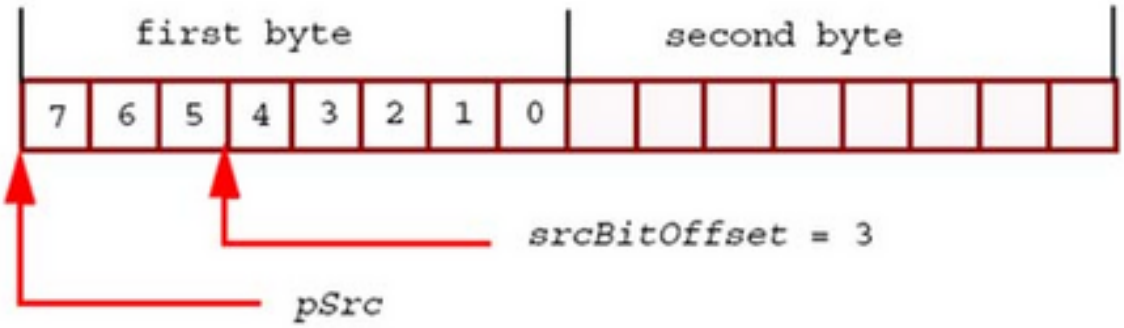
Description

This function copies the first len elements from a source vector pSrc into a destination vector pDst.

These functions copy elements of a vector that has a 8u data type. It means that each byte consists of eight consecutive elements of the vector (1 bit per element). You need to specify the start position of the source and destination vectors in the srcBitOffset and dstBitOffset parameters, respectively.

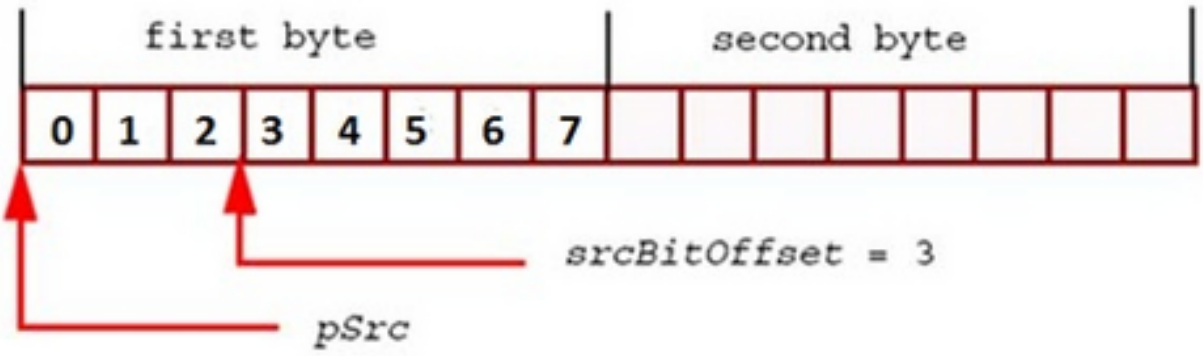
For the ippsCopyLE_1u function, the bit order of each byte is inverse to the element order. It means that the first element in a vector represents the last (seventh) bit of the first byte in a vector, as shown in the figure below.

Bit Layout for the ippsCopyLE_1u Function



For the ippsCopyBE_1u function, the bit order of each byte is ordinary. It means that the first element in a vector represents the last (zero) bit of the first byte in a vector, as shown in the figure below.

Bit Layout for the ippsCopyBE_1u Function



Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when the pSrc or pDst pointer is NULL.

ippStsSizeErr	Indicates an error when:

Move

Moves the contents of one vector to another vector.

Syntax

```
IppStatus ippsMove_8u(const Ipp8u* pSrc, Ipp8u* pDst, int len);
IppStatus ippsMove_16s(const Ipp16s* pSrc, Ipp16s* pDst, int len);
IppStatus ippsMove_32s(const Ipp32s* pSrc, Ipp32s* pDst, int len);
IppStatus ippsMove_32f(const Ipp32f* pSrc, Ipp32f* pDst, int len);
IppStatus ippsMove_64f(const Ipp64f* pSrc, Ipp64f* pDst, int len);
IppStatus ippsMove_64s(const Ipp64s* pSrc, Ipp64s* pDst, int len);
IppStatus ippsMove_16sc(const Ipp16sc* pSrc, Ipp16sc* pDst, int len);
IppStatus ippsMove_32sc(const Ipp32sc* pSrc, Ipp32sc* pDst, int len);
IppStatus ippsMove_32fc(const Ipp32fc* pSrc, Ipp32fc* pDst, int len);
IppStatus ippsMove_64sc(const Ipp64sc* pSrc, Ipp64sc* pDst, int len);
IppStatus ippsMove_64fc(const Ipp64fc* pSrc, Ipp64fc* pDst, int len);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pSrc`	Pointer to the source vector used to initialize `pDst`.
`pDst`	Pointer to the destination vector to be initialized.
`len`	Number of elements to move.

Description

This function moves the first `len` elements from a source vector `pSrc` into the destination vector `pDst`. If some parts of the source and destination vectors are overlapping, then the function ensures that the original source bytes in the overlapping parts are moved (it means that they are copied before being overwritten) to the appropriate parts of the destination vector.

Return Values

ippStsNoErr	Indicates no error.

ippStsNullPtrErr	Indicates an error when the pSrc or pDst pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to zero.

Example

```
Ipp8u pSrc[10] = { "123456789" };
Ipp8u pDst[6];
int len = 6;
IppStatus status;


status = ippsMove_8u ( pSrc, pDst, len );
if(ippStsNoErr != status)
  printf("Intel® IPP Error: %s",ippGetStatusString(status));
```

Result:

```
pSrc = 123456789
pDst = 123456
```

Set

Initializes vector elements to a specified common value.

Syntax

```
IppStatus ippsSet_8u(Ipp8u val, Ipp8u* pDst, int len);IppStatus ippsSet_16s(Ipp16s val,
Ipp16s* pDst, int len);IppStatus ippsSet_16sc(Ipp16sc val, Ipp16sc* pDst, int
len);IppStatus ippsSet_32s(Ipp32s val, Ipp32s* pDst, int len);IppStatus
ippsSet_32f(Ipp32f val, Ipp32f* pDst, int len);IppStatus ippsSet_32sc(Ipp32sc val,
Ipp32sc* pDst, int len);IppStatus ippsSet_32fc(Ipp32fc val, Ipp32fc* pDst, int
len);IppStatus ippsSet_64s(Ipp64s val, Ipp64s* pDst, int len);IppStatus
ippsSet_64f(Ipp64f val, Ipp64f* pDst, int len);IppStatus ippsSet_64sc(Ipp64sc val,
Ipp64sc* pDst, int len);IppStatus ippsSet_64fc(Ipp64fc val, Ipp64fc* pDst, int len);
```

Include Files

ipps.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

pDst	Pointer to the vector to be initialized.
len	Number of elements to initialize.
val	Value used to initialize the vector pDst.

Description

This function initializes the first `len` elements of the real or complex vector `pDst` to contain the same value `val`.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when the `pDst` pointer is `NULL`.
`ippStsSizeErr`	Indicates an error when `len` is less than or equal to zero.

Example

The code example below shows how to use the function `ippsSet`.

```
IppStatus set(void) {
      char src[] = "set";
      return ippsSet_8u('0', src, strlen(src));
}
```

Zero

Initializes a vector to zero.

Syntax

```
IppStatus ippsZero_8u(Ipp8u* pDst, int len);
IppStatus ippsZero_16s(Ipp16s* pDst, int len);
IppStatus ippsZero_32s(Ipp32s* pDst, int len);
IppStatus ippsZero_32f(Ipp32f* pDst, int len);
IppStatus ippsZero_64s(Ipp64s* pDst, int len);
IppStatus ippsZero_64f(Ipp64f* pDst, int len);
IppStatus ippsZero_16sc(Ipp16sc* pDst, int len);
IppStatus ippsZero_32sc(Ipp32sc* pDst, int len);
IppStatus ippsZero_32fc(Ipp32fc* pDst, int len);
IppStatus ippsZero_64sc(Ipp64sc* pDst, int len);
IppStatus ippsZero_64fc(Ipp64fc* pDst, int len);
```

Platform-aware functions:

```
IppStatus ippsZero_32f_L(Ipp32f* pDst, IppSizeL len);
IppStatus ippsZero_64f_L(Ipp64f* pDst, IppSizeL len);
IppStatus ippsZero_32fc_L(Ipp32fc* pDst, IppSizeL len);
IppStatus ippsZero_64fc_L(Ipp64fc* pDst, IppSizeL len);
```

Include Files

`ipps.h`

Flavors with the _L suffix: ipps_l.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

pDst	Pointer to the vector to be initialized to zero.
len	Number of elements to initialize.

Description

This function initializes the first len elements of the vector pDst to zero. If pDst is a complex vector, both real and imaginary parts are zeroed.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when the pDst pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to zero.

Example

The code example below shows how to use the ippsZero function.

```
IppStatus zero(void) {
   char src[] = "zero";
   return ippsZero_8u(src, strlen(src));
}
```

Sample-Generating Functions

This section describes Intel IPP functions which generate tone samples, triangle samples, pseudo-random samples with uniform distribution, and pseudo-random samples with Gaussian distribution, as well as special test samples.

Some sample-generating functions operate with data in the fixed point format. These functions have Q15 suffix in their name. This means that integer data are used in calculations inside the function as real numbers equal to the integer value multiplied by 2^{-15} (where " 15 " is called a *scale factor)*.

- Tone-Generating Functions
- Triangle-Generating Functions
- Uniform Distribution Functions
- Gaussian Distribution Functions
- Special Vector Functions

Tone-Generating Functions

The functions described below generate a tone (or "sinusoid") of a given frequency, phase, and magnitude. Tones are fundamental building blocks for analog signals. Thus, sampled tones are extremely useful in signal processing systems as test signals and as building blocks for more complex signals.

The use of tone functions is preferable against the analogous C math library's `sin()` function for many applications, because Intel IPP functions can use information retained from the computation of the previous sample to compute the next sample much faster than standard `sin()` or `cos()`.

- Tone

Tone

Generates a tone with a given frequency, phase, and magnitude.

Syntax

```
IppStatus ippsTone_16s(Ipp16s* pDst, int len, Ipp16s magn, Ipp32f rFreq, Ipp32f*
pPhase, IppHintAlgorithm hint);
```

```
IppStatus ippsTone_16sc(Ipp16sc* pDst, int len, Ipp16s magn, Ipp32f rFreq, Ipp32f*
pPhase, IppHintAlgorithm hint);
```

```
IppStatus ippsTone_32f(Ipp32f* pDst, int len, Ipp32f magn, Ipp32f rFreq, Ipp32f*
pPhase, IppHintAlgorithm hint);
```

```
IppStatus ippsTone_32fc(Ipp32fc* pDst, int len, Ipp32f magn, float rFreq, Ipp32f*
pPhase, IppHintAlgorithm hint);
```

```
IppStatus ippsTone_64f(Ipp64f* pDst, int len, Ipp64f magn, Ipp64f rFreq, Ipp64f*
pPhase, IppHintAlgorithm hint);
```

```
IppStatus ippsTone_64fc(Ipp64fc* pDst, int len, Ipp64f magn, Ipp64f rFreq, Ipp64f*
pPhase, IppHintAlgorithm hint);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`magn`	Magnitude of the tone, that is, the maximum value attained by the wave.
`pPhase`	Pointer to the phase of the tone relative to a cosine wave. It must be in range [0.0, 2π). You can use the returned value to compute the next continuous data block.
`rFreq`	Frequency of the tone relative to the sampling frequency. It must be in the interval [0.0, 0.5) for real tone and in [0.0, 1.0) for complex tone.
`pDst`	Pointer to the array that stores the samples.

len	Number of samples to be computed.
hint	Suggests using specific code. Possible values for the hint parameter are described in IppHintAlgorithm enum Type.

Description

This function generates the tone with the specified frequency rFreq, phase pPhase, and magnitude magn. The function computes len samples of the tone, and stores them in the array pDst. For real tones, each generated value x[n] is defined as:

x[n] =magn* cos(2πn*rFreq+phase)

For complex tones, x[n] is defined as:

x[n] =magn* (cos(2πn*rFreq+phase)+j* sin(2πn*rFreq+phase))

The parameter hint suggests using specific code, which provides for either fast but less accurate calculation, or more accurate but slower execution.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when the pDst or pPhase pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than, or equal to zero.
ippStsToneMagnErr	Indicates an error when magn is less than, or equal to zero.
ippStsToneFreqErr	Indicates an error when rFreq is negative, or greater than, or equal to 0.5 for real tone and to 1.0 for complex tone.
ippStsTonePhaseErr	Indicates an error when the pPhase value is negative, or greater than or equal to IPP_2PI.

Triangle-Generating Functions

This section describes the functions that generate a periodic signal with a triangular wave form (referred to as “triangle”) of a given frequency, phase, magnitude, and asymmetry.

A real periodic signal with triangular wave form x[n] (referred to as a real triangle) of a given frequency rFreq, phase value phase, magnitude magn, and asymmetry h is defined as follows:

x[n] =magn***ct``** :sub:`h` ``(2π*rFreq*n+phase),n = 0, 1, 2,…

A complexl periodic signal with triangular wave form x[n] (referred to as a complex triangle) of a given frequency rFreq, phase value phase, magnitude magn, and asymmetry h is defined as follows:

x[n] = magn * [**ct``** :sub:`h` ``(2π*rFreq*n+phase) + j ***st``** :sub:`h` ``(2π*rFreq*n+phase)],n = 0, 1, 2,…

The $\mathbf{ct}_h$ () function is determined as follows:

H = Π + h

$$ct_h(\alpha) = \begin{cases} -\frac{2}{H} \cdot \left(\alpha - \frac{H}{2}\right), & 0 \le \alpha \le H \\ \frac{2}{2\pi - H} \cdot \left(\alpha - \frac{2\pi + H}{2}\right), & H \le \alpha \le \pi \end{cases}$$

$$ct_h(\alpha + k \cdot 2\pi) = ct_h(\alpha), \; k = 0, \pm 1, \pm 2, \ldots$$

``**ct**``$_h$(α +k* 2π) =**ct``** :sub:`h` ``(α), k = 0, ±1, ±2, …

When H = π , asymmetry h = 0, and function **ct**$_h$ () is symmetric and a triangular analog of the **cos** () function. Note the following equations:

``**ct**``$_h$(H/2 +k*π) = 0,k = 0, ±1, ±2, …

``**ct**``$_h$(k* 2π) = 1, k = 0, ±1, ±2, …

``**ct**``$_h$(H+k* 2π) = -1,k = 0, ±1, ±2, …

The **st**$_h$ () function is determined as follows:

$$st_h(\alpha) = \begin{cases} \frac{2}{2\pi - H} \cdot \alpha, & 0 \le \alpha \le \frac{2\pi - H}{2} \\ -\frac{2}{H} \cdot (\alpha - \pi), & \frac{2\pi - H}{2} \le \alpha \le \frac{2\pi + H}{2} \\ \frac{2}{2\pi - H} \cdot (\alpha - 2\pi), & \frac{2\pi + H}{2} \le \alpha \le \pi \end{cases}$$

``**st**``$_h$(α +k* 2π) =**st``** :sub:`h` ``(α), k = 0, ±1, ±2, …

When H = π , asymmetry h = 0, and function **st**$_h$ () is symmetric and a triangular analog of the sine function. Note the following equations:

``**st**``$_h$(α) =**ct``** :sub:`h` ``(α + (3π +h)/2),k = 0, ±1, ±2, …

``**st**``$_h$(k* π) = 0, k = 0, ±1, ±2, …

``**st**``$_h$((π -h)/2 +k* 2π) = 1,k = 0, ±1, ±2, …

``**st**``$_h$((3π +h)/2 +k* 2π) = -1,k = 0, ±1, ±2, …

- Triangle

Triangle

Generates a triangle with a given frequency, phase, and magnitude.

Syntax

```
IppStatus ippsTriangle_16s(Ipp16s* pDst, int len, Ipp16s magn, Ipp32f rFreq, Ipp32f
asym, Ipp32f* pPhase);
```

```
IppStatus ippsTriangle_16sc(Ipp16sc* pDst, int len, Ipp16s magn, Ipp32f rFreq, Ipp32f
asym, Ipp32f* pPhase);
```

```
IppStatus ippsTriangle_32f(Ipp32f* pDst, int len, Ipp32f magn, Ipp32f rFreq, Ipp32f
asym, Ipp32f* pPhase);

IppStatus ippsTriangle_32fc(Ipp32fc* pDst, int len, Ipp32f magn, Ipp32f rFreq, Ipp32f
asym, Ipp32f* pPhase);

IppStatus ippsTriangle_64f(Ipp64f* pDst, int len, Ipp64f magn, Ipp64f rFreq, Ipp64f
asym, Ipp64f* pPhase);

IppStatus ippsTriangle_64fc(Ipp64fc* pDst, int len, Ipp64f magn, Ipp64f rFreq, Ipp64f
asym, Ipp64f* pPhase);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`rFreq`	Frequency of the triangle relative to the sampling frequency. It must be in range [0.0, 0.5).
`pPhase`	Pointer to the phase of the triangle relative to a cosine triangular analog wave. It must be in range [0.0, 2π). You can use the returned value to compute the next continuous data block.
`magn`	Magnitude of the triangle, that is, the maximum value attained by the wave.
`asym`	Asymmetry `h` of a triangle. It must be in range [-π , π). If `h = 0`, then the triangle is symmetric and a direct analog of a tone.
`pDst`	Pointer to the array that stores the samples.
`len`	Number of samples to be computed.

Description

This function generates the triangle with the specified frequency `rFreq`, phase pointed by `pPhase`, and magnitude `magn`. The function computes `len` samples of the triangle, and stores them in the array `pDst`. For real triangle, `x[n]` is defined as:

`x[n] = magn ***ct``** :sub:`h` ``(2π*rFreq*n+phase),n = 0`, 1, 2,…

For complex triangles, `x[n]` is defined as:

`x[n] = magn * [**ct``** :sub:`h` ``(2π*rFreq*n+phase) + j ***st``** :sub:`h` ``(2π*rFreq*n+phase)],n = 0`, 1, 2,…

See Triangle-Generating Functions for the definition of functions **``ct``**$_h$ and **``st``**$_h$.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when the pDst or pPhase pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to zero.
ippStsTrnglMagnErr	Indicates an error when magn is less than or equal to zero.
ippStsTrnglFreqErr	Indicates an error when rFreq is negative, or greater than or equal to 0.5.
ippStsTrnglPhaseErr	Indicates an error when the pPhase value is negative, or greater than or equal to IPP_2PI.
ippStsTrnglAsymErr	Indicates an error when asym is less than - IPP_PI, or greater than or equal to IPP_PI.

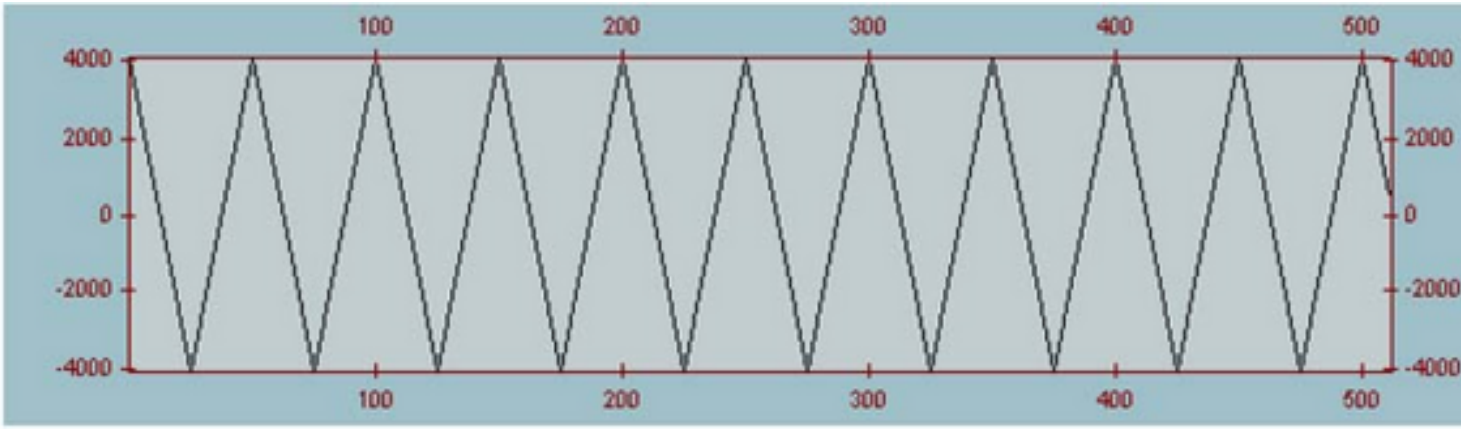
Example

The code example below demonstrates how to use the ippsTriangle function.

```
void func_triangle_direct()
{
    Ipp16s* pDst;
    int len = 512;
    Ipp16s magn = 4095;
    Ipp32f rFreq = 0.02;
    Ipp32f asym = 0.0;
    Ipp32f Phase = 0.0;
    IppStatus status;


    status = ippsTriangle_16s(pDst, len, magn, rFreq, asym, &Phase);
    if(ippStsNoErr != status)
      printf("Intel® IPP Error: %s",ippGetStatusString(status));
}
```

Result:



Uniform Distribution Functions

This section describes the functions that generate pseudo-random samples with uniform distribution.

- RandUniformInit
- RandUniformGetSize
- RandUniform

RandUniformInit

Initializes a noise generator with uniform distribution.

Syntax

```
IppStatus ippsRandUniformInit_8u(IppsRandUniState_8u* pRandUniState, Ipp8u low, Ipp8u high, unsigned int seed);
IppStatus ippsRandUniformInit_16s(IppsRandUniState_16s* pRandUniState, Ipp16s low, Ipp16s high, unsigned int seed);
IppStatus ippsRandUniformInit_32f(IppsRandUniState_32f* pRandUniState, Ipp32f low, Ipp32f high, unsigned int seed);
IppStatus ippsRandUniformInit_64f(IppsRandUniState_64f* pRandUniState, Ipp64f low, Ipp64f high, unsigned int seed);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pRandUniState`	Pointer to the structure containing parameters for the generator of noise.

low	Lower bound of the uniform distribution range.
high	Upper bound of the uniform distribution range.
seed	Seed value used by the pseudo-random number generation algorithm.

Description

This function initializes the pseudo-random generator state structure pRandUniState in the external buffer. The uniform distribution range is specified by the lower and upper bounds low and high, respectively. Before using this function, you need to compute the size of the external buffer by using the ippsRandUniformGetSize function.

The example of using this function is similar to the example provided with the Histogram function description.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when the pRandUniState pointer is NULL.
ippStsMemAllocErr	Indicates an error when there is not enough memory for the operation.

See Also

RandUniformGetSize Computes the length of the uniform distribution generator structure.

RandUniformGetSize

Computes the length of the uniform distribution generator structure.

Syntax

```
IppStatus ippsRandUniformGetSize_8u(int* pRandUniformStateSize);
IppStatus ippsRandUniformGetSize_16s(int* pRandUniformStateSize);
IppStatus ippsRandUniformGetSize_32f(int* pRandUniformStateSize);
IppStatus ippsRandUniformGetSize_64f(int* pRandUniformStateSize);
```

Include Files

ipps.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

pRandUniformStateSize	Pointer to the computed value of size in bytes of the generator specification structure.

Description

This function computes the length (in bytes) pRandUniformStateSize of the uniform distribution generator structure that is used by the ippsRandUniformInit function.

The example of using this function is similar to the example provided with the Histogram function description.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when the pointer pRandUniformStateSize is NULL .

See Also

RandUniformInit Initializes a noise generator with uniform distribution.

RandUniform

Generates the pseudo-random samples with a uniform distribution.

Syntax

```
IppStatus ippsRandUniform_8u(Ipp8u* pDst, int len, IppsRandUniState_8u* pRandUniState);
IppStatus ippsRandUniform_16s(Ipp16s* pDst, int len, IppsRandUniState_16s*
pRandUniState);
IppStatus ippsRandUniform_32f(Ipp32f* pDst, int len, IppsRandUniState_32f*
pRandUniState);
IppStatus ippsRandUniform_64f(Ipp64f* pDst, int len, IppsRandUniState_64f*
pRandUniState);
```

Include Files

ipps.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

pDst	Pointer to the array which stores the samples.
len	Number of samples to be computed.

pRandUniState	Pointer to the structure containing parameters for the generator of noise.

Description

This function generates len pseudo-random samples with a uniform distribution and stores them in the array pDst. Initial parameters of the generator are set in the generator state structure pRandUniState. Before calling ippsRandUniform, you must initialize the generator state by calling the function RandUniformInit.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when the pDst or pRandUniState pointer is NULL.
ippStsContextMatchErr	Indicates an error when the state identifier is incorrect.

Gaussian Distribution Functions

This section describes the function that generates pseudo-random samples with Gaussian distribution.

- RandGaussInit
- RandGaussGetSize
- RandGauss

RandGaussInit

Initializes a noise generator with Gaussian distribution.

Syntax

```
IppStatus ippsRandGaussInit_8u(IppsRandGaussState_8u* pRandGaussState, Ipp8u mean,
Ipp8u stdDev, unsigned int seed);
```

```
IppStatus ippsRandGaussInit_16s(IppsRandGaussState_16s* pRandGaussState, Ipp16s mean,
Ipp16s stdDev, unsigned int seed);
```

```
IppStatus ippsRandGaussInit_32f(IppsRandGaussState_32f* pRandGaussState, Ipp32f mean,
Ipp32f stdDev, unsigned int seed);
```

```
IppStatus ippsRandGaussInit_64f(IppsRandGaussState_64f* pRandGaussState, Ipp64f mean,
Ipp64f stdDev, unsigned int seed);
```

Include Files

ipps.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

pRandGaussState	Pointer to the structure containing parameters for the generator of noise.
mean	Mean of the Gaussian distribution.
stdDev	Standard deviation of the Gaussian distribution.
seed	Seed value used by the pseudo-random number generator algorithm.

Description

This function initializes the pseudo-random generator state structure pRandGaussState in the external buffer. This structure contains parameters of the required noise generator that are specified by the mean, stdDev, and seed values. Before using this function, you need to compute the size of the buffer by calling the ippsRandGaussGetSize function.

Return Values

ppStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when the pRandGaussState pointer is NULL.
ippStsMemAllocErr	Indicates an error when there is not enough memory for the operation.

See Also

RandGaussGetSize Computes the length of the Gaussian distribution generator structure.

RandGaussGetSize

Computes the length of the Gaussian distribution generator structure.

Syntax

```
IppStatus ippsRandGaussGetSize_8u(int* pRandGaussStateSize);
IppStatus ippsRandGaussGetSize_32f(int* pRandGaussStateSize);
IppStatus ippsRandGaussGetSize_16s(int* pRandGaussStateSize);
IppStatus ippsRandGaussGetSize_64f(int* pRandGaussStateSize);
```

Include Files

ipps.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

pRandGaussStateSize	Pointer to the size, in bytes, of the generator specification structure.

Description

This function computes the length (in bytes) pRandGaussStateSize of the uniform distribution generator structure that is used by the ippsRandGaussInit function.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when the pointer pRandGaussStateSize is NULL.

See Also

RandGaussInit Initializes a noise generator with Gaussian distribution.

RandGauss

Generates the pseudo-random samples with a Gaussian distribution.

Syntax

```
IppStatus ippsRandGauss_8u(Ipp8u* pDst, int len, IppsRandGaussState_8u*
pRandGaussState);
IppStatus ippsRandGauss_16s(Ipp16s* pDst, int len, IppsRandGaussState_16s*
pRandGaussState);
IppStatus ippsRandGauss_32f(Ipp32f* pDst, int len, IppsRandGaussState_32f*
pRandGaussState);
IppStatus ippsRandGauss_64f(Ipp64f* pDst, int len, IppsRandGaussState_64f*
pRandGaussState);
```

Include Files

ipps.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

pDst	Pointer to the array which stores the samples.
len	Number of samples to be computed.
pRandGaussState	Pointer to the structure containing parameters of the noise generator.

Description

This function generates `len` pseudo-random samples with a Gaussian distribution and stores them in the array `pDst`. The initial parameters of the generator are set in the generator state structure `pRandGaussState`. Before calling `ippsRandGauss`, you must initialize the generator state by calling the RandGaussInit function.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when the `pRandGaussState` pointer is `NULL`.
`ippStsContextMatchErr`	Indicates an error when the state identifier is incorrect.

Special Vector Functions

The functions described in this section create special vectors that can be used as a test signals to examine the effect of applying different signal processing functions.

- VectorJaehne
- VectorSlope

VectorJaehne

Creates a Jaehne vector.

Syntax

```
IppStatus ippsVectorJaehne_8u(Ipp8u* pDst, int len, Ipp8u magn);
IppStatus ippsVectorJaehne_16u(Ipp16u* pDst, int len, Ipp16u magn);
IppStatus ippsVectorJaehne_16s(Ipp16s* pDst, int len, Ipp16s magn);
IppStatus ippsVectorJaehne_32s(Ipp32s* pDst, int len, Ipp32s magn);
IppStatus ippsVectorJaehne_32f(Ipp32f* pDst, int len, Ipp32f magn);
IppStatus ippsVectorJaehne_64f(Ipp64f* pDst, int len, Ipp64f magn);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pDst`	Pointer to the destination vector.
`len`	Number of elements in the vector.

magn	Magnitude of the signal to be generated.

Description

This function creates a Jaehne vector and stores the result in pDst. The magnitude magn must be positive. The function generates the sinusoid with a variable frequency. The computation is performed as follows:

$pDst[n] = magn * \sin((0.5\pi n^2)/len), 0 \le n < len$

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when the pSrcDst pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.
ippStsJaehneErr	Indicates an error when magn is negative.

Example

The code example below shows how to use the function ippsVectorJaehne.

```
IppStatus Jaehne (void)
{
    Ipp16s buf[100] ;
    return ippsVectorJaehne_16s ( buf, 100, 255 );
}
```

VectorSlope

Creates a slope vector.

Syntax

```
IppStatus ippsVectorSlope_8u(Ipp8u* pDst, int len, Ipp32f offset, Ipp32f slope);
IppStatus ippsVectorSlope_16u(Ipp16u* pDst, int len, Ipp32f offset, Ipp32f slope);
IppStatus ippsVectorSlope_16s(Ipp16s* pDst, int len, Ipp32f offset, Ipp32f slope);
IppStatus ippsVectorSlope_32u(Ipp32u* pDst, int len, Ipp64f offset, Ipp64f slope);
IppStatus ippsVectorSlope_32s(Ipp32s* pDst, int len, Ipp64f offset, Ipp64f slope);
IppStatus ippsVectorSlope_32f(Ipp32f* pDst, int len, Ipp32f offset, Ipp32f slope);
IppStatus ippsVectorSlope_64f(Ipp64f* pDst, int len, Ipp64f offset, Ipp64f slope);
```

Include Files

ipps.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

pDst	Pointer to the destination vector.
len	Number of elements in the vector.
offset	Offset value.
slope	Slope coefficient.

Description

This function creates a slope vector and stores the result in pDst. The destination vector elements are computed according to the following formula:

pDst[n] =offset+slope*n,0≤n <len.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when the pDst pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.

Essential Functions

This chapter describes the Intel ® IPP functions that perform logical and shift, arithmetic, conversion, windowing, and statistical operations.

- Logical and Shift Functions
- Arithmetic Functions
- Conversion Functions
- Windowing Functions
- Statistical Functions
- Sampling Functions
- AI Inference Functions

Logical and Shift Functions

This section describes the Intel IPP signal processing functions that perform logical and shift operations on vectors. Logical and shift functions are only defined for integer arguments.

For binary logical operations AND, OR and XOR, the following functions are provided:

AndC, OrC, XorC for vector-scalar operations;

And, Or, Xor for vector-vector operations.

- AndC
- And
- OrC
- Or
- XorC
- Xor
- Not
- LShiftC

- RShiftC

AndC

Computes the bitwise AND of a scalar value and each element of a vector.

Syntax

```
IppStatus ippsAndC_8u(const Ipp8u* pSrc, Ipp8u val, Ipp8u* pDst, int len);
IppStatus ippsAndC_16u(const Ipp16u* pSrc, Ipp16u val, Ipp16u* pDst, int len);
IppStatus ippsAndC_32u(const Ipp32u* pSrc, Ipp32u val, Ipp32u* pDst, int len);
IppStatus ippsAndC_8u_I(Ipp8u val, Ipp8u* pSrcDst, int len);
IppStatus ippsAndC_16u_I(Ipp16u val, Ipp16u* pSrcDst, int len);
IppStatus ippsAndC_32u_I(Ipp32u val, Ipp32u* pSrcDst, int len);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`val`	Input scalar value.
`pSrc`	Pointer to the source vector.
`pDst`	Pointer to the destination vector.
`pSrcDst`	Pointer to the source and destination vector for the in-place operation.
`len`	Number of elements in the vector.

Description

This function computes the bitwise AND of a scalar value `val` and each element of the vector `pSrc`, and stores the result in `pDst`.

The in-place flavors of `ippsAndC` compute the bitwise AND of a scalar value `val` and each element of the vector `pSrcDst` and store the result in `pSrcDst`.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when the `pSrc`, `pDst`, or `pSrcDst` pointer is `NULL`.

ippStsSizeErr	Indicates an error when len is less than or equal to 0.

And

Computes the bitwise AND of two vectors.

Syntax

```
IppStatus ippsAnd_8u(const Ipp8u* pSrc1, const Ipp8u* pSrc2, Ipp8u* pDst, int len);
IppStatus ippsAnd_16u(const Ipp16u* pSrc1, const Ipp16u* pSrc2, Ipp16u* pDst, int len);
IppStatus ippsAnd_32u(const Ipp32u* pSrc1, const Ipp32u* pSrc2, Ipp32u* pDst, int len);
IppStatus ippsAnd_8u_I(const Ipp8u* pSrc, Ipp8u* pSrcDst, int len);
IppStatus ippsAnd_16u_I(const Ipp16u* pSrc, Ipp16u* pSrcDst, int len);
IppStatus ippsAnd_32u_I(const Ipp32u* pSrc, Ipp32u* pSrcDst, int len);
```

Include Files

ipps.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

pSrc1, pSrc2	Pointers to the two source vectors.
pDst	Pointer to the destination vector.
pSrc	Pointer to the source vector for the in-place operation.
pSrcDst	Pointer to the source and destination vector for the in-place operation.
len	Number of elements in the vector.

Description

This function computes the bitwise AND of the corresponding elements of the vectors pSrc1 and pSrc2, and stores the result in the vector pDst.

The in-place flavors of ippsAnd compute the bitwise AND of the corresponding elements of the vectors pSrc and pSrcDst and store the result in the vector pSrcDst.

Return Values

ippStsNoErr	Indicates no error.

ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.

OrC

Computes the bitwise OR of a scalar value and each element of a vector.

Syntax

```
IppStatus ippsOrC_8u(const Ipp8u* pSrc, Ipp8u val, Ipp8u* pDst, int len);
IppStatus ippsOrC_16u(const Ipp16u* pSrc, Ipp16u val, Ipp16u* pDst, int len);
IppStatus ippsOrC_32u(const Ipp32u* pSrc, Ipp32u val, Ipp32u* pDst, int len);
IppStatus ippsOrC_8u_I(Ipp8u val, Ipp8u* pSrcDst, int len);
IppStatus ippsOrC_16u_I(Ipp16u val, Ipp16u* pSrcDst, int len);
IppStatus ippsOrC_32u_I(Ipp32u val, Ipp32u* pSrcDst, int len);
```

Include Files

ipps.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

val	Input scalar value.
pSrc	Pointer to the source vector.
pDst	Pointer to the destination vector.
pSrcDst	Pointer to the source and destination vector for the in-place operation.
len	Number of elements in the vector.

Description

This function computes the bitwise OR of a scalar value val and each element of the vector pSrc, and stores the result in pDst.

The in-place flavors of ippsOrC compute the bitwise OR of a scalar value val and each element of the vector pSrcDst and store the result in pSrcDst.

Return Values

ippStsNoErr	Indicates no error.

ippStsNullPtrErr	Indicates an error when the pSrc, pDst, or pSrcDst pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.

Or

Computes the bitwise OR of two vectors.

Syntax

```
IppStatus ippsOr_8u(const Ipp8u* pSrc1, const Ipp8u* pSrc2, Ipp8u* pDst, int len);
IppStatus ippsOr_16u(const Ipp16u* pSrc1, const Ipp16u* pSrc2, Ipp16u* pDst, int len);
IppStatus ippsOr_32u(const Ipp32u* pSrc1, const Ipp32u* pSrc2, Ipp32u* pDst, int len);
IppStatus ippsOr_8u_I(const Ipp8u* pSrc, Ipp8u* pSrcDst, int len);
IppStatus ippsOr_16u_I(const Ipp16u* pSrc, Ipp16u* pSrcDst, int len);
IppStatus ippsOr_32u_I(const Ipp32u* pSrc, Ipp32u* pSrcDst, int len);
```

Include Files

ipps.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

pSrc1, pSrc2	Pointers to the two source vectors.
pDst	Pointer to the destination vector.
pSrc	Pointer to the source vector for the in-place operation.
pSrcDst	Pointer to the source and destination vector for the in-place operation.
len	Number of elements in the vector.

Description

This function computes the bitwise OR of the corresponding elements of the vectors pSrc1 and pSrc2, and stores the result in the vector pDst.

The in-place flavors of ippsOr compute the bitwise OR of the corresponding elements of the vectors pSrc and pSrcDst and store the result in the vector pSrcDst.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.

XorC

Computes the bitwise XOR of a scalar value and each element of a vector.

Syntax

```
IppStatus ippsXorC_8u(const Ipp8u* pSrc, Ipp8u val, Ipp8u* pDst, int len);
IppStatus ippsXorC_16u(const Ipp16u* pSrc, Ipp16u val, Ipp16u* pDst, int len);
IppStatus ippsXorC_32u(const Ipp32u* pSrc, Ipp32u val, Ipp32u* pDst, int len);
IppStatus ippsXorC_8u_I(Ipp8u val, Ipp8u* pSrcDst, int len);
IppStatus ippsXorC_16u_I(Ipp16u val, Ipp16u* pSrcDst, int len);
IppStatus ippsXorC_32u_I(Ipp32u val, Ipp32u* pSrcDst, int len);
```

Include Files

ipps.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

val	Input scalar value.
pSrc	Pointer to the source vector.
pDst	Pointer to the destination vector.
pSrcDst	Pointer to the source and destination vector for the in-place operation.
len	Number of elements in the vector.

Description

This function computes the bitwise XOR of a scalar value val and each element of the vector pSrc, and stores the result in pDst.

The in-place flavors of ippsXorC compute the bitwise XOR of a scalar value val and each element of the vector pSrcDst and store the result in pSrcDst.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when the pSrc, pDst, or pSrcDst pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.

Xor

Computes the bitwise XOR of two vectors.

Syntax

```
IppStatus ippsXor_8u(const Ipp8u* pSrc1, const Ipp8u* pSrc2, Ipp8u* pDst, int len);
IppStatus ippsXor_16u(const Ipp16u* pSrc1, const Ipp16u* pSrc2, Ipp16u* pDst, int len);
IppStatus ippsXor_32u(const Ipp32u* pSrc1, const Ipp32u* pSrc2, Ipp32u* pDst, int len);
IppStatus ippsXor_8u_I(const Ipp8u* pSrc, Ipp8u* pSrcDst, int len);
IppStatus ippsXor_16u_I(const Ipp16u* pSrc, Ipp16u* pSrcDst, int len);
IppStatus ippsXor_32u_I(const Ipp32u* pSrc, Ipp32u* pSrcDst, int len);
```

Include Files

ipps.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

pSrc1, pSrc2	Pointers to the two source vectors.
pDst	Pointer to the destination vector.
pSrc	Pointer to the source vector for the in-place operation.
pSrcDst	Pointer to the source and destination vector for the in-place operation.
len	Number of elements in the vector.

Description

This function computes the bitwise XOR of the corresponding elements of the vectors pSrc1 and pSrc2, and stores the result in the vector pDst.

The in-place flavors of ippsXor compute the bitwise XOR of the corresponding elements of the vectors pSrc and pSrcDst and store the result in the vector pSrcDst.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.

Not

Computes the bitwise NOT of the vector elements.

Syntax

```
IppStatus ippsNot_8u(const Ipp8u* pSrc, Ipp8u* pDst, int len);
IppStatus ippsNot_16u(const Ipp16u* pSrc, Ipp16u* pDst, int len);
IppStatus ippsNot_32u(const Ipp32u* pSrc, Ipp32u* pDst, int len);
IppStatus ippsNot_8u_I(Ipp8u* pSrcDst, int len);
IppStatus ippsNot_16u_I(Ipp16u* pSrcDst, int len);
IppStatus ippsNot_32u_I(Ipp32u* pSrcDst, int len);
```

Include Files

ipps.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

pSrc	Pointer to the source vector.
pDst	Pointer to the destination vector.
pSrcDst	Pointer to the source and destination vector for the in-place operation.
len	Number of elements in the vector.

Description

This function computes the bitwise NOT of the corresponding elements of the vectors pSrc, and stores the result in the vector pDst.

The in-place flavors of ippsNot compute the bitwise NOT of the corresponding elements of the vector pSrcDst and store the result in the vector pSrcDst.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when the pSrc, pDst, or pSrcDst pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.

LShiftC

Shifts bits in vector elements to the left.

Syntax

```
IppStatus ippsLShiftC_8u(const Ipp8u* pSrc, int val, Ipp8u* pDst, int len);
IppStatus ippsLShiftC_16s(const Ipp16s* pSrc, int val, Ipp16s* pDst, int len);
IppStatus ippsLShiftC_16u(const Ipp16u* pSrc, int val, Ipp16u* pDst, int len);
IppStatus ippsLShiftC_32s(const Ipp32s* pSrc, int val, Ipp32s* pDst, int len);
IppStatus ippsLShiftC_8u_I(int val, Ipp8u* pSrcDst, int len);
IppStatus ippsLShiftC_16u_I(int val, Ipp16u* pSrcDst, int len);
IppStatus ippsLShiftC_16s_I(int val, Ipp16s* pSrcDst, int len);
IppStatus ippsLShiftC_32s_I(int val, Ipp32s* pSrcDst, int len);
```

Include Files

ipps.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

val	Number of bits by which the function shifts each element of the vector pSrc or pSrcDst.
pSrc	Pointer to the source vector.
pDst	Pointer to the destination vector.
pSrcDst	Pointer to the source and destination vector for the in-place operation.
len	Number of elements in the vector.

Description

This function shifts each element of the vector pSrc by val bits to the left, and stores the result in pDst.

The in-place flavors of ippsLShiftC shift each element of the vector pSrcDst by val bits to the left and store the result in pSrcDst.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when the pSrc, pDst, or pSrcDst pointer is NULL .
ippStsSizeErr	Indicates an error when len is less than or equal to 0.

RShiftC

Shifts bits in vector elements to the right.

Syntax

```
IppStatus ippsRShiftC_8u(const Ipp8u* pSrc, int val, Ipp8u* pDst, int len);
IppStatus ippsRShiftC_16s(const Ipp16s* pSrc, int val, Ipp16s* pDst, int len);
IppStatus ippsRShiftC_16u(const Ipp16u* pSrc, int val, Ipp16u* pDst, int len);
IppStatus ippsRShiftC_32s(const Ipp32s* pSrc, int val, Ipp32s* pDst, int len);
IppStatus ippsRShiftC_8u_I(int val, Ipp8u* pSrcDst, int len);
IppStatus ippsRShiftC_16u_I(int val, Ipp16u* pSrcDst, int len);
IppStatus ippsRShiftC_16s_I(int val, Ipp16s* pSrcDst, int len);
IppStatus ippsRShiftC_32s_I(int val, Ipp32s* pSrcDst, int len);
```

Include Files

ipps.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

val	Number of bits by which the function shifts each element of the vector pSrc or pSrcDst.
pSrc	Pointer to the source vector.
pDst	Pointer to the destination vector.
pSrcDst	Pointer to the source and destination vector for the in-place operation.
len	Number of elements in the vector.

Description

This function shifts each element of the vector `pSrc` by `val` bits to the right, and stores the result in `pDst`.

The in-place flavors of `ippsRShiftC` shift each element of the vector `pSrcDst` by `val` bits to the right and store the result in `pSrcDst`.

Note that the arithmetic shift is realized for signed data, and the logical shift for unsigned data.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when the `pSrc`, `pDst`, or `pSrcDst` pointer is `NULL`.
`ippStsSizeErr`	Indicates an error when `len` is less than or equal to zero.

Example

The code example below shows how the logical and shift functions can be used in the saturate operation. The data are converted to the `unsigned char` range [0…255].

```
void saturate(void) {
      Ipp16s x[8] = {1000, -257, 127, 4, 5, 0, 7, 8}, lo[8], hi[8];
      IppStatus status = ippsNot_16u((Ipp16u*)x, (Ipp16u*)lo, 8);
      ippsRShiftC_16s_I(15, lo, 8);
      ippsCopy_16s(x, hi, 8);
      ippsSubCRev_16s_ISfs(255, hi, 8, 0);
      ippsRShiftC_16s_I(15, hi, 8);
      ippsAnd_16u_I((Ipp16u*)lo, (Ipp16u*)x, 8);
      ippsOr_16u_I((Ipp16u*)hi, (Ipp16u*)x, 8);
      ippsAndC_16u_I(255, (Ipp16u*)x, 8);
      printf_16s("saturate =", x, 8, status);
}
```

Output:

```
saturate =  255 0 127 4 5 0 7 8
```

Arithmetic Functions

This section describes the Intel IPP signal processing functions that perform vector arithmetic operations on vectors. The arithmetic functions include basic element-wise arithmetic operations between vectors, as well as more complex calculations such as computing absolute values, square and square root, natural logarithm and exponential of vector elements.

Intel IPP software provides two versions of each function. One version performs the operation in-place, while the other stores the results of the operation in a different destination vector, that is, executes an out-of-place operation.

- AddC
- Add
- AddProductC
- AddProduct
- MulC
- Mul
- SubC
- SubCRev

- Sub
- DivC
- DivCRev
- Div
- Div_Round
- Abs
- Sqr
- Sqrt
- Cubrt
- Exp
- Ln
- SumLn
- Arctan
- Normalize

AddC

Adds a constant value to each element of a vector.

Syntax

Case 1: Not-in-place operations on floating point data.

```
IppStatus ippsAddC_32f(const Ipp32f* pSrc, Ipp32f val, Ipp32f* pDst, int len);
IppStatus ippsAddC_64f(const Ipp64f* pSrc, Ipp64f val, Ipp64f* pDst, int len);
IppStatus ippsAddC_32fc(const Ipp32fc* pSrc, Ipp32fc val,Ipp32fc* pDst, int len);
IppStatus ippsAddC_64fc(const Ipp64fc* pSrc, Ipp64fc val, Ipp64fc* pDst, int len);
```

Case 2: Not-in-place operations on integer data.

```
IppStatus ippsAddC_8u_Sfs(const Ipp8u* pSrc, Ipp8u val, Ipp8u* pDst, int len, int
scaleFactor);
IppStatus ippsAddC_16s_Sfs(const Ipp16s* pSrc, Ipp16s val, Ipp16s* pDst, int len, int
scaleFactor);
IppStatus ippsAddC_16u_Sfs(const Ipp16u* pSrc, Ipp16u val, Ipp16u* pDst, int len, int
scaleFactor);
IppStatus ippsAddC_32s_Sfs(const Ipp32s* pSrc, Ipp32s val, Ipp32s* pDst, int len, int
scaleFactor);
IppStatus ippsAddC_16sc_Sfs(const Ipp16sc* pSrc, Ipp16sc val, Ipp16sc* pDst, int len,
int scaleFactor);
IppStatus ippsAddC_32sc_Sfs(const Ipp32sc* pSrc, Ipp32sc val, Ipp32sc* pDst, int len,
int scaleFactor);
IppStatus ippsAddC_64u_Sfs(const Ipp64u* pSrc, Ipp64u val, Ipp64u* pDst, Ipp32u len,
int scaleFactor, IppRoundMode rndMode);
IppStatus ippsAddC_64s_Sfs(const Ipp64s* pSrc, Ipp64s val, Ipp64s* pDst, Ipp32u len,
int scaleFactor, IppRoundMode rndMode);
```

Case 3: In-place operations on floating point data.

```
IppStatus ippsAddC_16s_I(Ipp16s val, Ipp16s* pSrcDst, int len);
IppStatus ippsAddC_32f_I(Ipp32f val, Ipp32f* pSrcDst, int len);
IppStatus ippsAddC_64f_I(Ipp64f val, Ipp64f* pSrcDst, int len);
```

```
IppStatus ippsAddC_32fc_I(Ipp32fc val, Ipp32fc* pSrcDst, int len);
IppStatus ippsAddC_64fc_I(Ipp64fc val, Ipp64fc* pSrcDst, int len);
```

Case 4: In-place operations on integer data.

```
IppStatus ippsAddC_8u_ISfs(Ipp8u val, Ipp8u* pSrcDst, int len, int scaleFactor);
IppStatus ippsAddC_16u_ISfs(Ipp16u val, Ipp16u* pSrcDst, int len, int scaleFactor);
IppStatus ippsAddC_16s_ISfs(Ipp16s val, Ipp16s* pSrcDst, int len, int scaleFactor);
IppStatus ippsAddC_32s_ISfs(Ipp32s val, Ipp32s* pSrcDst, int len, int scaleFactor);
IppStatus ippsAddC_16sc_ISfs(Ipp16sc val, Ipp16sc* pSrcDst, int len, int scaleFactor);
IppStatus ippsAddC_32sc_ISfs(Ipp32sc val, Ipp32sc* pSrcDst, int len, int scaleFactor);
```

Case 5: Not-in-place operations on floating point data.

```
IppStatus ippsAddC_32f_L(const Ipp32f* pSrc, Ipp32f val, Ipp32f* pDst, IppSizeL len);
IppStatus ippsAddC_64f_L(const Ipp64f* pSrc, Ipp64f val, Ipp64f* pDst, IppSizeL len);
IppStatus ippsAddC_32fc_L(const Ipp32fc* pSrc, Ipp32fc val, Ipp32fc* pDst, IppSizeL
len);
IppStatus ippsAddC_64fc_L(const Ipp64fc* pSrc, Ipp64fc val, Ipp64fc* pDst, IppSizeL
len);
```

Case 6: In-place operations on floating point data.

```
IppStatus ippsAddC_32f_I_L(Ipp32f val, Ipp32f* pSrcDst, IppSizeL len);
IppStatus ippsAddC_64f_I_L(Ipp64f val, Ipp64f* pSrcDst, IppSizeL len);
IppStatus ippsAddC_32fc_I_L(Ipp32fc val, Ipp32fc* pSrcDst, IppSizeL len);
IppStatus ippsAddC_64fc_I_L(Ipp64fc val, Ipp64fc* pSrcDst, IppSizeL len);
```

Include Files

`ipps.h`

Flavors with the `_L` suffix: `ipps_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pSrc`	Pointer to the source vector.
`val`	Scalar value used to increment each element of the source vector.
`pDst`	Pointer to the destination vector.
`pSrcDst`	Pointer to the source and destination vector for the in-place operation.
`len`	Number of elements in the vector.

scaleFactor	Scale factor, refer to Integer Scaling.
rndMode	Rounding mode, the following values are possible: • ippRndZero floating-point values are truncated to zero • ippRndNear floating-point values are rounded to the nearest even integer when the fractional part equals 0.5; otherwise they are rounded to the nearest integer • ippRndFinancial floating-point values are rounded down to the nearest integer when the fractional part is less than 0.5, or rounded up to the nearest integer if the fractional part is equal or greater than 0.5.

Description

This function adds a value val to each element of the source vector pSrc, and stores the result in the destination vector pDst.

The in-place flavors of **``ippsAddC``** add a value val to each element of the vector pSrcDst, and store the result in pSrcDst.

Functions with Sfs suffixe perform scaling of the result value in accordance with the scaleFactor value. If the output value exceeds the data range, the result is saturated.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when the pSrc, pDst, or pSrcDst pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to zero.

See Also

Integer Scaling

Add

Adds the elements of two vectors.

Syntax

Case 1. Not-in-place operations on floating point data, and integer data without scaling.

```
IppStatus ippsAdd_16s(const Ipp16s* pSrc1, const Ipp16s* pSrc2, Ipp16s* pDst, int len);
IppStatus ippsAdd_32f(const Ipp32f* pSrc1, const Ipp32f* pSrc2, Ipp32f* pDst, int len);
IppStatus ippsAdd_64f(const Ipp64f* pSrc1, const Ipp64f* pSrc2, Ipp64f* pDst, int len);
IppStatus ippsAdd_32fc(const Ipp32fc* pSrc1, const Ipp32fc* pSrc2, Ipp32fc* pDst, int
len);
```

```
IppStatus ippsAdd_64fc(const Ipp64fc* pSrc1, const Ipp64fc* pSrc2, Ipp64fc* pDst, int
len);

IppStatus ippsAdd_8u16u(const Ipp8u* pSrc1, const Ipp8u* pSrc2, Ipp16u* pDst, int len);

IppStatus ippsAdd_16u(const Ipp16u* pSrc1, const Ipp16u* pSrc2, Ipp16u* pDst, int len);

IppStatus ippsAdd_32u(const Ipp32u* pSrc1, const Ipp32u* pSrc2, Ipp32u* pDst, int len);

IppStatus ippsAdd_16s32f(const Ipp16s* pSrc1, const Ipp16s* pSrc2, Ipp32f* pDst, int
len);
```

Case 2. Not-in-place operations on integer data with scaling.

```
IppStatus ippsAdd_8u_Sfs(const Ipp8u* pSrc1, const Ipp8u* pSrc2, Ipp8u* pDst, int len,
int scaleFactor);

IppStatus ippsAdd_16u_Sfs(const Ipp16u* pSrc1, const Ipp16u* pSrc2, Ipp16u* pDst, int
len, int scaleFactor);

IppStatus ippsAdd_16s_Sfs(const Ipp16s* pSrc1, const Ipp16s* pSrc2, Ipp16s* pDst, int
len, int scaleFactor);

IppStatus ippsAdd_32s_Sfs(const Ipp32s* pSrc1, const Ipp32s* pSrc2, Ipp32s* pDst, int
len, int scaleFactor);

IppStatus ippsAdd_16sc_Sfs(const Ipp16sc* pSrc1, const Ipp16sc* pSrc2, Ipp16sc* pDst,
int len, int scaleFactor);

IppStatus ippsAdd_32sc_Sfs(const Ipp32sc* pSrc1, const Ipp32sc* pSrc2, Ipp32sc* pDst,
int len, int scaleFactor);

IppStatus ippsAdd_64s_Sfs(const Ipp64s* pSrc1, const Ipp64s* pSrc2, Ipp64s* pDst, int
len, int scaleFactor);
```

Case 3. In-place operations on floating point data, and integer data without scaling.

```
IppStatus ippsAdd_16s_I(const Ipp16s* pSrc, Ipp16s* pSrcDst, int len);

IppStatus ippsAdd_32f_I(const Ipp32f* pSrc, Ipp32f* pSrcDst, int len);

IppStatus ippsAdd_64f_I(const Ipp64f* pSrc, Ipp64f* pSrcDst, int len);

IppStatus ippsAdd_32fc_I(const Ipp32fc* pSrc, Ipp32fc* pSrcDst, int len);

IppStatus ippsAdd_64fc_I(const Ipp64fc* pSrc, Ipp64fc* pSrcDst, int len);

IppStatus ippsAdd_16s32s_I(const Ipp16s* pSrc, Ipp32s* pSrcDst, int len);

IppStatus ippsAdd_32u_I(const Ipp32u* pSrc, Ipp32u* pSrcDst, int len);
```

Case 4. In-place operations on integer data with scaling.

```
IppStatus ippsAdd_8u_ISfs(const Ipp8u* pSrc, Ipp8u* pSrcDst, int len, int scaleFactor);

IppStatus ippsAdd_16u_ISfs(const Ipp16u* pSrc, Ipp16u* pSrcDst, int len, int
scaleFactor);

IppStatus ippsAdd_16s_ISfs(const Ipp16s* pSrc, Ipp16s* pSrcDst, int len, int
scaleFactor);

IppStatus ippsAdd_32s_ISfs(const Ipp32s* pSrc, Ipp32s* pSrcDst, int len, int
scaleFactor);

IppStatus ippsAdd_16sc_ISfs(const Ipp16sc* pSrc, Ipp16sc* pSrcDst, int len, int
scaleFactor);
```

```
IppStatus ippsAdd_32sc_ISfs(const Ipp32sc* pSrc, Ipp32sc* pSrcDst, int len, int
scaleFactor);
```

Case 5. Not-in-place operations on floating point data, and integer data without scaling with platform-aware functions.

```
IppStatus ippsAdd_32f_L(const Ipp32f* pSrc1, const Ipp32f* pSrc2, Ipp32f* pDst,
IppSizeL len);

IppStatus ippsAdd_64f_L(const Ipp64f* pSrc1, const Ipp64f* pSrc2, Ipp64f* pDst,
IppSizeL len);

IppStatus ippsAdd_32fc_L(const Ipp32fc* pSrc1, const Ipp32fc* pSrc2, Ipp32fc* pDst,
IppSizeL len);

IppStatus ippsAdd_64fc_L(const Ipp64fc* pSrc1, const Ipp64fc* pSrc2, Ipp64fc* pDst,
IppSizeL len);
```

Case 6. In-place operations on floating point data, and integer data without scaling with platform-aware functions.

```
IppStatus ippsAdd_32f_I_L(const Ipp32f* pSrc, Ipp32f* pSrcDst, IppSizeL len);

IppStatus ippsAdd_64f_I_L(const Ipp64f* pSrc, Ipp64f* pSrcDst, IppSizeL len);

IppStatus ippsAdd_32fc_I_L(const Ipp32fc* pSrc, Ipp32fc* pSrcDst, IppSizeL len);

IppStatus ippsAdd_64fc_I_L(const Ipp64fc* pSrc, Ipp64fc* pSrcDst, IppSizeL len);
```

Include Files

`ipps.h`

Flavors with the `_L` suffix: `ipps_l.h`

Domain Dependencies

Flavors declared in `ipps.h`:

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Flavors declared in `ipps64x.h`:

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippcore_tl.lib`, `ipps_tl.lib`

Parameters

`pSrc1`, `pSrc2`	Pointers to the source vectors.
`pDst`	Pointer to the destination vector.
`pSrc`	Pointer to the source vector for in-place operations.
`pSrcDst`	Pointer to the source and destination vector for in-place operation.
`len`	Number of elements in the vector
`scaleFactor`	Scale factor, refer to Integer Scaling.

Description

This function adds the elements of the vector `pSrc1` to the elements of the vector `pSrc2`, and stores the result in `pDst`.

The in-place flavors of `ippsAdd` add the elements of the vector `pSrc` to the elements of the vector `pSrcDst` and store the result in `pSrcDst`.

Functions with `Sfs` suffix perform scaling of the result value in accordance with the `scaleFactor` value. If the output value exceeds the data range, the result is saturated.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error when `len` is less than or equal to 0.

Example

`Add`:

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

#include <stdio.h>
#include "ipp.h"

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define
check_sts(st)
                                    \
    if ((st) !=
ippStsNoErr)
                          \
        goto exitLine; /* Go to Exit if Intel(R) Integrated Performance Primitives (Intel(R)
IPP) function returned status different from \
                           ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */
```

```
int main()
{
    int len = 32;
    int i;
    Ipp64f *pDst = ippsMalloc_64f(len * sizeof(Ipp64f));
    Ipp64f *pSrc1 = ippsMalloc_64f(len * sizeof(Ipp64f));
    Ipp64f *pSrc2 = ippsMalloc_64f(len * sizeof(Ipp64f));
    IppStatus status;

    printf("\n\nSource vector 1\n");
    for (i = 0; i < len; i++) {
        pSrc1[i] = i;
        printf("%.0f; ", pSrc1[i]);
    }
    printf("\n\nSource vector 2\n");
    for (i = 0; i < len; i++) {
        pSrc2[i] = 10 + i;
        printf("%.0f; ", pSrc2[i]);
    }

    check_sts(status = ippsAdd_64f(pSrc2, pSrc1, pDst, len));

    printf("\n\nResult\n");
    for (i = 0; i < len; i++)
        printf("%.2f; ", pDst[i]);
    printf("\n\n");

    EXIT_MAIN
    ippsFree(pSrc1);
    ippsFree(pSrc2);
    ippsFree(pDst);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

Add_I:

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

#include <stdio.h>
#include "ipp.h"

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
```

```
    exitLine: /* Label for Exit */
#define
check_sts(st)
                                  \
    if ((st) !=
ippStsNoErr)
                           \
        goto exitLine; /* Go to Exit if Intel(R) Integrated Performance Primitives (Intel(R)
IPP) function returned status different from \
                           ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main()
{
    int len = 32;
    int i;
    Ipp64f *pSrc1 = ippsMalloc_64f(len * sizeof(Ipp64f));
    Ipp64f *pSrcDst = ippsMalloc_64f(len * sizeof(Ipp64f));
    IppStatus status;

    printf("\n\nSource vector 1\n");
    for (i = 0; i < len; i++) {
        pSrcDst[i] = i;
        printf("%.0f; ", pSrcDst[i]);
    }
    printf("\n\nSource vector 2\n");
    for (i = 0; i < len; i++) {
        pSrc1[i] = 10 + i;
        printf("%.0f; ", pSrc1[i]);
    }

    check_sts(status = ippsAdd_64f_I(pSrc1, pSrcDst, len));

    printf("\n\nResult\n");
    for (i = 0; i < len; i++)
        printf("%.2f; ", pSrcDst[i]);
    printf("\n\n");

    EXIT_MAIN
    ippsFree(pSrc1);
    ippsFree(pSrcDst);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

See Also

Integer Scaling

AddProductC

Adds product of a vector and a constant to the accumulator vector.

Syntax

```
IppStatus ippsAddProductC_32f(const Ipp32f* pSrc, const Ipp32f val, Ipp32f* pSrcDst,
int len);
```

```
IppStatus ippsAddProductC_64f(const Ipp64f* pSrc, const Ipp64f val, Ipp64f* pSrcDst,
int len);
```

```
IppStatus ippsAddProductC_32fc(const Ipp32fc* pSrc, const Ipp32fc val, Ipp32fc*
pSrcDst, int len);
```

```
IppStatus ippsAddProductC_64fc(const Ipp64fc* pSrc, const Ipp64fc val, Ipp64fc*
pSrcDst, int len);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pSrc`	Pointer to the source vector.
`val`	The value by which the source vector is multiplied.
`pSrcDst`	Pointer to the source and destination vector for the in-place operation.
`len`	Number of elements in the vector.

Description

This function multiplies each element of the source vector `pSrc` by a value `val` and adds the result to the corresponding element of the accumulator vector `pSrcDst` as given by:

For real data types:

`pSrcDst` [n] = `pSrcDst` [n] + `pSrc` [n]* `val`,

`0≤n <len`

For complex data types:

`pSrcDst.re` [n] = `pSrcDst.re` [n] + `pSrc.re` [n]* `val.re` - `pSrc.im` [n]* `val.im`,

`pSrcDst.im` [n] = `pSrcDst.im` [n] + `pSrc.re` [n]* `val.im` + `pSrc.im` [n]* `val.re`,

`0≤n <len`

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error if `len` is less than or equal to 0.

AddProduct

Adds product of two vectors to the accumulator vector.

Syntax

Case 1. Operations on floating point data.

```
IppStatus ippsAddProduct_32f(const Ipp32f* pSrc1, const Ipp32f* pSrc2, Ipp32f* pSrcDst,
int len);
```

```
IppStatus ippsAddProduct_64f(const Ipp64f* pSrc1, const Ipp64f* pSrc2, Ipp64f* pSrcDst,
int len);
```

```
IppStatus ippsAddProduct_32fc(const Ipp32fc* pSrc1, const Ipp32fc* pSrc2, Ipp32fc*
pSrcDst, int len);
```

```
IppStatus ippsAddProduct_64fc(const Ipp64fc* pSrc1, const Ipp64fc* pSrc2, Ipp64fc*
pSrcDst, int len);
```

Case 2. Operations on integer data with scaling.

```
IppStatus ippsAddProduct_16s_Sfs(const Ipp16s* pSrc1, const Ipp16s* pSrc2, Ipp16s*
pSrcDst, int len, int scaleFactor);
```

```
IppStatus ippsAddProduct_32s_Sfs(const Ipp32s* pSrc1, const Ipp32s* pSrc2, Ipp32s*
pSrcDst, int len, int scaleFactor);
```

```
IppStatus ippsAddProduct_16s32s_Sfs(const Ipp16s* pSrc1, const Ipp16s* pSrc2, Ipp32s*
pSrcDst, int len, int scaleFactor);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pSrc1`, `pSrc2`	Pointers to the source vectors.
`pSrcDst`	Pointer to the destination accumulator vector.
`len`	The number of elements in the vectors.
`scaleFactor`	Scale factor, refer to Integer Scaling.

Description

This function multiplies each element of the source vector `pSrc1` by the corresponding element of the vector `pSrc2`, and adds the result to the corresponding element of the accumulator vector `pSrcDst` as given by:

`pSrcDst[n ] =pSrcDst[n ] +pSrc1[n ] *pSrc2[n ],0≤n <len.`

Functions with `Sfs` suffixes perform scaling of the result value in accordance with the `scaleFactor` value. If the output value exceeds the data range, the result becomes saturated.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.

MulC

Multiplies each element of a vector by a constant value.

Syntax

Case 1. Not-in-place operations without scaling.

```
IppStatus ippsMulC_32f(const Ipp32f* pSrc, Ipp32f val, Ipp32f* pDst, int len);
IppStatus ippsMulC_64f(const Ipp64f* pSrc, Ipp64f val, Ipp64f* pDst, int len);
IppStatus ippsMulC_32fc(const Ipp32fc* pSrc, Ipp32fc val, Ipp32fc* pDst, int len);
IppStatus ippsMulC_64fc(const Ipp64fc* pSrc, Ipp64fc val, Ipp64fc* pDst, int len);
IppStatus ippsMulC_Low_32f16s(const Ipp32f* pSrc, Ipp32f val, Ipp16s* pDst, int len);
```

Case 2. Not-in-place operations with scaling.

```
IppStatus ippsMulC_8u_Sfs(const Ipp8u* pSrc, Ipp8u val, Ipp8u* pDst, int len, int
scaleFactor);
IppStatus ippsMulC_16s_Sfs(const Ipp16s* pSrc, Ipp16s val, Ipp16s* pDst, int len, int
scaleFactor);
IppStatus ippsMulC_16u_Sfs(const Ipp16u* pSrc, Ipp16u val, Ipp16u* pDst, int len, int
scaleFactor);
IppStatus ippsMulC_32s_Sfs(const Ipp32s* pSrc, Ipp32s val, Ipp32s* pDst, int len, int
scaleFactor);
IppStatus ippsMulC_16sc_Sfs(const Ipp16sc* pSrc, Ipp16sc val, Ipp16sc* pDst, int len,
int scaleFactor);
IppStatus ippsMulC_32sc_Sfs(const Ipp32sc* pSrc, Ipp32sc val, Ipp32sc* pDst, int len,
int scaleFactor);
IppStatus ippsMulC_32f16s_Sfs(const Ipp32f* pSrc, Ipp32f val, Ipp16s* pDst, int len,
int scaleFactor);
```

Case 3. In-place operations without scaling.

```
IppStatus ippsMulC_16s_I(Ipp16s val, Ipp16s* pSrcDst, int len);
IppStatus ippsMulC_32f_I(Ipp32f val, Ipp32f* pSrcDst, int len);
IppStatus ippsMulC_64f_I(Ipp64f val, Ipp64f* pSrcDst, int len);
IppStatus ippsMulC_32fc_I(Ipp32fc val, Ipp32fc* pSrcDst, int len);
IppStatus ippsMulC_64fc_I(Ipp64fc val, Ipp64fc* pSrcDst, int len);
```

Case 4. In-place operations with scaling.

```
IppStatus ippsMulC_8u_ISfs(Ipp8u val, Ipp8u* pSrcDst, int len, int scaleFactor);
IppStatus ippsMulC_16u_ISfs(Ipp16u val, Ipp16u* pSrcDst, int len, int scaleFactor);
IppStatus ippsMulC_16s_ISfs(Ipp16s val, Ipp16s* pSrcDst, int len, int scaleFactor);
IppStatus ippsMulC_32s_ISfs(Ipp32s val, Ipp32s* pSrcDst, int len, int scaleFactor);
IppStatus ippsMulC_64f64s_ISfs(Ipp64f val, Ipp64s* pSrcDst, Ipp32u len, int
scaleFactor);
IppStatus ippsMulC_16sc_ISfs(Ipp16sc val, Ipp16sc* pSrcDst, int len, int scaleFactor);
IppStatus ippsMulC_32sc_ISfs(Ipp32sc val, Ipp32sc* pSrcDst, int len, int scaleFactor);
IppStatus ippsMulC_64s_ISfs(Ipp64s val, Ipp64s* pSrcDst, Ipp32u len, int scaleFactor);
```

Case 5. Not In-place operations without scaling with platform-aware functions.

```
IppStatus ippsMulC_32f_L(const Ipp32f* pSrc, Ipp32f val, Ipp32f* pDst, IppSizeL len);
IppStatus ippsMulC_64f_L(const Ipp64f* pSrc, Ipp64f val, Ipp64f* pDst, IppSizeL len);
IppStatus ippsMulC_32fc_L(const Ipp32fc* pSrc, Ipp32fc val, Ipp32fc* pDst, IppSizeL
len);
IppStatus ippsMulC_64fc_L(const Ipp64fc* pSrc, Ipp64fc val, Ipp64fc* pDst, IppSizeL
len);
```

Case 6. Not-in-place operations with scaling with platform-aware functions.

```
IppStatus ippsMulC_32s_Sfs_L(const Ipp32s* pSrc, Ipp32s val, Ipp32s* pDst, IppSizeL
len, int scaleFactor);
```

Case 7. In-place operations without scaling with platform-aware functions.

```
IppStatus ippsMulC_32f_I_L(Ipp32f val, Ipp32f* pSrcDst, IppSizeL len);
IppStatus ippsMulC_64f_I_L(Ipp64f val, Ipp64f* pSrcDst, IppSizeL len);
IppStatus ippsMulC_32fc_I_L(Ipp32fc val, Ipp32fc* pSrcDst, IppSizeL len);
IppStatus ippsMulC_64fc_I_L(Ipp64fc val, Ipp64fc* pSrcDst, IppSizeL len);
```

Case 8. In-place operations with scaling with platform-aware functions.

```
IppStatus ippsMulC_32s_ISfs_L(Ipp32s val, Ipp32s* pSrcDst, IppSizeL len, int
scaleFactor);
```

Include Files

`ipps.h`

Flavors with the `_L` suffix: `ipps_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pSrc`	Pointer to the source vector.

val	The scalar value used to multiply each element of the source vector.
pDst	Pointer to the destination vector.
pSrcDst	Pointer to the source and destination vector for in-place operation.
len	The number of elements in the vector
scaleFactor	Scale factor, refer to Integer Scaling.

Description

This function multiplies each element of the vector pSrc by a value val and stores the result in pDst.

The in-place flavors of ippsMulC multiply each element of the vector pSrcDst by a value val and store the result in pSrcDst.

The function flavor with Low suffix in its name requires that each value of the product pSrc*val does not exceed the Ipp32s data type range.

The function flavors with Sfs suffix perform scaling of the result value in accordance with the scaleFactor value. If the output value exceeds the data range, the result is saturated.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when the pSrc, pDst, or pSrcDst pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than, or equal to 0.

See Also

Integer Scaling

Mul

Multiplies the elements of two vectors.

Syntax

Case 1. Not-in-place operations on floating point and integer data without scaling.

```
IppStatus ippsMul_16s(const Ipp16s* pSrc1, const Ipp16s* pSrc2, Ipp16s* pDst, int len);
IppStatus ippsMul_32f(const Ipp32f* pSrc1, const Ipp32f* pSrc2, Ipp32f* pDst, int len);
IppStatus ippsMul_64f(const Ipp64f* pSrc1, const Ipp64f* pSrc2, Ipp64f* pDst, int len);
IppStatus ippsMul_32fc(const Ipp32fc* pSrc1, const Ipp32fc* pSrc2, Ipp32fc* pDst, int
len);
IppStatus ippsMul_64fc(const Ipp64fc* pSrc1, const Ipp64fc* pSrc2, Ipp64fc* pDst, int
len);
IppStatus ippsMul_8u16u(const Ipp8u* pSrc1, const Ipp8u* pSrc2, Ipp16u* pDst, int len);
```

```
IppStatus ippsMul_32f32fc(const Ipp32f* pSrc1, const Ipp32fc* pSrc2, Ipp32fc* pDst, int
len);

IppStatus ippsMul_16s32f(const Ipp16s* pSrc1, const Ipp16s* pSrc2, Ipp32f* pDst, int
len);
```

Case 2. Not-in-place operations on integer data with scaling.

```
IppStatus ippsMul_8u_Sfs(const Ipp8u* pSrc1, const Ipp8u* pSrc2, Ipp8u* pDst, int len,
int scaleFactor);

IppStatus ippsMul_16u_Sfs(const Ipp16u* pSrc1, const Ipp16u* pSrc2, Ipp16u* pDst, int
len, int scaleFactor);

IppStatus ippsMul_16s_Sfs(const Ipp16s* pSrc1, const Ipp16s* pSrc2, Ipp16s* pDst, int
len, int scaleFactor);

IppStatus ippsMul_32s_Sfs(const Ipp32s* pSrc1, const Ipp32s* pSrc2, Ipp32s* pDst, int
len, int scaleFactor);

IppStatus ippsMul_16sc_Sfs(const Ipp16sc* pSrc1, const Ipp16sc* pSrc2, Ipp16sc* pDst,
int len, int scaleFactor);

IppStatus ippsMul_32sc_Sfs(const Ipp32sc* pSrc1, const Ipp32sc* pSrc2, Ipp32sc* pDst,
int len, int scaleFactor);

IppStatus ippsMul_16s32s_Sfs(const Ipp16s* pSrc1, const Ipp16s* pSrc2, Ipp32s* pDst,
int len, int scaleFactor);

IppStatus ippsMul_16u16s_Sfs(const Ipp16u* pSrc1, const Ipp16s* pSrc2, Ipp16s* pDst,
int len, int scaleFactor);
```

Case 3. In-place operations on floating point and integer data without scaling

```
IppStatus ippsMul_16s_I(const Ipp16s* pSrc, Ipp16s* pSrcDst, int len);

IppStatus ippsMul_32f_I(const Ipp32f* pSrc, Ipp32f* pSrcDst, int len);

IppStatus ippsMul_64f_I(const Ipp64f* pSrc, Ipp64f* pSrcDst, int len);

IppStatus ippsMul_32fc_I(const Ipp32fc* pSrc, Ipp32fc* pSrcDst, int len);

IppStatus ippsMul_64fc_I(const Ipp64fc* pSrc, Ipp64fc* pSrcDst, int len);

IppStatus ippsMul_32f32fc_I(const Ipp32f* pSrc, Ipp32fc* pSrcDst, int len);
```

Case 4. In-place operations on integer data with scaling

```
IppStatus ippsMul_8u_ISfs(const Ipp8u* pSrc, Ipp8u* pSrcDst, int len, int scaleFactor);

IppStatus ippsMul_16u_ISfs(const Ipp16u* pSrc, Ipp16u* pSrcDst, int len, int
scaleFactor);

IppStatus ippsMul_16s_ISfs(const Ipp16s* pSrc, Ipp16s* pSrcDst, int len, int
scaleFactor);

IppStatus ippsMul_32s_ISfs(const Ipp32s* pSrc, Ipp32s* pSrcDst, int len, int
scaleFactor);

IppStatus ippsMul_16sc_ISfs(const Ipp16sc* pSrc, Ipp16sc* pSrcDst, int len, int
scaleFactor);

IppStatus ippsMul_32sc_ISfs(const Ipp32sc* pSrc, Ipp32sc* pSrcDst, int len, int
scaleFactor);
```

Case 5. Not-in-place operations on floating point data without scaling with platform-aware functions.

```
IppStatus ippsMul_32f_L(const Ipp32f* pSrc1, const Ipp32f* pSrc2, Ipp32f* pDst,
IppSizeL len);

IppStatus ippsMul_64f_L(const Ipp64f* pSrc1, const Ipp64f* pSrc2, Ipp64f* pDst,
IppSizeL len);

IppStatus ippsMul_32fc_L(const Ipp32fc* pSrc1, const Ipp32fc* pSrc2, Ipp32fc* pDst,
IppSizeL len);

IppStatus ippsMul_64fc_L(const Ipp64fc* pSrc1, const Ipp64fc* pSrc2, Ipp64fc* pDst,
IppSizeL len);
```

Case 6. In-place operations on floating point data without scaling with platform-aware functions.

```
IppStatus ippsMul_32f_I_L(const Ipp32f* pSrc, Ipp32f* pSrcDst, IppSizeL len);

IppStatus ippsMul_64f_I_L(const Ipp64f* pSrc, Ipp64f* pSrcDst, IppSizeL len);

IppStatus ippsMul_32fc_I_L(const Ipp32fc* pSrc, Ipp32fc* pSrcDst, IppSizeL len);

IppStatus ippsMul_64fc_I_L(const Ipp64fc* pSrc, Ipp64fc* pSrcDst, IppSizeL len);
```

Include Files

`ipps.h`

Flavors with the `_L` suffix: `ipps_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pSrc1`, `pSrc2`	Pointers to the source vectors.
`pDst`	Pointer to the destination vector.
`pSrc`	Pointer to the source vector for in-place operation.
`pSrcDst`	Pointer to the source and destination vector for in-place operation.
`len`	Number of elements in the vector
`scaleFactor`	Scale factor, refer to Integer Scaling.

Description

This function multiplies the elements of the vector `pSrc1` by the elements of the vector `pSrc2` and stores the result in `pDst`.

The in-place flavors of `ippsMul` multiply the elements of the vector `pSrc` by the elements of the vector `pSrcDst` and store the result in `pSrcDst`.

Function flavors with `Sfs` suffix perform scaling of the result value in accordance with the `scaleFactor` value. If the output value exceeds the data range, the result is saturated.

Function flavor with `Low` suffix requires that each value of the product does not exceed the `Ipp32s` data type range.

Return Values

`ippStsNoErr`	Indicates no error
`ippStsNullPtrErr`	Indicates an error when any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error when `len` is less than, or equal to 0,

See Also

Integer Scaling

SubC

Subtracts a constant value from each element of a vector.

Syntax

Case 1. Not-in-place operations on floating point data.

```
IppStatus ippsSubC_32f(const Ipp32f* pSrc, Ipp32f val, Ipp32f* pDst, int len);
IppStatus ippsSubC_32fc(const Ipp32fc* pSrc, Ipp32fc val, Ipp32fc* pDst, int len);
IppStatus ippsSubC_64f(const Ipp64f* pSrc, Ipp64f val, Ipp64f* pDst, int len);
IppStatus ippsSubC_64fc(const Ipp64fc* pSrc, Ipp64fc val, Ipp64fc* pDst, int len);
```

Case 2. Not-in-place operations on integer data.

```
IppStatus ippsSubC_8u_Sfs(const Ipp8u* pSrc, Ipp8u val, Ipp8u* pDst, int len, int scaleFactor);
IppStatus ippsSubC_16u_Sfs(const Ipp16u* pSrc, Ipp16u val, Ipp16u* pDst, int len, int scaleFactor);
IppStatus ippsSubC_16s_Sfs(const Ipp16s* pSrc, Ipp16s val, Ipp16s* pDst, int len, int scaleFactor);
IppStatus ippsSubC_32s_Sfs(const Ipp32s* pSrc, Ipp32s val, Ipp32s* pDst, int len, int scaleFactor);
IppStatus ippsSubC_16sc_Sfs(const Ipp16sc* pSrc, Ipp16sc val, Ipp16sc* pDst, int len, int scaleFactor);
IppStatus ippsSubC_32sc_Sfs(const Ipp32sc* pSrc, Ipp32sc val, Ipp32sc* pDst, int len, int scaleFactor);
```

Case 3. In-place operations on floating point data.

```
IppStatus ippsSubC_16s_I(Ipp16s val, Ipp16s* pSrcDst, int len);
IppStatus ippsSubC_32f_I(Ipp32f val, Ipp32f* pSrcDst, int len);
IppStatus ippsSubC_64f_I(Ipp64f val, Ipp64f* pSrcDst, int len);
IppStatus ippsSubC_32fc_I(Ipp32fc val, Ipp32fc* pSrcDst, int len);
IppStatus ippsSubC_64fc_I(Ipp64fc val, Ipp64fc* pSrcDst, int len);
```

Case 4. In-place operations on integer data.

```
IppStatus ippsSubC_8u_ISfs(Ipp8u val, Ipp8u* pSrcDst, int len, int scaleFactor);
IppStatus ippsSubC_16u_ISfs(Ipp16u val, Ipp16u* pSrcDst, int len, int scaleFactor);
IppStatus ippsSubC_16s_ISfs(Ipp16s val, Ipp16s* pSrcDst, int len, int scaleFactor);
IppStatus ippsSubC_32s_ISfs(Ipp32s val, Ipp32s* pSrcDst, int len, int scaleFactor);
IppStatus ippsSubC_16sc_ISfs(Ipp16sc val, Ipp16sc* pSrcDst, int len, int scaleFactor);
IppStatus ippsSubC_32sc_ISfs(Ipp32sc val, Ipp32sc* pSrcDst, int len, int scaleFactor);
```

Case 5: Not-in-place operations on floating point data.

```
IppStatus ippsSubC_32f_L(const Ipp32f* pSrc, Ipp32f val, Ipp32f* pDst, IppSizeL len);
IppStatus ippsSubC_64f_L(const Ipp64f* pSrc, Ipp64f val, Ipp64f* pDst, IppSizeL len);
IppStatus ippsSubC_32fc_L(const Ipp32fc* pSrc, Ipp32fc val, Ipp32fc* pDst, IppSizeL
len);
IppStatus ippsSubC_64fc_L(const Ipp64fc* pSrc, Ipp64fc val, Ipp64fc* pDst, IppSizeL
len);
```

Case 6: In-place operations on floating point data.

```
IppStatus ippsSubC_32f_I_L(Ipp32f val, Ipp32f* pSrcDst, IppSizeL len);
IppStatus ippsSubC_64f_I_L(Ipp64f val, Ipp64f* pSrcDst, IppSizeL len);
IppStatus ippsSubC_32fc_I_L(Ipp32fc val, Ipp32fc* pSrcDst, IppSizeL len);
IppStatus ippsSubC_64fc_I_L(Ipp64fc val, Ipp64fc* pSrcDst, IppSizeL len);
```

Include Files

`ipps.h`

Flavors with the `_L` suffix: `ipps_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pSrc`	Pointer to the source vector.
`val`	Scalar value used to decrement each element of the source vector.
`pDst`	Pointer to the destination vector.
`pSrcDst`	Pointer to the source and destination vector for in-place operation.
`len`	Number of elements in the vector
`scaleFactor`	Scale factor, refer to Integer Scaling.

Description

This function subtracts a value `val` from each element of the vector `pSrc`, and stores the result in `pDst`.

The in-place flavors of `ippsSubC` subtract a value `val` from each element of the vector `pSrcDst` and store the result in `pSrcDst`.

Functions with `Sfs` suffixes perform scaling of the result value in accordance with the `scaleFactor` value. If the output value exceeds the data range, the result becomes saturated.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when the `pSrc`, `pDst`, or `pSrcDst` pointer is `NULL`.
`ippStsSizeErr`	Indicates an error when `len` is less than or equal to 0.

SubCRev

Subtracts each element of a vector from a constant value.

Syntax

Case 1. Not-in-place operations on floating point data.

```
IppStatus ippsSubCRev_32f(const Ipp32f* pSrc, Ipp32f val, Ipp32f* pDst, int len);
IppStatus ippsSubCRev_64f(const Ipp64f* pSrc, Ipp64f val, Ipp64f* pDst, int len);
IppStatus ippsSubCRev_32fc(const Ipp32fc* pSrc, Ipp32fc val, Ipp32fc* pDst, int len);
IppStatus ippsSubCRev_64fc(const Ipp64fc* pSrc, Ipp64fc val, Ipp64fc* pDst, int len);
```

Case 2. Not-in-place operations on integer data.

```
IppStatus ippsSubCRev_8u_Sfs(const Ipp8u* pSrc, Ipp8u val, Ipp8u* pDst, int len, int
scaleFactor);
IppStatus ippsSubCRev_16u_Sfs(const Ipp16u* pSrc, Ipp16u val, Ipp16u* pDst, int len,
int scaleFactor);
IppStatus ippsSubCRev_16s_Sfs(const Ipp16s* pSrc, Ipp16s val, Ipp16s* pDst, int len,
int scaleFactor);
IppStatus ippsSubCRev_32s_Sfs(const Ipp32s* pSrc, Ipp32s val, Ipp32s* pDst, int len,
int scaleFactor);
IppStatus ippsSubCRev_16sc_Sfs(const Ipp16sc* pSrc, Ipp16sc val, Ipp16sc* pDst, int
len, int scaleFactor);
IppStatus ippsSubCRev_32sc_Sfs(const Ipp32sc* pSrc, Ipp32sc val, Ipp32sc* pDst, int
len, int scaleFactor);
```

Case 3. In-place operations on floating point data.

```
IppStatus ippsSubCRev_32f_I(Ipp32f val, Ipp32f* pSrcDst, int len);
IppStatus ippsSubCRev_64f_I(Ipp64f val, Ipp64f* pSrcDst, int len);
IppStatus ippsSubCRev_32fc_I(Ipp32fc val, Ipp32fc* pSrcDst, int len);
```

```
IppStatus ippsSubCRev_64fc_I(Ipp64fc val, Ipp64fc* pSrcDst, int len);
```

Case 4. In-place operations on integer data.

```
IppStatus ippsSubCRev_8u_ISfs(Ipp8u val, Ipp8u* pSrcDst, int len, int scaleFactor);
IppStatus ippsSubCRev_16u_ISfs(Ipp16u val, Ipp16u* pSrcDst, int len, int scaleFactor);
IppStatus ippsSubCRev_16s_ISfs(Ipp16s val, Ipp16s* pSrcDst, int len, int scaleFactor);
IppStatus ippsSubCRev_32s_ISfs(Ipp32s val, Ipp32s* pSrcDst, int len, int scaleFactor);
IppStatus ippsSubCRev_16sc_ISfs(Ipp16sc val, Ipp16sc* pSrcDst, int len, int
scaleFactor);
IppStatus ippsSubCRev_32sc_ISfs(Ipp32sc val, Ipp32sc* pSrcDst, int len, int
scaleFactor);
```

Case 5: Not-in-place operations on floating point data.

```
IppStatus ippsSubCRev_32f_L(const Ipp32f* pSrc, Ipp32f val, Ipp32f* pDst, IppSizeL
len);
IppStatus ippsSubCRev_64f_L(const Ipp64f* pSrc, Ipp64f val, Ipp64f* pDst, IppSizeL
len);
IppStatus ippsSubCRev_32fc_L(const Ipp32fc* pSrc, Ipp32fc val, Ipp32fc* pDst, IppSizeL
len);
IppStatus ippsSubCRev_64fc_L(const Ipp64fc* pSrc, Ipp64fc val, Ipp64fc* pDst, IppSizeL
len);
```

Case 6: In-place operations on floating point data.

```
IppStatus ippsSubCRev_32f_I_L(Ipp32f val, Ipp32f* pSrcDst, IppSizeL len);
IppStatus ippsSubCRev_64f_I_L(Ipp64f val, Ipp64f* pSrcDst, IppSizeL len);
IppStatus ippsSubCRev_32fc_I_L(Ipp32fc val, Ipp32fc* pSrcDst, IppSizeL len);
IppStatus ippsSubCRev_64fc_I_L(Ipp64fc val, Ipp64fc* pSrcDst, IppSizeL len);
```

Include Files

`ipps.h`

Flavors with the `_L` suffix: `ipps_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`val`	Scalar value from which vector elements are subtracted.
`pSrc`	Pointer to the source vector.
`pDst`	Pointer to the destination vector.

pSrcDst	Pointer to the vector whose elements are to be subtracted from the value val in case of the in-place operation. The destination vector which stores the result of the subtraction val-pSrcDst[n].
len	Number of elements in the vector
scaleFactor	Scale factor, refer to Integer Scaling.

Description

This function subtracts each element of the vector pSrc from a value val and stores the result in pDst.

The in-place flavors of ippsSubCRev subtract each element of the vector pSrcDst from a value val and store the result in pSrcDst.

Functions with Sfs suffixes perform scaling of the result value in accordance with the scaleFactor value. If the output value exceeds the data range, the result becomes saturated.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when the pSrc, pDst, or pSrcDst pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.

Sub

Subtracts the elements of two vectors.

Syntax

Case 1. Not-in-place operations on floating point data, and integer data without scaling.

```
IppStatus ippsSub_16s(const Ipp16s* pSrc1, const Ipp16s* pSrc2, Ipp16s* pDst, int len);
IppStatus ippsSub_32f(const Ipp32f* pSrc1, const Ipp32f* pSrc2, Ipp32f* pDst, int len);
IppStatus ippsSub_64f(const Ipp64f* pSrc1, const Ipp64f* pSrc2, Ipp64f* pDst, int len);
IppStatus ippsSub_32fc(const Ipp32fc* pSrc1, const Ipp32fc* pSrc2, Ipp32fc* pDst, int len);
IppStatus ippsSub_64fc(const Ipp64fc* pSrc1, const Ipp64fc* pSrc2, Ipp64fc* pDst, int len);
IppStatus ippsSub_16s32f(const Ipp16s* pSrc1, const Ipp16s* pSrc2, Ipp32f* pDst, int len);
```

Case 2. Not-in-place operations on integer data with scaling.

```
IppStatus ippsSub_8u_Sfs(const Ipp8u* pSrc1, const Ipp8u* pSrc2, Ipp8u* pDst, int len, int scaleFactor);
IppStatus ippsSub_16u_Sfs(const Ipp16u* pSrc1, const Ipp16u* pSrc2, Ipp16u* pDst, int len, int scaleFactor);
```

```
IppStatus ippsSub_16s_Sfs(const Ipp16s* pSrc1, const Ipp16s* pSrc2, Ipp16s* pDst, int
len, int scaleFactor);

IppStatus ippsSub_32s_Sfs(const Ipp32s* pSrc1, const Ipp32s* pSrc2, Ipp32s* pDst, int
len, int scaleFactor);

IppStatus ippsSub_16sc_Sfs(const Ipp16sc* pSrc1, const Ipp16sc* pSrc2, Ipp16sc* pDst,
int len, int scaleFactor);

IppStatus ippsSub_32sc_Sfs(const Ipp32sc* pSrc1, const Ipp32sc* pSrc2, Ipp32sc* pDst,
int len, int scaleFactor);
```

Case 3. In-place operations on floating point data and integer data without scaling.

```
IppStatus ippsSub_16s_I(const Ipp16s* pSrc, Ipp16s* pSrcDst, int len);

IppStatus ippsSub_32f_I(const Ipp32f* pSrc, Ipp32f* pSrcDst, int len);

IppStatus ippsSub_64f_I(const Ipp64f* pSrc, Ipp64f* pSrcDst, int len);

IppStatus ippsSub_32fc_I(const Ipp32fc* pSrc, Ipp32fc* pSrcDst, int len);

IppStatus ippsSub_64fc_I(const Ipp64fc* pSrc, Ipp64fc* pSrcDst, int len);
```

Case 4. In-place operations on integer data with scaling.

```
IppStatus ippsSub_8u_ISfs(const Ipp8u* pSrc, Ipp8u* pSrcDst, int len, int scaleFactor);

IppStatus ippsSub_16u_ISfs(const Ipp16u* pSrc, Ipp16u* pSrcDst, int len, int
scaleFactor);

IppStatus ippsSub_16s_ISfs(const Ipp16s* pSrc, Ipp16s* pSrcDst, int len, int
scaleFactor);

IppStatus ippsSub_32s_ISfs(const Ipp32s* pSrc, Ipp32s* pSrcDst, int len, int
scaleFactor);

IppStatus ippsSub_16sc_ISfs(const Ipp16sc* pSrc, Ipp16sc* pSrcDst, int len, int
scaleFactor);

IppStatus ippsSub_32sc_ISfs(const Ipp32sc* pSrc, Ipp32sc* pSrcDst, int len, int
scaleFactor);
```

Case 5. Not-in-place operations on floating point data, and integer data without scaling with platform-aware functions.

```
IppStatus ippsSub_32f_L(const Ipp32f* pSrc1, const Ipp32f* pSrc2, Ipp32f* pDst,
IppSizeL len);

IppStatus ippsSub_64f_L(const Ipp64f* pSrc1, const Ipp64f* pSrc2, Ipp64f* pDst,
IppSizeL len);

IppStatus ippsSub_32fc_L(const Ipp32fc* pSrc1, const Ipp32fc* pSrc2, Ipp32fc* pDst,
IppSizeL len);

IppStatus ippsSub_64fc_L(const Ipp64fc* pSrc1, const Ipp64fc* pSrc2, Ipp64fc* pDst,
IppSizeL len);
```

Case 6. In-place operations on floating point data, and integer data without scaling with platform-aware functions.

```
IppStatus ippsSub_32f_I_L(const Ipp32f* pSrc, Ipp32f* pSrcDst, IppSizeL len);

IppStatus ippsSub_64f_I_L(const Ipp64f* pSrc, Ipp64f* pSrcDst, IppSizeL len);

IppStatus ippsSub_32fc_I_L(const Ipp32fc* pSrc, Ipp32fc* pSrcDst, IppSizeL len);
```

```
IppStatus ippsSub_64fc_I_L(const Ipp64fc* pSrc, Ipp64fc* pSrcDst, IppSizeL len);
```

Include Files

`ipps.h`

Flavors with the `_L` suffix: `ipps_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pSrc1`	Pointer to the source vector-subtrahend, whose elements are to be subtracted.
`pSrc2`	Pointer to the source vector-minuend from whose elements the elements of `pSrc1` are to be subtracted.
`pDst`	Pointer to the destination vector.
`pSrc`	Pointer to the source vector-subtrahend for in-place operation.
`pSrcDst`	Pointer to the source vector-minuend and destination vector for in-place operation.
`len`	Number of elements in the vector.
`scaleFactor`	Scale factor, refer to Integer Scaling.

Description

This function subtracts the elements of the vector `pSrc1` from the elements of the vector `pSrc2`, and stores the result in `pDst`.

The in-place flavors of `ippsSub` subtract the elements of the vector `pSrc` from the elements of a vector `pSrcDst` and store the result in `pSrcDst`.

Functions with `Sfs` suffixes perform scaling of the result value in accordance with the `scaleFactor` value. If the output value exceeds the data range, the result becomes saturated.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error when `len` is less than or equal to 0.

DivC

Divides each element of a vector by a constant value.

Syntax

Case 1. Not-in-place operations on floating point data.

```
IppStatus ippsDivC_32f(const Ipp32f* pSrc, Ipp32f val, Ipp32f* pDst, int len);
IppStatus ippsDivC_64f(const Ipp64f* pSrc, Ipp64f val, Ipp64f* pDst, int len);
IppStatus ippsDivC_32fc(const Ipp32fc* pSrc, Ipp32fc val, Ipp32fc* pDst, int len);
IppStatus ippsDivC_64fc(const Ipp64fc* pSrc, Ipp64fc val, Ipp64fc* pDst, int len);
```

Case 2. Not-in-place operations on integer data with scaling.

```
IppStatus ippsDivC_8u_Sfs(const Ipp8u* pSrc, Ipp8u val, Ipp8u* pDst, int len, int
scaleFactor);
IppStatus ippsDivC_16u_Sfs(const Ipp16u* pSrc, Ipp16u val, Ipp16u* pDst, int len, int
scaleFactor);
IppStatus ippsDivC_16s_Sfs(const Ipp16s* pSrc, Ipp16s val, Ipp16s* pDst, int len, int
scaleFactor);
IppStatus ippsDivC_16sc_Sfs(const Ipp16sc* pSrc, Ipp16sc val, Ipp16sc* pDst, int len,
int scaleFactor);
```

Case 3. In-place operations on floating point data.

```
IppStatus ippsDivC_32f_I(Ipp32f val, Ipp32f* pSrcDst, int len);
IppStatus ippsDivC_64f_I(Ipp64f val, Ipp64f* pSrcDst, int len);
IppStatus ippsDivC_32fc_I(Ipp32fc val, Ipp32fc* pSrcDst, int len);
IppStatus ippsDivC_64fc_I(Ipp64fc val, Ipp64fc* pSrcDst, int len);
```

Case 4. In-place operations on integer data with scaling.

```
IppStatus ippsDivC_8u_ISfs(Ipp8u val, Ipp8u* pSrcDst, int len, int scaleFactor);
IppStatus ippsDivC_16u_ISfs(Ipp16u val, Ipp16u* pSrcDst, int len, int scaleFactor);
IppStatus ippsDivC_16s_ISfs(Ipp16s val, Ipp16s* pSrcDst, int len, int scaleFactor);
IppStatus ippsDivC_64s_ISfs(Ipp64s val, Ipp64s* pSrcDst, Ipp32u len, int scaleFactor);
IppStatus ippsDivC_16sc_ISfs(Ipp16sc val, Ipp16sc* pSrcDst, int len, int scaleFactor);
```

Case 5. Not-in-place operations on floating point data with platform-aware functions.

```
IppStatus ippsDivC_32f_L(const Ipp32f* pSrc, Ipp32f val, Ipp32f* pDst, IppSizeL len);
IppStatus ippsDivC_64f_L(const Ipp64f* pSrc, Ipp64f val, Ipp64f* pDst, IppSizeL len);
IppStatus ippsDivC_32fc_L(const Ipp32fc* pSrc, Ipp32fc val, Ipp32fc* pDst, IppSizeL
len);
IppStatus ippsDivC_64fc_L(const Ipp64fc* pSrc, Ipp64fc val, Ipp64fc* pDst, IppSizeL
len);
```

Case 6. In-place operations on floating point data with platform-aware functions.

```
IppStatus ippsDivC_32f_I_L(Ipp32f val, Ipp32f* pSrcDst, IppSizeL len);
IppStatus ippsDivC_64f_I_L(Ipp64f val, Ipp64f* pSrcDst, IppSizeL len);
IppStatus ippsDivC_32fc_I_L(Ipp32fc val, Ipp32fc* pSrcDst, IppSizeL len);
```

```
IppStatus ippsDivC_64fc_I_L(Ipp64fc val, Ipp64fc* pSrcDst, IppSizeL len);
```

Include Files

`ipps.h`

Flavors with the `_L` suffix: `ipps_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`val`	Scalar value used as a divisor.
`pSrc`	Pointer to the source vector.
`pDst`	Pointer to the destination vector.
`pSrcDst`	Pointer to the source and destination vector for in-place operation.
`len`	Number of elements in the vector
`scaleFactor`	Scale factor, refer to Integer Scaling.

Description

This function divides each element of the vector `pSrc` by a value `val` and stores the result in `pDst`.

The in-place flavors of `ippsDivC` divide each element of the vector `pSrcDst` by a value `val` and store the result in `pSrcDst`.

Functions with `Sfs` suffixes perform scaling of the result value in accordance with the `scaleFactor` value. If the output value exceeds the data range, the result becomes saturated.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when the `pSrc`, `pDst`, or `pSrcDst` pointer is `NULL`.
`ippStsSizeErr`	Indicates an error when `len` is less than or equal to 0.
`ippStsDivByZeroErr`	Indicates an error when `val` is equal to 0.

DivCRev

Divides a constant value by each element of a vector.

Syntax

```
IppStatus ippsDivCRev_16u(const Ipp16u* pSrc, Ipp16u val, Ipp16u* pDst, int len);
```

```
IppStatus ippsDivCRev_32f(const Ipp32f* pSrc, Ipp32f val, Ipp32f* pDst, int len);
IppStatus ippsDivCRev_16u_I(Ipp16u val, Ipp16u* pSrcDst, int len);
IppStatus ippsDivCRev_32f_I(Ipp32f val, Ipp32f* pSrcDst, int len);
IppStatus ippsDivCRev_32f_L(const Ipp32f* pSrc, Ipp32f val, Ipp32f* pDst, IppSizeL
len);
IppStatus ippsDivCRev_32f_I_L(Ipp32f val, Ipp32f* pSrcDst, IppSizeL len);
```

Include Files

`ipps.h`

Flavors with the `_L` suffix: `ipps_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`val`	Constant value used as a dividend in the operation.
`pSrc`	Pointer to the source vector whose elements are used as divisors.
`pDst`	Pointer to the destination vector.
`pSrcDst`	Pointer to the source and destination vector for in-place operation.
`len`	Number of elements in the vector

Description

This function divides the constant value `val` by each element of the vector `pSrc` and stores the results in `pDst`.

The in-place flavors of `ippsDivC` divide the constant value `val` by each element of the vector `pSrcDst` and store the results in `pSrcDst`.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when the `pSrc`, `pDst`, or `pSrcDst` pointer is `NULL`.
`ippStsSizeErr`	Indicates an error when `len` is less than or equal to 0.
`ippStsDivByZeroErr`	Indicates an error when any element of the vector `pSource` is equal to 0.

Div

Divides the elements of two vectors.

Syntax

Case 1. Not-in-place operations on integer data.

```
IppStatus ippsDiv_8u_Sfs(const Ipp8u* pSrc1, const Ipp8u* pSrc2, Ipp8u* pDst, int len,
int scaleFactor);

IppStatus ippsDiv_16u_Sfs(const Ipp16u* pSrc1, const Ipp16u* pSrc2, Ipp16u* pDst, int
len, int scaleFactor);

IppStatus ippsDiv_16s_Sfs(const Ipp16s* pSrc1, const Ipp16s* pSrc2, Ipp16s* pDst, int
len, int scaleFactor);

IppStatus ippsDiv_32s_Sfs(const Ipp32s* pSrc1, const Ipp32s* pSrc2, Ipp32s* pDst, int
len, int scaleFactor);

IppStatus ippsDiv_16sc_Sfs(const Ipp16sc* pSrc1, const Ipp16sc* pSrc2, Ipp16sc* pDst,
int len, int scaleFactor);

IppStatus ippsDiv_32s16s_Sfs(const Ipp16s* pSrc1, const Ipp32s* pSrc2, Ipp16s* pDst,
int len, int scaleFactor);
```

Case 2. Not-in-place operations on floating point data.

```
IppStatus ippsDiv_32f(const Ipp32f* pSrc1, const Ipp32f* pSrc2, Ipp32f* pDst, int len);

IppStatus ippsDiv_64f(const Ipp64f* pSrc1, const Ipp64f* pSrc2, Ipp64f* pDst, int len);

IppStatus ippsDiv_32fc(const Ipp32fc* pSrc1, const Ipp32fc* pSrc2, Ipp32fc* pDst, int
len);

IppStatus ippsDiv_64fc(const Ipp64fc* pSrc1, const Ipp64fc* pSrc2, Ipp64fc* pDst, int
len);
```

Case 3. In-place operations on integer data.

```
IppStatus ippsDiv_8u_ISfs(const Ipp8u* pSrc, Ipp8u* pSrcDst, int len, int scaleFactor);

IppStatus ippsDiv_16u_ISfs(const Ipp16u* pSrc, Ipp16u* pSrcDst, int len, int
scaleFactor);

IppStatus ippsDiv_16s_ISfs(const Ipp16s* pSrc, Ipp16s* pSrcDst, int len, int
scaleFactor);

IppStatus ippsDiv_16sc_ISfs(const Ipp16sc* pSrc, Ipp16sc* pSrcDst, int len, int
scaleFactor);

IppStatus ippsDiv_32s_ISfs(const Ipp32s* pSrc, Ipp32s* pSrcDst, int len, int
scaleFactor);
```

Case 4. In-place operations on floating point data.

```
IppStatus ippsDiv_32f_I(const Ipp32f* pSrc, Ipp32f* pSrcDst, int len);

IppStatus ippsDiv_64f_I(const Ipp64f* pSrc, Ipp64f* pSrcDst, int len);

IppStatus ippsDiv_32fc_I(const Ipp32fc* pSrc, Ipp32fc* pSrcDst, int len);

IppStatus ippsDiv_64fc_I(const Ipp64fc* pSrc, Ipp64fc* pSrcDst, int len);
```

Case 5. Not-in-place operations on floating point data with platform-aware functions.

```
IppStatus ippsDiv_32f_L(const Ipp32f* pSrc1, const Ipp32f* pSrc2, Ipp32f* pDst,
IppSizeL len);
```

```
IppStatus ippsDiv_64f_L(const Ipp64f* pSrc1, const Ipp64f* pSrc2, Ipp64f* pDst,
IppSizeL len);

IppStatus ippsDiv_32fc_L(const Ipp32fc* pSrc1, const Ipp32fc* pSrc2, Ipp32fc* pDst,
IppSizeL len);

IppStatus ippsDiv_64fc_L(const Ipp64fc* pSrc1, const Ipp64fc* pSrc2, Ipp64fc* pDst,
IppSizeL len);
```

Case 6. In-place operations on floating point data with platform-aware functions.

```
IppStatus ippsDiv_32f_I_L(const Ipp32f* pSrc, Ipp32f* pSrcDst, IppSizeL len);

IppStatus ippsDiv_64f_I_L(const Ipp64f* pSrc, Ipp64f* pSrcDst, IppSizeL len);

IppStatus ippsDiv_32fc_I_L(const Ipp32fc* pSrc, Ipp32fc* pSrcDst, IppSizeL len);

IppStatus ippsDiv_64fc_I_L(const Ipp64fc* pSrc, Ipp64fc* pSrcDst, IppSizeL len);
```

Include Files

`ipps.h`

Flavors with the `_L` suffix: `ipps_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pSrc1`	Pointer to the divisor vector.
`pSrc2`	Pointer to the dividend vector.
`pDst`	Pointer to the destination vector.
`pSrc`	Pointer to the divisor vector for in-place operations.
`pSrcDst`	Pointer to the source and destination vector for in-place operations.
`len`	Number of elements in the vector.
`scaleFactor`	Scale factor, refer to Integer Scaling.

Description

This function divides the elements of the `pSrc2` vector by the elements of the `pSrc1` vector , and stores the result in `pDst`.

The in-place flavors of `ippsDiv` divide the elements of the vector `pSrcDst` by the elements of the vector `pSrc` and store the result in `pSrcDst`.

Functions with `Sfs` suffixe perform scaling of the result in accordance with the `scaleFactor` value. If the output value exceeds the data range, the result is saturated.

If any of the divisor vector elements is equal to zero, the function returns a warning and continues execution with the corresponding result value. For more information see “Handling of Special Cases”.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to zero.
ippStsDivByZero	Indicates a warning when any of the divisor vector elements is equal to zero.

Example

Div:

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

#include <stdio.h>
#include "ipp.h"

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define
check_sts(st)
                                  \
    if ((st) !=
ippStsNoErr)
                          \
        goto exitLine; /* Go to Exit if Intel(R) Integrated Performance Primitives (Intel(R)
IPP) function returned status different from \
                           ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main()
{
    int len = 32;
    int i;
    Ipp64f *pDst = ippsMalloc_64f(len * sizeof(Ipp64f));
    Ipp64f *pSrc1 = ippsMalloc_64f(len * sizeof(Ipp64f));
    Ipp64f *pSrc2 = ippsMalloc_64f(len * sizeof(Ipp64f));
```

```
    IppStatus status;

    printf("\n\nSource vector 1\n");
    for (i = 0; i < len; i++) {
        pSrc1[i] = i;
        printf("%.0f; ", pSrc1[i]);
    }
    printf("\n\nSource vector 2\n");
    for (i = 0; i < len; i++) {
        pSrc2[i] = 10 + i;
        printf("%.0f; ", pSrc2[i]);
    }

    check_sts(status = ippsDiv_64f(pSrc2, pSrc1, pDst, len));

    printf("\n\nResult\n");
    for (i = 0; i < len; i++)
        printf("%.2f; ", pDst[i]);
    printf("\n\n");

    EXIT_MAIN
    ippsFree(pSrc1);
    ippsFree(pSrc2);
    ippsFree(pDst);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

Div_I:

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

#include <stdio.h>
#include "ipp.h"

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define
check_sts(st)
                                  \
    if ((st) !=
ippStsNoErr)
                          \
        goto exitLine; /* Go to Exit if Intel(R) Integrated Performance Primitives (Intel(R)
```

```
IPP) function returned status different from \
                          ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main()
{
    int len = 32;
    int i;
    Ipp64f *pSrc1 = ippsMalloc_64f(len * sizeof(Ipp64f));
    Ipp64f *pSrcDst = ippsMalloc_64f(len * sizeof(Ipp64f));
    IppStatus status;

    printf("\n\nSource vector 1\n");
    for (i = 0; i < len; i++) {
        pSrcDst[i] = i;
        printf("%.0f; ", pSrcDst[i]);
    }
    printf("\n\nSource vector 2\n");
    for (i = 0; i < len; i++) {
        pSrc1[i] = 10 + i;
        printf("%.0f; ", pSrc1[i]);
    }

    check_sts(status = ippsDiv_64f_I(pSrc1, pSrcDst, len));

    printf("\n\nResult\n");
    for (i = 0; i < len; i++)
        printf("%.2f; ", pSrcDst[i]);
    printf("\n\n");

    EXIT_MAIN
    ippsFree(pSrc1);
    ippsFree(pSrcDst);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

See Also

Integer Scaling

Appendix A Handling of Special Cases

Div_Round

Divides the elements of two vectors with rounding.

Syntax

Case 1. Not-in-place operations on integer data.

```
IppStatus ippsDiv_Round_8u_Sfs(const Ipp8u* pSrc1, const Ipp8u* pSrc2, Ipp8u* pDst, int
len, IppRoundMode rndMode, int scaleFactor);

IppStatus ippsDiv_Round_16u_Sfs(const Ipp16u* pSrc1, const Ipp16u* pSrc2, Ipp16u* pDst,
int len, IppRoundMode rndMode, int scaleFactor);
```

```
IppStatus ippsDiv_Round_16s_Sfs(const Ipp16s* pSrc1, const Ipp16s* pSrc2, Ipp16s* pDst,
int len, IppRoundMode rndMode, int scaleFactor);
```

Case 2. In-place operations on integer data.

```
IppStatus ippsDiv_Round_8u_ISfs(const Ipp8u* pSrc, Ipp8u* pSrcDst, int len,
IppRoundMode rndMode, int scaleFactor);
```

```
IppStatus ippsDiv_Round_16u_ISfs(const Ipp16u* pSrc, Ipp16u* pSrcDst, int len,
IppRoundMode rndMode, int scaleFactor);
```

```
IppStatus ippsDiv_Round_16s_ISfs(const Ipp16s* pSrc, Ipp16s* pSrcDst, int len,
IppRoundMode rndMode, int scaleFactor);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pSrc1`	Pointer to the vector whose elements are used as divisors.
`pSrc2`	Pointer to the vector whose elements are used as dividends.
`pDst`	Pointer to the destination vector.
`pSrc`	Pointer to the source vector whose elements are used as divisors for in-place operations.
`pSrcDst`	Pointer to the source and destination vector for in-place operations.
`len`	Number of elements in the vector.
`rndMode`	Rounding mode, the following values are possible: • `ippRndZero` - specifies that floating-point values are truncated toward zero, • `ippRndNear` - specifies that floating-point values are rounded to the nearest even integer when the fractional part equals 0.5; otherwise they are rounded to the nearest integer, • `ippRndFinancial` - specifies that floating-point values are rounded down to the nearest integer when the fractional part is less than 0.5, or rounded up to

	the nearest integer if the fractional part is equal or greater than 0.5.
scaleFactor	Scale factor, refer to Integer Scaling.

Description

This function divides the elements of the vector pSrc2 by the elements of the vector pSrc1, the result is rounded using the rounding method specified by the parameter roundMode and stored in the vector pDst.

The in-place flavors of ippsDiv_Round divide the elements of the vector pSrcDst by the elements of the vector pSrc, the result is rounded using the rounding method specified by the parameter roundMode and stored in the vector pSrcDst.

Functions perform scaling of the result value in accordance with the scaleFactor value. If the output value exceeds the data range, the result becomes saturated.

If the function ippsDiv_Round encounters a zero-valued divisor vector element, it returns a warning status and continues execution with the corresponding result value (see appendix A "Handling of Special Cases" for more information).

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.
ippStsDivByZero	Indicates a warning for zero-valued divisor vector element. The function execution is continued.
ippStsRoundModeNotSupportedErr	Indicates an error condition if the roundMode has an illegal value.

Abs

Computes absolute values of vector elements.

Syntax

```
IppStatus ippsAbs_16s(const Ipp16s* pSrc, Ipp16s* pDst, int len);
IppStatus ippsAbs_32s(const Ipp32s* pSrc, Ipp32s* pDst, int len);
IppStatus ippsAbs_32f(const Ipp32f* pSrc, Ipp32f* pDst, int len);
IppStatus ippsAbs_64f(const Ipp64f* pSrc, Ipp64f* pDst, int len);
IppStatus ippsAbs_16s_I(Ipp16s* pSrcDst, int len);
IppStatus ippsAbs_32s_I(Ipp32s* pSrcDst, int len);
IppStatus ippsAbs_32f_I(Ipp32f* pSrcDst, int len);
IppStatus ippsAbs_64f_I(Ipp64f* pSrcDst, int len);
```

Platform-aware functions

```
IppStatus ippsAbs_32f_L(const Ipp32f* pSrc, Ipp32f* pDst, IppSizeL len);
IppStatus ippsAbs_64f_L(const Ipp64f* pSrc, Ipp64f* pDst, IppSizeL len);
IppStatus ippsAbs_32f_I_L(Ipp32f* pSrcDst, IppSizeL len);
IppStatus ippsAbs_64f_I_L(Ipp64f* pSrcDst, IppSizeL len);
```

Include Files

`ipps.h`

For functions with the `_L` suffix: `ipps_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pSrc`	Pointer to the source vector.
`pDst`	Pointer to the destination vector.
`pSrcDst`	Pointer to the source and destination vector for in-place operations.
`len`	Number of elements in the vector.

Description

This function computes the absolute values of each element of the vector `pSrc` and stores the result in `pDst`. The in-place flavors of `ippsAbs` compute the absolute values of each element of the vector `pSrcDst` and store the result in `pSrcDst`.

To compute the absolute values of complex data, use the function ippsMagnitude ippsMagnitude.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when the `pSrc`, `pDst`, or `pSrcDst` pointer is `NULL`.
`ippStsSizeErr`	Indicates an error when `len` is less than or equal to 0.

Example

`Abs`:

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
```

```
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *****************************************************************************/

#include <stdio.h>
#include "ipp.h"

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define
check_sts(st)
                                  \
    if ((st) !=
ippStsNoErr)
                          \
        goto exitLine; /* Go to Exit if Intel(R) Integrated Performance Primitives (Intel(R)
IPP) function returned status different from \
                           ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main()
{
    int len = 32;
    int i;
    Ipp64f *pSrc = ippsMalloc_64f(len * sizeof(Ipp64f));
    Ipp64f *pDst = ippsMalloc_64f(len * sizeof(Ipp64f));
    IppStatus status;

    printf("\n\nSource vector\n");
    for (i = 0; i < len; i++) {
        pSrc[i] = 10 - i;
        printf("%.0f; ", pSrc[i]);
    }

    check_sts(status = ippsAbs_64f(pSrc, pDst, len));

    printf("\n\nResult\n");
    for (i = 0; i < len; i++)
        printf("%.0f; ", pDst[i]);
    printf("\n\n");

    EXIT_MAIN
    ippsFree(pSrc);
    ippsFree(pDst);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

Abs_I:

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

#include <stdio.h>
#include "ipp.h"

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define
check_sts(st)
                                  \
    if ((st) !=
ippStsNoErr)
                          \
        goto exitLine; /* Go to Exit if Intel(R) Integrated Performance Primitives (Intel(R)
IPP) function returned status different from \
                           ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main()
{
    int len = 32;
    int i;
    Ipp64f *pSrcDst = ippsMalloc_64f(len * sizeof(Ipp64f));
    IppStatus status;

    printf("\n\nSource vector\n");
    for (i = 0; i < len; i++) {
        pSrcDst[i] = 10 - i;
        printf("%.0f; ", pSrcDst[i]);
    }

    check_sts(status = ippsAbs_64f_I(pSrcDst, len));

    printf("\n\nResult\n");
    for (i = 0; i < len; i++)
        printf("%.0f; ", pSrcDst[i]);
    printf("\n\n");

    EXIT_MAIN
    ippsFree(pSrcDst);
```

```
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

Sqr

Computes a square of each element of a vector.

Syntax

```
IppStatus ippsSqr_32f(const Ipp32f* pSrc, Ipp32f* pDst, int len);
IppStatus ippsSqr_64f(const Ipp64f* pSrc, Ipp64f* pDst, int len);
IppStatus ippsSqr_32fc(const Ipp32fc* pSrc, Ipp32fc* pDst, int len);
IppStatus ippsSqr_64fc(const Ipp64fc* pSrc, Ipp64fc* pDst, int len);
IppStatus ippsSqr_8u_Sfs(const Ipp8u* pSrc, Ipp8u* pDst, int len, int scaleFactor);
IppStatus ippsSqr_16s_Sfs(const Ipp16s* pSrc, Ipp16s* pDst, int len, int scaleFactor);
IppStatus ippsSqr_16u_Sfs(const Ipp16u* pSrc, Ipp16u* pDst, int len, int scaleFactor);
IppStatus ippsSqr_16sc_Sfs(const Ipp16sc* pSrc, Ipp16sc* pDst, int len, int
scaleFactor);
IppStatus ippsSqr_32f_I(Ipp32f* pSrcDst, int len);
IppStatus ippsSqr_64f_I(Ipp64f* pSrcDst, int len);
IppStatus ippsSqr_32fc_I(Ipp32fc* pSrcDst, int len);
IppStatus ippsSqr_64fc_I(Ipp64fc* pSrcDst, int len);
IppStatus ippsSqr_8u_ISfs(Ipp8u* pSrcDst, int len, int scaleFactor);
IppStatus ippsSqr_16s_ISfs(Ipp16s* pSrcDst, int len, int scaleFactor);
IppStatus ippsSqr_16u_ISfs(Ipp16u* pSrcDst, int len, int scaleFactor);
IppStatus ippsSqr_16sc_ISfs(Ipp16sc* pSrcDst, int len, int scaleFactor);
```

Platform-aware functions:

```
IppStatus ippsSqr_32f_L(const Ipp32f* pSrc, Ipp32f* pDst, IppSizeL len);
IppStatus ippsSqr_64f_L(const Ipp64f* pSrc, Ipp64f* pDst, IppSizeL len);
IppStatus ippsSqr_32fc_L(const Ipp32fc* pSrc, Ipp32fc* pDst, IppSizeL len);
IppStatus ippsSqr_64fc_L(const Ipp64fc* pSrc, Ipp64fc* pDst, IppSizeL len);
IppStatus ippsSqr_32f_I_L(Ipp32f* pSrcDst, IppSizeL len);
IppStatus ippsSqr_64f_I_L(Ipp64f* pSrcDst, IppSizeL len);
IppStatus ippsSqr_32fc_I_L(Ipp32fc* pSrcDst, IppSizeL len);
IppStatus ippsSqr_64fc_I_L(Ipp64fc* pSrcDst, IppSizeL len);
```

Include Files

`ipps.h`

Flavors with the `_L` suffix: `ipps_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pSrc`	Pointer to the source vector.
`pDst`	Pointer to the destination vector.
`pSrcDst`	Pointer to the source and destination vector for in-place operations.
`len`	Number of elements in the vector
`scaleFactor`	Scale factor, refer to Integer Scaling.

Description

This function computes the square of each element of the vector `pSrc`, and stores the result in `pDst`. The computation is performed as follows:

$pDst[n] = pSrc[n]^2$

The in-place flavors of `ippsSqr` compute the square of each element of the vector `pSrcDst` and store the result in `pSrcDst`. The computation is performed as follows:

$pSrcDst[n] = pSrcDst[n]^2$

When computing the square of an integer number, the output result can exceed the data range and become saturated. To get a precise result, use the scale factor.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when the `pSrc`, `pDst`, or `pSrcDst` pointer is `NULL`.
`ippStsSizeErr`	Indicates an error when `len` is less than or equal to zero.

Example

`Sqr`:

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
```

```
 * those that are expressly stated in the License.
 *******************************************************************************/

#include <stdio.h>
#include "ipp.h"

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define
check_sts(st)
                                  \
    if ((st) !=
ippStsNoErr)
                          \
        goto exitLine; /* Go to Exit if Intel(R) Integrated Performance Primitives (Intel(R)
IPP) function returned status different from \
                           ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main()
{
    int len = 32;
    int i;
    Ipp16s *pSrc = ippsMalloc_16s(len * sizeof(Ipp16s));
    Ipp16s *pDst = ippsMalloc_16s(len * sizeof(Ipp16s));
    IppStatus status;
    int scale = 0; // without scaling
    printf("\n\nSource vector\n");
    for (i = 0; i < len; i++) {
        pSrc[i] = 2 + i;
        printf("%d; ", pSrc[i]);
    }

    check_sts(status = ippsSqr_16s_Sfs(pSrc, pDst, len, scale));

    printf("\n\nResult\n");
    for (i = 0; i < len; i++)
        printf("%d; ", pDst[i]);
    printf("\n\n");

    EXIT_MAIN
    ippsFree(pSrc);
    ippsFree(pDst);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

Sqr_I:

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
```

```
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 ****************************************************************************/

#include <stdio.h>
#include "ipp.h"

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define
check_sts(st)
                                    \
    if ((st) !=
ippStsNoErr)
                           \
        goto exitLine; /* Go to Exit if Intel(R) Integrated Performance Primitives (Intel(R)
IPP) function returned status different from \
                            ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main()
{
    int len = 32;
    int i;
    Ipp16u *pSrcDst = ippsMalloc_16u(len * sizeof(Ipp16u));
    IppStatus status;
    int scale = 0; // without scaling
    printf("\n\nSource vector\n");
    for (i = 0; i < len; i++) {
        pSrcDst[i] = i;
        printf("%d; ", pSrcDst[i]);
    }

    check_sts(status = ippsSqr_16u_ISfs(pSrcDst, len, scale));

    printf("\n\nResult\n");
    for (i = 0; i < len; i++)
        printf("%d; ", pSrcDst[i]);
    printf("\n\n");

    EXIT_MAIN
    ippsFree(pSrcDst);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

Sqrt

Computes a square root of each element of a vector.

Syntax

```
IppStatus ippsSqrt_32f(const Ipp32f* pSrc, Ipp32f* pDst, int len);
IppStatus ippsSqrt_64f(const Ipp64f* pSrc, Ipp64f* pDst, int len);
IppStatus ippsSqrt_32fc(const Ipp32fc* pSrc, Ipp32fc* pDst, int len);
IppStatus ippsSqrt_64fc(const Ipp64fc* pSrc, Ipp64fc* pDst, int len);
IppStatus ippsSqrt_8u_Sfs(const Ipp8u* pSrc, Ipp8u* pDst, int len, int scaleFactor);
IppStatus ippsSqrt_16s_Sfs(const Ipp16s* pSrc, Ipp16s* pDst, int len, int scaleFactor);
IppStatus ippsSqrt_16u_Sfs(const Ipp16u* pSrc, Ipp16u* pDst, int len, int scaleFactor);
IppStatus ippsSqrt_16sc_Sfs(const Ipp16sc* pSrc, Ipp16sc* pDst, int len, int
scaleFactor);
IppStatus ippsSqrt_32s16s_Sfs(const Ipp32s* pSrc, Ipp16s* pDst, int len, int
scaleFactor);
IppStatus ippsSqrt_32f_I(Ipp32f* pSrcDst, int len);
IppStatus ippsSqrt_64f_I(Ipp64f* pSrcDst, int len);
IppStatus ippsSqrt_32fc_I(Ipp32fc* pSrcDst, int len);
IppStatus ippsSqrt_64fc_I(Ipp64fc* pSrcDst, int len);
IppStatus ippsSqrt_8u_ISfs(Ipp8u* pSrcDst, int len, int scaleFactor);
IppStatus ippsSqrt_16s_ISfs(Ipp16s* pSrcDst, int len, int scaleFactor);
IppStatus ippsSqrt_16u_ISfs(Ipp16u* pSrcDst, int len, int scaleFactor);
IppStatus ippsSqrt_16sc_ISfs(Ipp16sc* pSrcDst, int len, int scaleFactor);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pSrc`	Pointer to the source vector.
`pDst`	Pointer to the destination vector.
`pSrcDst`	Pointer to the source and destination vector for in-place operations.
`len`	Number of elements in the vector
`scaleFactor`	Scale factor, refer to Integer Scaling.

Description

This function computes the square root of each element of the vector `pSrc`, and stores the result in `pDst`. The computation is performed as follows:

`pDst[n]` = (`pSrc[n]`) $^{1/2}$

The in-place flavors of `ippsSqrt` compute the square root of each element of the vector `pSrcDst` and store the result in `pSrcDst`. The computation is performed as follows:

`pSrcDst[n]` = (`pSrcDst[n]`) $^{1/2}$.

The square root of complex vector elements is computed as follows:

$$\sqrt{a+j\cdot b} = \sqrt{\frac{\sqrt{a^2+b^2}+a}{2}} + j\cdot sign(b)\cdot \sqrt{\frac{\sqrt{a^2+b^2}-a}{2}}$$

If the function `ippsSqrt` encounters a negative value in the input, it returns a warning status and continues execution with the corresponding result value (see appendix A "Handling of Special Cases" for more information).

To increase precision of an integer output, use the scale factor.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when the `pSrc`, `pDst`, or `pSrcDst` pointer is `NULL`.
`ippStsSizeErr`	Indicates an error when `len` is less than or equal to 0.
`ippStsSqrtNegArg`	Indicates a warning that a source element has a negative value.

Example

`Sqrt`:

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

#include <stdio.h>
#include "ipp.h"

/* Next two defines are created to simplify code reading and understanding */
```

```
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define
check_sts(st)
                                  \
    if ((st) !=
ippStsNoErr)
                           \
        goto exitLine; /* Go to Exit if Intel(R) Integrated Performance Primitives (Intel(R)
IPP) function returned status different from \
                            ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main()
{
    int len = 32;
    int i;
    Ipp16s *pSrc = ippsMalloc_16s(len * sizeof(Ipp16s));
    Ipp16s *pDst = ippsMalloc_16s(len * sizeof(Ipp16s));
    IppStatus status;
    int scale = 0; // without scaling
    printf("\n\nSource vector\n");
    for (i = 0; i < len; i++) {
        pSrc[i] = 2 + i;
        printf("%d; ", pSrc[i]);
    }

    check_sts(status = ippsSqrt_16s_Sfs(pSrc, pDst, len, scale));

    printf("\n\nResult\n");
    for (i = 0; i < len; i++)
        printf("%d; ", pDst[i]);
    printf("\n\n");

    EXIT_MAIN
    ippsFree(pSrc);
    ippsFree(pDst);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

Sqrt_I:

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
```

```
 *****************************************************************************/

#include <stdio.h>
#include "ipp.h"

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define
check_sts(st)
                                  \
    if ((st) !=
ippStsNoErr)
                          \
        goto exitLine; /* Go to Exit if Intel(R) Integrated Performance Primitives (Intel(R)
IPP) function returned status different from \
                           ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main()
{
    int len = 32;
    int i;
    Ipp16u *pSrcDst = ippsMalloc_16u(len * sizeof(Ipp16u));
    IppStatus status;
    int scale = 0; // without scaling
    printf("\n\nSource vector\n");
    for (i = 0; i < len; i++) {
        pSrcDst[i] = i;
        printf("%d; ", pSrcDst[i]);
    }

    check_sts(status = ippsSqrt_16u_ISfs(pSrcDst, len, scale));

    printf("\n\nResult\n");
    for (i = 0; i < len; i++)
        printf("%d; ", pSrcDst[i]);
    printf("\n\n");

    EXIT_MAIN
    ippsFree(pSrcDst);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

Cubrt

Computes cube root of each element of a vector.

Syntax

```
IppStatus ippsCubrt_32f(const Ipp32f* pSrc, Ipp32f* pDst, int len);

IppStatus ippsCubrt_32s16s_Sfs(const Ipp32s* pSrc, Ipp16s* pDst, int len, int
scaleFactor);
```

Include Files

ipps.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

pSrc	Pointer to the source vector.
pDst	Pointer to the destination vector.
len	Number of elements in the vector
scaleFactor	Scale factor, refer to Integer Scaling.

Description

This function computes cube root of each element of pSrc and stores the result in the corresponding element of pDst.

The computation is performed as follows:

pDst[n] = (pSrc[n]) $^{1/3}$, 0≤n <len.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when the pDst or pSrc pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.

Exp

Computes e to the power of each element of a vector.

Syntax

```
IppStatus ippsExp_32f(const Ipp32f* pSrc, Ipp32f* pDst, int len);
IppStatus ippsExp_64f(const Ipp64f* pSrc, Ipp64f* pDst, int len);
IppStatus ippsExp_32f_I(Ipp32f* pSrcDst, int len);
IppStatus ippsExp_64f_I(Ipp64f* pSrcDst, int len);
IppStatus ippsExp_16s_Sfs(const Ipp16s* pSrc, Ipp16s* pDst, int len, int scaleFactor);
IppStatus ippsExp_32s_Sfs(const Ipp32s* pSrc, Ipp32s* pDst, int len, int scaleFactor);
IppStatus ippsExp_16s_ISfs(Ipp16s* pSrcDst, int len, int scaleFactor);
IppStatus ippsExp_32s_ISfs(Ipp32s* pSrcDst, int len, int scaleFactor);
```

Include Files

ipps.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

pSrc	Pointer to the source vector.
pDst	Pointer to the destination vector.
pSrcDst	Pointer to the source and destination vector pSrcDst for the in-place operation.
len	Number of elements in the vector
scaleFactor	Scale factor, refer to Integer Scaling.

Description

This function computes the exponential function of each element of the vector pSrc, and stores the result in pDst.

The computation is performed as follows:

$pDst[n] = e^{pSrc[n]}$

The in-place flavors of ippsExp compute the exponential function of each element of the vector pSrcDst and store the result in pSrcDst.

The computation is performed as follows:

$pSrcDst[n] = e^{pSrcDst[n]}$

When an overflow occurs, the function continues operation with the corresponding result value (see appendix A "Handling of Special Cases" for more information).

When computing the exponent of an integer number, the output result can exceed the data range and become saturated. The scaling retains the output data range but results in precision loss in low-order bits. The function ippsExp_32f64f computes the output result in a higher precision data range.

Application Notes

For the functions ippsExp and ippsLn the result is rounded to the nearest integer after scaling.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when the pSrc, pDst, or pSrcDst pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to zero.

Example

Exp:

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

#include <stdio.h>
#include "ipp.h"

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define
check_sts(st)
                                    \
    if ((st) !=
ippStsNoErr)
                           \
        goto exitLine; /* Go to Exit if Intel(R) Integrated Performance Primitives (Intel(R)
IPP) function returned status different from \
                            ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main()
{
    int len = 32;
    int i;
    Ipp16s *pSrc = ippsMalloc_16s(len * sizeof(Ipp16s));
    Ipp16s *pDst = ippsMalloc_16s(len * sizeof(Ipp16s));
    IppStatus status;
    int scale = 0; // without scaling
    printf("\n\nSource vector\n");
    for (i = 0; i < len; i++) {
        pSrc[i] = 2 + i;
        printf("%d; ", pSrc[i]);
    }

    check_sts(status = ippsExp_16s_Sfs(pSrc, pDst, len, scale));

    printf("\n\nResult\n");
    for (i = 0; i < len; i++)
        printf("%d; ", pDst[i]);
    printf("\n\n");
```

```
    EXIT_MAIN
    ippsFree(pSrc);
    ippsFree(pDst);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

Exp_I:

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

#include <stdio.h>
#include "ipp.h"

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define
check_sts(st)
                                  \
    if ((st) !=
ippStsNoErr)
                          \
        goto exitLine; /* Go to Exit if Intel(R) Integrated Performance Primitives (Intel(R)
IPP) function returned status different from \
                           ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main()
{
    int len = 32;
    int i;
    Ipp16s *pSrcDst = ippsMalloc_16s(len * sizeof(Ipp16s));
    IppStatus status;
    int scale = 0; // without scaling
    printf("\n\nSource vector\n");
    for (i = 0; i < len; i++) {
        pSrcDst[i] = 2 + i;
        printf("%d; ", pSrcDst[i]);
    }

    check_sts(status = ippsExp_16s_ISfs(pSrcDst, len, scale));
```

```
    printf("\n\nResult\n");
    for (i = 0; i < len; i++)
        printf("%d; ", pSrcDst[i]);
    printf("\n\n");

    EXIT_MAIN
    ippsFree(pSrcDst);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

Ln

Computes the natural logarithm of each element of a vector.

Syntax

```
IppStatus ippsLn_32f(const Ipp32f* pSrc, Ipp32f* pDst, int len);
IppStatus ippsLn_64f(const Ipp64f* pSrc, Ipp64f* pDst, int len);
IppStatus ippsLn_16s_Sfs(const Ipp16s* pSrc, Ipp16s* pDst, int len, int scaleFactor);
IppStatus ippsLn_32s_Sfs(const Ipp32s* pSrc, Ipp32s* pDst, int len, int scaleFactor);
IppStatus ippsLn_16s_ISfs(Ipp16s* pSrcDst, int len, int scaleFactor);
IppStatus ippsLn_32s_ISfs(Ipp32s* pSrcDst, int len, int scaleFactor);
IppStatus ippsLn_32f_I(Ipp32f* pSrcDst, int len);
IppStatus ippsLn_64f_I(Ipp64f* pSrcDst, int len);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pSrc`	Pointer to the source vector.
`pDst`	Pointer to the destination vector.
`pSrcDst`	Pointer to the source and destination vector for the in-place operation.
`len`	Number of elements in the vector
`scaleFactor`	Scale factor, refer to Integer Scaling.

Description

This function computes the natural logarithm of each element of the vector `pSrc` and stores the result in `pDst` as given by

$pDst[n] = \log_e(pSrc[n])$

The in-place flavors of `ippsLn` compute the natural logarithm of each element of the vector `pSrcDst` and store the result in `pSrcDst` as given by

$pSrcDst[n] = \log_e(pSrcDst[n])$

If the function `ippsLn` encounters a zero or negative value in the input, it returns a warning status and continues execution with the corresponding result value (see appendix A "Handling of Special Cases" for more information).

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when the `pSrc`, `pDst`, or `pSrcDst` pointer is `NULL`.
`ippStsSizeErr`	Indicates an error when `len` is less than or equal to zero.
`ippStsLnZeroArg`	Indicates a warning for zero-valued input vector elements.
`ippStsLnNegArg`	Indicates a warning for negative input vector elements.

Example

`Ln`:

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

#include <stdio.h>
#include "ipp.h"

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define
check_sts(st)
```

```
                                    \
    if ((st) !=
ippStsNoErr)
                            \
        goto exitLine; /* Go to Exit if Intel(R) Integrated Performance Primitives (Intel(R)
IPP) function returned status different from \
                           ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main()
{
    int len = 32;
    int i;
    Ipp16s *pSrc = ippsMalloc_16s(len * sizeof(Ipp16s));
    Ipp16s *pDst = ippsMalloc_16s(len * sizeof(Ipp16s));
    IppStatus status;
    int scale = 0; // without scaling
    printf("\n\nSource vector\n");
    for (i = 0; i < len; i++) {
        pSrc[i] = 2 + i;
        printf("%d; ", pSrc[i]);
    }

    check_sts(status = ippsLn_16s_Sfs(pSrc, pDst, len, scale));

    printf("\n\nResult\n");
    for (i = 0; i < len; i++)
        printf("%d; ", pDst[i]);
    printf("\n\n");

    EXIT_MAIN
    ippsFree(pSrc);
    ippsFree(pDst);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

Ln_I:

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

#include <stdio.h>
#include "ipp.h"
```

```
/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define
check_sts(st)
                                  \
    if ((st) !=
ippStsNoErr)
                           \
        goto exitLine; /* Go to Exit if Intel(R) Integrated Performance Primitives (Intel(R)
IPP) function returned status different from \
                            ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main()
{
    int len = 32;
    int i;
    Ipp16s *pSrcDst = ippsMalloc_16s(len * sizeof(Ipp16s));
    IppStatus status;
    int scale = 0; // without scaling
    printf("\n\nSource vector\n");
    for (i = 0; i < len; i++) {
        pSrcDst[i] = 2 + i;
        printf("%d; ", pSrcDst[i]);
    }

    check_sts(status = ippsLn_16s_ISfs(pSrcDst, len, scale));

    printf("\n\nResult\n");
    for (i = 0; i < len; i++)
        printf("%d; ", pSrcDst[i]);
    printf("\n\n");

    EXIT_MAIN
    ippsFree(pSrcDst);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

SumLn

Sums natural logarithms of each element of a vector.

Syntax

```
IppStatus ippsSumLn_32f(const Ipp32f* pSrc, int len, Ipp32f* pSum);
IppStatus ippsSumLn_64f(const Ipp64f* pSrc, int len, Ipp64f* pSum);
IppStatus ippsSumLn_32f64f(const Ipp32f* pSrc, int len, Ipp64f* pSum);
IppStatus ippsSumLn_16s32f(const Ipp16s* pSrc, int len, Ipp32f* pSum);
```

Include Files

ipps.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

pSrc	Pointer to the source vector.
pSum	Pointer to the output result.
len	Number of elements in the vector.

Description

This function computes the sum of natural logarithms of each element of the vector pSrc and stores the result value in pSum. The summation is given by:

$$sum = \sum_{n=0}^{len-1} \ln(pSrc[n])$$

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when the pSrc or pSum pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.
ippStsLnZeroArg	Indicates a warning for zero-valued input vector elements. Operation execution is not aborted. The value of the destination vector element for floating-point operations is set to -Inf.
ippStsLnNegArg	Indicates a warning for negative input vector elements. Operation execution is not aborted. The value of the destination vector element for floating-point operations is set to NaN.

Arctan

Computes the inverse tangent of each element of a vector.

Syntax

```
IppStatus ippsArctan_32f(const Ipp32f* pSrc, Ipp32f* pDst, int len);
```

```
IppStatus ippsArctan_64f(const Ipp64f* pSrc, Ipp64f* pDst, int len);
IppStatus ippsArctan_32f_I(Ipp32f* pSrcDst, int len);
IppStatus ippsArctan_64f_I(Ipp64f* pSrcDst, int len);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pSrc`	Pointer to the source vector.
`pDst`	Pointer to the destination vector.
`pSrcDst`	Pointer to the source and destination vector `pSrcDst` for the in-place operation.
`len`	Number of elements in the vector.

Description

This function computes the inverse tangent of each element of `pSrc` and stores the result in the corresponding element of `pDst`.

The computation is performed as follows:

`pDst[n] = arctan(pSrc[n]),0≤n <len.`

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when the `pSrc`, `pDst`, or `pSrcDst` pointer is `NULL`.
`ippStsSizeErr`	Indicates an error when `len` is less than or equal to 0.

Normalize

Normalizes elements of a real or complex vector using offset and division operations.

Syntax

Case 1: Not-in-place operations on floating point and integer data

```
IppStatus ippsNormalize_32f(const Ipp32f* pSrc, Ipp32f* pDst, int len, Ipp32f vSub,
Ipp32f vDiv);
IppStatus ippsNormalize_64f(const Ipp64f* pSrc, Ipp64f* pDst, int len, Ipp64f vSub,
Ipp64f vDiv);
```

```
IppStatus ippsNormalize_32fc(const Ipp32fc* pSrc, Ipp32fc* pDst, int len, Ipp32fc vSub,
Ipp32f vDiv);

IppStatus ippsNormalize_64fc(const Ipp64fc* pSrc, Ipp64fc* pDst, int len, Ipp64fc vSub,
Ipp64f vDiv);

IppStatus ippsNormalize_16s_Sfs(const Ipp16s* pSrc, Ipp16s* pDst, int len, Ipp16s vSub,
int vDiv, int scaleFactor);

IppStatus ippsNormalize_16sc_Sfs(const Ipp16sc* pSrc, Ipp16sc* pDst, int len, Ipp16sc
vSub, int vDiv, int scaleFactor);
```

Case 2: In-place operations on floating point and integer data

```
IppStatus ippsNormalize_32f_I(Ipp32f* pSrcDst, int len, Ipp32f vSub, Ipp32f vDiv);

IppStatus ippsNormalize_64f_I(Ipp64f* pSrcDst, int len, Ipp64f vSub, Ipp64f vDiv);

IppStatus ippsNormalize_32fc_I(Ipp32fc* pSrcDst, int len, Ipp32fc vSub, Ipp32f vDiv);

IppStatus ippsNormalize_64fc_I(Ipp64fc* pSrcDst, int len, Ipp64fc vSub, Ipp64f vDiv);

IppStatus ippsNormalize_16s_ISfs(Ipp16s* pSrcDst, int len, Ipp16s vSub, int vDiv, int
scaleFactor);

IppStatus ippsNormalize_16sc_ISfs(Ipp16sc* pSrcDst, int len, Ipp16sc vSub, int vDiv,
int scaleFactor);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pSrc`	Pointer to the source vector.
`pSrcDst`	Pointer to the source and destination vector for in-place operations.
`vSub`	Subtrahend value.
`vDiv`	Denominator value.
`pDst`	Pointer to the vector which stores the normalized elements.
`len`	Number of elements in the vector
`scaleFactor`	Scale factor, refer to Integer Scaling.

Description

This function subtracts `vSub` from elements of the input vector `pSrc` (`pSrcDst` for in-place operations), divides the differences by `vDiv`, and stores the result in `pDst` (`pSrcDst` for in-place operations). The computation is performed as follows:

pDst[n] = (pSrc[n] -vSub)/vDiv.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when the pSrc or pDst pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.
ippStsDivByZeroErr	Indicates an error when vDiv is equal to 0 or less than the minimum floating-point positive number.

Conversion Functions

The functions described in this section perform the following conversion operations for vectors:

- Sorting all elements of a vector
- Data type conversion (including floating-point to integer and integer to floating-point)
- Joining several vectors
- Extracting components from a complex vector and constructing a complex vector
- Computing the complex conjugates of vectors
- Cartesian to polar and polar to Cartesian coordinate conversion.

This section also describes the Intel IPP functions that extract real and imaginary components from a complex vector or construct a complex vector using its real and imaginary components. The functions ippsReal and ippsImag return the real and imaginary parts of a complex vector in a separate vector, respectively. The function ippsRealToCplx constructs a complex vector from real and imaginary components stored in two respective vectors. The function ippsCplxToReal returns the real and imaginary parts of a complex vector in two respective vectors. The function ippsMagnitude computes the magnitude of a complex vector elements.

- SortAscend, SortDescend
- SortIndexAscend, SortIndexDescend
- SortRadixGetBufferSize
- SortRadixAscend, SortRadixDescend
- SortRadixIndexGetBufferSize
- SortRadixIndexAscend, SortRadixIndexDescend
- TopKGetBufferSize
- TopKInit
- TopK
- SwapBytes
- Convert
- Conj
- ConjFlip
- Magnitude
- Phase
- PowerSpectr
- Real
- Imag
- RealToCplx
- CplxToReal
- Threshold
- Threshold_LT, Threshold_GT
- Threshold_LTAbs, Threshold_GTAbs
- Threshold_LTVal, Threshold_LTAbsVal, Threshold_GTVal, Threshold_LTValGTVal
- Threshold_LTInv

- CartToPolar
- PolarToCart
- MaxOrder
- Flip
- FindNearestOne
- FindNearest

SortAscend, SortDescend

Sorts all elements of a vector.

Syntax

```
IppStatus ippsSortAscend_8u_I(Ipp8u* pSrcDst, int len);
IppStatus ippsSortAscend_16u_I(Ipp16u* pSrcDst, int len);
IppStatus ippsSortAscend_16s_I(Ipp16s* pSrcDst, int len);
IppStatus ippsSortAscend_32s_I(Ipp32s* pSrcDst, int len);
IppStatus ippsSortAscend_32f_I(Ipp32f* pSrcDst, int len);
IppStatus ippsSortAscend_64f_I(Ipp64f* pSrcDst, int len);
IppStatus ippsSortDescend_8u_I(Ipp8u* pSrcDst, int len);
IppStatus ippsSortDescend_16u_I(Ipp16u* pSrcDst, int len);
IppStatus ippsSortDescend_16s_I(Ipp16s* pSrcDst, int len);
IppStatus ippsSortDescend_32s_I(Ipp32s* pSrcDst, int len);
IppStatus ippsSortDescend_32f_I(Ipp32f* pSrcDst, int len);
IppStatus ippsSortDescend_64f_I(Ipp64f* pSrcDst, int len);
```

Platform-aware functions:

```
IppStatus ippsSortAscend_32f_I_L(Ipp32f* pSrcDst, IppSizeL len);
IppStatus ippsSortAscend_64f_I_L(Ipp64f* pSrcDst, IppSizeL len);
IppStatus ippsSortDescend_32f_I_L(Ipp32f* pSrcDst, IppSizeL len);
IppStatus ippsSortDescend_64f_I_L(Ipp64f* pSrcDst, IppSizeL len);
```

Include Files

`ipps.h`

Flavors with the `_L` suffix: `ipps_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pSrcDst`	Pointer to the source and destination vector.
`len`	Number of elements in the vector

Description

These functions rearrange all elements of the source vector `pSrcDst` in the ascending or descending order, respectively, and store the result in the destination vector `pSrcDst`.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when the `pSrcDst` is `NULL`.
`ippStsSizeErr`	Indicates an error when `len` is less than or equal to zero.

Example

`SortAscend`:

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

#include <stdio.h>
#include "ipp.h"

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define
check_sts(st)
                                    \
    if ((st) !=
ippStsNoErr)
                          \
        goto exitLine; /* Go to Exit if Intel(R) Integrated Performance Primitives (Intel(R)
IPP) function returned status different from \
                           ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main()
{
    Ipp16u pSrc[] = {5, 7, 2, 1, 6, 8, 9, 0, 4, 3};
    IppStatus status;
    int i;
```

```
    printf("\nSource vector\n");
    for (i = 0; i < 10; i++)
        printf("%d ", pSrc[i]);

    check_sts(status = ippsSortAscend_16u_I(pSrc, 10));

    printf("\nResult values vector\n");
    for (i = 0; i < 10; i++)
        printf("%d ", pSrc[i]);

    EXIT_MAIN
    printf("\nExit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

SortDescend:

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

#include <stdio.h>
#include "ipp.h"

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define
check_sts(st)
                                    \
    if ((st) !=
ippStsNoErr)
                           \
        goto exitLine; /* Go to Exit if Intel(R) Integrated Performance Primitives (Intel(R)
IPP) function returned status different from \
                           ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main()
{
    Ipp16u pSrc[] = {5, 7, 2, 1, 6, 8, 9, 0, 4, 3};
    IppStatus status;
    int i;

    printf("\nSource vector\n");
```

```
    for (i = 0; i < 10; i++)
        printf("%d ", pSrc[i]);

    check_sts(status = ippsSortDescend_16u_I(pSrc, 10));

    printf("\nResult values vector\n");
    for (i = 0; i < 10; i++)
        printf("%d ", pSrc[i]);

    EXIT_MAIN
    printf("\nExit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

SortIndexAscend, SortIndexDescend

Rearranges elements of the vector and their indexes.

Syntax

```
IppStatus ippsSortIndexAscend_8u_I(Ipp8u* pSrcDst, int* pDstIdx, int len);
IppStatus ippsSortIndexAscend_16u_I(Ipp16u* pSrcDst, int* pDstIdx, int len);
IppStatus ippsSortIndexAscend_16s_I(Ipp16s* pSrcDst, int* pDstIdx, int len);
IppStatus ippsSortIndexAscend_32s_I(Ipp32s* pSrcDst, int* pDstIdx, int len);
IppStatus ippsSortIndexAscend_32f_I(Ipp32f* pSrcDst, int* pDstIdx, int len);
IppStatus ippsSortIndexAscend_64f_I(Ipp64f* pSrcDst, int* pDstIdx, int len);
IppStatus ippsSortIndexDescend_8u_I(Ipp8u* pSrcDst, int* pDstIdx, int len);
IppStatus ippsSortIndexDescend_16u_I(Ipp16u* pSrcDst, int* pDstIdx, int len);
IppStatus ippsSortIndexDescend_16s_I(Ipp16s* pSrcDst, int* pDstIdx, int len);
IppStatus ippsSortIndexDescend_32s_I(Ipp32s* pSrcDst, int* pDstIdx, int len);
IppStatus ippsSortIndexDescend_32f_I(Ipp32f* pSrcDst, int* pDstIdx, int len);
IppStatus ippsSortIndexDescend_64f_I(Ipp64f* pSrcDst, int* pDstIdx, int len);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pSrcDst`	Pointer to the source and destination vector.
`pDstIdx`	Pointer to the destination vector containing indexes.
`len`	Number of elements in the vector

Description

These functions rearrange all elements of the source vector `pSrcDst` in the ascending or descending order, respectively, and store the elements in the destination vector `pSrcDst`, and their indexes in the desalination vector `pDstIdx`. If some elements are identical, their indexes are not ordered.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error when `len` is less than or equal to 0.

SortRadixGetBufferSize

Computes the size of the buffer for the `SortRadixAscend` and `SortRadixDescend` functions.

Syntax

```
IppStatus ippsSortRadixGetBufferSize(int len, IppDataType dataType, int* pBuffferSize);
IppStatus ippsSortRadixGetBufferSize_L(IppSizeL len, IppDataType dataType, IppSizeL* pBufferSize);
```

Include Files

`ipps.h`

Flavors with the `_L` suffix: `ipps_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`len`	Number of elements in the vector
`dataType`	Data type of the vector.
`pBufferSize`	Pointer to the buffer size.

Description

This function calculates the size of the buffer for the **ippsSortRadixAscend`/:ref:`ippsSortRadixDescend <c_sortradixascend-sortradixdescend>** functions.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when `pBufSize` is `NULL`.

ippStsSizeErr	Indicates an error when len is less than, or equal to 0.
ippStsDataTypeErr	Indicates an error when the dataType value is not supported.

See Also

SortRadixAscend SortRadixDescend Sorts all elements of a vector using radix sorting algorithm.

SortRadixAscend, SortRadixDescend

Sorts all elements of a vector using radix sorting algorithm.

Syntax

```
IppStatus ippsSortRadixAscend_<mod>(Ipp<datatype>* pSrcDst, int len, Ipp8u* pBuffer);
```

Supported values for mod:

8u_I	32u_I	64u_I
16u_I	32s_I	64s_I
16s_I	32f_I	64f_I

```
IppStatus ippsSortRadixDescend_<mod>(Ipp<datatype>* pSrcDst, int len, Ipp8u* pBuffer);
```

Supported values for mod:

8u_I	32u_I	64u_I
16u_I	32s_I	64s_I
16s_I	32f_I	64f_I

Radix Sorting Algorithm for platform-aware functions

```
IppStatus ippsSortRadixAscend_<mod>(Ipp<datatype>* pSrcDst, IppSizeL len, Ipp8u*
pBuffer);
```

Supported values for mod:

	64u_I_L
32s_I_L	64s_I_L
32f_I_L	64f_I_L

```
IppStatus ippsSortRadixDescend_<mod>(Ipp<datatype>* pSrcDst, IppSizeL len, Ipp8u*
pBuffer);
```

Supported values for mod:

	64u_I_L
32s_I_L	64s_I_L
32f_I_L	64f_I_L

Include Files

ipps.h

Flavors with the _L suffix: ipps_l.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

pSrcDst	Pointer to the source and destination vector.
len	Number of elements in the vector
pBuffer	Pointer to the buffer for internal calculations. To compute the required buffer size, use the SortRadixGetBufferSize function.

Description

These functions rearrange all elements of the source vector pSrcDst in the ascending or descending order, respectively, using "radix sort" algorithm, and store the result in the destination vector pSrcDst.

Flavors with the _L suffix operate on larger data size.

These functions require the work buffer for internal calculations, to compute the size of the buffer, use the SortRadixGetBufferSize or SortRadixGetBufferSize_L (for the flavors with the _L suffix) function.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pSrcDst or pBuffer is NULL.
ippStsSizeErr	Indicates an error when len is less than, or equal to 0.

See Also

SortRadixGetBufferSize Computes the size of the buffer for the SortRadixAscend and SortRadixDescend functions.

SortRadixIndexGetBufferSize

Computes the size of the buffer for the SortRadixIndexAscend and SortRadixIndexDescend functions.

Syntax

```
IppStatus ippsSortRadixIndexGetBufferSize(int len, IppDataType dataType, int*
pBufSize);
```

```
IppStatus ippsSortRadixIndexGetBufferSize_L(IppSizeL len, IppDataType dataType,
IppSizeL* pBufSize);
```

Include Files

`ipps.h`

Flavors with the `_L` suffix: `ipps_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`len`	Number of elements in the vector
`dataType`	Data type pf the vector.
`pBufSize`	Pointer to the buffer size.

Description

This function calculates the size of the buffer for the **ippsSortRadixIndexAscend`/:ref:`ippsSortRadixIndexDescend <c_sortradixindexascend-sortradixindexdescend>** functions.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when `pBufSize` is `NULL`.
`ippStsSizeErr`	Indicates an error when `len` is less than, or equal to 0.
`ippStsDataTypeErr`	Indicates an error when the `dataType` value is not supported.

See Also

SortRadixIndexAscend SortRadixIndexDescend Indirectly sorts all elements of a vector using radix sorting algorithm.

SortRadixIndexAscend, SortRadixIndexDescend

Indirectly sorts all elements of a vector using radix sorting algorithm.

Syntax

```
IppStatus ippsSortRadixIndexAscend_8u(const Ipp8u* pSrc, Ipp32s srcStrideBytes, Ipp32s* pDstIndx, int len, Ipp8u* pBuffer);
```

```
IppStatus ippsSortRadixIndexAscend_16u(const Ipp16u* pSrc, Ipp32s srcStrideBytes, Ipp32s* pDstIndx, int len, Ipp8u* pBuffer);
```

```
IppStatus ippsSortRadixIndexAscend_16s(const Ipp16s* pSrc, Ipp32s srcStrideBytes, Ipp32s* pDstIndx, int len, Ipp8u* pBuffer);
```

```
IppStatus ippsSortRadixIndexAscend_32s(const Ipp32s* pSrc, Ipp32s srcStrideBytes, Ipp32s* pDstIndx, int len, Ipp8u* pBuffer);
```

```
IppStatus ippsSortRadixIndexAscend_32u(const Ipp32u* pSrc, Ipp32s srcStrideBytes,
Ipp32s* pDstIndx, int len, Ipp8u* pBuffer);

IppStatus ippsSortRadixIndexAscend_32f(const Ipp32f* pSrc, Ipp32s srcStrideBytes,
Ipp32s* pDstIndx, int len, Ipp8u* pBuffer);

IppStatus ippsSortRadixIndexAscend_64f(const Ipp64f* pSrc, Ipp32s srcStrideBytes,
Ipp32s* pDstIndx, int len, Ipp8u* pBuffer);

IppStatus ippsSortRadixIndexAscend_64s(const Ipp64s* pSrc, Ipp32s srcStrideBytes,
Ipp32s* pDstIndx, int len, Ipp8u* pBuffer);

IppStatus ippsSortRadixIndexAscend_64u(const Ipp64u* pSrc, Ipp32s srcStrideBytes,
Ipp32s* pDstIndx, int len, Ipp8u* pBuffer);

IppStatus ippsSortRadixIndexDescend_8u(const Ipp8u* pSrc, Ipp32s srcStrideBytes,
Ipp32s* pDstIndx, int len, Ipp8u* pBuffer);

IppStatus ippsSortRadixIndexDescend_16u(const Ipp16u* pSrc, Ipp32s srcStrideBytes,
Ipp32s* pDstIndx, int len, Ipp8u* pBuffer);

IppStatus ippsSortRadixIndexDescend_16s(const Ipp16s* pSrc, Ipp32s srcStrideBytes,
Ipp32s* pDstIndx, int len, Ipp8u* pBuffer);

IppStatus ippsSortRadixIndexDescend_32s(const Ipp32s* pSrc, Ipp32s srcStrideBytes,
Ipp32s* pDstIndx, int len, Ipp8u* pBuffer);

IppStatus ippsSortRadixIndexDescend_32u(const Ipp32u* pSrc, Ipp32s srcStrideBytes,
Ipp32s* pDstIndx, int len, Ipp8u* pBuffer);

IppStatus ippsSortRadixIndexDescend_32f(const Ipp32f* pSrc, Ipp32s srcStrideBytes,
Ipp32s* pDstIndx, int len, Ipp8u* pBuffer);

IppStatus ippsSortRadixIndexDescend_64f(const Ipp64f* pSrc, Ipp32s srcStrideBytes,
Ipp32s* pDstIndx, int len, Ipp8u* pBuffer);

IppStatus ippsSortRadixIndexDescend_64s(const Ipp64s* pSrc, Ipp32s srcStrideBytes,
Ipp32s* pDstIndx, int len, Ipp8u* pBuffer);

IppStatus ippsSortRadixIndexDescend_64u(const Ipp64u* pSrc, Ipp32s srcStrideBytes,
Ipp32s* pDstIndx, int len, Ipp8u* pBuffer);
```

Radix Sorting Algorithm for platform-aware functions

```
IppStatus ippsSortRadixIndexAscend_64s_L(const Ipp64s* pSrc, IppSizeL srcStrideBytes,
IppSizeL* pDstIndx, IppSizeL len, Ipp8u* pBuffer);

IppStatus ippsSortRadixIndexAscend_64u_L(const Ipp64u* pSrc, IppSizeL srcStrideBytes,
IppSizeL* pDstIndx, IppSizeL len, Ipp8u* pBuffer);

IppStatus ippsSortRadixIndexDescend_64s_L(const Ipp64s* pSrc, IppSizeL srcStrideBytes,
IppSizeL* pDstIndx, IppSizeL len, Ipp8u* pBuffer);

IppStatus ippsSortRadixIndexDescend_64u_L(const Ipp64u* pSrc, IppSizeL srcStrideBytes,
IppSizeL* pDstIndx, IppSizeL len, Ipp8u* pBuffer);
```

Include Files

`ipps.h`

Flavors with the `_L` suffix: `ipps_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pSrc`	Pointer the source sparse keys vector.
`srcStrideBytes`	Distance in bytes between two consecutive elements of the source vector.
`pDstIndx`	Pointer to the destination vector of indexes.
`len`	Number of elements in the vectors.
`pBuffer`	Pointer to the work buffer for internal calculations. To compute the size of the buffer, use the SortRadixIndexGetBufferSize function.

Description

These functions indirectly sort all elements of the source sparse keys vector `pSrc` in the ascending or descending order, respectively, using “radix sort” algorithm and store the indexes of resulting arrangement order in the destination vector `pDstIndx`. Elements of the source vector are not rearranged.

These functions require the work buffer for internal calculations, to compute the size of the required buffer, use the SortRadixIndexGetBufferSize function. Intervals between the elements of the source sparse vector `pSrc` in memory must be equal to the value of `srcStrideBytes`, minimum value of which is equal to the size of the data type of the key value. The sorting algorithm does not change the relative order of the elements with equal keys.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when the `pSrc` or `pBuffer` is `NULL`.
`ippStsSizeErr`	Indicates an error when `len` is less than or equal to zero, or `srcStrideBytes` is less than `sizeof(key type)`.

See Also

SortRadixIndexGetBufferSize Computes the size of the buffer for the SortRadixIndexAscend and SortRadixIndexDescend functions.

TopKGetBufferSize

Computes the size of the buffer for the `TopK` function.

Syntax

```
IppStatus ippsTopKGetBufferSize(Ipp64s srcLen, Ipp64s dstLen, IppDataType dataType,
IppTopKMode hint, Ipp64s* bufSize);
```

Include Files

ipps.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

srcLen	Number of elements in the source vector.
dstLen	Number of K values to be returned, the function returns min(K, dstLen) elements.
dataType	Data type of the vector.
hint	Parameter to choose the optimization that is most suitable for the srcLen+dstlen(K) combination, supported values: ippTopKAuto/ ippTopKDirect/ ippTopKRadix.
bufSize	Size of the required work buffer.

Description

This function computes the size of the work buffer required for the ippsTopK function.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when bufSize is NULL.
ippStsSizeErr	Indicates an error when at least one of the srcLen or dstLen values is less than, or equal to zero.
ippStsBadArgErr	Indicates an error when the hint value is not supported.

See Also

TopK Returns maximum K values of an array.

TopKInit

Initializes pDstValue and pDstIndex arrays for the ippsTopK function.

Syntax

```
IppStatus ippsTopKInit_32s(Ipp32s* pDstValue, Ipp64s* pDstIndex, Ipp64s dstLen);
```

Include Files

ipps.h

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pDstValue`	Pointer to an array with maximum values found by the `TopK` function.
`pDstIndex`	Pointer to an array with indexes of maximum values found by the `TopK` function.
`dstLen`	Number of `K` values to be returned, the function returns `min(K, dstLen)` elements.

Description

This function initializes the `pDstValue` array with minimal values and `pDstIndex` with -1 value.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one of the pointers is `NULL`.
`ippStsSizeErr`	Indicates an error when at least one of the `dstLen` values is less than, or equal to zero.

See Also

TopK Returns maximum K values of an array.

TopK

Returns maximum `K` values of an array.

Syntax

```
IppStatus ippsTopK_32f(const Ipp32f* pSrc, Ipp64s srcIndex, Ipp64s srcStride, Ipp64s srcLen, Ipp32f* pDstValue, Ipp64s* pDstIndex, Ipp64s dstLen, IppTopKMode hint, Ipp8u* pBuffer);
```

```
IppStatus ippsTopK_32s(const Ipp32s* pSrc, Ipp64s srcIndex, Ipp64s srcStride, Ipp64s srcLen, Ipp32s* pDstValue, Ipp64s* pDstIndex, Ipp64s dstLen, IppTopKMode hint, Ipp8u* pBuffer);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

pSrc	Pointer to the source vector.
srcIndex	Index of the pSrc[0] element, index of pSrc[n] is equal to srcIndex+n.
srcStride	The stride between elements in a source array. The unit of srcStride is Byte.
srcLen	Number of elements in the source vector.
pDstValue	Maximum values found by the function.
dstIndex	Indexes of maximum values.
dstLen	Number of K values to be returned, the function returns min(K, dstLen) elements.
hint	Parameter to choose the optimization that is most suitable for the srcLen+dstlen(K) combination, supported values: ippTopKAuto/ ippTopKDirect/ ippTopKRadix.
pBuffer	Pointer to the work buffer.

Description

This function searches for dstLen maximumvalues and their indexes in an input vector. The function isdesigned to process large input vectors by small blocks, it takesinto account results of previous blocks processing getting maximumvalues of pSrc and pDstValue and then combining the final results into pDstValue.

The srcIndex parameter stores the index of thefirst element pSrc[0] of each new block, thus supporting thecontinuous numbering of elements. Before calling the ippsTopK function, compute the required buffer size and initialize pDstValue and pDstIndex using the ippsTopKGetBufferSize and ippsTopKInit_32s functions, respectively.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one of the pointers is NULL.
ippStsSizeErr	Indicates an error when at least one of the srcLen or dstLen values is less than, or equal to zero.
ippStsBadArgErr	Indicates an error when the hint value is not supported.

Example

```
/*******************************************************************************
 * Copyright (C) 2023 Intel Corporation.
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
```

```
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *
 *****************************************************************************/

#include <stdio.h>
#include "ipp.h"

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define check_sts(st)        \
    if ((st) != ippStsNoErr) \
        goto exitLine; /* Go to Exit if Intel® Integrated Primitives (Intel® IPP) function
returned status different from ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel® IPP functions perform bad arguments
check and will return an appropriate status  */

int main()
{
    Ipp32f src[10] = {2.5, 1.0, 3.2, 4.7, 0.8, 5.6, 6.1, 3.9, 2.3, 4.0};
    Ipp32f dst[5];
    Ipp64s idx[5];
    IppStatus status;
    int i;

    printf("\nSource vector\n");
    for (i = 0; i < 10; i++)
        printf("%f ", src[i]);

    Ipp64s buf_size;
    ippsTopKGetBufferSize(10, 5, ipp32f, ippTopKAuto, &buf_size);

    Ipp8u buf[buf_size];
    ippsTopKInit_32f(dst, idx, 5);

    check_sts(status = ippsTopK_32f(src, 0, 4, 10, dst, idx, 5, ippTopKAuto, buf));

    printf("\nTop k values:\n");
    for (i = 0; i < 5; i++)
        printf("%f ", dst[i]);

    EXIT_MAIN
    printf("\nExit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

See Also

TopKGetBufferSize Computes the size of the buffer for the TopK function.

TopKInit Initializes pDstValue and pDstIndex arrays for the ippsTopK function.

SwapBytes

Reverses the byte order of a vector.

Syntax

```
IppStatus ippsSwapBytes_16u(const Ipp16u* pSrc, Ipp16u* pDst, int len);
IppStatus ippsSwapBytes_24u(const Ipp8u* pSrc, Ipp8u* pDst, int len);
IppStatus ippsSwapBytes_32u(const Ipp32u* pSrc, Ipp32u* pDst, int len);
IppStatus ippsSwapBytes_64u(const Ipp64u* pSrc, Ipp64u* pDst, int len);
IppStatus ippsSwapBytes_16u_I(Ipp16u* pSrcDst, int len);
IppStatus ippsSwapBytes_24u_I(Ipp8u* pSrcDst, int len);
IppStatus ippsSwapBytes_32u_I(Ipp32u* pSrcDst, int len);
IppStatus ippsSwapBytes_64u_I(Ipp64u* pSrcDst, int len);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pSrc`	Pointer to the source vector.
`pDst`	Pointer to the destination vector.
`pSrcDst`	Pointer to the source and destination vector for the in-place operation.
`len`	Number of elements in the vector.

Description

This function reverses the endian order (byte order) of the source vector `pSrc(pSrcDst``for the in-place operation) and stores the result in ``pDst(pSrcDst)`. When the low-order byte is stored in memory at the lowest address, and the high-order byte at the highest address, the little-endian order is implemented.When the high-order byte is stored in memory at the lowest address, and the low-order byte at the highest address, the big-endian order is implemented. The function `ippsSwapBytes` allows to switch from one order to the other in either direction.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when any of the specified pointers is `NULL`.

ippStsSizeErr	Indicates an error when `len` is less than or equal to zero.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

#include <stdio.h>
#include "ipp.h"

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define
check_sts(st)
                                  \
    if ((st) !=
ippStsNoErr)
                          \
        goto exitLine; /* Go to Exit if Intel(R) Integrated Performance Primitives (Intel(R)
IPP) function returned status different from \
                           ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main()
{
    Ipp16u vec[2] = {0x1234, 0x5678};
    IppStatus status;

    printf("Source vector %X %X\n", vec[0], vec[1]);
    check_sts(status = ippsSwapBytes_16u_I(vec, 2));
    printf("After swap \nResult vector %X %X\n", vec[0], vec[1]);

    EXIT_MAIN
    printf("\nExit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

Convert

Converts the data type of a vector and stores the results in a second vector.

Syntax

```
IppStatus ippsConvert_8s16s(const Ipp8s* pSrc,Ipp16s* pDst, int len);

IppStatus ippsConvert_8s32f(const Ipp8s* pSrc, Ipp32f* pDst, int len);

IppStatus ippsConvert_8u32f(const Ipp8u* pSrc, Ipp32f* pDst, int len);

IppStatus ippsConvert_8u8s_Sfs(const Ipp8u* pSrc, Ipp8s* pDst, int len, IppRoundMode
rndMode, int scaleFactor);

IppStatus ippsConvert_8s8u(const Ipp8s* pSrc, Ipp8u* pDst, int len);

IppStatus ippsConvert_16s8s_Sfs(const Ipp16s* pSrc, Ipp8s* pDst, Ipp32u len,
IppRoundMode rndMode, int scaleFactor);

IppStatus ippsConvert_16s32s(const Ipp16s* pSrc, Ipp32s* pDst, int len);

IppStatus ippsConvert_16s32f(const Ipp16s* pSrc, Ipp32f* pDst, int len);

IppStatus ippsConvert_16u32f(const Ipp16u* pSrc, Ipp32f* pDst, int len);

IppStatus ippsConvert_32s16s(const Ipp32s* pSrc, Ipp16s* pDst, int len);

IppStatus ippsConvert_32s32f(const Ipp32s* pSrc, Ipp32f* pDst, int len);

IppStatus ippsConvert_32s64f(const Ipp32s* pSrc, Ipp64f* pDst, int len);

IppStatus ippsConvert_32f64f(const Ipp32f* pSrc, Ipp64f* pDst, int len);

IppStatus ippsConvert_64s64f(const Ipp64s* pSrc, Ipp64f* pDst, Ipp32u len);

IppStatus ippsConvert_64f32f(const Ipp64f* pSrc, Ipp32f* pDst, int len);

IppStatus ippsConvert_16s32f_Sfs(const Ipp16s* pSrc, Ipp32f* pDst, int len, int
scaleFactor);

IppStatus ippsConvert_16s64f_Sfs(const Ipp16s* pSrc, Ipp64f* pDst, int len, int
scaleFactor);

IppStatus ippsConvert_32s16s_Sfs(const Ipp32s* pSrc, Ipp16s* pDst, int len, int
scaleFactor);

IppStatus ippsConvert_32s32f_Sfs(const Ipp32s* pSrc, Ipp32f* pDst, int len, int
scaleFactor);

IppStatus ippsConvert_32s64f_Sfs(const Ipp32s* pSrc, Ipp64f* pDst, int len, int
scaleFactor);

IppStatus ippsConvert_32f8s_Sfs(const Ipp32f* pSrc, Ipp8s* pDst, int len, IppRoundMode
rndMode, int scaleFactor);

IppStatus ippsConvert_32f8u_Sfs(const Ipp32f* pSrc, Ipp8u* pDst, int len, IppRoundMode
rndMode, int scaleFactor);

IppStatus ippsConvert_32f16s_Sfs(const Ipp32f* pSrc, Ipp16s* pDst, int len,
IppRoundMode rndMode, int scaleFactor);

IppStatus ippsConvert_32f16u_Sfs(const Ipp32f* pSrc, Ipp16u* pDst, int len,
IppRoundMode rndMode, int scaleFactor);

IppStatus ippsConvert_32f32s_Sfs(const Ipp32f* pSrc, Ipp32s* pDst, int len,
IppRoundMode rndMode, int scaleFactor);

IppStatus ippsConvert_64f8s_Sfs(const Ipp64f* pSrc, Ipp8s* pDst, int len, IppRoundMode
rndMode, int scaleFactor);
```

```
IppStatus ippsConvert_64f8u_Sfs(const Ipp64f* pSrc, Ipp8u* pDst, int len, IppRoundMode
rndMode, int scaleFactor);

IppStatus ippsConvert_64f16u_Sfs(const Ipp64f* pSrc, Ipp16u* pDst, int len,
IppRoundMode rndMode, int scaleFactor);

IppStatus ippsConvert_64s32s_Sfs(const Ipp64s* pSrc, Ipp32s* pDst, int len,
IppRoundMode rndMode, int scaleFactor);

IppStatus ippsConvert_64f16s_Sfs(const Ipp64f* pSrc, Ipp16s* pDst, int len,
IppRoundMode rndMode, int scaleFactor);

IppStatus ippsConvert_64f32s_Sfs(const Ipp64f* pSrc, Ipp32s* pDst, int len,
IppRoundMode rndMode, int scaleFactor);

IppStatus ippsConvert_64f64s_Sfs(const Ipp64f* pSrc, Ipp64s* pDst, Ipp32u len,
IppRoundMode rndMode, int scaleFactor);

IppStatus ippsConvert_24u32u(const Ipp8u* pSrc, Ipp32u* pDst, int len);

IppStatus ippsConvert_24u32f(const Ipp8u* pSrc, Ipp32f* pDst, int len);

IppStatus ippsConvert_32u24u_Sfs(const Ipp32u* pSrc, Ipp8u* pDst, int len, int
scaleFactor);

IppStatus ippsConvert_32f24u_Sfs(const Ipp32f* pSrc, Ipp8u* pDst, int len, int
scaleFactor);

IppStatus ippsConvert_24s32s(const Ipp8u* pSrc, Ipp32s* pDst, int len);

IppStatus ippsConvert_24s32f(const Ipp8u* pSrc, Ipp32f* pDst, int len);

IppStatus ippsConvert_32s24s_Sfs(const Ipp32s* pSrc, Ipp8u* pDst, int len, int
scaleFactor);

IppStatus ippsConvert_32f24s_Sfs(const Ipp32f* pSrc, Ipp8u* pDst, int len, int
scaleFactor);

IppStatus ippsConvert_16s16f(const Ipp16s* pSrc, Ipp16f* pDst, int len, IppRoundMode
rndMode);

IppStatus ippsConvert_32f16f(const Ipp32f* pSrc, Ipp16f* pDst, int len, IppRoundMode
rndMode);

IppStatus ippsConvert_16f16s_Sfs(const Ipp16f* pSrc, Ipp16s* pDst, int len,
IppRoundMode rndMode, int scaleFactor);

IppStatus ippsConvert_16f32f(const Ipp16f* pSrc, Ipp32f* pDst, int len);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pSrc`	Pointer to the source vector.

pDst	Pointer to the destination vector.
rndMode	Rounding mode, the following values are possible: • ippRndZero floating-point values are truncated to zero • ippRndNear floating-point values are rounded to the nearest even integer when the fractional part equals 0.5; otherwise they are rounded to the nearest integer • ippRndFinancial floating-point values are rounded down to the nearest integer when the fractional part is less than 0.5, or rounded up to the nearest integer if the fractional part is equal or greater than 0.5.
len	Number of elements in the vector.
scaleFactor	Scale factor, refer to Integer Scaling.

Description

This function converts the type of data contained in the vector pSrc and stores the results in pDst.

Functions with the Sfs suffix perform scaling of the result value in accordance with the scaleFactor value. The converted result is saturated if it exceeds the output data range.

Functions that operate with 16f data do not support the ippRndFinancial rounding mode.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when the pDst or pSrc pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.
ippStsRoundModeNotSupportedErr	Indicates an error when the specified rounding mode is not supported.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
```

```
warranties, other than
 * those that are expressly stated in the License.
 *****************************************************************************/

#include <stdio.h>
#include "ipp.h"

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define
check_sts(st)
                                  \
    if ((st) !=
ippStsNoErr)
                          \
        goto exitLine; /* Go to Exit if Intel(R) Integrated Performance Primitives (Intel(R)
IPP) function returned status different from \
                           ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main()
{
    IppStatus status;
    Ipp32s src32s[2] = {33000, -33000};
    Ipp32f src32f[2] = {126.6, -125.4};
    Ipp32f src_32f[2] = {113.12, -113.6};
    Ipp32f src1_32f[5] = {-2.5, -2.4, 1.4, 1.5, 1.6};
    Ipp8s src_8s[2] = {125, -125};
    Ipp8u src8[1] = {255};
    Ipp32f dst32f[1];
    Ipp16s dst16[2];
    Ipp16s dst_16s[2];
    Ipp8u dst8u[2];
    Ipp8s dstN8[2];
    Ipp8s dstZ8[2];
    Ipp8s dstF8[5];

    int scaleFactor = 0; // no scaling

    check_sts(status = ippsConvert_8s16s(src_8s, dst_16s, 2));
    printf("\n ippsConvert_8s16s Source: %d %d; Result: %d %d\n", src_8s[0], src_8s[1],
dst_16s[0], dst_16s[1]);

    check_sts(status = ippsConvert_8u32f(src8, dst32f, 1));
    printf("\n ippsConvert_8u32f Source: %d; Result: %.1f\n", src8[0], dst32f[0]);

    check_sts(status = ippsConvert_32s16s(src32s, dst16, 2));
    printf("\n ippsConvert_32s16s Source: %d %d; Result: %d %d\n", src32s[0], src32s[1],
dst16[0], dst16[1]);

    check_sts(status = ippsConvert_32f8s_Sfs(src32f, dstN8, 2, ippRndNear, scaleFactor));
    printf("\n ippsConvert_32f8s_Sfs : ippRndNear\n Source: %.1f %.1f; Result: %d %d\n",
src32f[0], src32f[1], dstN8[0], dstN8[1]);

    check_sts(status = ippsConvert_32f8s_Sfs(src32f, dstZ8, 2, ippRndZero, scaleFactor));
```

```
    printf("\n ippsConvert_32f8s_Sfs : ippRndZero\n Source: %.1f %.1f; Result: %d %d\n",
src32f[0], src32f[1], dstZ8[0], dstZ8[1]);

    check_sts(status = ippsConvert_32f8u_Sfs(src_32f, dst8u, 2, ippRndNear, scaleFactor));
    printf("\n ippsConvert_32f8u_Sfs : ippRndNear\n Source: %.1f %.1f; Result: %d %d\n",
src_32f[0], src_32f[1], dst8u[0], dst8u[1]);

    check_sts(status = ippsConvert_32f8s_Sfs(src1_32f, dstF8, 5, ippRndFinancial, scaleFactor));
    printf("\n ippsConvert_32f8u_Sfs : ippRndFinancial\n Source: %.1f %.1f %.1f %.1f %.1f;
Result: %d %d %d %d %d\n", src1_32f[0], src1_32f[1],
           src1_32f[2], src1_32f[3], src1_32f[4], dstF8[0], dstF8[1], dstF8[2], dstF8[3],
dstF8[4]);

    EXIT_MAIN
    printf("\nExit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
/*
The function ippsConvert_32f16f has the following specific when it processes the number that are
not in the range [MinVal16f..MaxVal16f]:
If x > MaxVal16f then{
    If(rndMode == IppRndNear) then{
        y = Convert_32f16f(x) = +INF
    }
    If(rndMode == IppRndZero) then{
        If(x == +INF) then{
            y = Convert_32f16f(x) = +INF
        }else {
            y = Convert_32f16f(x) = MaxVal16f
        }
    }
}

If x < MinVal16f then{
    If(rndMode == IppRndNear) then{
        y = Convert_32f16f(x) = -INF
    }
    If(rndMode == IppRndZero) then{
        If(x == -INF) then{
            y = Convert_32f16f(x) = -INF
        }else {
            y = Convert_32f16f(x) = M in Val16f
        }
    }
}
*/
```

See Also

Integer Scaling

Conj

Stores the complex conjugate values of a vector in a second vector or in-place.

Syntax

```
IppStatus ippsConj_16sc(const Ipp16sc* pSrc, Ipp16sc* pDst, int len);
```

```
IppStatus ippsConj_32fc(const Ipp32fc* pSrc, Ipp32fc* pDst, int len);
IppStatus ippsConj_64fc(const Ipp64fc* pSrc, Ipp64fc* pDst, int len);
IppStatus ippsConj_16sc_I(Ipp16sc* pSrcDst, int len);
IppStatus ippsConj_32fc_I(Ipp32fc* pSrcDst, int len);
IppStatus ippsConj_64fc_I(Ipp64fc* pSrcDst, int len);
IppStatus ippsConj_32fc_L(const Ipp32fc* pSrc, Ipp32fc* pDst, IppSizeL len);
IppStatus ippsConj_64fc_L(const Ipp64fc* pSrc, Ipp64fc* pDst, IppSizeL len);
IppStatus ippsConj_32fc_I_L(Ipp32fc* pSrcDst, IppSizeL len);
IppStatus ippsConj_64fc_I_L(Ipp64fc* pSrcDst, IppSizeL len);
```

Include Files

`ipps.h`

Flavors with the `_L` suffix: `ipps_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pSrc`	Pointer to the source vector.
`pDst`	Pointer to the destination vector.
`pSrcDst`	Pointer to the source and destination vector for the in-place operation.
`len`	Number of elements in the vector.

Description

This function stores in `pDst` the element-wise conjugation of the complex vector `pSrc`. The element-wise conjugation of the vector is defined as follows:

`pDst[n].re=pSrc[n].re`

`pDst[n].im= -pSrc[n].im`

The in-place flavors of `ippsConj` store in `pSrcDst` the element-wise conjugation of the complex vector `pSrcDst`.

The element-wise conjugation of the vector is defined as follows:

`pSrcDst[n].re=pSrcDst[n].re`

`pSrcDst[n].im= -pSrcDst[n].im`

Return Values

`ippStsNoErr`	Indicates no error.

ippStsNullPtrErr	Indicates an error when the pSrc, pDst, or pSrcDst pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.

ConjFlip

Computes the complex conjugate of a vector and stores the result in reverse order.

Syntax

```
IppStatus ippsConjFlip_16sc(const Ipp16sc* pSrc, Ipp16sc* pDst, int len);
IppStatus ippsConjFlip_32fc(const Ipp32fc* pSrc, Ipp32fc* pDst, int len);
IppStatus ippsConjFlip_64fc(const Ipp64fc* pSrc, Ipp64fc* pDst, int len);
```

Include Files

ipps.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

pSrc	Pointer to the source vector.
pDst	Pointer to the destination vector.
len	Number of elements in the vector.

Description

This function computes the conjugate of the vector pSrc and stores the result, in reverse order, in pDst. The complex conjugate, stored in reverse order, is defined as follows:

pDst[n] = conj(pSrc[len- n - 1]).

Note that if pSrc and pDst overlap in memory, the function returns unpredictable results.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when the pSrc or pDst pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.

Magnitude

Computes the magnitudes of the elements of a complex vector.

Syntax

```
IppStatus ippsMagnitude_32f(const Ipp32f* pSrcRe, const Ipp32f* pSrcIm, Ipp32f* pDst, int len);

IppStatus ippsMagnitude_64f(const Ipp64f* pSrcRe, const Ipp64f* pSrcIm, Ipp64f* pDst, int len);

IppStatus ippsMagnitude_32fc(const Ipp32fc* pSrc, Ipp32f* pDst, int len);

IppStatus ippsMagnitude_64fc(const Ipp64fc* pSrc, Ipp64f* pDst, int len);

IppStatus ippsMagnitude_16s32f(const Ipp16s* pSrcRe, const Ipp16s* pSrcIm, Ipp32f* pDst, int len);

IppStatus ippsMagnitude_16sc32f(const Ipp16sc* pSrc, Ipp32f* pDst, int len);

IppStatus ippsMagnitude_16s_Sfs(const Ipp16s* pSrcRe, const Ipp16s* pSrcIm, Ipp16s* pDst, int len, int scaleFactor);

IppStatus ippsMagnitude_16sc_Sfs(const Ipp16sc* pSrc, Ipp16s* pDst, int len, int scaleFactor);

IppStatus ippsMagnitude_32sc_Sfs(const Ipp32sc* pSrc, Ipp32s* pDst, int len, int scaleFactor);
```

Platform-aware functions:

```
IppStatus ippsMagnitude_32f_L(const Ipp32f* pSrcRe, const Ipp32f* pSrcIm, Ipp32f* pDst, IppSizeL len);

IppStatus ippsMagnitude_64f_L(const Ipp64f* pSrcRe, const Ipp64f* pSrcIm, Ipp64f* pDst, IppSizeL len);

IppStatus ippsMagnitude_32fc_L(const Ipp32fc* pSrc, Ipp32f* pDst, IppSizeL len);

IppStatus ippsMagnitude_64fc_L(const Ipp64fc* pSrc, Ipp64f* pDst, IppSizeL len);
```

Include Files

`ipps.h`

Flavors with the `_L` suffix: `ipps_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pSrc`	Pointer to the source vector.
`pSrcRe`	Pointer to the vector with the real parts of complex elements.
`pSrcIm`	Pointer to the vector with the imaginary parts of complex elements.
`pDst`	Pointer to the destination vector.

len	Number of elements in the vector
scaleFactor	Scale factor, refer to Integer Scaling.

Description

The complex flavor of this function computes the element-wise magnitude of the complex vector pSrc and stores the result in pDst. The element-wise magnitude is defined by the formula:

$magn[n] = (pSrc[n].re^2 + pSrc[n].im^2)^{1/2}$

The real flavor of the function ippsMagnitude computes the element-wise magnitude of the complex vector whose real and imaginary components are specified in the vectors pSrcRe and pSrcIm, respectively, and stores the result in pDst. The element-wise magnitude is defined by the formula:

$magn[n] = (pSrcRe[n]^2 + pSrcIm[n]^2)^{1/2}$

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.

Example

```
/*******************************************************************************
 * Copyright 2017 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

#include <stdio.h>
#include "ipp.h"

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define
check_sts(st)
                                    \
    if ((st) !=
ippStsNoErr)
                          \
        goto exitLine; /* Go to Exit if Intel(R) Integrated Performance Primitives (Intel(R)
```

```
IPP) function returned status different from \
                          ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main()
{
    Ipp32fc pSrc[5] = {{1.2, 3.2}, {0.0, 0.3}, {4.3, 6.7}, {4.2, 0.0}, {1.2, 1.2}};
    Ipp32f pDstIm[5];
    IppStatus status;
    int i;

    printf("\nSource vector\n");
    for (i = 0; i < 5; i++)
        printf("%.1f, %.1f; ", pSrc[i].re, pSrc[i].im);

    check_sts(status = ippsMagnitude_32fc(pSrc, pDstIm, 5));

    printf("\nResult values vector\n");
    for (i = 0; i < 5; i++)
        printf("%.1f ", pDstIm[i]);

    EXIT_MAIN
    printf("\nExit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

Phase

Computes the phase angles of elements of a complex vector.

Syntax

`IppStatus ippsPhase_64fc(const Ipp64fc* pSrc, Ipp64f* pDst, int len);`

`IppStatus ippsPhase_32fc(const Ipp32fc* pSrc, Ipp32f* pDst, int len);`

`IppStatus ippsPhase_16sc32f(const Ipp16sc* pSrc, Ipp32f* pDst, int len);`

`IppStatus ippsPhase_64f(const Ipp64f* pSrcRe, const Ipp64f* pSrcIm, Ipp64f* pDst, int len);`

`IppStatus ippsPhase_32f(const Ipp32f* pSrcRe, const Ipp32f* pSrcIm, Ipp32f* pDst, int len);`

`IppStatus ippsPhase_16s32f(const Ipp16s* pSrcRe, const Ipp16s* pSrcIm, Ipp32f* pDst, int len);`

`IppStatus ippsPhase_16sc_Sfs(const Ipp16sc* pSrc, Ipp16s* pDst, int len, int scaleFactor);`

`IppStatus ippsPhase_16s_Sfs(const Ipp16s* pSrcRe, const Ipp16s* pSrcIm, Ipp16s* pDst, int len, int scaleFactor);`

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pSrc`	Pointer to the source vector.
`pSrcRe`	Pointer to the source vector which stores the real components.
`pSrcIm`	Pointer to the source vector which stores the imaginary components.
`pDst`	Pointer to the vector which stores the phase (angle) components of the elements in radians. Phase values are in the range (-π, π].
`len`	Number of elements in the vector
`scaleFactor`	Scale factor, refer to Integer Scaling .

Description

This function returns the phase angles of elements of the complex input vector `pSrc`, or the complex input vector whose real and imaginary components are specified in the vectors `pSrcRe` and `pSrcIm`, respectively, and stores the result in the vector `pDst`. Phase values are returned in radians and are in the range (-π, π] .

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error when `len` is less than or equal to zero.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

#include <stdio.h>
```

```
#include "ipp.h"

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define
check_sts(st)
                                  \
    if ((st) !=
ippStsNoErr)
                           \
        goto exitLine; /* Go to Exit if Intel(R) Integrated Performance Primitives (Intel(R)
IPP) function returned status different from \
                            ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main()
{
    Ipp32fc pSrc[5] = {{1.2, 3.2}, {0.0, 0.3}, {4.3, 6.7}, {4.2, 0.0}, {1.2, 1.2}};
    Ipp32f pDstIm[5];
    IppStatus status;
    int i;

    printf("\nSource vector\n");
    for (i = 0; i < 5; i++)
        printf("%.1f, %.1f; ", pSrc[i].re, pSrc[i].im);

    check_sts(status = ippsPhase_32fc(pSrc, pDstIm, 5));

    printf("\nResult values vector\n");
    for (i = 0; i < 5; i++)
        printf("%.1f ", pDstIm[i]);

    EXIT_MAIN
    printf("\nExit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

PowerSpectr

Computes the power spectrum of a complex vector.

Syntax

```
IppStatus ippsPowerSpectr_64fc(const Ipp64fc* pSrc, Ipp64f* pDst, int len);

IppStatus ippsPowerSpectr_32fc(const Ipp32fc* pSrc, Ipp32f* pDst, int len);

IppStatus ippsPowerSpectr_16sc_Sfs(const Ipp16sc* pSrc, Ipp16s* pDst, int len, int
scaleFactor);

IppStatus ippsPowerSpectr_64f(const Ipp64f* pSrcRe, const Ipp64f* pSrcIm, Ipp64f* pDst,
int len);

IppStatus ippsPowerSpectr_32f(const Ipp32f* pSrcRe, const Ipp32f* pSrcIm, Ipp32f* pDst,
int len);
```

```
IppStatus ippsPowerSpectr_16s_Sfs(const Ipp16s* pSrcRe, const Ipp16s* pSrcIm, Ipp16s*
pDst, int len, int scaleFactor);

IppStatus ippsPowerSpectr_16s32f(const Ipp16s* pSrcRe, const Ipp16s* pSrcIm, Ipp32f*
pDst, int len);

IppStatus ippsPowerSpectr_16sc32f(const Ipp16sc* pSrc, Ipp32f* pDst, int len);
```

Platform-aware functions:

```
IppStatus ippsPowerSpectr_32fc_L(const Ipp32fc* pSrc, Ipp32f* pDst, IppSizeL len);

IppStatus ippsPowerSpectr_64fc_L(const Ipp64fc* pSrc, Ipp64f* pDst, IppSizeL len);

IppStatus ippsPowerSpectr_32f_L(const Ipp32f* pSrcRe, const Ipp32f* pSrcIm, Ipp32f*
pDst, IppSizeL len);

IppStatus ippsPowerSpectr_64f_L(const Ipp64f* pSrcRe, const Ipp64f* pSrcIm, Ipp64f*
pDst, IppSizeL len);
```

Include Files

`ipps.h`

Flavors with the `_L` suffix: `ipps_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pSrc`	Pointer to the source vector.
`pSrcRe`	Pointer to the source vector which stores the real components.
`pSrcIm`	Pointer to the source vector which stores the imaginary components.
`pDst`	Pointer to the vector which stores the spectrum components of the elements.
`len`	Number of elements in the vector
`scaleFactor`	Scale factor, refer to Integer Scaling.

Description

This function returns the power spectrum of the complex input vector `pSrc`, or the complex input vector whose real and imaginary components are specified in the vectors `pSrcRe` and `pSrcIm`, respectively, and stores the results in the vector `pDst`. The power spectrum elements are squares of the magnitudes of the complex input vector elements:

`pDst[n] = (pSrc[n].re)`^{2}`+ (pSrc[n].im)`^{2} , or `pDst[n] = (pSrcRe[n])`^{2}`+ (pSrcIm[n])`^{2} .

To compute magnitudes, use the function ippsMagnitude.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.

Real

Returns the real part of a complex vector in a second vector.

Syntax

```
IppStatus ippsReal_16sc(const Ipp16sc* pSrc, Ipp16s* pDstRe, int len);
IppStatus ippsReal_32fc(const Ipp32fc* pSrc, Ipp32f* pDstRe, int len);
IppStatus ippsReal_64fc(const Ipp64fc* pSrc, Ipp64f* pDstRe, int len);
```

Include Files

ipps.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

pSrc	Pointer to the complex source vector.
pDstRe	Pointer to the destination vector with real parts.
len	Number of elements in the vector.

Description

This function returns the real part of the complex vector pSrc in the vector pDstRe.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when the pDstRe or pSrc pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to zero.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
```

```
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *****************************************************************************/

#include <stdio.h>
#include "ipp.h"

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define
check_sts(st)
                                  \
    if ((st) !=
ippStsNoErr)
                           \
        goto exitLine; /* Go to Exit if Intel(R) Integrated Performance Primitives (Intel(R)
IPP) function returned status different from \
                            ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main()
{
    Ipp32fc pSrc[5] = {{1.2, 3.2}, {0.0, 0.3}, {4.3, 6.7}, {4.2, 0.0}, {1.2, 1.2}};
    Ipp32f pDstIm[5];
    IppStatus status;
    int i;

    printf("\nSource vector\n");
    for (i = 0; i < 5; i++)
        printf("%.1f, %.1f; ", pSrc[i].re, pSrc[i].im);

    check_sts(status = ippsReal_32fc(pSrc, pDstIm, 5));

    printf("\nResult values vector\n");
    for (i = 0; i < 5; i++)
        printf("%.1f ", pDstIm[i]);

    EXIT_MAIN
    printf("\nExit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

Imag

Returns the imaginary part of a complex vector in a second vector.

Syntax

```
IppStatus ippsImag_16sc(const Ipp16sc* pSrc, Ipp16s* pDstIm, int len);
IppStatus ippsImag_32fc(const Ipp32fc* pSrc, Ipp32f* pDstIm, int len);
IppStatus ippsImag_64fc(const Ipp64fc* pSrc, Ipp64f* pDstIm, int len);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pSrc`	Pointer to the complex source vector.
`pDstIm`	Pointer to the destination vector with imaginary parts.
`len`	Number of elements in the vector.

Description

This function returns the imaginary part of a complex vector `pSrc` in the vector `pDstIm`.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when the `pDstIm` or `pSrc` pointer is `NULL`.
`ippStsSizeErr`	Indicates an error when `len` is less than or equal to zero.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

#include <stdio.h>
```

```
#include "ipp.h"

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define
check_sts(st)
                                  \
    if ((st) !=
ippStsNoErr)
                           \
        goto exitLine; /* Go to Exit if Intel(R) Integrated Performance Primitives (Intel(R)
IPP) function returned status different from \
                            ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main()
{
    Ipp32fc pSrc[5] = {{1.2, 3.2}, {0.0, 0.3}, {4.3, 6.7}, {4.2, 0.0}, {1.2, 1.2}};
    Ipp32f pDstIm[5];
    IppStatus status;
    int i;

    printf("\nSource vector\n");
    for (i = 0; i < 5; i++)
        printf("%.1f, %.1f; ", pSrc[i].re, pSrc[i].im);

    check_sts(status = ippsImag_32fc(pSrc, pDstIm, 5));

    printf("\nResult values vector\n");
    for (i = 0; i < 5; i++)
        printf("%.1f ", pDstIm[i]);

    EXIT_MAIN
    printf("\nExit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

RealToCplx

Returns a complex vector constructed from the real and imaginary parts of two real vectors.

Syntax

```
IppStatus ippsRealToCplx_16s(const Ipp16s* pSrcRe, const Ipp16s* pSrcIm, Ipp16sc* pDst,
int len);
```

```
IppStatus ippsRealToCplx_32f(const Ipp32f* pSrcRe, const Ipp32f* pSrcIm, Ipp32fc* pDst,
int len);
```

```
IppStatus ippsRealToCplx_64f(const Ipp64f* pSrcRe, const Ipp64f* pSrcIm, Ipp64fc* pDst,
int len);
```

Platform-aware functions:

```
IppStatus ippsRealToCplx_32f_L(const Ipp32f* pSrcRe, const Ipp32f* pSrcIm, Ipp32fc*
pDst, IppSizeL len);
```

```
IppStatus ippsRealToCplx_64f_L(const Ipp64f* pSrcRe, const Ipp64f* pSrcIm, Ipp64fc*
pDst, IppSizeL len);
```

Include Files

`ipps.h`

Flavors with the `_L` suffix: `ipps_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pSrcRe`	Pointer to the vector with real parts of complex elements.
`pSrcIm`	Pointer to the vector with imaginary parts of complex elements.
`pDst`	Pointer to the destination vector.
`len`	Number of elements in the vector.

Description

This function returns a complex vector `pDst` constructed from the real and imaginary parts of the input vectors `pSrcRe` and `pSrcIm`.

If `pSrcRe` is `NULL`, the real component of the vector is set to zero.

If `pSrcIm` is `NULL`, the imaginary component of the vector is set to zero.

Note that the pointers cannot be both `NULL`.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when the `pDst` pointer is `NULLNULL`. The pointer `pSrcRe` or `pSrcIm` can be `NULL`.
`ippStsSizeErr`	Indicates an error when `len` is less than or equal to zero.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
```

```
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *****************************************************************************/

#include <stdio.h>
#include "ipp.h"

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define
check_sts(st)
                                  \
    if ((st) !=
ippStsNoErr)
                           \
        goto exitLine; /* Go to Exit if Intel(R) Integrated Performance Primitives (Intel(R)
IPP) function returned status different from \
                            ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main()
{
    Ipp16s pValsIm[] = {1, -3, 5, -7, 18, -24, 35, 48};
    Ipp16s pValsRe[] = {1, -3, 5, -7, 18, -24, 35, 48};
    Ipp16sc pOutVals[8];
    IppStatus status;
    int i;

    printf("\nSource vector\n");
    printf("\nRe=");
    for (i = 0; i < 8; i++)
        printf("%d ", pValsRe[i]);
    printf("\nIm=");
    for (i = 0; i < 8; i++)
        printf("%d ", pValsRe[i]);

    check_sts(status = ippsRealToCplx_16s(pValsRe, pValsIm, pOutVals, 8));

    printf("\nResult values vector\n");
    for (i = 0; i < 8; i++)
        printf("%d, %d; ", pOutVals[i].re, pOutVals[i].im);

    EXIT_MAIN
    printf("\nExit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

CplxToReal

Returns the real and imaginary parts of a complex vector in two respective vectors.

Syntax

```
IppStatus ippsCplxToReal_16sc(const Ipp16sc* pSrc, Ipp16s* pDstRe, Ipp16s* pDstIm, int len);
```

```
IppStatus ippsCplxToReal_32fc(const Ipp32fc* pSrc, Ipp32f* pDstRe, Ipp32f* pDstIm, int len);
```

```
IppStatus ippsCplxToReal_64fc(const Ipp64fc* pSrc, Ipp64f* pDstRe, Ipp64f* pDstIm, int len);
```

Platform-aware functions:

```
IppStatus ippsCplxToReal_32fc_L(const Ipp32fc* pSrc, Ipp32f* pDstRe, Ipp32f* pDstIm, IppSizeL len);
```

```
IppStatus ippsCplxToReal_64fc_L(const Ipp64fc* pSrc, Ipp64f* pDstRe, Ipp64f* pDstIm, IppSizeL len);
```

Include Files

`ipps.h`

Flavors with the `_L` suffix: `ipps_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pSrc`	Pointer to the complex vector `pSrc`.
`pDstRe`	Pointer to the output vector with real parts.
`pDstIm`	Pointer to the output vector with imaginary parts.
`len`	Number of elements in the vector.

Description

This function returns the real and imaginary parts of a complex vector `pSrc` in two vectors `pDstRe` and `pDstIm`.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when the data vector pointer is `NULL`.
`ippStsSizeErr`	Indicates an error when `len` is less than or equal to 0.

Threshold

Performs the threshold operation on the elements of a vector by limiting the element values by specified value.

Syntax

```
IppStatus ippsThreshold_16s(const Ipp16s* pSrc, Ipp16s* pDst, int len, Ipp16s level,
IppCmpOp relOp);

IppStatus ippsThreshold_32f(const Ipp32f* pSrc, Ipp32f* pDst, int len, Ipp32f level,
IppCmpOp relOp);

IppStatus ippsThreshold_64f(const Ipp64f* pSrc, Ipp64f* pDst, int len, Ipp64f level,
IppCmpOp relOp);

IppStatus ippsThreshold_32fc(const Ipp32fc* pSrc, Ipp32fc* pDst, int len, Ipp32f level,
IppCmpOp relOp);

IppStatus ippsThreshold_64fc(const Ipp64fc* pSrc, Ipp64fc* pDst, int len, Ipp64f level,
IppCmpOp relOp);

IppStatus ippsThreshold_16sc(const Ipp16sc* pSrc, Ipp16sc* pDst, int len, Ipp16s level,
IppCmpOp relOp);

IppStatus ippsThreshold_16s_I(Ipp16s* pSrcDst, int len, Ipp16s level,IppCmpOp relOp);

IppStatus ippsThreshold_32f_I(Ipp32f* pSrcDst, int len, Ipp32f level, IppCmpOp relOp);

IppStatus ippsThreshold_64f_I(Ipp64f* pSrcDst, int len, Ipp64f level, IppCmpOp relOp);

IppStatus ippsThreshold_32fc_I(Ipp32fc* pSrcDst, int len, Ipp32f level, IppCmpOp
relOp);

IppStatus ippsThreshold_64fc_I(Ipp64fc* pSrcDst, int len, Ipp64f level, IppCmpOp
relOp);

IppStatus ippsThreshold_16sc_I(Ipp16sc* pSrcDst, int len, Ipp16s level, IppCmpOp
relOp);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pSrc`	Pointer to the source vector.
`pDst`	Pointer to the destination vector.
`pSrcDst`	Pointer to the source and destination vector for the in-place operation.
`len`	Number of elements in the vector.

level	Value used to limit each element of pSrc or pSrcDst.This parameter must always be real. For complex versions, it must be positive and represent magnitude.
relOp	Values of this argument specify which relational operator to use and whether level is an upper or lower bound for the input. The relOp must have one of the following values: • ippCmpLess Specifies the "less than" operator and level is a lower bound. • ippCmpGreater Specifies the "greater than" operator and level is an upper bound.

Description

This function performs the threshold operation on the vector pSrc by limiting each element by the threshold value level. Function operation is similar to that of the functions ippsThreshold_LT, ippsThreshold_GT but its interface contains the relOp parameter that specifies the type of the comparison operation to perform.

The in-place flavors of ippsThreshold perform the threshold operation on the vector pSrcDst by limiting each element by the threshold value level.

The relOp argument specifies which relational operator to use: when its value is ippCmpGreater- "greater than", when ippCmpLess - "less than", and determines whether level is an upper or lower bound for the input, respectively.

The formula for ippsThreshold called with the relOp= ippCmpLess is:

$$pDst[n] = \begin{cases} level, & pSrc[n] < level \\ pSrc[n], & otherwise \end{cases}$$

The formula for ippsThreshold called with the relOp= ippCmpGreater is:

$$pDst[n] = \begin{cases} level, & pSrc[n] > level \\ pSrc[n], & otherwise \end{cases}$$

For complex versions of the function ippsThreshold, the level argument is always real. The formula for complex ippsThreshold called with the relOp = ippCmpLess is:

$$pDst[n] = \begin{cases} \frac{pSrc[n] \cdot level}{abs(pSrc[n])}, & abs(pSrc[n]) < level \\ pSrc[n], & otherwise \end{cases}$$

The formula for complex `ippsThreshold` called with the `relOp= ippCmpGreater` is:

$$pDst[n] = \begin{cases} \frac{pSrc[n] \cdot level}{abs(pSrc[n])}, & abs(pSrc[n]) > level \\ pSrc[n], & otherwise \end{cases}$$

Application Notes

For all complex versions, `level` must be positive and represents a magnitude. The magnitude of the input is limited, but the phase remains unchanged. Zero-valued input is assumed to have zero phase.

A special rule is applied to the integer complex versions of the function `ippsThreshold`. In general, the resulting point coordinates at the complex plane are not integer. The function rounds them off to integer in such a way that the threshold operation is not performed. Thus, for the "less than" operation (with the `ippCmpLess` flag) the coordinates are rounded to the infinity (`+Inf` for positive coordinates, and `–Inf` for negative), and for the "greater than" operation (with the `ippCmpGreater` flag) the coordinates are rounded to zero.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when the `pSrc`, `pDst`, or `pSrcDst` pointer is `NULL`.
`ippStsSizeErr`	Indicates an error when `len` is less than or equal to 0.
`ippStsBadArgErr`	Indicates an error when `relOp` has an invalid value.
`ippStsThreshNegLevelErr`	Indicates an error when `level` for the complex version is negative (see appendix A "Handling of Special Cases" for more information).

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
```

```
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *****************************************************************************/

#include <stdio.h>
#include "ipp.h"

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define
check_sts(st)
                                  \
    if ((st) !=
ippStsNoErr)
                          \
        goto exitLine; /* Go to Exit if Intel(R) Integrated Performance Primitives (Intel(R)
IPP) function returned status different from \
                           ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main()
{
    Ipp32f pVals[] = {1, -3, 5, -7, 18, -24, 35, 48};
    Ipp32f pOutVals[8];
    IppStatus status;
    int i;

    printf("\nSource vector\n");
    for (i = 0; i < 8; i++)
        printf("%f ", pVals[i]);

    check_sts(status = ippsThreshold_32f(pVals, pOutVals, 8, 10, ippCmpGreater));

    printf("\nResult values vector\n");
    for (i = 0; i < 8; i++)
        printf("%f ", pOutVals[i]);

    EXIT_MAIN
    printf("\nExit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

Threshold_LT, Threshold_GT

Performs the threshold operation on the elements of a vector by limiting the element values by the specified value.

Syntax

```
IppStatus ippsThreshold_LT_16s(const Ipp16s* pSrc, Ipp16s* pDst, int len, Ipp16s
level);
```

```
IppStatus ippsThreshold_LT_32s(const Ipp32s* pSrc, Ipp32s* pDst, int len, Ipp32s
level);
```

```
IppStatus ippsThreshold_LT_32f(const Ipp32f* pSrc, Ipp32f* pDst, int len, Ipp32f
level);

IppStatus ippsThreshold_LT_64f(const Ipp64f* pSrc, Ipp64f* pDst, int len, Ipp64f
level);

IppStatus ippsThreshold_LT_32fc(const Ipp32fc* pSrc, Ipp32fc* pDst, int len, Ipp32f
level);

IppStatus ippsThreshold_LT_64fc(const Ipp64fc* pSrc, Ipp64fc* pDst, int len, Ipp64f
level);

IppStatus ippsThreshold_LT_16sc(const Ipp16sc* pSrc, Ipp16sc* pDst, int len, Ipp16s
level);

IppStatus ippsThreshold_GT_16s(const Ipp16s* pSrc, Ipp16s* pDst, int len, Ipp16s
level);

IppStatus ippsThreshold_GT_32s(const Ipp32s* pSrc, Ipp32s* pDst, int len, Ipp32s
level);

IppStatus ippsThreshold_GT_32f(const Ipp32f* pSrc, Ipp32f* pDst, int len, Ipp32f
level);

IppStatus ippsThreshold_GT_64f(const Ipp64f* pSrc, Ipp64f* pDst, int len, Ipp64f
level);

IppStatus ippsThreshold_GT_32fc(const Ipp32fc* pSrc, Ipp32fc* pDst, int len, Ipp32f
level);

IppStatus ippsThreshold_GT_64fc(const Ipp64fc* pSrc, Ipp64fc* pDst, int len, Ipp64f
level);

IppStatus ippsThreshold_GT_16sc(const Ipp16sc* pSrc, Ipp16sc* pDst, int len, Ipp16s
level);

IppStatus ippsThreshold_GT_16s_I(Ipp16s* pSrcDst, int len, Ipp16s level);

IppStatus ippsThreshold_GT_32s_I(Ipp32s* pSrcDst, int len, Ipp32s level);

IppStatus ippsThreshold_GT_32f_I(Ipp32f* pSrcDst, int len, Ipp32f level);

IppStatus ippsThreshold_GT_64f_I(Ipp64f* pSrcDst, int len, Ipp64f level);

IppStatus ippsThreshold_GT_32fc_I(Ipp32fc* pSrcDst, int len, Ipp32f level);

IppStatus ippsThreshold_GT_64fc_I(Ipp64fc* pSrcDst, int len, Ipp64f level);

IppStatus ippsThreshold_GT_16sc_I(Ipp16sc* pSrcDst, int len, Ipp16s level);

IppStatus ippsThreshold_LT_16s_I(Ipp16s* pSrcDst, int len, Ipp16s level);

IppStatus ippsThreshold_LT_32s_I(Ipp32s* pSrcDst, int len, Ipp32s level);

IppStatus ippsThreshold_LT_32f_I(Ipp32f* pSrcDst, int len, Ipp32f level);

IppStatus ippsThreshold_LT_64f_I(Ipp64f* pSrcDst, int len, Ipp64f level);

IppStatus ippsThreshold_LT_32fc_I(Ipp32fc* pSrcDst, int len, Ipp32f level);

IppStatus ippsThreshold_LT_64fc_I(Ipp64fc* pSrcDst, int len, Ipp64f level);

IppStatus ippsThreshold_LT_16sc_I(Ipp16sc* pSrcDst, int len, Ipp16s level);
```

Include Files

ipps.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

pSrc	Pointer to the source vector.
pDst	Pointer to the destination vector.
pSrcDst	Pointer to the source and destination vector for the in-place operation.
len	Number of elements in the vector.
level	Value used to limit each element of pSrc or pSrcDst.This argument must always be real. For complex versions, it must be positive and represent magnitude.

Description

They implement thresholding of the vector pSrc by limiting each element by the threshold value level. These functions perform the similar operation to the ippsThreshold function but are designed for the fixed type of the compare operation to use: ippsThreshold_LT is for the "less than" comparison, while ippsThreshold_GT is for the "greater than" comparison.

The in-place flavors perform the threshold operation on the vector pSrcDst by limiting each element by the threshold value level.

``ippsThreshold_LT``. The ippsThreshold_LT function performs the operation "less than", and level is a lower bound for the input. The formula for ippsThreshold_LT is the following:

$$pDst[n] = \begin{cases} level, & pSrc[n] < level \\ pSrc[n], & otherwise \end{cases}$$

For complex versions of the function ippsThreshold_LT, the parameter level is always real.

The formula for complex ippsThreshold_LT is:

$$pDst[n] = \begin{cases} \frac{pSrc[n] \cdot level}{abs(pSrc[n])}, & abs(pSrc[n]) < level \\ pSrc[n], & otherwise \end{cases}$$

``ippsThreshold_GT``. The function ippsThreshold_GT performs the operation "greater than" and level is an upper bound for the input.

The formula for ippsThreshold_GT is the following:

$$pDst[n] = \begin{cases} level, & pSrc[n] > level \\ pSrc[n], & otherwise \end{cases}$$

For complex versions of the function `ippsThreshold_GT`, the parameter `level` is always real.

The formula for complex `ippsThreshold_GT` is:

$$pDst[n] = \begin{cases} \frac{pSrc[n] \cdot level}{abs(pSrc[n])}, & abs(pSrc[n]) > level \\ pSrc[n], & otherwise \end{cases}$$

Application Notes

For all complex versions, `level` must be positive and represents a magnitude. The magnitude of the input is limited, but the phase remains unchanged. Zero-valued input is assumed to have zero phase.

A special rule is applied to the integer complex versions of the threshold functions. In general, the resulting point coordinates at the complex plane are not integer. The function rounds them off to integer in such a way that the threshold operation is not performed. Thus, for the "less than" operation (the `ippsThreshold_LT` function) the coordinates are rounded to the infinity (`+Inf` for positive coordinates, and `–Inf` for negative), and for the "greater than" operation (the `ippsThreshold_GT` function) the coordinates are rounded to 0.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when the `pSrc`, `pDst`, or `pSrcDst` pointer is `NULL`.
`ippStsSizeErr`	Indicates an error when `len` is less than or equal to 0
`ippStsThreshNegLevelErr`	Indicates an error when `level` for the complex version is negative (see appendix A "Handling of Special Cases" for more information).

Threshold_LTAbs, Threshold_GTAbs

Performs the threshold operation on the absolute values of elements of a vector.

Syntax

```
IppStatus ippsThreshold_LTAbs_16s(const Ipp16s* pSrc, Ipp16s* pDst, int len, Ipp16s level);
IppStatus ippsThreshold_LTAbs_32s(const Ipp32s* pSrc, Ipp32s* pDst, int len, Ipp32s level);
IppStatus ippsThreshold_LTAbs_32f(const Ipp32f* pSrc, Ipp32f* pDst, int len, Ipp32f level);
```

```
IppStatus ippsThreshold_LTAbs_64f(const Ipp64f* pSrc, Ipp64f* pDst, int len, Ipp64f
level);
IppStatus ippsThreshold_GTAbs_16s(const Ipp16s* pSrc, Ipp16s* pDst, int len, Ipp16s
level);
IppStatus ippsThreshold_GTAbs_32s(const Ipp32s* pSrc, Ipp32s* pDst, int len, Ipp32s
level);
IppStatus ippsThreshold_GTAbs_32f(const Ipp32f* pSrc, Ipp32f* pDst, int len, Ipp32f
level);
IppStatus ippsThreshold_GTAbs_64f(const Ipp64f* pSrc, Ipp64f* pDst, int len, Ipp64f
level);
IppStatus ippsThreshold_GTAbs_16s_I(Ipp16s* pSrcDst, int len, Ipp16s level);
IppStatus ippsThreshold_GTAbs_32s_I(Ipp32s* pSrcDst, int len, Ipp32s level);
IppStatus ippsThreshold_GTAbs_32f_I(Ipp32f* pSrcDst, int len, Ipp32f level);
IppStatus ippsThreshold_GTAbs_64f_I(Ipp64f* pSrcDst, int len, Ipp64f level);
IppStatus ippsThreshold_LTAbs_16s_I(Ipp16s* pSrcDst, int len, Ipp16s level);
IppStatus ippsThreshold_LTAbs_32s_I(Ipp32s* pSrcDst, int len, Ipp32s level);
IppStatus ippsThreshold_LTAbs_32f_I(Ipp32f* pSrcDst, int len, Ipp32f level);
IppStatus ippsThreshold_LTAbs_64f_I(Ipp64f* pSrcDst, int len, Ipp64f level);
```

Platform-aware functions:

```
IppStatus ippsThreshold_LTAbs_32f_L(const Ipp32f* pSrc, Ipp32f* pDst, IppSizeL len,
Ipp32f level);
IppStatus ippsThreshold_LTAbs_64f_L(const Ipp64f* pSrc, Ipp64f* pDst, IppSizeL len,
Ipp64f level);
IppStatus ippsThreshold_LTAbs_32f_I_L(Ipp32f* pSrcDst, IppSizeL len, Ipp32f level);
IppStatus ippsThreshold_LTAbs_64f_I_L(Ipp64f* pSrcDst, IppSizeL len, Ipp64f level);
```

Include Files

`ipps.h`

Flavors with the `_L` suffix: `ipps_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pSrc`	Pointer to the source vector.
`pDst`	Pointer to the destination vector.
`pSrcDst`	Pointer to the source and destination vector for the in-place operation.

len	Number of elements in the vector.
level	Value used to limit each element of source vector. This argument can not be negative.

Description

These functions implement thresholding of the vector pSrc by limiting absolute value of each element by the threshold value level. These functions perform the compare operation of the fixed type: ippsThreshold_LTAbs is for the "less than" comparison, while ippsThreshold_GTAbs is for the "greater than" comparison. Elements of the result vector pDst have the same sign that the source elements.

The in-place flavors perform the threshold operation on the vector pSrcDst.

``ippsThreshold_LTAbs``. The ippsThreshold_LTAbs function performs the operation "less than", and level is a lower bound for the input. The formula for ippsThreshold_LTAbs is the following:

$$pDst[n] = \begin{cases} level \text{ if } abs(pSrc[n]) < level, & pSrc[n] \geq 0 \\ -level \text{ if } abs(pSrc[n]) < level, & pSrc[n] < 0 \\ pSrc[n], \quad otherwise \end{cases}$$

``ippsThreshold_GTAbs``. The function ippsThreshold_GTAbs performs the operation "greater than" and level is an upper bound for the input. The formula for ippsThreshold_GTAbs is the following:

$$pDst[n] = \begin{cases} level \text{ if } abs(pSrc[n]) > level, & pSrc[n] \geq 0 \\ -level \text{ if } abs(pSrc[n]) > level, & pSrc[n] < 0 \\ pSrc[n], \quad otherwise \end{cases}$$

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if pSrc, pDst, or pSrcDst pointer is NULL.
ippStsSizeErr	Indicates an error if len is less than or equal to zero.
ippStsThreshNegLevelErr	Indicates an error if level is negative.

Example

Threshold_LtAbs:

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
```

```
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *****************************************************************************/

#include <stdio.h>
#include "ipp.h"

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define
check_sts(st)
                                  \
    if ((st) !=
ippStsNoErr)
                          \
        goto exitLine; /* Go to Exit if Intel(R) Integrated Performance Primitives (Intel(R)
IPP) function returned status different from \
                           ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main()
{
    Ipp32f pVals[] = {1, -3, 5, -7, 18, -24, 35, 48};
    Ipp32f pOutVals[8];
    IppStatus status;
    int i;

    printf("Source vector\n");
    for (i = 0; i < 8; i++)
        printf("%f ", pVals[i]);

    check_sts(status = ippsThreshold_LTAbs_32f(pVals, pOutVals, 8, 10));

    printf("Result values vector\n");
    for (i = 0; i < 8; i++)
        printf("%f ", pOutVals[i]);

    EXIT_MAIN
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

Threshold_LtAbs_I:

```
/*****************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
```

```
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *****************************************************************************/

#include <stdio.h>
#include "ipp.h"

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define
check_sts(st)
                                  \
    if ((st) !=
ippStsNoErr)
                          \
        goto exitLine; /* Go to Exit if Intel(R) Integrated Performance Primitives (Intel(R)
IPP) function returned status different from \
                           ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main()
{
    Ipp32f pVals[] = {1, -3, 5, -7, 18, -24, 35, 48};
    IppStatus status;
    int i;

    printf("Source vector\n");
    for (i = 0; i < 8; i++)
        printf("%f ", pVals[i]);

    check_sts(status = ippsThreshold_LTAbs_32f_I(pVals, 8, 10));

    printf("Result values vector\n");
    for (i = 0; i < 8; i++)
        printf("%f ", pVals[i]);

    EXIT_MAIN
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

Threshold_LTVal, Threshold_LTAbsVal, Threshold_GTVal, Threshold_LTValGTVal

Performs the threshold operation on the elements of a vector by limiting the element values by the specified level and replacing them with the specified value.

Syntax

```
IppStatus ippsThreshold_LTAbsVal_32f(const Ipp32f* pSrc, Ipp32f* pDst, int len, Ipp32f level, Ipp32f value);

IppStatus ippsThreshold_LTAbsVal_64f(const Ipp64f* pSrc, Ipp64f* pDst, int len, Ipp64f level, Ipp64f value);

IppStatus ippsThreshold_LTAbsVal_16s(const Ipp16s* pSrc, Ipp16s* pDst, int len, Ipp16s level, Ipp16s value);

IppStatus ippsThreshold_LTAbsVal_32s(const Ipp32s* pSrc, Ipp32s* pDst, int len, Ipp32s level, Ipp32s value);

IppStatus ippsThreshold_LTAbsVal_32f_I(Ipp32f* pSrcDst, int len, Ipp32f level, Ipp32f value);

IppStatus ippsThreshold_LTAbsVal_64f_I(Ipp64f* pSrcDst, int len, Ipp64f level, Ipp64f value);

IppStatus ippsThreshold_LTAbsVal_16s_I(Ipp16s* pSrcDst, int len, Ipp16s level, Ipp16s value);

IppStatus ippsThreshold_LTAbsVal_32s_I(Ipp32s* pSrcDst, int len, Ipp32s level, Ipp32s value);

IppStatus ippsThreshold_LTVal_16s(const Ipp16s* pSrc, Ipp16s* pDst, int len, Ipp16s level, Ipp16s value);

IppStatus ippsThreshold_LTVal_32f(const Ipp32f* pSrc, Ipp32f* pDst, int len, Ipp32f level, Ipp32f value);

IppStatus ippsThreshold_LTVal_64f(const Ipp64f* pSrc, Ipp64f* pDst, int len, Ipp64f level, Ipp64f value);

IppStatus ippsThreshold_LTVal_16sc(const Ipp16sc* pSrc, Ipp16sc* pDst, int len, Ipp16s level, Ipp16sc value);

IppStatus ippsThreshold_LTVal_32fc(const Ipp32fc* pSrc, Ipp32fc* pDst, int len, Ipp32f level, Ipp32fc value);

IppStatus ippsThreshold_LTVal_64fc(const Ipp64fc* pSrc, Ipp64fc* pDst, int len, Ipp64f level, Ipp64fc value);

IppStatus ippsThreshold_GTVal_16s(const Ipp16s* pSrc, Ipp16s* pDst, int len, Ipp16s level, Ipp16s value);

IppStatus ippsThreshold_GTVal_32f(const Ipp32f* pSrc, Ipp32f* pDst, int len, Ipp32f level, Ipp32f value);

IppStatus ippsThreshold_GTVal_64f(const Ipp64f* pSrc, Ipp64f* pDst, int len, Ipp64f level, Ipp64f value);

IppStatus ippsThreshold_GTVal_16sc(const Ipp16sc* pSrc, Ipp16sc* pDst, int len, Ipp16s level, Ipp16sc value);

IppStatus ippsThreshold_GTVal_32fc(const Ipp32fc* pSrc, Ipp32fc* pDst, int len, Ipp32f level, Ipp32fc value);

IppStatus ippsThreshold_GTVal_64fc(const Ipp64fc* pSrc, Ipp64fc* pDst, int len, Ipp64f level, Ipp64fc value);
```

```
IppStatus ippsThreshold_LTValGTVal_16s(const Ipp16s* pSrc, Ipp16s* pDst, int len,
Ipp16s levelLT, Ipp16s valueLT, Ipp16s levelGT, Ipp16s valueGT);

IppStatus ippsThreshold_LTValGTVal_32s(const Ipp32s* pSrc, Ipp32s* pDst, int len,
Ipp32s levelLT, Ipp32s valueLT, Ipp32s levelGT, Ipp32s valueGT);

IppStatus ippsThreshold_LTValGTVal_32f(const Ipp32f* pSrc, Ipp32f* pDst, int len,
Ipp32f levelLT, Ipp32f valueLT, Ipp32f levelGT, Ipp32f valueGT);

IppStatus ippsThreshold_LTValGTVal_64f(const Ipp64f* pSrc, Ipp64f* pDst, int len,
Ipp64f levelLT, Ipp64f valueLT, Ipp64f levelGT, Ipp64f valueGT);

IppStatus ippsThreshold_LTVal_16s_I(Ipp16s* pSrcDst, int len, Ipp16s level, Ipp16s
value);

IppStatus ippsThreshold_LTVal_32f_I(Ipp32f* pSrcDst, int len, Ipp32f level, Ipp32f
value);

IppStatus ippsThreshold_LTVal_64f_I(Ipp64f* pSrcDst, int len, Ipp64f level, Ipp64f
value);

IppStatus ippsThreshold_LTVal_16sc_I(Ipp16sc* pSrcDst, int len, Ipp16s level, Ipp16sc
value);

IppStatus ippsThreshold_LTVal_32fc_I(Ipp32fc* pSrcDst, int len, Ipp32f level, Ipp32fc
value);

IppStatus ippsThreshold_LTVal_64fc_I(Ipp64fc* pSrcDst, int len, Ipp64f level, Ipp64fc
value);

IppStatus ippsThreshold_GTVal_16s_I(Ipp16s* pSrcDst, int len, Ipp16s level, Ipp16s
value);

IppStatus ippsThreshold_GTVal_32f_I(Ipp32f* pSrcDst, int len, Ipp32f level, Ipp32f
value);

IppStatus ippsThreshold_GTVal_64f_I(Ipp64f* pSrcDst, int len, Ipp64f level, Ipp64f
value);

IppStatus ippsThreshold_GTVal_16sc_I(Ipp16sc* pSrcDst, int len, Ipp16s level, Ipp16sc
value);

IppStatus ippsThreshold_GTVal_32fc_I(Ipp32fc* pSrcDst, int len, Ipp32f level, Ipp32fc
value);

IppStatus ippsThreshold_GTVal_64fc_I(Ipp64fc* pSrcDst, int len, Ipp64f level, Ipp64fc
value);

IppStatus ippsThreshold_LTValGTVal_16s_I(Ipp16s* pSrcDst, int len, Ipp16s levelLT,
Ipp16s valueLT, Ipp16s levelGT, Ipp16s valueGT);

IppStatus ippsThreshold_LTValGTVal_32s_I(Ipp32s* pSrcDst, int len, Ipp32s levelLT,
Ipp32s valueLT, Ipp32s levelGT, Ipp32s valueGT);

IppStatus ippsThreshold_LTValGTVal_32f_I(Ipp32f* pSrcDst, int len, Ipp32f levelLT,
Ipp32f valueLT, Ipp32f levelGT, Ipp32f valueGT);

IppStatus ippsThreshold_LTValGTVal_64f_I(Ipp64f* pSrcDst, int len, Ipp64f levelLT,
Ipp64f valueLT, Ipp64f levelGT, Ipp64f valueGT);
```

Include Files

ipps.h

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pSrc`	Pointer to the source vector.
`pDst`	Pointer to the destination vector.
`pSrcDst`	Pointer to the source and destination vector for the in-place operation.
`len`	Number of elements in the vector.
`level`	Value used to limit each element of `pSrc` or `pSrcDst`.This argument must always be real. For complex versions, it must be positive and represent magnitude.
`levelLT`	Low bound used to limit each element of `pSrc` or `pSrcDst` for the `ippsThreshold_LTValGTVal` function.
`levelGT`	Upper bound used to limit each element of `pSrc` or `pSrcDst` for the `ippsThreshold_LTValGTVal` function.
`value`	Value to be assigned to vector elements which are “less than” or “greater than” `level`.
`valueLT`	Value to be assigned to vector elements which are less than `levelLT` for the `ippsThreshold_LTValGTVal` function.
`valueGT`	Value to be assigned to vector elements which are greater than `levelGT` for the `ippsThreshold_LTValGTVal` function.

Description

These functions perform the threshold operation on the vector `pSrc` by limiting each element by the threshold level and replacing it with the specified value.

The in-place flavors of the function perform the threshold operation on the vector `pSrcDst` by limiting each element by the threshold value.

``ippsThreshold_LTAbsVal``. The `ippsThreshold_LTAbsVal` function substitutes each element of the source vector that is less by absolute value than specified `level` with the specified constant `value`.

The formula for `ippsThreshold_LTAbsVal` is:

```
if( ABS(x[i]) < level ) y[i] = value;
else y[i] = x[i];
```

``ippsThreshold_LTVal``. The function `ippsThreshold_LTVal` performs the operation “less than” and `level` is a lower bound for the input. The vector elements less than `level` are set to `value`.

The formula for `ippsThreshold_LTVal` is:

$$pDst[n] = \begin{cases} value, & pSrc[n] < level \\ pSrc[n], & otherwise \end{cases}$$

For complex versions of the function `ippsThreshold_LTVal`, the parameter `level` is always real.

The formula for complex `ippsThreshold_LTVal` is:

$$pDst[n] = \begin{cases} value, & abs(pSrc[n]) < level \\ pSrc[n], & otherwise \end{cases}$$

``ippsThreshold_GTVal``. The function `ippsThreshold_GTVal` performs the operation "greater than" and `level` is an upper bound for the input. The vector elements greater than `level` are set to `value`.

The formula for `ippsThreshold_GtVal` is:

$$pDst[n] = \begin{cases} value, & pSrc[n] > level \\ pSrc[n], & otherwise \end{cases}$$

For complex versions of the function `ippsThreshold_GTVal`, the parameter `level` is always real.

The formula for complex `ippsThreshold_GTVal` is:

$$pDst[n] = \begin{cases} value, & abs(pSrc[n]) > level \\ pSrc[n], & otherwise \end{cases}$$

``ippsThreshold_LTValGTVal``. The function `ippsThreshold_LTValGTVal` checks both the "less than" and "greater than" conditions. The parameter `levelLT` is a lower bound and the parameter `levelGT` is an upper bound for the input. The source vector elements less than `levelLT` are set to `valueLT`, and the source vector elements greater than `levelGT` are set to `valueGT`. The value of `levelLT` must be less than or equal to `levelGT`.

The formula for `ippsThreshold_LTValGTVal` is:

$$pDst[n] = \begin{cases} valueLT, & pSrc[n] < levelLT \\ pSrc[n], & levelLT \le pSrc[n] \le levelGT \\ valueGT, & pSrc[n] > levelGT \end{cases}$$

For all complex versions, `level` must be positive and represent a magnitude.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when the pSrc, pDst, or pSrcDst pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.
ippStsThresholdErr	Indicates an error when levelLT is greater than levelGT.
ippStsThreshNegLevelErr	Indicates an error when level for the complex version is negative (see appendix A "Handling of Special Cases" for more information).

Threshold_LTInv

Computes the inverse of vector elements after limiting their magnitudes by the given lower bound.

Syntax

```
IppStatus ippsThreshold_LTInv_32f(const Ipp32f* pSrc, Ipp32f* pDst, int len, Ipp32f level);
IppStatus ippsThreshold_LTInv_64f(const Ipp64f* pSrc, Ipp64f* pDst, int len, Ipp64f level);
IppStatus ippsThreshold_LTInv_32fc(const Ipp32fc* pSrc, Ipp32fc* pDst, int len, Ipp32f level);
IppStatus ippsThreshold_LTInv_64fc(const Ipp64fc* pSrc, Ipp64fc* pDst, int len, Ipp64f level);
IppStatus ippsThreshold_LTInv_32f_I(Ipp32f* pSrcDst, int len, Ipp32f level);
IppStatus ippsThreshold_LTInv_64f_I(Ipp64f* pSrcDst, int len, Ipp64f level);
IppStatus ippsThreshold_LTInv_32fc_I(Ipp32fc* pSrcDst, int len, Ipp32f level);
IppStatus ippsThreshold_LTInv_64fc_I(Ipp64fc* pSrcDst, int len, Ipp64f level);
```

Include Files

ipps.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

pSrc	Pointer to the source vector.
pDst	Pointer to the destination vector.

pSrcDst	Pointer to the source and destination vector for the in-place operation.
len	Number of elements in the vector.
level	Value used to limit each element of pSrc or pSrcDst.This argument must always be real and positive.

Description

This function computes the inverse of elements of the vector pSrc and stores the result in pDst. The computation occurs after first limiting the magnitude of each element by the threshold value level.

The in-place flavors of ippsThreshold_LTInv compute the inverse of elements of the vector pSrcDst and store the result in pSrcDst. The computation occurs after first limiting the magnitude of each element by the threshold value level.

The threshold operation is performed to avoid division by zero. Since level represents a magnitude, it is always real and must be positive. The formula for ippsThreshold_LTInv is the following:

$$pDst[n] = \begin{cases} \frac{1}{level}, & abs(pSrc[n]) = 0 \\ \frac{abs(pSrc[n])}{pSrc[n] \cdot level}, & 0 < abs(pSrc[n]) < level \\ \frac{1}{pSrc[n]}, & otherwise \end{cases}$$

If the function encounters zero-valued vector elements and level is also 0 (see appendix A "Handling of Special Cases"), the output value is set to Inf (infinity), but operation execution is not aborted:

$$pDst[n] = \begin{cases} Inf, & pSrc[n] = 0 \\ \frac{1}{pSrc[n]}, & otherwise \end{cases}$$

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when the pSrc, pDst, or pSrcDst pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to zero.
ippStsThreshNegLevelErr	Indicates an error when level is negative.

ippStsInvZero	Indicates a warning when `level` and a vector element are equal to zero. Operation execution is not aborted. The value of the destination vector element is `Inf`.

Example

Threshold_LtInv:

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

#include <stdio.h>
#include "ipp.h"

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define
check_sts(st)
                                  \
    if ((st) !=
ippStsNoErr)
                          \
        goto exitLine; /* Go to Exit if Intel(R) Integrated Performance Primitives (Intel(R)
IPP) function returned status different from \
                           ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main()
{
    Ipp32f pVals[] = {1, 3, 5, 7, 18, 24, 35, 48};
    Ipp32f pOutVals[8];
    IppStatus status;
    int i;

    printf("Source vector\n");
    for (i = 0; i < 8; i++)
        printf("%f ", pVals[i]);

    check_sts(status = ippsThreshold_LTInv_32f(pVals, pOutVals, 8, 2));

    printf("Result values vector\n");
```

```
    for (i = 0; i < 8; i++)
        printf("%f ", pOutVals[i]);

    EXIT_MAIN
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

Threshold_LtInv_I:

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

#include <stdio.h>
#include "ipp.h"

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define
check_sts(st)
                                  \
    if ((st) !=
ippStsNoErr)
                          \
        goto exitLine; /* Go to Exit if Intel(R) Integrated Performance Primitives (Intel(R)
IPP) function returned status different from \
                           ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main()
{
    Ipp32f pVals[] = {1, 3, 5, 7, 18, 24, 35, 48};
    IppStatus status;
    int i;

    printf("Source vector\n");
    for (i = 0; i < 8; i++)
        printf("%f ", pVals[i]);

    check_sts(status = ippsThreshold_LTInv_32f_I(pVals, 8, 2));

    printf("Result values vector\n");
    for (i = 0; i < 8; i++)
```

```
        printf("%f ", pVals[i]);

    EXIT_MAIN
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

CartToPolar

Converts the elements of a complex vector to polar coordinate form.

Syntax

```
IppStatus ippsCartToPolar_32f(const Ipp32f* pSrcRe, const Ipp32f* pSrcIm, Ipp32f*
pDstMagn, Ipp32f* pDstPhase, int len);

IppStatus ippsCartToPolar_64f(const Ipp64f* pSrcRe, const Ipp64f* pSrcIm, Ipp64f*
pDstMagn, Ipp64f* pDstPhase, int len);

IppStatus ippsCartToPolar_32fc(const Ipp32fc* pSrc, Ipp32f* pDstMagn, Ipp32f*
pDstPhase, int len);

IppStatus ippsCartToPolar_64fc(const Ipp64fc* pSrc, Ipp64f* pDstMagn, Ipp64f*
pDstPhase, int len);

IppStatus ippsCartToPolar_16sc_Sfs(const Ipp16sc* pSrc, Ipp16s* pDstMagn, Ipp16s*
pDstPhase, int len, int magnScaleFactor, int phaseScaleFactor);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pSrc`	Pointer to the source vector.
`pSrcRe`	Pointer to the source vector which stores the real components of Cartesian X,Y pairs.
`pSrcIm`	Pointer to the source vector which stores the imaginary components of Cartesian X,Y pairs.
`pDstMagn`	Pointer to the vector which stores the magnitude (radius) component of the elements of the vector `pSrc`.
`pDstPhase`	Pointer to the vector which stores the phase (angle) component of the elements of the vector `pSrc` in radians. Phase values are in the range (-π, π].
`len`	Number of elements in the vector.

magnScaleFactor	Integer scale factor for the magnitude component, refer to Integer Scaling.
phaseScaleFactor	Integer scale factor for the phase component, refer to Integer Scaling.

Description

This function converts the elements of a complex input vector pSrc or the complex input vector whose real and imaginary components are specified in the vectors pSrcRe and pSrcIm, respectively, to polar coordinate form, and stores the magnitude (radius) component of each element in the vector pDstMagn and the phase (angle) component of each element in the vector pDstPhase.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to zero.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

#include <stdio.h>
#include "ipp.h"
#include <math.h>

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define
check_sts(st)
                                    \
    if ((st) !=
ippStsNoErr)
                          \
        goto exitLine; /* Go to Exit if Intel(R) Integrated Performance Primitives (Intel(R)
IPP) function returned status different from \
                           ippStsNoErr */
```

```
/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main()
{
    int i;
    IppStatus status;
    Ipp64f cart[6], magn[4], phase[4];
    for (i = 0; i < 6; ++i)
        cart[i] = sin(IPP_2PI * i / 8);

    printf("Source:\n");
    printf("\nre= ");
    for (i = 0; i < 4; i++)
        printf("%f ", cart[i]);
    printf("\nim= ");
    for (i = 2; i < 6; i++)
        printf("%f ", cart[i]);

    check_sts(status = ippsCartToPolar_64f(cart, cart + 2, magn, phase, 4));

    printf("\nResult:");
    printf("\nmagn= ");
    for (i = 0; i < 4; i++)
        printf("%f ", magn[i]);
    printf("\nphase= ");
    for (i = 0; i < 4; i++)
        printf("%f ", phase[i]);

    EXIT_MAIN
    printf("\nExit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

PolarToCart

Converts the polar form magnitude/phase pairs stored in input vectors to Cartesian coordinate form.

Syntax

```
IppStatus ippsPolarToCart_32f(const Ipp32f* pSrcMagn, const Ipp32f* pSrcPhase, Ipp32f*
pDstRe, Ipp32f* pDstIm, int len);

IppStatus ippsPolarToCart_64f(const Ipp64f* pSrcMagn, const Ipp64f* pSrcPhase, Ipp64f*
pDstRe, Ipp64f* pDstIm, int len);

IppStatus ippsPolarToCart_32fc(const Ipp32f* pSrcMagn, const Ipp32f* pSrcPhase,
Ipp32fc* pDst, int len);

IppStatus ippsPolarToCart_64fc(const Ipp64f* pSrcMagn, const Ipp64f* pSrcPhase,
Ipp64fc* pDst, int len);

IppStatus ippsPolarToCart_16sc_Sfs(const Ipp16s* pSrcMagn, const Ipp16s* pSrcPhase,
Ipp16sc* pDst, int len, int magnScaleFactor, int phaseScaleFactor);
```

Include Files

ipps.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

pSrcMagn	Pointer to the source vector which stores the magnitude (radius) components of the elements in polar coordinate form.
pSrcPhase	Pointer to the vector which stores the phase (angle) components of the elements in polar coordinate form in radians.
pDst	Pointer to the resulting vector which stores the complex pairs in Cartesian coordinates (X + iY).
pDstRe	Pointer to the resulting vector which stores the real components of Cartesian X,Y pairs.
pDstIm	Pointer to the resulting vector which stores the imaginary components of Cartesian X,Y pairs.
len	Number of elements in the vectors.
magnScaleFactor	Integer scale factor for the magnitude component, refer to Integer Scaling.
phaseScaleFactor	Integer scale factor for the phase component, refer to Integer Scaling.

Description

This function converts the polar form magnitude/phase pairs stored in the input vectors pSrcMagn and pSrcPhase into a complex vector and stores the results in the vector pDst, or stores the real components of the result in the vector pDstRe and the imaginary components in the vector pDstIm.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.

MaxOrder

Computes the maximum order of a vector.

Syntax

```
IppStatus ippsMaxOrder_16s(const Ipp16s* pSrc, int len, int* pOrder);
IppStatus ippsMaxOrder_32s(const Ipp32s* pSrc, int len, int* pOrder);
IppStatus ippsMaxOrder_32f(const Ipp32f* pSrc, int len, int* pOrder);
IppStatus ippsMaxOrder_64f(const Ipp64f* pSrc, int len, int* pOrder);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pSrc`	Pointer to the source vector.
`len`	Number of elements in the vector.
`pOrder`	Pointer to the result value.

Description

This function finds the maximum binary number in elements of the exponent vector `pSrc`, and stores the result in `pOrder`.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when the `pSrc` or `pOrder` pointer is `NULL`.
`ippStsSizeErr`	Indicates an error when `len` is less than or equal to 0.
`ippStsNanArg`	Indicates a warning when `NaN` is encountered in the input data vector.

Flip

Reverses the order of elements in a vector.

Syntax

```
IppStatus ippsFlip_8u(const Ipp8u* pSrc, Ipp8u* pDst, int len);
IppStatus ippsFlip_16u(const Ipp16u* pSrc, Ipp16u* pDst, int len);
IppStatus ippsFlip_32f(const Ipp32f* pSrc, Ipp32f* pDst, int len);
IppStatus ippsFlip_64f(const Ipp64f* pSrc, Ipp64f* pDst, int len);
IppStatus ippsFlip_32fc(const Ipp32fc* pSrc, Ipp32fc* pDst, int len);
```

```
IppStatus ippsFlip_64fc(const Ipp64fc* pSrc, Ipp64fc* pDst, int len);
IppStatus ippsFlip_16u_I(Ipp16u* pSrcDst, int len);
IppStatus ippsFlip_8u_I(Ipp8u* pSrcDst, int len);
IppStatus ippsFlip_32f_I(Ipp32f* pSrcDst, int len);
IppStatus ippsFlip_64f_I(Ipp64f* pSrcDst, int len);
IppStatus ippsFlip_32fc_I(Ipp32fc* pSrcDst, int len);
IppStatus ippsFlip_64fc_I(Ipp64fc* pSrcDst, int len);
```

Platform-aware functions:

```
IppStatus ippsFlip_32f_L(const Ipp32f* pSrc, Ipp32f* pDst, IppSizeL len);
IppStatus ippsFlip_64f_L(const Ipp64f* pSrc, Ipp64f* pDst, IppSizeL len);
IppStatus ippsFlip_32fc_L(const Ipp32fc* pSrc, Ipp32fc* pDst, IppSizeL len);
IppStatus ippsFlip_64fc_L(const Ipp64fc* pSrc, Ipp64fc* pDst, IppSizeL len);
IppStatus ippsFlip_32f_I_L(Ipp32f* pSrcDst, IppSizeL len);
IppStatus ippsFlip_64f_I_L(Ipp64f* pSrcDst, IppSizeL len);
IppStatus ippsFlip_32fc_I_L(Ipp32fc* pSrcDst, IppSizeL len);
IppStatus ippsFlip_64fc_I_L(Ipp64fc* pSrcDst, IppSizeL len);
```

Include Files

`ipps.h`

Flavors with the `_L` suffix: `ipps_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pSrc`	Pointer to the source vector.
`pDst`	Pointer to the destination vector.
`pSrcDst`	Pointer to the source and destination vector for the in-place operation.
`len`	Number of elements in the vector.

Description

This function stores the elements of a source vector `pSrc` to a destination vector `pDst` in reverse order according to the following formula:

```
pDst[n] =pSrc[len- n-1], n = 0 ..len-1
```

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.

FindNearestOne

Finds an element of the table which is closest to the specified value.

Syntax

```
IppStatus ippsFindNearestOne_16u(Ipp16u inpVal, Ipp16u* pOutVal, int* pOutIndex, const Ipp16u *pTable, int tblLen);
```

Include Files

ipps.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

inpVal	Reference value.
pOutVal	Pointer to the output value.
pOutIndex	Pointer to the output index.
pTable	Pointer to the table for searching.
tblLen	Number of elements in the table.

Description

This function searches through the table pTable for an element which is closest to the specified reference value inpVal. The resulting element and its index are stored in pOutVal and pOutIndex, respectively. The table elements must satisfy the condition $pTable[n] \le pTable[n+1]$. The function uses the following distance criterion for determining the table closest element closest: $\min(|inpVal - pTable[n]|)$.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one of the specified pointers is NULL.

ippStsSizeErr	Indicates an error when tblLen is less than or equal to 0.

FindNearest

Finds table elements that are closest to the elements of the specified vector.

Syntax

```
IppStatus ippsFindNearest_16u(const Ipp16u* pVals, Ipp16u* pOutVals, int* pOutIndexes,
int len, const Ipp16u *pTable, int tblLen);
```

Include Files

ipps.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

pVals	Pointer to the vector containing reference values.
pOutVals	Pointer to the output vector.
pOutIndexes	Pointer to the array that stores output indexes.
len	Number of elements in the input vector.
pTable	Pointer to the table for searching.
tblLen	Number of elements in the table.

Description

This function searches through the table pTable for elements which are closest to the reference elements of the input vector pVals. The resulting elements and their indexes are stored in pOutVals and pOutIndexes, respectively. The table elements must satisfy the condition pTable[n]≤pTable[n+1]. The function uses the following distance criterion for determining the table element closest to pVals[k] : min(|pVals[k] – pTable[n]|).

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when tblLen or len is less than or equal to zero.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

#include <stdio.h>
#include "ipp.h"

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define
check_sts(st)
                                  \
    if ((st) !=
ippStsNoErr)
                          \
        goto exitLine; /* Go to Exit if Intel(R) Integrated Performance Primitives (Intel(R)
IPP) function returned status different from \
                           ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main()
{
    Ipp16u pVals[] = {12, 6, 78};
    Ipp16u pTable[] = {1, 3, 5, 7, 18, 24, 35, 48};
    Ipp16u pOutVals[3];
    int pOutIndexes[3];
    IppStatus status;
    int i;

    printf("Source table\n");
    for (i = 0; i < 8; i++)
        printf("%d ", pTable[i]);
    printf("\nSource vector\n");
    for (i = 0; i < 3; i++)
        printf("%d ", pVals[i]);

    check_sts(status = ippsFindNearest_16u(pVals, pOutVals, pOutIndexes, 3, pTable, 8));

    printf("\nResult values vector\n");
    for (i = 0; i < 3; i++)
        printf("%d ", pOutVals[i]);
```

```
    printf("\nResult indexes vector\n");
    for (i = 0; i < 3; i++)
        printf("%d ", pOutIndexes[i]);

    EXIT_MAIN
    printf("\nExit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

Windowing Functions

This chapter describes several of the windowing functions commonly used in signal processing. A window is a mathematical function by which a signal is multiplied to improve the characteristics of some subsequent analysis. Windows are commonly used in FFT-based spectral analysis.

- Understanding Window Functions
- WinBartlett
- WinBlackman
- WinHamming
- WinHann
- WinKaiser

Understanding Window Functions

The Intel IPP provides the following functions to generate window samples:

- Bartlett windowing function
- Blackman family of windowing functions
- Hamming windowing function
- WinHann windowing function
- WinKaiser windowing function

These functions generate the window samples and multiply them into an existing signal. To obtain the window samples themselves, initialize the vector argument to the unity vector before calling the window function.

If you want to multiply different frames of a signal by the same window multiple times, it is better to first calculate the window by calling one of the windowing functions (`ippsWinHann`, for example) on a vector with all elements set to 1.0. Then use one of the vector multiplication functions (`ippsMul`, for example) to multiply the window into the signal each time a new set of input samples is available. This avoids repeatedly calculating the window samples. This is illustrated in the following code example.

Example

```
void multiFrameWin( void ) {
     Ipp32f win[LEN], x[LEN], X[LEN];
     IppsFFTSpec_R_32f* ctx;
     ippsSet_32f( 1, win, LEN );
     ippsWinHann_32f_I( win, LEN );
     /// ... initialize FFT context
     while(1 ){
        /// ... get x signal
        ///
        ippsMul_32f_I( win, x, LEN );
        ippsFFTFwd_RToPack_32f( x, X, ctx, 0 );
     }
}
```

For more information on windowing, see: [Jack89], section 7.3, *Windows in Spectrum Analysis*; [Jack89], section 9.1, *Window-Function Technique*; and [Mit93], section 16-2, *Fourier Analysis of Finite-Time Signals*. For more information on these references, see also the Bibliography at the end of this document.

WinBartlett

Multiplies a vector by a Bartlett windowing function.

Syntax

```
IppStatus ippsWinBartlett_16s(const Ipp16s* pSrc, Ipp16s* pDst, int len);
IppStatus ippsWinBartlett_32f(const Ipp32f* pSrc, Ipp32f* pDst, int len);
IppStatus ippsWinBartlett_64f(const Ipp64f* pSrc, Ipp64f* pDst, int len);
IppStatus ippsWinBartlett_16sc(const Ipp16sc* pSrc, Ipp16sc* pDst, int len);
IppStatus ippsWinBartlett_32fc(const Ipp32fc* pSrc, Ipp32fc* pDst, int len);
IppStatus ippsWinBartlett_64fc(const Ipp64fc* pSrc, Ipp64fc* pDst, int len);
IppStatus ippsWinBartlett_16s_I(Ipp16s* pSrcDst, int len);
IppStatus ippsWinBartlett_32f_I(Ipp32f* pSrcDst, int len);
IppStatus ippsWinBartlett_64f_I(Ipp64f* pSrcDst, int len);
IppStatus ippsWinBartlett_16sc_I(Ipp16sc* pSrcDst, int len);
IppStatus ippsWinBartlett_32fc_I(Ipp32fc* pSrcDst, int len);
IppStatus ippsWinBartlett_64fc_I(Ipp64fc* pSrcDst, int len);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pSrc`	Pointer to the source vector.
`pDst`	Pointer to the destination vector.
`pSrcDst`	Pointer to the source and destination vector for the in-place operation.
`len`	Number of elements in the vector.

Description

This function multiplies the vector `pSrc` by the Bartlett (triangle) window, and stores the result in `pDst`.

The in-place flavors of `ippsWinBartlett` multiply the `pSrcDst` by the Bartlett (triangle) window and store the result in `pSrcDst`.

The complex types multiply both the real and imaginary parts of the vector by the same window.

The Bartlett window is defined as follows:

$$w_{bartlett}(n) = \begin{cases} \frac{2n}{len-1}, & 0 \le n \le \frac{len-1}{2} \\ 2 - \frac{2n}{len-1}, & \frac{len-1}{2} < n \le len-1 \end{cases}$$

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when the pSrc, pDst, or pSrcDst pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than 3.

Example

The example below shows how to use the function ippsWinBartlett_32f_I.

```
void bartlett(void) {
      Ipp32f x[8];
      ippsSet_32f(1, x, 8);
      ippsWinBartlett_32f_I(x, 8);
      printf_32f("bartlett (half) =", x, 4, ippStsNoErr);
}
```

Output:

```
bartlett (half) =  0.000000 0.285714 0.571429 0.857143
```

Matlab* Analog: `>> b = bartlett(8); b(1:4)'`

WinBlackman

Multiplies a vector by a Blackman windowing function.

Syntax

```
IppStatus ippsWinBlackman_16s(const Ipp16s* pSrc, Ipp16s* pDst, int len, Ipp32f alpha);
IppStatus ippsWinBlackman_16sc(const Ipp16sc* pSrc, Ipp16sc* pDst, int len, Ipp32f alpha);
IppStatus ippsWinBlackman_32f(const Ipp32f* pSrc, Ipp32f* pDst, int len, Ipp32f alpha);
IppStatus ippsWinBlackman_32fc(const Ipp32fc* pSrc, Ipp32fc* pDst, int len, Ipp32f alpha);
IppStatus ippsWinBlackman_64f(const Ipp64f* pSrc, Ipp64f* pDst, int len, Ipp64f alpha);
IppStatus ippsWinBlackman_64fc(const Ipp64fc* pSrc, Ipp64fc* pDst, int len, Ipp64f alpha);
IppStatus ippsWinBlackmanStd_16s(const Ipp16s* pSrc, Ipp16s* pDst, int len);
IppStatus ippsWinBlackmanStd_16sc(const Ipp16sc* pSrc, Ipp16sc* pDst, int len);
```

```
IppStatus ippsWinBlackmanStd_32f(const Ipp32f* pSrc, Ipp32f* pDst, int len);
IppStatus ippsWinBlackmanStd_32fc(const Ipp32fc* pSrc, Ipp32fc* pDst, int len);
IppStatus ippsWinBlackmanStd_64f(const Ipp64f* pSrc, Ipp64f* pDst, int len);
IppStatus ippsWinBlackmanStd_64fc(const Ipp64fc* pSrc, Ipp64fc* pDst, int len);
IppStatus ippsWinBlackmanOpt_16s(const Ipp16s* pSrc, Ipp16s* pDst, int len);
IppStatus ippsWinBlackmanOpt_16sc(const Ipp16sc* pSrc, Ipp16sc* pDst, int len);
IppStatus ippsWinBlackmanOpt_32f(const Ipp32f* pSrc, Ipp32f* pDst, int len);
IppStatus ippsWinBlackmanOpt_32fc(const Ipp32fc* pSrc, Ipp32fc* pDst, int len);
IppStatus ippsWinBlackmanOpt_64f(const Ipp64f* pSrc, Ipp64f* pDst, int len);
IppStatus ippsWinBlackmanOpt_64fc(const Ipp64fc* pSrc, Ipp64fc* pDst, int len);
IppStatus ippsWinBlackman_16s_I(Ipp16s* pSrcDst, int len, Ipp32f alpha);
IppStatus ippsWinBlackman_16sc_I(Ipp16sc* pSrcDst, int len, Ipp32f alpha);
IppStatus ippsWinBlackman_32f_I(Ipp32f* pSrcDst, int len, Ipp32f alpha);
IppStatus ippsWinBlackman_32fc_I(Ipp32fc* pSrcDst, int len, Ipp32f alpha);
IppStatus ippsWinBlackman_64f_I(Ipp64f* pSrcDst, int len, Ipp64f alpha);
IppStatus ippsWinBlackman_64fc_I(Ipp64fc* pSrcDst, int len, Ipp64f alpha);
IppStatus ippsWinBlackmanOpt_16s_I(Ipp16s* pSrcDst, int len);
IppStatus ippsWinBlackmanOpt_16sc_I(Ipp16sc* pSrcDst, int len);
IppStatus ippsWinBlackmanOpt_32f_I(Ipp32f* pSrcDst, int len);
IppStatus ippsWinBlackmanOpt_32fc_I(Ipp32fc* pSrcDst, int len);
IppStatus ippsWinBlackmanOpt_64f_I(Ipp64f* pSrcDst, int len);
IppStatus ippsWinBlackmanOpt_64fc_I(Ipp64fc* pSrcDst, int len);
IppStatus ippsWinBlackmanStd_16s_I(Ipp16s* pSrcDst, int len);
IppStatus ippsWinBlackmanStd_16sc_I(Ipp16sc* pSrcDst, int len);
IppStatus ippsWinBlackmanStd_32f_I(Ipp32f* pSrcDst, int len);
IppStatus ippsWinBlackmanStd_32fc_I(Ipp32fc* pSrcDst, int len);
IppStatus ippsWinBlackmanStd_64f_I(Ipp64f* pSrcDst, int len);
IppStatus ippsWinBlackmanStd_64fc_I(Ipp64fc* pSrcDst, int len);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

pSrc	Pointer to the source vector.
pDst	Pointer to the destination vector.
pSrcDst	Pointer to the source and destination vector for the in-place operation.
alpha	Adjustable parameter associated with the Blackman windowing equation.
len	Number of elements in the vector

Description

These functions multiply the vector pSrc by the Blackman window, and store the result in pDst.

The in-place flavors of ippsWinBlackman multiply the vector pSrcDst by the Blackman window, and store the result in pSrcDst.

The complex types multiply both the real and imaginary parts of the vector by the same window. The functions for the Blackman family of windows are defined below.

``ippsWinBlackman``. The function ippsWinBlackman allows the application to specify alpha. The Blackman window is defined as follows:

$$w_{blackman}(n) = \frac{alpha+1}{2} - 0.5\cos\left(\frac{2\pi n}{len-1}\right) - \frac{alpha}{2}\cos\left(\frac{4\pi n}{len-1}\right)$$

``ippsWinBlackmanStd``. The standard Blackman window is provided by the function ippsWinBlackmanStd, which simply multiplies a vector by a Blackman window with the standard value of alpha shown below:

alpha = -0.16

``ippsWinBlackmanOpt``. The function ippsWinBlackmanOpt provides a modified window that has a 30 dB/octave roll-off by multiplying a vector by a Blackman window with the optimal value of alpha shown below:

$$alpha = -\frac{0.5}{1+\cos\frac{2\pi}{len-1}}$$

The minimum len is equal to 4. For large len, the optimal alpha converges asymptotically to the asymptotic alpha; the application can use the asymptotic value of alpha shown below:

alpha = -0.25

Return Values

ippStsNoErr	Indicates no error.

ippStsNullPtrErr	Indicates an error when the pSrc, pDst, or pSrcDst pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than 4 for the function ippsWinBlackmanOpt and less than 3 for all other functions of the family.

Example

The example below shows how to use the function ippsWinBlackmanStd_32f_I

```
void blackman(void) {
      Ipp32f x[8];
      ippsSet_32f(1, x, 8);
      ippsWinBlackmanStd_32f_I(x, 8);
      printf_32f("blackman (half) =", x, 4, ippStsNoErr);
}
```

Output:

```
blackman(half) = 0.000000 0.090453 0.459183 0.920364
```

Matlab* Analog: `>> b = blackman(8)'; b(1:4)`

WinHamming

Multiplies a vector by a Hamming windowing function.

Syntax

```
IppStatus ippsWinHamming_16s(const Ipp16s* pSrc, Ipp16s* pDst, int len);
IppStatus ippsWinHamming_32f(const Ipp32f* pSrc, Ipp32f* pDst, int len);
IppStatus ippsWinHamming_64f(const Ipp64f* pSrc, Ipp64f* pDst, int len);
IppStatus ippsWinHamming_16sc(const Ipp16sc* pSrc, Ipp16sc* pDst, int len);
IppStatus ippsWinHamming_32fc(const Ipp32fc* pSrc, Ipp32fc* pDst, int len);
IppStatus ippsWinHamming_64fc(const Ipp64fc* pSrc, Ipp64fc* pDst, int len);
IppStatus ippsWinHamming_16s_I(Ipp16s* pSrcDst, int len);
IppStatus ippsWinHamming_32f_I(Ipp32f* pSrcDst, int len);
IppStatus ippsWinHamming_64f_I(Ipp64f* pSrcDst, int len);
IppStatus ippsWinHamming_16sc_I(Ipp16sc* pSrcDst, int len);
IppStatus ippsWinHamming_32fc_I(Ipp32fc* pSrcDst, int len);
IppStatus ippsWinHamming_64fc_I(Ipp64fc* pSrcDst, int len);
```

Include Files

ipps.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

pSrc	Pointer to the source vector.
pDst	Pointer to the destination vector.
pSrcDst	Pointer to the source and destination vector for the in-place operation.
len	Number of elements in the vector.

Description

This function multiplies the vector pSrc by the Hamming window and stores the result in pDst.

The in-place flavors of ippsWinHamming multiply the vector pSrcDst by the Hamming window and store the result in pSrcDst.

The complex types multiply both the real and imaginary parts of the vector by the same window. The Hamming window is defined as follows:

$$w_{hamming}(n) = 0.54 - 0.46\cos\left(\frac{2\pi n}{len-1}\right)$$

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when the pSrc, pDst, or pSrcDst pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than 3.

Example

The example below shows how to use the function ippsWinHamming_32f_I.

```
void hamming(void) {
      Ipp32f x[8];
      ippsSet_32f(1, x, 8);
      ippsWinHamming_32f_I(x, 8);
      printf_32f("hamming(half) =", x, 4, ippStsNoErr);
}
```

Output:

```
hamming(half) =  0.080000 0.253195 0.642360 0.954446
```

Matlab* Analog: `>> b = hamming(8); b(1:4)'`

WinHann

Multiplies a vector by a Hann windowing function.

Syntax

```
IppStatus ippsWinHann_16s(const Ipp16s* pSrc, Ipp16s* pDst, int len);
```

```
IppStatus ippsWinHann_16sc(const Ipp16sc* pSrc, Ipp16sc* pDst, int len);
IppStatus ippsWinHann_32f(const Ipp32f* pSrc, Ipp32f* pDst, int len);
IppStatus ippsWinHann_32fc(const Ipp32fc* pSrc, Ipp32fc* pDst, int len);
IppStatus ippsWinHann_64f(const Ipp64f* pSrc, Ipp64f* pDst, int len);
IppStatus ippsWinHann_64fc(const Ipp64fc* pSrc, Ipp64fc* pDst, int len);
IppStatus ippsWinHann_16s_I(Ipp16s* pSrcDst, int len);
IppStatus ippsWinHann_16sc_I(Ipp16sc* pSrcDst, int len);
IppStatus ippsWinHann_32f_I(Ipp32f* pSrcDst, int len);
IppStatus ippsWinHann_32fc_I(Ipp32fc* pSrcDst, int len);
IppStatus ippsWinHann_64f_I(Ipp64f* pSrcDst, int len);
IppStatus ippsWinHann_64fc_I(Ipp64fc* pSrcDst, int len);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pSrc`	Pointer to the source vector.
`pDst`	Pointer to the destination vector.
`pSrcDst`	Pointer to the source and destination vector for the in-place operation.
`len`	Number of elements in the vector.

Description

This function multiplies the vector `pSrc` by the Hann window and stores the result in `pDst`.

The in-place flavors of `ippsWinHann` multiply the vector `pSrcDst` by the Hann window and store the result in `pSrcDst`.

The complex types multiply both the real and imaginary parts of the vector by the same window. The Hann window is defined as follows:

$$w_{hann}(n) = 0.5 - 0.5\cos\left(\frac{2\pi n}{len-1}\right)$$

Return Values

`ippStsNoErr`	Indicates no error.

ippStsNullPtrErr	Indicates an error when the pSrc, pDst, or pSrcDst pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than 3.

Example

The example below shows how to use the function ippsWinHann_32f_I

```
void hann(void) {
      Ipp32f x[8];
      ippsSet_32f(1, x, 8);
      ippsWinHann_32f_I(x, 8);
      printf_32f("hann(half) =", x, 4, ippStsNoErr);
}
```

Output:

```
 hann(half) =  0.000000 0.188255 0.611260 0.950484
Matlab* Analog:
    >> N = 8; n = 0:N-1; 0.5*(1-cos(2*pi*n/(N-1)))
```

WinKaiser

Multiplies a vector by a Kaiser windowing function.

Syntax

```
IppStatus ippsWinKaiser_16s(const Ipp16s* pSrc, Ipp16s* pDst, int len, Ipp32f alpha);
IppStatus ippsWinKaiser_32f(const Ipp32f* pSrc, Ipp32f* pDst, int len, Ipp32f alpha);
IppStatus ippsWinKaiser_64f(const Ipp64f* pSrc, Ipp64f* pDst, int len, Ipp64f alpha);
IppStatus ippsWinKaiser_16sc(const Ipp16sc* pSrc, Ipp16sc* pDst, int len, Ipp32f
alpha);
IppStatus ippsWinKaiser_32fc(const Ipp32fc* pSrc, Ipp32fc* pDst, int len, Ipp32f
alpha);
IppStatus ippsWinKaiser_64fc(const Ipp64fc* pSrc, Ipp64fc* pDst, int len, Ipp64f
alpha);
IppStatus ippsWinKaiser_16s_I(Ipp16s* pSrcDst, int len, Ipp32f alpha);
IppStatus ippsWinKaiser_32f_I(Ipp32f* pSrcDst, int len, Ipp32f alpha);
IppStatus ippsWinKaiser_64f_I(Ipp64f* pSrcDst, int len, Ipp64f alpha);
IppStatus ippsWinKaiser_16sc_I(Ipp16sc* pSrcDst, int len, Ipp32f alpha);
IppStatus ippsWinKaiser_32fc_I(Ipp32fc* pSrcDst, int len, Ipp32f alpha);
IppStatus ippsWinKaiser_64fc_I(Ipp64fc* pSrcDst, int len, Ipp64f alpha);
```

Include Files

ipps.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

pSrc	Pointer to the source vector.
pDst	Pointer to the destination vector.
pSrcDst	Pointer to the source and destination vector for the in-place operation.
alpha	Adjustable parameter associated with the Kaiser windowing equation.
len	Number of elements in the vector.

Description

This function multiplies the vector pSrc by the Kaiser window, and stores the result in pDst.

The in-place flavors of ippsWinKaiser multiply the vector pSrcDst by the Kaiser window and store the result in pSrcDst.

``ippsWinKaiser``. The function ippsWinKaiser allows the application to specify alpha. The function multiplies both real and imaginary parts of the complex vector by the same window. The Kaiser family of windows are defined as follows:

$$w_{kaiser}(n) = \frac{I_0\left(alpha\sqrt{\left(\frac{len-1}{2}\right)^2-\left(n-\left(\frac{len-1}{2}\right)\right)^2}\right)}{I_0\left(alpha\left(\frac{len-1}{2}\right)\right)}$$

Here $I_0()$ is the modified zero-order Bessel function of the first kind.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when the pDst, pSrc, or pSrcDst pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than 1.
ippStsHugeWinErr	Indicates an error when the Kaiser window is too big.

Example

The example below shows how to use the function ippsWinKaiser_32f_I.

```
void kaiser(void) {
     Ipp32f x[8];
     IppStatus st;
     ippsSet_32f(1, x, 8);
```

```
    st = ippsWinKaiser_32f_I( x, 8, 1.0f );
    printf_32f("kaiser(half) =", x, 4, ippStsNoErr);
}
```

Output:

```
kaiser(half) =  0.135534 0.429046 0.755146 0.970290
```

Matlab* Analog: `>> kaiser(8,7/2)'`

Statistical Functions

This section describes the Intel IPP functions that compute the vector measure values: maximum, minimum, mean, and standard deviation.

- Sum
- Max
- MaxIndx
- MaxAbs
- MaxAbsIndx
- Min
- MinIndx
- MinAbs
- MinAbsIndx
- MinMax
- MinMaxIndx
- ReplaceNAN
- Mean
- StdDev
- MeanStdDev
- Norm
- NormDiff
- DotProd
- MaxEvery, MinEvery
- ZeroCrossing
- CountInRange

Sum

Computes the sum of the elements of a vector.

Syntax

```
IppStatus ippsSum_32f(const Ipp32f* pSrc, int len, Ipp32f* pSum, IppHintAlgorithm
hint);

IppStatus ippsSum_32fc(const Ipp32fc* pSrc, int len, Ipp32fc* pSum, IppHintAlgorithm
hint);

IppStatus ippsSum_64f(const Ipp64f* pSrc, int len, Ipp64f* pSum);

IppStatus ippsSum_64fc(const Ipp64fc* pSrc, int len, Ipp64fc* pSum);

IppStatus ippsSum_16s_Sfs(const Ipp16s* pSrc, int len, Ipp16s* pSum, int scaleFactor);

IppStatus ippsSum_32s_Sfs(const Ipp32s* pSrc, int len, Ipp32s* pSum, int scaleFactor);

IppStatus ippsSum_16s32s_Sfs(const Ipp16s* pSrc, int len, Ipp32s* pSum, int
scaleFactor);
```

```
IppStatus ippsSum_16sc_Sfs(const Ipp16sc* pSrc, int len, Ipp16sc* pSum, int
scaleFactor);

IppStatus ippsSum_16sc32sc_Sfs(const Ipp16sc* pSrc, int len, Ipp32sc* pSum, int
scaleFactor);
```

Platform-aware functions:

```
IppStatus ippsSum_32f_L(const Ipp32f* pSrc, IppSizeL len, Ipp32f* pSum,
IppHintAlgorithm hint);

IppStatus ippsSum_32fc_L(const Ipp32fc* pSrc, IppSizeL len, Ipp32fc* pSum,
IppHintAlgorithm hint);

IppStatus ippsSum_64f_L(const Ipp64f* pSrc, IppSizeL len, Ipp64f* pSum);

IppStatus ippsSum_64fc_L(const Ipp64fc* pSrc, IppSizeL len, Ipp64fc* pSum);
```

Include Files

`ipps.h`

Flavors with the `_L` suffix: `ipps_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pSrc`	Pointer to the source vector.
`pSum`	Pointer to the output result.
`len`	Number of elements in the vector.
`hint`	Suggests using specific code. The possible values for the `hint` argument are described in IppHintAlgorithm enum Type.
`scaleFactor`	Scale factor, refer to Integer Scaling.

Description

This function computes the sum of the elements of the vector `pSrc` and stores the result in `pSum`.

The sum of the elements of `pSrc` is defined by the formula:

$$sum = \sum_{n=0}^{len-1} pSrc[n]$$

The `hint` argument suggests using specific code, either faster but less accurate calculation, or more accurate but slower calculation.

When computing the sum of integer numbers, the output result can exceed the data range and become saturated. To get a precise result, use the scale factor. The scaling is performed in accordance with the `scaleFactor` value.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when the `pSum` or `pSrc` pointer is `NULL`.
`ippStsSizeErr`	Indicates an error when `len` is less than or equal to 0.

Example

The example below shows how to use the function `ippsSum`.

```
void sum(void) {
      Ipp16s x[4] = {-32768, 32767, 32767, 32767}, sm;
      ippsSum_16s_Sfs(x, 4, &sm, 1);
      printf_16s("sum =", &sm, 1, ippStsNoErr);
}
```

Output:

```
  sum = 32766
Matlab* Analog:
    >> x = [-32768, 32767, 32767, 32767]; sum(x)/2
```

Max

Returns the maximum value of a vector.

Syntax

```
IppStatus ippsMax_16s(const Ipp16s* pSrc, int len, Ipp16s* pMax);
IppStatus ippsMax_32s(const Ipp32s* pSrc, int len, Ipp32s* pMax);
IppStatus ippsMax_32f(const Ipp32f* pSrc, int len, Ipp32f* pMax);
IppStatus ippsMax_64f(const Ipp64f* pSrc, int len, Ipp64f* pMax);
```

Platform-aware functions:

```
IppStatus ippsMax_32f_L(const Ipp32f* pSrc, IppSizeL len, Ipp32f* pMax);
IppStatus ippsMax_64f_L(const Ipp64f* pSrc, IppSizeL len, Ipp64f* pMax);
```

Include Files

`ipps.h`

Flavors with the `_L` suffix: `ipps_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

pSrc	Pointer to the source vector.
pMax	Pointer to the output result.
len	Number of elements in the vector

Description

This function returns the maximum value of the input vector pSrc, and stores the result in pMax.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when the pMax or pSrc pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.

MaxIndx

Returns the maximum value of a vector and the index of the maximum element.

Syntax

```
IppStatus ippsMaxIndx_16s(const Ipp16s* pSrc, int len, Ipp16s* pMax, int* pIndx);
IppStatus ippsMaxIndx_32s(const Ipp32s* pSrc, int len, Ipp32s* pMax, int* pIndx);
IppStatus ippsMaxIndx_32f(const Ipp32f* pSrc, int len, Ipp32f* pMax, int* pIndx);
IppStatus ippsMaxIndx_64f(const Ipp64f* pSrc, int len, Ipp64f* pMax, int* pIndx);
```

Platform-aware functions:

```
IppStatus ippsMaxIndx_32f_L(const Ipp32f* pSrc, IppSizeL len, Ipp32f* pMax, IppSizeL* pIndx);
IppStatus ippsMaxIndx_64f_L(const Ipp64f* pSrc, IppSizeL len, Ipp64f* pMax, IppSizeL* pIndx);
```

Include Files

ipps.h

Flavors with the _L suffix: ipps_l.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

pSrc	Pointer to the source vector.

pMax	Pointer to the output result.
len	Number of elements in the vector.
pIndx	Pointer to the index value of the maximum element.

Description

This function returns the maximum value of the input vector pSrc, and stores the result in pMax. If pIndx is not a NULL pointer, the function returns the index of the maximum element and stores it in pIndx. If there are several equal maximum elements, the first index from the beginning is returned.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.

Example

The code example below demonstrates how to use the function ippsMaxIndx.

```
Ipp16s src[] = { 1, -2, 3, 8, -6 };
Ipp16s max;
int len = 5;
int  indx;


ippsMaxIndx_16s ( src, len, &max, &indx );
```

Result:

```
max = 8  indx = 3
```

MaxAbs

Returns the maximum absolute value of a vector.

Syntax

```
IppStatus ippsMaxAbs_16s(const Ipp16s* pSrc, int len, Ipp16s* pMaxAbs);
IppStatus ippsMaxAbs_32s(const Ipp32s* pSrc, int len, Ipp32s* pMaxAbs);
IppStatus ippsMaxAbs_32f(const Ipp32f* pSrc, int len, Ipp32f* pMaxAbs);
IppStatus ippsMaxAbs_64f(const Ipp64f* pSrc, int len, Ipp64f* pMaxAbs);
```

Platform-aware functions:

```
IppStatus ippsMaxAbs_32f_L(const Ipp32f* pSrc, IppSizeL len, Ipp32f* pMaxAbs);
IppStatus ippsMaxAbs_64f_L(const Ipp64f* pSrc, IppSizeL len, Ipp64f* pMaxAbs);
```

Include Files

ipps.h

Flavors with the _L suffix: ipps_l.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

pSrc	Pointer to the source vector.
pMaxAbs	Pointer to the output result.
len	Number of elements in the vector.

Description

This function returns the maximum absolute value of the input vector pSrc, and stores the result in pMaxAbs.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when the pMaxAbs or pSrc pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.

Example

The example below shows how to use the function ippsMaxAbs_16s.

```
Ipp16s src[5] = { 2, -8, -3, -1, 7 };
Ipp16s maxAbs;
ippsMaxAbs_16s ( src, 5, &maxAbs );
```

Result:

```
maxAbs = 8
```

MaxAbsIndx

Returns the maximum absolute value of a vector and the index of the corresponding element.

Syntax

```
IppStatus ippsMaxAbsIndx_16s(const Ipp16s* pSrc, int len, Ipp16s* pMaxAbs, int* pIndx);
IppStatus ippsMaxAbsIndx_32s(const Ipp32s* pSrc, int len, Ipp32s* pMaxAbs, int* pIndx);
IppStatus ippsMaxAbsIndx_32f(const Ipp32f* pSrc, int len, Ipp32f* pMaxAbs, int* pIndx);
IppStatus ippsMaxAbsIndx_64f(const Ipp64f* pSrc, int len, Ipp64f* pMaxAbs, int* pIndx);
```

Platform-aware functions:

```
IppStatus ippsMaxAbsIndx_32f_L(const Ipp32f* pSrc, IppSizeL len, Ipp32f* pMaxAbs,
IppSizeL* pIndx);

IppStatus ippsMaxAbsIndx_64f_L(const Ipp64f* pSrc, IppSizeL len, Ipp64f* pMaxAbs,
IppSizeL* pIndx);
```

Include Files

`ipps.h`

Flavors with the `_L` suffix: `ipps_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pSrc`	Pointer to the source vector.
`pMaxAbs`	Pointer to the output result.
`len`	Number of elements in the vector.
`pIndx`	Pointer to the index value of the maximum element.

Description

This function returns the maximum absolute value `pMaxAbs` of the input vector `pSrc`, and the index of the corresponding element `pIndx`. If there are several elements with the equal maximum absolute value, the first index from the beginning is returned.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error when `len` is less than or equal to 0.

Min

Returns the minimum value of a vector.

Syntax

```
IppStatus ippsMin_16s(const Ipp16s* pSrc, int len, Ipp16s* pMin);

IppStatus ippsMin_32s(const Ipp32s* pSrc, int len, Ipp32s* pMin);

IppStatus ippsMin_32f(const Ipp32f* pSrc, int len, Ipp32f* pMin);

IppStatus ippsMin_64f(const Ipp64f* pSrc, int len, Ipp64f* pMin);
```

Platform-aware functions:

```
IppStatus ippsMin_32f_L(const Ipp32f* pSrc, IppSizeL len, Ipp32f* pMin);
IppStatus ippsMin_64f_L(const Ipp64f* pSrc, IppSizeL len, Ipp64f* pMin);
```

Include Files

`ipps.h`

Flavors with the `_L` suffix: `ipps_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pSrc`	Pointer to the source vector.
`pMin`	Pointer to the output result.
`len`	Number of elements in the vector.

Description

This function returns the minimum value of the input vector `pSrc`, and stores the result in `pMin`.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when the `pMin` or `pSrc` pointer is `NULL`.
`ippStsSizeErr`	Indicates an error when `len` is less than or equal to 0.

Example

The example below shows how to use the function `ippsMin`.

```
Ipp16s src = { 1, -2, 3, 8, -6};
Ipp16s min;
int len = 5;
ippsMin_16s (src, len, &min );
```

Result:

```
min = -6
```

MinIndx

Returns the minimum value of a vector and the index of the minimum element.

Syntax

```
IppStatus ippsMinIndx_16s(const Ipp16s* pSrc, int len, Ipp16s* pMin, int* pIndx);
```

```
IppStatus ippsMinIndx_32s(const Ipp32s* pSrc, int len, Ipp32s* pMin, int* pIndx);
IppStatus ippsMinIndx_32f(const Ipp32f* pSrc, int len, Ipp32f* pMin, int* pIndx);
IppStatus ippsMinIndx_64f(const Ipp64f* pSrc, int len, Ipp64f* pMin, int* pIndx);
```

Platform-aware functions:

```
IppStatus ippsMinIndx_32f_L(const Ipp32f* pSrc, IppSizeL len, Ipp32f* pMin, IppSizeL* pIndx);
IppStatus ippsMinIndx_64f_L(const Ipp64f* pSrc, IppSizeL len, Ipp64f* pMin, IppSizeL* pIndx);
```

Include Files

`ipps.h`

Flavors with the `_L` suffix: `ipps_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pSrc`	Pointer to the source vector.
`pMin`	Pointer to the output result.
`len`	Number of elements in the vector.
`pIndx`	Pointer to the index value of the minimum element.

Description

This function returns the minimum value of the input vector `pSrc` and stores the result in `pMin`. If `pIndx` is not a `NULL` pointer, the function returns the index of the minimum element and stores it in `pIndx`. If there are several equal minimum elements, the first index from the beginning is returned.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when the `pMin` or `pSrc` pointer is `NULL`.
`ippStsSizeErr`	Indicates an error when `len` is less than or equal to 0.

MinAbs

Returns the minimum absolute value of a vector.

Syntax

```
IppStatus ippsMinAbs_16s(const Ipp16s* pSrc, int len, Ipp16s* pMinAbs);
```

```
IppStatus ippsMinAbs_32s(const Ipp32s* pSrc, int len, Ipp32s* pMinAbs);
IppStatus ippsMinAbs_16f(const Ipp16f* pSrc, int len, Ipp16f* pMinAbs);
IppStatus ippsMinAbs_32f(const Ipp32f* pSrc, int len, Ipp32f* pMinAbs);
IppStatus ippsMinAbs_64f(const Ipp64f* pSrc, int len, Ipp64f* pMinAbs);
```

Platform-aware functions:

```
IppStatus ippsMinAbs_32f_L(const Ipp32f* pSrc, IppSizeL len, Ipp32f* pMinAbs);
IppStatus ippsMinAbs_64f_L(const Ipp64f* pSrc, IppSizeL len, Ipp64f* pMinAbs);
```

Include Files

`ipps.h`

Flavors with the `_L` suffix: `ipps_l.h`

Parameters

`pSrc`	Pointer to the source vector.
`pMinAbs`	Pointer to the output result.
`len`	Number of elements in the vector.

Description

This function returns the minimum absolute value of the input vector `pSrc`, and stores the result in `pMinAbs`.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when the `pMinAbs` or `pSrc` pointer is `NULL`.
`ippStsSizeErr`	Indicates an error when `len` is less than or equal to 0.

MinAbsIndx

Returns the minimum absolute value of a vector and the index of the corresponding element.

Syntax

```
IppStatus ippsMinAbsIndx_16s(const Ipp16s* pSrc, int len, Ipp16s* pMinAbs, int* pIndx);
IppStatus ippsMinAbsIndx_32s(const Ipp32s* pSrc, int len, Ipp32s* pMinAbs, int* pIndx);
IppStatus ippsMinAbsIndx_32f(const Ipp32f* pSrc, int len, Ipp32f* pMinAbs, int* pIndx);
IppStatus ippsMinAbsIndx_64f(const Ipp64f* pSrc, int len, Ipp64f* pMinAbs, int* pIndx);
```

Platform-aware functions:

```
IppStatus ippsMinAbsIndx_32f_L(const Ipp32f* pSrc, IppSizeL len, Ipp32f* pMinAbs,
IppSizeL* pIndx);
```

```
IppStatus ippsMinAbsIndx_64f_L(const Ipp64f* pSrc, IppSizeL len, Ipp64f* pMinAbs,
IppSizeL* pIndx);
```

Include Files

`ipps.h`

Flavors with the `_L` suffix: `ipps_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pSrc`	Pointer to the source vector.
`pMinAbs`	Pointer to the output result.
`len`	Number of elements in the vector.
`pIndx`	Pointer to the index value of the corresponding element.

Description

This function returns the minimum absolute value `pMinAbs` of the input vector `pSrc`, and the index of the corresponding element `pIndx`. If there are several elements with equal maximum absolute value, the first index from the beginning is returned.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error when `len` is less than or equal to 0.

MinMax

Returns the maximum and minimum values of a vector.

Syntax

```
IppStatus ippsMinMax_8u(const Ipp8u* pSrc, int len, Ipp8u* pMin, Ipp8u* pMax);
IppStatus ippsMinMax_16u(const Ipp16u* pSrc, int len, Ipp16u* pMin, Ipp16u* pMax);
IppStatus ippsMinMax_16s(const Ipp16s* pSrc, int len, Ipp16s* pMin, Ipp16s* pMax);
IppStatus ippsMinMax_32u(const Ipp32u* pSrc, int len, Ipp32u* pMin, Ipp32u* pMax);
IppStatus ippsMinMax_32s(const Ipp32s* pSrc, int len, Ipp32s* pMin, Ipp32s* pMax);
IppStatus ippsMinMax_32f(const Ipp32f* pSrc, int len, Ipp32f* pMin, Ipp32f* pMax);
```

```
IppStatus ippsMinMax_64f(const Ipp64f* pSrc, int len, Ipp64f* pMin, Ipp64f* pMax);
```

Platform-aware functions:

```
IppStatus ippsMinMax_32f_L(const Ipp32f* pSrc, IppSizeL len, Ipp32f* pMin, Ipp32f* pMax);
```

```
IppStatus ippsMinMax_64f_L(const Ipp64f* pSrc, IppSizeL len, Ipp64f* pMin, Ipp64f* pMax);
```

Include Files

`ipps.h`

Flavors with the `_L` suffix: `ipps_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pSrc`	Pointer to the source vector.
`pMin`	Pointer to the minimum value.
`pMax`	Pointer to the maximum value.
`len`	Number of elements in the vector.

Description

This function returns the minimum and maximum values of the input vector `pSrc`, and stores the results in `pMin` and `pMax`, respectively.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when the `pMin` or `pSrc` pointer is `NULL`.
`ippStsSizeErr`	Indicates an error when `len` is less than or equal to 0.

MinMaxIndx

Returns the maximum and minimum values of a vector and the indexes of the corresponding elements.

Syntax

```
IppStatus ippsMinMaxIndx_8u(const Ipp8u* pSrc, int len, Ipp8u* pMin, int* pMinIndx, Ipp8u* pMax, int* pMaxIndx);
```

```
IppStatus ippsMinMaxIndx_16u(const Ipp16u* pSrc, int len, Ipp16u* pMin, int* pMinIndx, Ipp16u* pMax, int* pMaxIndx);
```

```
IppStatus ippsMinMaxIndx_16s(const Ipp16s* pSrc, int len, Ipp16s* pMin, int* pMinIndx,
Ipp16s* pMax, int* pMaxIndx);

IppStatus ippsMinMaxIndx_32u(const Ipp32u* pSrc, int len, Ipp32u* pMin, int* pMinIndx,
Ipp32u* pMax, int* pMaxIndx);

IppStatus ippsMinMaxIndx_32s(const Ipp32s* pSrc, int len, Ipp32s* pMin, int* pMinIndx,
Ipp32s* pMax, int* pMaxIndx);

IppStatus ippsMinMaxIndx_32f(const Ipp32f* pSrc, int len, Ipp32f* pMin, int* pMinIndx,
Ipp32f* pMax, int* pMaxIndx);

IppStatus ippsMinMaxIndx_64f(const Ipp64f* pSrc, int len, Ipp64f* pMin, int* pMinIndx,
Ipp64f* pMax, int* pMaxIndx);
```

Platform-aware functions:

```
IppStatus ippsMinMaxIndx_32f_L(const Ipp32f* pSrc, IppSizeL len, Ipp32f* pMin,
IppSizeL* pMinIndx, Ipp32f* pMax, IppSizeL* pMaxIndx);

IppStatus ippsMinMaxIndx_64f_L(const Ipp64f* pSrc, IppSizeL len, Ipp64f* pMin,
IppSizeL* pMinIndx, Ipp64f* pMax, IppSizeL* pMaxIndx);
```

Include Files

`ipps.h`

Flavors with the `_L` suffix: `ipps_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pSrc`	Pointer to the source vector.
`pMin`	Pointer to the minimum value.
`pMax`	Pointer to the maximum value.
`len`	Number of elements in the vector.
`pMinIndx`	Pointer to the index value of the minimum element.
`pMaxIndx`	Pointer to the index value of the maximum element.

Description

This function returns the minimum and maximum values of the input vector `pSrc` and stores the result in `pMin` and `pMax`, respectively. The function also returns the indexes of the minimum and maximum elements and stores them in `pMinIndx` and `pMaxIndx`, respectively. If there are several equal minimum or maximum elements, the first index from the beginning is returned.

Return Values

`ippStsNoErr`	Indicates no error.

ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.

ReplaceNAN

Replaces not-a-number (NaN) values of vector elements with a constant value.

Syntax

```
IppStatus ippsReplaceNAN_32f_I(Ipp32f* pSrcDst, int len, Ipp32f value);
IppStatus ippsReplaceNAN_64f_I(Ipp64f* pSrcDst, int len, Ipp64f value);
```

Include Files

ipps.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

pSrcDst	Pointer to the source and destination vector.
len	Number of elements in the vector.
value	Constant value to be assigned to NaN elements of the vector.

Description

This function replaces not-a-number (NaN) elements of the source vector with value, other vector elements remain unchanged:

```
pSrcDst[i] = (pSrcDst[i] == NAN) ? value : pSrcDst[i]
```

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pSrcDst is NULL.
ippStsSizeErr	Indicates an error when len is less than, or equal to zero.

Mean

Computes the mean value of a vector.

Syntax

```
IppStatus ippsMean_32f(const Ipp32f* pSrc, int len, Ipp32f* pMean, IppHintAlgorithm hint);
IppStatus ippsMean_32fc(const Ipp32fc* pSrc, int len, Ipp32fc* pMean, IppHintAlgorithm hint);
IppStatus ippsMean_64f(const Ipp64f* pSrc, int len, Ipp64f* pMean);
IppStatus ippsMean_64fc(const Ipp64fc* pSrc, int len, Ipp64fc* pMean);
IppStatus ippsMean_16s_Sfs(const Ipp16s* pSrc, int len, Ipp16s* pMean, int scaleFactor);
IppStatus ippsMean_32s_Sfs(const Ipp32s* pSrc,int len,Ipp32s* pMean,int scaleFactor);
IppStatus ippsMean_16sc_Sfs(const Ipp16sc* pSrc, int len, Ipp16sc* pMean, int scaleFactor);
```

Platform-aware functions:

```
IppStatus ippsMean_32f_L(const Ipp32f* pSrc, IppSizeL len, Ipp32f* pMean, IppHintAlgorithm hint);
IppStatus ippsMean_32fc_L(const Ipp32fc* pSrc, IppSizeL len, Ipp32fc* pMean, IppHintAlgorithm hint);
IppStatus ippsMean_64f_L(const Ipp64f* pSrc, IppSizeL len, Ipp64f* pMean);
IppStatus ippsMean_64fc_L(const Ipp64fc* pSrc, IppSizeL len, Ipp64fc* pMean);
```

Include Files

`ipps.h`

Flavors with the `_L` suffix: `ipps_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pSrc`	Pointer to the source vector.
`pMean`	Pointer to the output result.
`len`	Number of elements in the vector
`hint`	Suggests using specific code. The possible values for the `hint` argument are described in IppHintAlgorithm enum Type.
`scaleFactor`	Scale factor, refer to Integer Scaling.

Description

This function computes the mean (average) of the vector `pSrc`, and stores the result in `pMean`. The mean of `pSrc` is defined by the formula:

$$mean = \frac{1}{len} \sum_{n=0}^{len-1} pSrc[n]$$

The `hint` argument suggests using specific code, either faster but less accurate calculation, or more accurate but slower calculation.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when the `pMean` or `pSrc` pointer is `NULL`.
`ippStsSizeErr`	Indicates an error when `len` is less than or equal to 0.

Example

The example below shows how to use the function `ippsMean_32f`

```
void mean(void) {
      Ipp32f *x = ippsMalloc_32f(1000), mean;
      int i;
      for(i = 0; i<1000; ++i) x[i] = (float)rand() / RAND_MAX;
      ippsMean_32f(x, 1000, &mean, ippAlgHintFast);
      printf_32f("mean =", &mean, 1, ippStsNoErr);
      ippsFree(x);
}
```

Output:

```
 mean = 0.492591
Matlab* Analog:
    >> x = rand(1,1000); mean(x)
```

StdDev

Computes the standard deviation value of a vector.

Syntax

`IppStatus ippsStdDev_32f(const Ipp32f* pSrc, int len, Ipp32f* pStdDev, IppHintAlgorithm hint);`

`IppStatus ippsStdDev_64f(const Ipp64f* pSrc, int len, Ipp64f* pStdDev);`

`IppStatus ippsStdDev_16s_Sfs(const Ipp16s* pSrc, int len, Ipp16s* pStdDev, int scaleFactor);`

`IppStatus ippsStdDev_16s32s_Sfs(const Ipp16s* pSrc, int len, Ipp32s* pStdDev, int scaleFactor);`

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pSrc`	Pointer to the source vector.
`pStdDev`	Pointer to the output result.
`len`	Number of elements in the vector.
`hint`	Suggests using specific code. The possible values for the `hint` argument are described in IppHintAlgorithm enum Type.
`scaleFactor`	Scale factor, refer to Integer Scaling.

Description

This function computes the standard deviation of the input vector `pSrc`, and stores the result in `pStdDev`. The vector length can not be less than 2. The standard deviation of `pSrc` is defined by the unbiased estimate formula:

$$stdDev = \sqrt{\frac{\sum_{n=0}^{len-1} (pSrc[n] - mean(pSrc))^2}{len - 1}}$$

The `hint` argument suggests using specific code, either faster but less accurate calculation, or more accurate but slower calculation.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when the `pStdDev` or `pSrc` pointer is `NULL`.
`ippStsSizeErr`	Indicates an error when `len` is less than or equal to 1.

Example

The example below shows how to use the function `ippsStdDev_32f`.

```
void stdev(void) {
      Ipp32f *x = ippsMalloc_32f(1000), stdev;
      int i;
      for (i = 0; i<1000; ++i) x[i] = (float)rand() / RAND_MAX;
      ippsStdDev_32f(x, 1000, &stdev, ippAlgHintFast);
      printf_32f("stdev =", &stdev, 1, ippStsNoErr);
      ippsFree(x);
}
```

Output:

```
 stdev = 0.286813
Matlab* Analog:
    >> x = rand(1,1000); std(x)
```

MeanStdDev

Computes the mean value and the standard deviation value of a vector.

Syntax

```
IppStatus ippsMeanStdDev_32f(const Ipp32f* pSrc, int len, Ipp32f* pMean, Ipp32f* pStdDev, IppHintAlgorithm hint);
```

```
IppStatus ippsMeanStdDev_64f(const Ipp64f* pSrc, int len, Ipp64f* pMean, Ipp64f* pStdDev);
```

```
IppStatus ippsMeanStdDev_16s_Sfs(const Ipp16s* pSrc, int len, Ipp16s* pMean, Ipp16s* pStdDev, int scaleFactor);
```

```
IppStatus ippsMeanStdDev_16s32s_Sfs(const Ipp16s* pSrc, int len, Ipp32s* pMean, Ipp32s* pStdDev, int scaleFactor);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pSrc`	Pointer to the source vector.
`pMean`	Pointer to the output result - mean value.
`pStdDev`	Pointer to the output result - standard deviation.
`len`	Number of elements in the vector
`hint`	Suggests using specific code. The possible values for the `hint` argument are described in IppHintAlgorithm enum Type.
`scaleFactor`	Scale factor, refer to Integer Scaling.

Description

This function computes both the mean value and the standard deviation of the input vector `pSrc`, and stores the results in `pMean` and `pStdDev` respectively. The vector length can not be less than 2. The mean of `pSrc` is defined by the formula:

$$mean = \frac{1}{len} \sum_{n=0}^{len-1} pSrc[n]$$

The standard deviation of `pSrc` is defined by the unbiased estimate formula:

$$stdDev = \sqrt{\frac{\sum_{n=0}^{len-1} (pSrc[n] - mean(pSrc))^2}{len-1}}$$

The `hint` argument suggests using specific code, either faster but less accurate calculation, or more accurate but slower calculation.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error when `len` is less than or equal to 1.

Norm

Computes the C, L1, L2, or L2Sqr norm of a vector.

Syntax

```
IppStatus ippsNorm_Inf_32f(const Ipp32f* pSrc, int len, Ipp32f* pNorm);
IppStatus ippsNorm_Inf_64f(const Ipp64f* pSrc, int len, Ipp64f* pNorm);
IppStatus ippsNorm_Inf_16s32f(const Ipp16s* pSrc, int len, Ipp32f* pNorm);
IppStatus ippsNorm_Inf_32fc32f(const Ipp32fc* pSrc, int len, Ipp32f* pNorm);
IppStatus ippsNorm_Inf_64fc64f(const Ipp64fc* pSrc, int len, Ipp64f* pNorm);
IppStatus ippsNorm_L1_32f(const Ipp32f* pSrc, int len, Ipp32f* pNorm);
IppStatus ippsNorm_L1_64f(const Ipp64f* pSrc, int len, Ipp64f* pNorm);
IppStatus ippsNorm_L1_16s32f(const Ipp16s* pSrc, int len, Ipp32f* pNorm);
IppStatus ippsNorm_L1_32fc64f(const Ipp32fc* pSrc, int len, Ipp64f* pNorm);
IppStatus ippsNorm_L1_64fc64f(const Ipp64fc* pSrc, int len, Ipp64f* pNorm);
IppStatus ippsNorm_L2_32f(const Ipp32f* pSrc, int len, Ipp32f* pNorm);
IppStatus ippsNorm_L2_64f(const Ipp64f* pSrc, int len, Ipp64f* pNorm);
IppStatus ippsNorm_L2_16s32f(const Ipp16s* pSrc, int len, Ipp32f* pNorm);
```

```
IppStatus ippsNorm_L2_32fc64f(const Ipp32fc* pSrc, int len, Ipp64f* pNorm);

IppStatus ippsNorm_L2_64fc64f(const Ipp64fc* pSrc, int len, Ipp64f* pNorm);

IppStatus ippsNorm_L2Sqr_16s64s_Sfs(const Ipp16s* pSrc, int len, Ipp64s* pNorm, int
scaleFactor);

IppStatus ippsNorm_L2Sqr_16s64s_Sfs(const Ipp16s* pSrc, int len, Ipp64s* pNorm, int
scaleFactor);

IppStatus ippsNorm_Inf_16s32s_Sfs(const Ipp16s* pSrc, int len, Ipp32s* pNorm, int
scaleFactor);

IppStatus ippsNorm_L1_16s32s_Sfs(const Ipp16s* pSrc, int len, Ipp32s* pNorm, int
scaleFactor);

IppStatus ippsNorm_L1_16s64s_Sfs(const Ipp16s* pSrc, int len, Ipp64s* pNorm, int
scaleFactor);

IppStatus ippsNorm_L2_16s32s_Sfs(const Ipp16s* pSrc, int len, Ipp32s* pNorm, int
scaleFactor);
```

Platform-aware functions:

```
IppStatus ippsNorm_L2_32f_L(const Ipp32f* pSrc, IppSizeL len, Ipp32f* pNorm);

IppStatus ippsNorm_L2_64f_L(const Ipp64f* pSrc, IppSizeL len, Ipp64f* pNorm);

IppStatus ippsNorm_L2_32fc64f_L(const Ipp32fc* pSrc, IppSizeL len, Ipp64f* pNorm);

IppStatus ippsNorm_L2_64fc64f_L(const Ipp64fc* pSrc, IppSizeL len, Ipp64f* pNorm);
```

Include Files

`ipps.h`

Flavors with the `_L` suffix: `ipps_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pSrc`	Pointer to the source vector.
`pNorm`	Pointer to the output result.
`len`	Number of elements in the vector.
`scaleFactor`	Scale factor, refer to Integer Scaling.

Description

This function computes the `C`, `L1`, `L2`, or `L2Sqr` norm of the source vector `pSrc` and stores the result in `pNorm`.

``ippsNorm_Inf``. The function `ippsNorm_Inf` computes the `C` norm defined by the formula:

$$Norm_C = \max_{n=0}^{len-1} |pSrc[n]|$$

``ippsNorm_L1``. The function `ippsNorm_L1` computes the `L1` norm defined by the formula:

$$Norm_{L1} = \sum_{n=0}^{len-1} |pSrc[n]|$$

``ippsNorm_L2``. The function `ippsNorm_L2` computes the `L2` norm defined by the formula:

$$Norm_{L2} = \sqrt{\sum_{n=0}^{len-1} |pSrc[n]|^2}$$

``ippsNorm_L2Sqr``. The function `ippsNorm_L2Sqr` computes the `L2Sqr` norm defined as square of the `L2` norm.

Functions with `Sfs` suffixes perform scaling of the result value in accordance with the `scaleFactor` value.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when the `pSrc` or `pNorm` pointer is `NULL`.
`ippStsSizeErr`	Indicates an error when `len` is less than or equal to 0.

NormDiff

Computes the C, L1, L2, or L2Sqr norm of two vectors' difference.

Syntax

```
IppStatus ippsNormDiff_Inf_32f(const Ipp32f* pSrc1, const Ipp32f* pSrc2, int len,
Ipp32f* pNorm);
```

```
IppStatus ippsNormDiff_Inf_64f(const Ipp64f* pSrc1, const Ipp64f* pSrc2, int len,
Ipp64f* pNorm);
```

```
IppStatus ippsNormDiff_Inf_16s32f(const Ipp16s* pSrc1, const Ipp16s* pSrc2, int len,
Ipp32f* pNorm);
```

```
IppStatus ippsNormDiff_Inf_32fc32f(const Ipp32fc* pSrc1, const Ipp32fc* pSrc2, int len,
Ipp32f* pNorm);
```

```
IppStatus ippsNormDiff_Inf_64fc64f(const Ipp64fc* pSrc1, const Ipp64fc* pSrc2, int len,
Ipp64f* pNorm);
```

```
IppStatus ippsNormDiff_L1_32f(const Ipp32f* pSrc1, const Ipp32f* pSrc2, int len,
Ipp32f* pNorm);

IppStatus ippsNormDiff_L1_64f(const Ipp64f* pSrc1, const Ipp64f* pSrc2, int len,
Ipp64f* pNorm);

IppStatus ippsNormDiff_L1_16s32f(const Ipp16s* pSrc1, const Ipp16s* pSrc2, int len,
Ipp32f* pNorm);

IppStatus ippsNormDiff_L1_32fc64f(const Ipp32fc* pSrc1, const Ipp32fc* pSrc2, int len,
Ipp64f* pNorm);

IppStatus ippsNormDiff_L1_64fc64f(const Ipp64fc* pSrc1, const Ipp64fc* pSrc2, int len,
Ipp64f* pNorm);

IppStatus ippsNormDiff_L2_32f(const Ipp32f* pSrc1, const Ipp32f* pSrc2, int len,
Ipp32f* pNorm);

IppStatus ippsNormDiff_L2_64f(const Ipp64f* pSrc1, const Ipp64f* pSrc2, int len,
Ipp64f* pNorm);

IppStatus ippsNormDiff_L2_16s32f(const Ipp16s* pSrc1, const Ipp16s* pSrc2, int len,
Ipp32f* pNorm);

IppStatus ippsNormDiff_L2_32fc64f(const Ipp32fc* pSrc1, const Ipp32fc* pSrc2, int len,
Ipp64f* pNorm);

IppStatus ippsNormDiff_L2_64fc64f(const Ipp64fc* pSrc1, const Ipp64fc* pSrc2, int len,
Ipp64f* pNorm);

IppStatus ippsNormDiff_L2Sqr_16s64s_Sfs(const Ipp16s* pSrc1, const Ipp16s* pSrc2, int
len, Ipp64s* pNorm, int scaleFactor);

IppStatus ippsNormDiff_Inf_16s32s_Sfs(const Ipp16s* pSrc1, const Ipp16s* pSrc2, int
len, Ipp32s* pNorm, int scaleFactor);

IppStatus ippsNormDiff_L1_16s32s_Sfs(const Ipp16s* pSrc1, const Ipp16s* pSrc2, int len,
Ipp32s* pNorm, int scaleFactor);

IppStatus ippsNormDiff_L1_16s64s_Sfs(const Ipp16s* pSrc1, const Ipp16s* pSrc2, int len,
Ipp64s* pNorm, int scaleFactor);

IppStatus ippsNormDiff_L2_16s32s_Sfs(const Ipp16s* pSrc1, const Ipp16s* pSrc2, int len,
Ipp32s* pNorm, int scaleFactor);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pSrc1`, `pSrc2`	Pointers to the two source vectors; `pSrc2` can be `NULL`.
`pNorm`	Pointer to the output result.

len	Number of elements in the vector.
scaleFactor	Scale factor, refer to Integer Scaling.

Description

This function computes the C, L1, L2, or L2Sqr norm of the source vectors' difference, and stores the result in pNorm.

``ippsNormDiff_Inf``. The function ippsNormDiff_Inf computes the C norm defined by the formula:

$$Norm_{Inf} = \max_{n=0}^{len-1} |pSrc1[n] - pSrc2[n]|$$

``ippsNormDiff_L1``. The function ippsNormDiff_L1 computes the L1 norm defined by the formula:

$$Norm_{L1} = \sum_{n=0}^{len-1} |pSrc1[n] - pSrc2[n]|$$

``ippsNormDiff_L2``. The function ippsNormDiff_L2 computes the L2 norm defined by the formula:

$$Norm_{L2} = \sqrt{\sum_{n=0}^{len-1} |pSrc1[n] - pSrc2[n]|^2}$$

``ippsNormDiff_L2Sqr``. The function ippsNormDiff_L2Sqr computes the L2Sqr norm defined as square of the L2 norm.

Functions with Sfs suffixes perform scaling of the result value in accordance with the scaleFactor value.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when the pSrc1, pSrc2, or pNorm pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.

Example

The example below shows how to use the function ippsNormDiff.

```
int norm( void ) {
      Ipp16s x[LEN];
      Ipp32f Norm[3];
      IppStatus st;
      int i;
      for( i=0; i<LEN; ++i ) x[i] = (Ipp16s)rand();
```

```
      ippsNormDiff_Inf_16s32f( x, 0, LEN, Norm );
      ippsNormDiff_L1_16s32f( x, 0, LEN, Norm+1 );
      st = ippsNormDiff_L2_16s32f( x, 0, LEN, Norm+2 );
      printf_32f("Norm (oo,L1,L2) =", Norm, 3, st );
      return Norm[2] <= Norm[1] && Norm[1] <= LEN*Norm[0];
}
```

Output:

```
 Norm (oo,L1,L2) =  31993.000000 1526460.000000 180270.781250
Matlab* analog:
    >> x = 32767*rand(1,100);norm(x,inf),norm(x,1),norm(x,2)
```

DotProd

Computes the dot product of two vectors.

Syntax

```
IppStatus ippsDotProd_32f(const Ipp32f* pSrc1, const Ipp32f* pSrc2, int len, Ipp32f*
pDp);

IppStatus ippsDotProd_32fc(const Ipp32fc* pSrc1, const Ipp32fc* pSrc2, int len,
Ipp32fc* pDp);

IppStatus ippsDotProd_32f32fc(const Ipp32f* pSrc1, const Ipp32fc* pSrc2, int len,
Ipp32fc* pDp);

IppStatus ippsDotProd_32f64f(const Ipp32f* pSrc1, const Ipp32f* pSrc2, int len, Ipp64f*
pDp);

IppStatus ippsDotProd_32fc64fc(const Ipp32fc* pSrc1, const Ipp32fc* pSrc2, int len,
Ipp64fc* pDp);

IppStatus ippsDotProd_32f32fc64fc(const Ipp32f* pSrc1, const Ipp32fc* pSrc2, int len,
Ipp64fc* pDp);

IppStatus ippsDotProd_64f(const Ipp64f* pSrc1, const Ipp64f* pSrc2, int len, Ipp64f*
pDp);

IppStatus ippsDotProd_64fc(const Ipp64fc* pSrc1, const Ipp64fc* pSrc2, int len,
Ipp64fc* pDp);

IppStatus ippsDotProd_64f64fc(const Ipp64f* pSrc1, const Ipp64fc* pSrc2, int len,
Ipp64fc* pDp);

IppStatus ippsDotProd_16s64s(const Ipp16s* pSrc1, const Ipp16s* pSrc2, int len, Ipp64s*
pDp);

IppStatus ippsDotProd_16sc64sc(const Ipp16sc* pSrc1, const Ipp16sc* pSrc2, int len,
Ipp64sc* pDp);

IppStatus ippsDotProd_16s16sc64sc(const Ipp16s* pSrc1, const Ipp16sc* pSrc2, int len,
Ipp64sc* pDp);

IppStatus ippsDotProd_16s32f(const Ipp16s* pSrc1, const Ipp16s* pSrc2, int len, Ipp32f*
pDp);

IppStatus ippsDotProd_32s_Sfs(const Ipp32s* pSrc1, const Ipp32s* pSrc2, int len,
Ipp32s* pDp, int scaleFactor);
```

```
IppStatus ippsDotProd_16s32s_Sfs(const Ipp16s* pSrc1, const Ipp16s* pSrc2, int len,
Ipp32s* pDp, int scaleFactor);

IppStatus ippsDotProd_16s32s32s_Sfs(const Ipp16s* pSrc1, const Ipp32s* pSrc2, int len,
Ipp32s* pDp, int scaleFactor);
```

Platform-aware functions:

```
IppStatus ippsDotProd_32f64f_L(const Ipp32f* pSrc1, const Ipp32f* pSrc2, IppSizeL len,
Ipp64f* pDp);

IppStatus ippsDotProd_32fc64fc_L(const Ipp32fc* pSrc1, const Ipp32fc* pSrc2, IppSizeL
len, Ipp64fc* pDp);

IppStatus ippsDotProd_32f32fc64fc_L(const Ipp32f* pSrc1, const Ipp32fc* pSrc2, IppSizeL
len, Ipp64fc* pDp);

IppStatus ippsDotProd_64f_L(const Ipp64f* pSrc1, const Ipp64f* pSrc2, IppSizeL len,
Ipp64f* pDp);

IppStatus ippsDotProd_64fc_L(const Ipp64fc* pSrc1, const Ipp64fc* pSrc2, IppSizeL len,
Ipp64fc* pDp);

IppStatus ippsDotProd_64f64fc_L(const Ipp64f* pSrc1, const Ipp64fc* pSrc2, IppSizeL
len, Ipp64fc* pDp);
```

Note that ippsDotProd_32f_L, ippsDotProd_32fc_L, ippsDotProd_32f32fc_L are not provided because the appropriate accuracy cannot be achieved with the 32-bit floating-point data types and IppSizeL length. Please use ippsDotProd_32f64f, ippsDotProd_32fc64fc, ippsDotProd_32f32fc64fc instead.

Include Files

`ipps.h`

`ipps_l.h` for platform-aware functions.

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pSrc1`	Pointer to the first vector to compute the dot product value.
`pSrc2`	Pointer to the second vector to compute the dot product value.
`pDp`	Pointer to the output result.
`len`	Number of elements in the vector.
`scaleFactor`	Scale factor, refer to Integer Scaling.

Description

This function computes the dot product (scalar value) of two vectors, `pSrc1` and `pSrc2`, and stores the result in `pDp`.

The computation is performed as follows:

$$dp = \sum_{n=0}^{len-1} pSrc1[n] * pSrc2[n]$$

To compute the dot product of complex data, use the function `ippsConj` to conjugate one of the operands. The vectors `pSrc1` and `pSrc2` must be of equal length.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when the `pDp`, `pSrc1`, or `pSrc` 2 pointer is `NULL`.
`ippStsSizeErr`	Indicates an error when `len` is less than or equal to 0.

Example

The example below shows how to use the function `ippsDotProd_64f` to verify orthogonality of the sine and cosine functions. Two vectors are orthogonal to each other when the dot product of the two vectors is zero.

```
void dotprod(void) {
      Ipp64f x[10], dp;
      int n;
      for (n = 0; n<10; ++n) x[n] = sin(IPP_2PI * n / 8);
      ippsDotProd_64f(x, x+2, 8, &dp);
      printf_64f("dp =", &dp, 1, ippStsNoErr);
}
```

Output:

```
dp = 0.000000
```

Matlab* Analog: `>> n = 0:9; x = sin(2*pi*n/8); a = x(1:8); b = x(3:10); a*b'`

MaxEvery, MinEvery

Computes maximum or minimum value for each pair of elements of two vectors.

Syntax

```
IppStatus ippsMaxEvery_8u(const Ipp8u* pSrc1, const Ipp8u* pSrc2, Ipp8u* pDst, Ipp32u len);

IppStatus ippsMaxEvery_16u(const Ipp16u* pSrc1, const Ipp16u* pSrc2, Ipp16u* pDst, Ipp32u len);

IppStatus ippsMaxEvery_32f(const Ipp32f* pSrc1, const Ipp32f* pSrc2, Ipp32f* pDst, Ipp32u len);

IppStatus ippsMaxEvery_64f(const Ipp64f* pSrc1, const Ipp64f* pSrc2, Ipp64f* pDst, Ipp32u len);

IppStatus ippsMinEvery_8u(const Ipp8u* pSrc1, const Ipp8u* pSrc2, Ipp8u* pDst, Ipp32u len);
```

```
IppStatus ippsMinEvery_16u(const Ipp16u* pSrc1, const Ipp16u* pSrc2, Ipp16u* pDst,
Ipp32u len);
IppStatus ippsMinEvery_32f(const Ipp32f* pSrc1, const Ipp32f* pSrc2, Ipp32f* pDst,
Ipp32u len);
IppStatus ippsMinEvery_64f(const Ipp64f* pSrc1, const Ipp64f* pSrc2, Ipp64f* pDst,
Ipp32u len);
IppStatus ippsMaxEvery_8u_I(const Ipp8u* pSrc, Ipp8u* pSrcDst, int len);
IppStatus ippsMaxEvery_16u_I(const Ipp16u* pSrc, Ipp16u* pSrcDst, int len);
IppStatus ippsMaxEvery_16s_I(const Ipp16s* pSrc, Ipp16s* pSrcDst, int len);
IppStatus ippsMaxEvery_32s_I(const Ipp32s* pSrc, Ipp32s* pSrcDst, int len);
IppStatus ippsMaxEvery_32f_I(const Ipp32f* pSrc, Ipp32f* pSrcDst, int len);
IppStatus ippsMaxEvery_64f_I(const Ipp64f* pSrc, Ipp64f* pSrcDst, Ipp32u len);
IppStatus ippsMinEvery_8u_I(const Ipp8u* pSrc, Ipp8u* pSrcDst, int len);
IppStatus ippsMinEvery_16u_I(const Ipp16u* pSrc, Ipp16u* pSrcDst, int len);
IppStatus ippsMinEvery_16s_I(const Ipp16s* pSrc, Ipp16s* pSrcDst, int len);
IppStatus ippsMinEvery_32s_I(const Ipp32s* pSrc, Ipp32s* pSrcDst, int len);
IppStatus ippsMinEvery_32f_I(const Ipp32f* pSrc, Ipp32f* pSrcDst, int len);
IppStatus ippsMinEvery_64f_I(const Ipp64f* pSrc, Ipp64f* pSrcDst, Ipp32u len);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pSrc`, `pSrc1`, `pSrc2`	Pointer to the input vector.
`pSrcDst`, `pDst`	Pointer to the vector which stores the result.
`len`	Number of elements in the vector.

Description

This function computes the maximum between each pair of corresponding elements of two input vectors and stores the result in `pSrcDst`.

This function computes minimum values likewise.

Return Values

`ippStsNoErr`	Indicates no error.

ippStsNullPtrErr	Indicates an error when the pSrc or pSrcDst pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.

ZeroCrossing

Computes specific zero crossing measure.

Syntax

```
IppStatus ippsZeroCrossing_16s32f(const Ipp16s* pSrc, Ipp32u len, Ipp32f* pValZCR, IppsZCType zcType);
```

```
IppStatus ippsZeroCrossing_32f(const Ipp32f* pSrc, Ipp32u len, Ipp32f* pValZCR, IppsZCType zcType);
```

Include Files

ipps.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

pSrc	Pointer to the source vector.
len	Number of elements in the vector.
pValZCR	Pointer to the output value of the zero crossing measure.
zcType	Type of the zero crossing measure, possible values are ippsZCR, ippsZCXor or ippsZCC.

Description

This function computes specific zero crossing measure according to the parameter zcType. The result of zero crossing measurement is stored in pValZCR. The calculations are performed in accordance with the formulas below.

If zcType = ippZCR, the function uses the following formula:

$$\sum_{i=1}^{len-1} (x_i \cdot x_{i-1}) < 0$$

If zcType = ippZCXor, the function uses the following formula:

$$\sum_{i=1}^{len-1} sign(x_i) \wedge sign(x_{i-1}) \text{ , where } sign(x) = \begin{cases} 0: & x>0, \quad x = +0 \\ 1: & x<0, \quad x = -0 \end{cases}$$

If `zcType = ippZCC`, the function uses the following formula:

$$\sum_{i=1}^{len-1} \frac{abs(sign(x_i) - sign(x_{i-1}))}{2} \text{, where } sign(x) = \begin{cases} 1: & x>0 \\ 0: & x = 0. \\ -1: & x<0 \end{cases}$$

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when the `pSrc` or `pValZCR` pointer is `NULL`.
`ippStsRangeErr`	Indicates an error when `zcType` has an invalid value.

CountInRange

Computes the number of elements of the vector whose values are in the specified range.

Syntax

```
IppStatus ippsCountInRange_32s(const Ipp32s* pSrc, int len, int* pCounts, Ipp32s lowerBound, Ipp32s upperBound);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pSrc`	Pointer to the first input vector.
`pCounts`	Pointer to the second input vector which stores the result.
`len`	Number of elements in the vector.

lowerBound	Lower boundary of the range.
upperBound	Upper boundary of the range.

Description

This function computes the number of elements of the vector pSrc whose values are in the range lowerBound<pSrc[n] <upperBound. The total number of such elements are stored in the pCounts.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when the pSrc or pSCounts pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.

Sampling Functions

The functions described in this section manipulate signal samples. Sampling functions are used to change the sampling rate of the input signal and thus to obtain the signal vector of a required length. The functions perform the following operations:

- Insert zero-valued samples between neighboring samples of a signal (up-sample).
- Remove samples from between neighboring samples of a signal (down-sample).

The upsampling and downsampling functions are used by some filtering functions described in Chapter 6.

- SampleUp
- SampleDown

SampleUp

Up-samples a signal, conceptually increasing its sampling rate by an integer factor.

Syntax

```
IppStatus ippsSampleUp_16s (const Ipp16s* pSrc, int srcLen, Ipp16s* pDst, int* pDstLen,
int factor, int* pPhase);

IppStatus ippsSampleUp_32f (const Ipp32f* pSrc, int srcLen, Ipp32f* pDst, int* pDstLen,
int factor, int* pPhase);

IppStatus ippsSampleUp_64f (const Ipp64f* pSrc, int srcLen, Ipp64f* pDst, int* pDstLen,
int factor, int* pPhase);

IppStatus ippsSampleUp_16sc (const Ipp16sc* pSrc, int srcLen, Ipp16sc* pDst, int*
pDstLen, int factor, int* pPhase);

IppStatus ippsSampleUp_32fc (const Ipp32fc* pSrc, int srcLen, Ipp32fc* pDst, int*
pDstLen, int factor, int* pPhase);

IppStatus ippsSampleUp_64fc (const Ipp64fc* pSrc, int srcLen, Ipp64fc* pDst, int*
pDstLen, int factor, int* pPhase);
```

Include Files

ipps.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

pSrc	Pointer to the source array (the signal to be up-sampled).
srcLen	Number of samples in the source array pSrc.
pDst	Pointer to the destination array.
pDstLen	Pointer to the length of the destination array pDst.
factor	Factor by which the signal is up-sampled. That is, factor – 1 zeros are inserted after each sample of the source array pSrc.
pPhase	Pointer to the input phase value which determines where each sample from pSrc lies within each output block of factor samples in pDst. The value of pPhase is required to be in the range [0; factor – 1].

Description

This function up-samples the srcLen – length source array pSrc by factor factor with phase pPhase, and stores the result in the array pDst, ignoring its length value by the pDstLen address.

Up-sampling inserts factor – 1 zeros between each sample of pSrc. The pPhase argument determines where each sample from the input array lies within each output block of factor samples. The value of pPhase is required to be in the range [0; factor – 1].

For example, if the input phase is 0, then every factor samples of the destination array begin with the corresponding source array sample, the other factor– 1 samples are equal to 0. The length of the destination array is stored by the pDstLen address.

The pPhase value is the phase of an source array sample. It is also a returned output phase which can be used as an input phase for the first sample in the next block to process. Use pPhase for block mode processing to get a continuous output signal.

The ippsSampleUp functionality can be described as follows:

pDst[factor* n +phase] =pSrc[n],0≤n <srcLen

pDst[factor * n + m] = 0, 0≤n <srcLen, 0≤m<factor, m ≠phase

pDstLen=factor*srcLen.

Return Values

ippStsNoErr	Indicates no error.

ippStsNullPtrErr	Indicates an error if the pDst, pSrc, pDstLen, or pPhase pointer is NULL.
ippStsSizeErr	Indicates an error if srcLen is less than or equal to 0.
ippStsSampleFactorErr	Indicates an error if factor is less than or equal to 0.
ippStsSamplePhaseErr	Indicates an error when pPhase is negative, or bigger than or equal to factor.

SampleDown

Down-samples a signal, conceptually decreasing its sampling rate by an integer factor.

Syntax

```
IppStatus ippsSampleDown_16s(const Ipp16s* pSrc, int srcLen, Ipp16s* pDst, int*
pDstLen, int factor, int* pPhase);
```

```
IppStatus ippsSampleDown_32f(const Ipp32f* pSrc, int srcLen, Ipp32f* pDst, int*
pDstLen, int factor, int* pPhase);
```

```
IppStatus ippsSampleDown_64f(const Ipp64f* pSrc, int srcLen, Ipp64f* pDst, int*
pDstLen, int factor, int* pPhase);
```

```
IppStatus ippsSampleDown_16sc(const Ipp16sc* pSrc, int srcLen, Ipp16sc* pDst, int*
pDstLen, int factor, int* pPhase);
```

```
IppStatus ippsSampleDown_32fc(const Ipp32fc* pSrc, int srcLen, Ipp32fc* pDst, int*
pDstLen, int factor, int* pPhase);
```

```
IppStatus ippsSampleDown_64fc(const Ipp64fc* pSrc, int srcLen, Ipp64fc* pDst, int*
pDstLen, int factor, int* pPhase);
```

Include Files

ipps.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

pSrc	Pointer to the source array holding the samples to be down-sampled.
srcLen	Number of samples in the input array pSrc.
pDst	Pointer to the destination array.
pDstLen	Pointer to the length of the destination array pDst.

factor	Factor by which the signal is down-sampled. That is, factor - 1 samples are discarded from every block of factor samples in pSrc.
pPhase	Pointer to the input phase value that determines which of the samples within each block of factor samples from pSrc is not discarded and copied to pDst. The value of pPhase is required to be in the range [0; factor - 1].

Description

This function down-samples the srcLen - length source array pSrc by factor factor with phase pPhase, and stores the result in the array pDst, ignoring its length value by the pDstLen address.

Down-sampling discards factor - 1 samples from pSrc, copying one sample from each block of factor samples from pSrc to pDst. The pPhase argument determines which of the samples in each block is not discarded and where it lies within each input block of factor samples. The value of pPhase is required to be in the range [0; factor - 1]. The length of the destination array is stored by the pDstLen address.

The pPhase value is the phase of an source array sample. It is also a returned output phase which can be used as an input phase for the first sample in the next block to process. Use pPhase for block mode processing to get a continuous output signal.

You can use the FIR multi-rate filter to combine filtering and resampling, for example, for antialiasing filtering before the sub-sampling procedure.

The ippsSampleDown functionality can be described as follows:

pDstLen= (srcLen+factor- 1 -phase)/factor

pDst[n]=pSrc[factor* n +phase],0≤n <pDstLen

phase= (factor+phase-srcLen%factor)%factor.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when the pDst, pSrc, pDstLen, or pPhase pointer is NULL.
ippStsSizeErr	Indicates an error when srcLen is less than or equal to 0.
ippStsSampleFactorErr	Indicates an error when factor is less than or equal to 0.
ippStsSamplePhaseErr	Indicates an error when pPhase is negative, or bigger than or equal to factor.

Example

The example below shows how to use the function ippsSampleDown.

```
void sampling( void ) {
      Ipp16s x[8] = { 1,2,3,4,5,6,7,8 };
      Ipp16s y[8] = {9,10,11,12,13,14,15,16}, z[8];
      int dstLen1, dstLen2, phase = 2;
```

```
    IppStatus st = ippsSampleDown_16s(x, 8, z, &dstLen1, 3, &phase);
    st = ippsSampleDown_16s(y, 8, z+dstLen1, &dstLen2, 3, &phase);
    printf_16s("down-sampling =", z, dstLen1+dstLen2, st);
}
```

Output:

```
down-sampling = 3 6 9 12 15
```

AI Inference Functions

This section describes the Intel IPP signal processing functions that are used in inference engines in the AI field.

- PatternMatchGetBufferSize
- PatternMatch

PatternMatchGetBufferSize

Computes the size of the work buffer for the `ippsPatternMatch` function.

Syntax

```
IppStatus ippsPatternMatchGetBufferSize (int srcLen, int patternLen, int patternSize,
IppPatternMatchMode hint, int* bufSize);
```

Include Files

`ipps.h`

Parameters

`srcLen`	Number of patterns in the source array.
`patternLen`	Number of elements in the templates array.
`patternSize`	The size of a pattern, in bytes.
`hint`	Option to run specially optimized code branch, supported values: • `ippPatternMatchAuto` • The function selects optimization automatically. • `ippPatternMatchDirect` • The function uses direct method, no additional memory is • `ippPatternMatchTable` • The function uses conversion data for internal
`bufSize`	Size of the required buffer.

Description

This function computes the size, in bytes, of the external work buffer needed for the ippsPatternMatch function. The result is stored in the `bufSize` parameter.

Return Values

ippStsNoErr	Indicates no error.
ippStsSizeErr	Indicates an error when at least one of the srcLen, dstLen, or patternSize values is less than, or equal to zero; or patternSize is too big.
ippStsBadArg	Indicates an error when the value of hint is not supported.

See Also

PatternMatch Compares given array of binary patterns with an array of templates.

PatternMatch

Compares given array of binary patterns with an array of templates.

Syntax

```
IppStatus ippsPatternMatch_8u16u(const Ipp8u* pSrc, int srcStep, int srcLen, const Ipp8u* pPattern, int patternStep, int patternLen, int patternSize, Ipp16u* pDst, IppPatternMatchMode hint, int* pBufSize);
```

Include Files

ipps.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

pSrc	Pointer to the source array of patterns.
srcStep	Stride between patterns in the source array.
srcLen	Number of patterns in the source array.
pPattern	Pointer to the array of templates.
patternStep	Stride between templates.
patternLen	Number of elements in the array of templates.
patternSize	Size of a pattern, in bytes.
pDst	Pointer to the result of comparison.
hint	Option to run specially optimized code branch, supported values: • ippPatternMatchAuto • The function selects optimization automatically.

	• ippPatternMatchDirect • The function uses direct method, no additional memory is • ippPatternMatchTable • The function uses conversion data for internal
pBufSize	Pointer to the work buffer size. The length of the buffer is srcLen*patternLen*sizeof(Ipp16u).

Description

This function compares a provided array of binary patterns with the array of templates. The *pattern* is an array containing bits of fixed size (8/16/../128/256/512 bits). The pattern size provided to the function is calculated in bytes. The *template* is some fixed pattern. The function compares the provided source patterns with the existing templates grouped into an array. To compare patterns, the function performs bitwise XOR operation between two patterns and calculates the number of resulting nonzero bits (population counter). This operation is applied to all source and template patterns.

Returned sequence is:

```
(patternLen = B, srcLen=A)
pat<0>^src<0>,      …,      pat<0>^src<A-1>,
pat<1>^src<0>,      …,      pat<1>^src<A-1>,
                            …
pat<B-1>^src<B-1>,  …,    pat<B-1>^src<A-1>,
```

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one of the pointers is NULL.
ippStsSizeErr	Indicates an error when at least one of the srcLen, dstLen, or size values is less than, or equal to zero; or one of the size values is too big.
ippStsBadArg	Indicates an error when the value of hint is not supported.

Filtering Functions

This chapter describes the Intel ® IPP functions that perform convolution and correlation operations, as well as linear and non-linear filtering.

- Convolution and Correlation Functions
- Filtering Functions

Convolution and Correlation Functions

Convolution is an operation used to define an output signal from any linear time-invariant (LTI) processor in response to any input signal.

The correlation functions described in this section estimate either the auto-correlation of a source vector or the cross-correlation of two vectors.

- Special Arguments
- AutoCorrNormGetBufferSize
- AutoCorrNorm
- CrossCorrNormGetBufferSize
- CrossCorrNorm
- ConvolveGetBufferSize
- Convolve
- ConvBiased

Special Arguments

Some Convolution and Correlation functions described in this section have two implementations of the algorithm:

- For small data size, function processes data as described by the formula
- For big data size, function uses FFT-inherited algorithms

The optimal algorithm is selected automatically according to the input data size. You can manually choose which algorithm to use by passing one of the following predefined values to the `algType` parameter of the function:

`ippAlgAuto`	Select the optimal algorithm automatically.
`ippAlgDirect`	Use direct algorithm as described by the formula.
`ippAlgFFT`	Use FFT-based algorithm implementation.

These values are declared in the `IppAlgType` enumerator.

Several functions support normalization of the output data. You can choose which normalization to apply by passing one of the following values to the function:

`ippsNormNone`	No normalization (default).
`ippsNormA`	Biased normalization.
`ippsNormB`	Unbiased normalization.

These values are declared in the `IppsNormOp` enumerator.

See Also

Structures and Enumerators

AutoCorrNormGetBufferSize

Computes the size of the work buffer for the `ippsAutoCorrNorm` function.

Syntax

```
IppStatus ippsAutoCorrNormGetBufferSize (int srcLen, int dstLen, IppDataType dataType,
IppEnum algType, int* pBufferSize);
```

Include Files

`ipps.h`

Parameters

`srcLen`	Number of elements in the source vector.

dstLen	Number of elements in the destination vector (length of auto-correlation).
dataType	Data type for auto-correlation. Possible values are ipp32f, ipp32fc, ipp64f, or ipp64fc.
algType	Bit-field mask for the algorithm type definition. Possible values are the results of composition of the IppAlgType and IppsNormOp values.
pBufferSize	Pointer to the size of the work buffer.

Description

The ippsAutoCorrNormGetBufferSize function computes the size in bytes of the external work buffer needed for the function that performs auto-correlation. The result is stored in the pBufferSize parameter.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pBufferSize is NULL.
ippStsSizeErr	Indicates an error when srcLen or dstLen is less than, or equal to zero.
ippStsAlgTypeErr	Indicates an error when: • the result of the bitwise AND operation between the algType and ippAlgMask differs from the ippAlgAuto, ippAlgDirect, or ippAlgFFT values; • the result of the bitwise AND operation between the algType and ippsNormMask differs from the ippsNormNone, ippsNormA, or ippsNormB values.
ippStsDataTypeErr	Indicates an error when the dataType value differs from the ipp32f, ipp32fc, ipp64f, or ipp64fc.

See Also

Enumerators

Special Arguments

AutoCorrNorm Calculates normal, biased, and unbiased auto-correlation of a vector.

AutoCorrNorm

Calculates normal, biased, and unbiased auto-correlation of a vector.

Syntax

```
IppStatus ippsAutoCorrNorm_32f (const Ipp32f* pSrc, int srcLen, Ipp32f* pDst, int dstLen, IppEnum algType, Ipp8u* pBuffer);
```

```
IppStatus ippsAutoCorrNorm_64f (const Ipp64f* pSrc, int srcLen, Ipp64f* pDst, int
dstLen, IppEnum algType, Ipp8u* pBuffer);

IppStatus ippsAutoCorrNorm_32fc (const Ipp32fc* pSrc, int srcLen, Ipp32fc* pDst, int
dstLen, IppEnum algType, Ipp8u* pBuffer);

IppStatus ippsAutoCorrNorm_64fc (const Ipp64fc* pSrc, int srcLen, Ipp64fc* pDst, int
dstLen, IppEnum algType, Ipp8u* pBuffer);
```

Include Files

`ipps.h`

Parameters

`pSrc`	Pointer to the source vector.
`srcLen`	Number of elements in the source vector.
`pDst`	Pointer to the destination vector. This vector stores the calculated auto-correlation of the source vector.
`dstLen`	Number of elements in the destination vector (length of auto-correlation).
`algType`	Bit-field mask for the algorithm type definition. Possible values are the results of composition of the `IppAlgType` and `IppsNormOp` values.
`pBuffer`	Pointer to the buffer for internal calculations.

Description

Before using these functions, you need to compute the size of the work buffer using the `ippsAutoCorrNormGetBufferSize` function.

These functions calculate the normalized auto-correlation of the `pSrc` vector of `srcLen` length and store the results in the `pDst` vector of `dstLen` length. The result vector `pDst` is calculated by the following equations:

$$pDst[n] = \sum_{i=0}^{srcLen-1} conj(pSrc[i]) \cdot pSrc[i+n], \quad 0 \le n < dstLen \qquad \text{(normal)}$$

$$pDst[n] = \frac{1}{srcLen} \sum_{i=0}^{srcLen-1} conj(pSrc[i]) \cdot pSrc[i+n], \quad 0 \le n < dstLen \qquad \text{(biased)}$$

$$pDst[n] = \frac{1}{srcLen - n} \sum_{i=0}^{srcLen-1} conj(pSrc[i]) \cdot pSrc[i+n], \quad 0 \le n < dstLen \qquad \text{(unbiased)}$$

where

$$pSrc[i] = \begin{cases} pSrc[i], & 0 \le i < srcLen \\ 0, & otherwise \end{cases}$$

NOTE The auto-correlation is computed for positive lags only. Auto-correlation for a negative lag is a complex conjugate of the auto-correlation for the equivalent positive lag.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error when `srcLen` or `dstLen` is less than, or equal to zero.
`ippStsAlgTypeErr`	Indicates an error when:

Example

The code example below demonstrates how to use the `ippsAutoCorrNormGetBufferSize` and `ippsAutoCorrNorm` functions.

```
IppStatus AutoCorrNormExample (void) {
   IppStatus status;
   const int srcLen = 5, dstLen = 10;
   Ipp32f pSrc[srcLen] = {0.2f, 3.1f, 2.0f, 1.2f, -1.1f}, pDst[dstLen];
   IppEnum funCfg = (IppEnum)(ippAlgAuto|ippsNormB);
   int bufSize = 0;
   Ipp8u *pBuffer;


   status = ippsAutoCorrNormGetBufferSize(srcLen, dstLen, ipp32f, funCfg, &bufSize);


   if ( status != ippStsNoErr )
      return status;


   pBuffer = ippsMalloc_8u( bufSize );


   status = ippsAutoCorrNorm_32f(pSrc, srcLen, pDst, dstLen, funCfg, pBuffer);


   printf_32f("pDst", pDst, dstLen);


   ippsFree( pBuffer );
   return status;
}
```

The result is as follows:

```
pDst -> 3.3 2.0 0.6 -1.6 -0.2 0.0 0.0 0.0 0.0 0.0
```

See Also

Enumerators

Special Arguments

AutoCorrNormGetBufferSize Computes the size of the work buffer for the ippsAutoCorrNorm function.

CrossCorrNormGetBufferSize

Computes the size of the work buffer for the `ippsCrossCorrNorm` function.

Syntax

```
IppStatus ippsCrossCorrNormGetBufferSize (int src1Len, int src2Len, int dstLen, int lowLag, IppDataType dataType, IppEnum algType, int* pBufferSize);
```

Include Files

`ipps.h`

Parameters

`src1Len`	Number of elements in the first source vector.
`src2Len`	Number of elements in the second source vector.
`dstLen`	Number of elements in the destination vector (length of cross-correlation).
`lowLag`	Lower value of the range of lags at which the correlation is computed.
`dataType`	Data type for cross-correlation. Possible values are `ipp32f`, `ipp32fc`, `ipp64f`, or `ipp64fc`.
`algType`	Bit-field mask for the algorithm type definition. Possible values are the results of composition of the `IppAlgType` and `IppsNormOp` values.
`pBufferSize`	Pointer to the size of the work buffer.

Description

The `ippsCrossCorrNormGetBufferSize` function computes the size in bytes of the external work buffer needed for the function that performs cross-correlation. The result is stored in the `pBufferSize` parameter.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when `pBufferSize` is `NULL`.

ippStsSizeErr	Indicates an error when the length of the vector is negative, or equal to zero.
ippStsAlgTypeErr	Indicates an error when: • the result of the bitwise AND operation between the algType and ippAlgMask values differs from the ippAlgAuto, ippAlgDirect, or ippAlgFFT values. • the result of the bitwise AND operation between the algType and ippsNormMask values differs from the ippsNormNone, ippsNormA, or ippsNormB values.
ippStsDataTypeErr	Indicates an error when the dataType value differs from the Ipp32f, Ipp32fc, Ipp64f, or Ipp64fc.

See Also

Enumerators

CrossCorrNorm Calculates the cross-correlation of two vectors.

Special Arguments

CrossCorrNorm

Calculates the cross-correlation of two vectors.

Syntax

```
IppStatus ippsCrossCorrNorm_32f (const Ipp32f* pSrc1, int src1Len, const Ipp32f* pSrc2,
int src2Len, Ipp32f* pDst, int dstLen, int lowLag, IppEnum algType, Ipp8u* pBuffer);

IppStatus ippsCrossCorrNorm_64f (const Ipp64f* pSrc1, int src1Len, const Ipp64f* pSrc2,
int src2Len, Ipp64f* pDst, int dstLen, int lowLag, IppEnum algType, Ipp8u* pBuffer);

IppStatus ippsCrossCorrNorm_32fc (const Ipp32fc* pSrc1, int src1Len, const Ipp32fc*
pSrc2, int src2Len, Ipp32fc* pDst, int dstLen, int lowLag, IppEnum algType, Ipp8u*
pBuffer);

IppStatus ippsCrossCorrNorm_64fc (const Ipp64fc* pSrc1, int src1Len, const Ipp64fc*
pSrc2, int src2Len, Ipp64fc* pDst, int dstLen, int lowLag, IppEnum algType, Ipp8u*
pBuffer);
```

Include Files

ipps.h

Parameters

pSrc1	Pointer to the first source vector.
src1Len	Number of elements in the first source vector.

pSrc2	Pointer to the second source vector.
src2Len	Number of elements in the second source vector.
pDst	Pointer to the destination vector. This vector stores the calculated cross-correlation of the pSrc1 and pSrc2 vectors.
dstLen	Number of elements in the destination vector. This value determines the range of lags at which the cross-correlation is calculated.
lowLag	Cross-correlation lowest lag.
algType	Bit-field mask for the algorithm type definition. Possible values are the results of composition of the IppAlgType and IppsNormOp values.
pBuffer	Pointer to the buffer for internal calculations.

Description

These functions calculate the cross-correlation of the pSrc1 vector and the pSrc2 vector, and store the results in the pDst vector. The result vector pDst is calculated by the following equations:

$$pDst[n] = \sum_{i=0}^{len1-1} conj(pSrc1[i]) \cdot pSrc2[n+i+lowLag] ,$$

where

$$0 \le n < dstLen,$$

$$pSrc2[j] = \begin{cases} pSrc2[j], & 0 < j < len2 \\ 0, & otherwise \end{cases}$$

Before using this function, you need to compute the size of the work buffer using the `ippsCrossCorrNormGetBufferSize` function.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error when the length of a vector is less than, or equal to zero.
`ippStsAlgTypeErr`	Indicates an error when:

Example

The code example below demonstrates how to use the `ippsCrossCorrNormGetBufferSize` and `ippsCrossCorrNorm` functions.

```
IppStatus CrossCorrNormExample (void) {
   IppStatus status;
   const int src1Len=5, src2Len=7, dstLen=16;
   int lowLag = -5;
   Ipp32f pSrc1[src1Len] = {1.f,1.f,1.f,1.f,1.f}, pSrc2[src2Len] =
{1.f,1.f,1.f,1.f,1.f,1.f,1.f}, pDst[dstLen];
   IppEnum funCfgNormNo = (IppEnum)(ippAlgAuto|ippsNormNone);
   IppEnum funCfgNormA = (IppEnum)(ippAlgAuto|ippsNormA);
   IppEnum funCfgNormB = (IppEnum)(ippAlgAuto|ippsNormB);
   int bufSizeNo=0, bufSizeA=0, bufSizeB=0, bufSizeMax=0;
   Ipp8u *pBuffer;


   status = ippsCrossCorrNormGetBufferSize(src1Len, src2Len, dstLen, -5, ipp32f, funCfgNormNo,
&bufSizeNo);
   if ( status != ippStsNoErr ) return status;
   status = ippsCrossCorrNormGetBufferSize(src1Len, src2Len, dstLen, -5, ipp32f, funCfgNormA,
&bufSizeA);
   if ( status != ippStsNoErr ) return status;
   status = ippsCrossCorrNormGetBufferSize(src1Len, src2Len, dstLen, -5, ipp32f, funCfgNormB,
&bufSizeB);
   if ( status != ippStsNoErr ) return status;


   bufSizeMax = IPP_MAX(bufSizeNo, IPP_MAX(bufSizeA, bufSizeB));// get max buffer size


   pBuffer = ippsMalloc_8u( bufSizeMax );
```

```
   status = ippsCrossCorrNorm_32f(pSrc1, src1Len, pSrc2, src2Len, pDst, dstLen, lowLag,
funCfgNormNo, pBuffer);
   printf_32f("pDst_NormNone", pDst, dstLen);


   status = ippsCrossCorrNorm_32f(pSrc1, src1Len, pSrc2, src2Len, pDst, dstLen, lowLag,
funCfgNormA, pBuffer);
   printf_32f("pDst_NormA", pDst, dstLen);


   status = ippsCrossCorrNorm_32f(pSrc1, src1Len, pSrc2, src2Len, pDst, dstLen, lowLag,
funCfgNormB, pBuffer);
   printf_32f("pDst_NormB", pDst, dstLen);


   ippsFree( pBuffer );
   return status;
}
```

The result is as follows:

```
pDst_NormNone -> 0.0 1.0 2.0 3.0 4.0 5.0 5.0 5.0 4.0 3.0 2.0 1.0 0.0 0.0 0.0 0.0
pDst_NormA    -> 0.0 0.2 0.4 0.6 0.8 1.0 1.0 1.0 0.8 0.6 0.4 0.2 0.0 0.0 0.0 0.0
pDst_NormB    -> 0.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 0.0 0.0 0.0 0.0
```

See Also

Enumerators

Special Arguments

CrossCorrNormGetBufferSize Computes the size of the work buffer for the ippsCrossCorrNorm function.

ConvolveGetBufferSize

Computes the size of the work buffer for the `ippsConvolve` function.

Syntax

```
IppStatus ippsConvolveGetBufferSize (int src1Len, int src2Len, IppDataType dataType,
IppEnum algType, int* pBufferSize);
```

Include Files

`ipps.h`

Parameters

`src1Len`	Number of elements in the first source vector.
`src2Len`	Number of elements in the second source vector.
`dataType`	Data type for convolution. Possible values are `ipp32f` and `ipp64f`.

algType	Bit-field mask for the algorithm type definition. Possible values are listed in the IppAlgType enumerator.
pBufferSize	Pointer to the size of the work buffer.

Description

The ippsConvolveGetBufferSize function computes the size, in bytes, of the external work buffer needed for the functions that perform convolution operations. The result is stored in the pBufferSize parameter.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pBufferSize is NULL.
ippStsSizeErr	Indicates an error when the length of the vector is negative, or equal to zero.
ippStsAlgTypeErr	Indicates an error when the result of the bitwise AND operation between the algType and ippAlgMask differs from the ippAlgAuto, ippAlgDirect, or ippAlgFFT values.
ippStsDataTypeErr	Indicates an error when the dataType value differs from the ipp32f or ipp64f.

See Also

Enumerators

Special Arguments

Convolve Performs a finite linear convolution of two vectors.

Convolve

Performs a finite linear convolution of two vectors.

Syntax

```
IppStatus ippsConvolve_32f (const Ipp32f* pSrc1, int src1Len, const Ipp32f* pSrc2, int src2Len, Ipp32f* pDst, IppEnum algType, Ipp8u* pBuffer);
```

```
IppStatus ippsConvolve_64f (const Ipp64f* pSrc1, int src1Len, const Ipp64f* pSrc2, int src2Len, Ipp64f* pDst, IppEnum algType, Ipp8u* pBuffer);
```

Include Files

ipps.h

Parameters

pSrc1	Pointer to the first source vector.
src1Len	Number of elements in the first source vector.

pSrc2	Pointer to the second source vector.
src2Len	Number of elements in the second source vector.
pDst	Pointer to the destination vector. This vector stores the result of the convolution of the pSrc1 and pSrc2 vectors.
algType	Bit-field mask for the algorithm type definition. Possible values are listed in the IppAlgType enumerator.
pBuffer	Pointer to the buffer for internal calculations.

Description

These functions perform the finite linear convolution of the pSrc1 and pSrc2 vectors. The src1Len elements of the pSrc1 vector are convolved with the src2Len elements of the pSrc2 vector. The result of the convolution is stored in the pDst vector with the length equal to src1Len + src2Len – 1. The result vector pDst is calculated by the following equations:

$$pDst[n] = \sum_{k=0}^{n} pSrc1[k] \cdot pSrc2[n-k] \qquad 0 \le n \le src1Len + src2Len - 1$$

where

- pSrc1[i]=0, if i ≥ src1Len
- pSrc2[j]=0, if j ≥ src2Len

Before using this function, you need to compute the size of the work buffer using the ippsConvolveGetBufferSize function.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when the length of a vector is less than, or equal to zero.
ippStsAlgTypeErr	Indicates an error when the result of the bitwise AND operation between algType and ippAlgMask differs from the ippAlgAuto, ippAlgDirect, or ippAlgFFT values.

Example

The code example below demonstrates how to use the `ippsConvolveGetBufferSize` and `ippsConvolve_32f` functions.

```
IppStatus ConvolveExample (void) {
   IppStatus status;
   const int src1Len = 5, src2Len = 2, dstLen = src1Len+src2Len-1;
   Ipp32f pSrc1[src1Len] = {-2.f,0.f,1.f,-1.f,3.f}, pSrc2[src2Len]={0.f,1.f}, pDst[dstLen];
   IppEnum funCfg = (IppEnum)(ippAlgAuto);
   int bufSize = 0;
   Ipp8u *pBuffer;


   status = ippsConvolveGetBufferSize(src1Len, src2Len, ipp32f, funCfg, &bufSize);


   if ( status != ippStsNoErr )
      return status;


   pBuffer = ippsMalloc_8u( bufSize );


   status = ippsConvolve_32f(pSrc1, src1Len, pSrc2, src2Len, pDst, funCfg, pBuffer);


   printf_32f("pDst", pDst, dstLen);


   ippsFree( pBuffer );
   return status;
}
```

The result is as follows:

```
pDst -> 0.0 -2.0 0.0 1.0 -1.0 3.0
```

See Also

Enumerators

Special Arguments

ConvolveGetBufferSize Computes the size of the work buffer for the ippsConvolve function.

ConvBiased

Computes the specified number of elements of the full finite linear convolution of two vectors.

Syntax

```
IppStatus ippsConvBiased_32f(const Ipp32f* pSrc1, int src1Len, const Ipp32f* pSrc2, int
src2Len, Ipp32f* pDst, int dstLen, int bias);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pSrc1`, `pSrc2`	Pointers to the two vectors to be convolved.
`src1Len`	Number of elements in the vector `pSrc1`.
`src2Len`	Number of elements in the vector `pSrc2`.
`pDst`	Pointer to the vector `pDst`. This vector stores the result of the convolution.
`dstLen`	Number of elements in the vector `pDst`.
`bias`	Parameter that specifes the starting element of the convolution.

Description

This function computes `dstLen` elements of finite linear convolution of two specified vectors `pSrc1` and `pSrc2` starting with an element that is specified by the `bias`. The result is stored in the vector `pDst`.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when the `pDst` or `pSrc` pointer is `NULL`.
`ippStsSizeErr`	Indicates an error when `src1Len` or `src2Len` is less than or equal to 0.

Example

The example below shows how to call the function `ippsConvBiased`.

```
void func_convbiased()
{
    Ipp32f pSrc1[5] = {1.1, -2.0, 3.5, 2.2, 0.0};
    Ipp32f pSrc2[4] = {0.0, 0.2, 2.5, -1.0};
    const int len = 10;
    Ipp32f pDst[len];
    int bias = 1;


    ippsZero_32f(pDst, len);
    ippsConvBiased_32f(pSrc1, 5, &pSrc2[1], 3, pDst, len, bias);
}
```

Result:

```
pDst -> 0.2  2.3  -4.3  9.2  5.5  0.0  0.0  0.0  0.0  0.0
```

Filtering Functions

The Intel IPP functions described in this section implement the following types of filters:

- Finite impulse response (FIR) filter
- Adaptive finite impulse response using least mean squares (LMS) filter
- Infinite impulse response (IIR) filter
- Median filter

A special set of functions is designed to generate filter coefficients for different types of FIR filters.

- SumWindow
- FIR Filter Functions
- FIR Filter Coefficient Generating Functions
- Single-Rate FIR LMS Filter Functions
- IIR Filter Functions
- IIRIIR Filter Functions
- Median Filter Functions
- Polyphase Resampling Functions

SumWindow

Sums elements in the mask applied to each element of a vector.

Syntax

```
IppStatus ippsSumWindow_8u32f(const Ipp8u* pSrc, Ipp32f* pDst, int len, int maskSize);
```

```
IppStatus ippsSumWindow_16s32f(const Ipp16s* pSrc, Ipp32f* pDst, int len, int maskSize);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pSrc`	Pointer to the source vector.
`pDst`	Pointer to the destination vector.
`len`	Number of elements of the vector.
`maskSize`	Size of the mask.

Description

This function sets each element in the destination vector `pDst` as the sum of `maskSize` elements of the source vector `pSrc`. The computation is performed as follows:

$$pDst[n] = \sum_{k=n}^{maskSize} pSrc[k] \quad , 0 \le n < len$$

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsMaskSizeErr	Indicates an error when maskSize is less than or equal to 0.

FIR Filter Functions

The functions described in this section perform a finite impulse response (FIR) filtering of input data. The functions initialize different FIR filter structures, get and set the delay lines and filter coefficients (taps), and perform filtering. Intel IPP contains the functions that implement the FIR filters without the delay line - stream FIR filters.

Special set of functions allows to compute the filter coefficients for different filters.

To perform single-rate FIR filtering with the ippsFIRSR function, follow this scheme:

1. Call ippsFIRSRGetSize function to get the size of the filter specification structure and the work buffer.
2. Call ippsFIRSRInit function to initialize the filter specification structure.
3. Call ippsFIRSR function to apply the single-rate FIR filter to a source vector.

- FIRMRGetSize
- FIRMRInit
- FIRMR
- FIRSRGetSize
- FIRSRInit
- FIRSR
- FIRSparseInit
- FIRSparseGetStateSize
- FIRSparseGetDlyLine
- FIRSparseSetDlyLine
- FIRSparse
- Examples of Using FIR Functions

FIRMRGetSize

Computes the size of the context structure and work buffer for multi-rate FIR filtering.

Syntax

Case 1: Operation on a signal that has the same data type as the coefficients

```
IppStatus ippsFIRMRGetSize(int tapsLen, int upFactor, int downFactor, IppDataType
tapsType, int* pSpecSize, int* pBufSize);
```

Case 2: Operation on a signal that has a data type different from the data type of the coefficients

```
IppStatus ippsFIRMRGetSize32f_32fc(int tapsLen, int upFactor, int downFactor, int*
pSpecSize, int* pBufSize);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`tapsLen`	Length of the FIR filter.
`upFactor`	Multi-rate up factor.
`downFactor`	Multi-rate down factor.
`tapsType`	Data type of the coefficients. Supported values are `Ipp32f`, `Ipp32fc`, `Ipp64f`, and `Ipp64fc`.
`pSpecSize`	Pointer to the size of the FIR specification structure.
`pBufSize`	Pointer to the size of the work buffer required for FIR filtering.

Description

This function computes the following:

- Size of the internal specification structure for multi-rate FIR filtering. The structure can be shared between all threads of the application.
- Size of the work buffer for each thread.

For an example on how to use this function, refer to the example provided with the FIRMR function description.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error when the `tapsLen` value is less than or equal tozero.
`ippStsDataTypeErr`	Indicates an error when the specified taps type is not supported.
`ippStsFIRMRFactorErr`	Indicates an error when the `upFactor` value or the `downFactor` value is less than zero.
`ippStsExceededSizeErr`	Indicates an error when the size of the work buffer exceeds the maximum of the data type positive value pointed by `*intpBufSize`.

See Also

FIRMR Performs multi-rate FIR filtering of a source vector.

FIRMRInit

Initializes the context structure for multi-rate FIR filtering.

Syntax

Case 1: Operation on a signal that has the same data type as the coefficients

```
IppStatus ippsFIRMRInit_<mod>(const Ipp<dataType>* pTaps, int tapsLen, int upFactor,
int upPhase, int downFactor, int downPhase, IppsFIRSpec_<dataType>* pSpec);
```

Supported values for `mod`:

`32f`	`64f`	`32fc`	`64fc`

Case 2: Operation on a signal that has a data type different from the data type of the coefficients

```
IppStatus ippsFIRMRInit32f_32fc(const Ipp32f* pTaps, int tapsLen, int upFactor, int
upPhase, int downFactor, int downPhase, IppsFIRSpec32f_32fc* pSpec);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pTaps`	Pointer to the array containing filter coefficients. The number of elements in the array is `tapsLen`.
`tapsLen`	Number of filter coefficients.
`upFactor`	Multi-rate upsampling factor.
`upPhase`	Phase for upsampled signal.
`downFactor`	Multi-rate downsampling factor.
`downPhase`	Phase for downsampled signal.
`pSpec`	Pointer to the internal FIR specification structure.

Description

This function initializes the multi-rate FIR filter specification structure in the external buffer. Before using this function, compute the size of the specification structure and the size of the work buffer using the FIRMRGetSize function.

The parameter `upFactor` is the factor by which the filtered signal is internally upsampled (see description of the function SampleUp for more details). That is, `upFactor - 1` zeros are inserted between each sample of the input signal.

The parameter `upPhase` is the parameter, which determines where a non-zero sample lies within the `upFactor - length` block of the upsampled input signal.

The parameter `downFactor` is the factor by which the FIR response obtained by filtering an upsampled input signal, is internally downsampled (see description of the function SampleDown for more details). That is, `downFactor - 1` output samples are discarded from each `downFactor - length` output block of the upsampled filter response.

The `downPhase` parameter determines where non-discarded sample lies within a block of upsampled filter response.

For an example on how to use this function, refer to the example provided with the FIRMR function description.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when one of the specified pointers is `NULL`.
`ippStsFIRLenErr`	Indicates an error when `tapsLen` is less than, or equal to zero.
`ippStsFIRMRFactorErr`	Indicates an error when `upFactor` or `downFactor` is less than, or equal to zero.
`ippStsFIRMRPhaseErr`	Indicates an error when `upPhase/downPhase` is negative, or greater than or equal to `upFactor/downFactor`.

See Also

FIRMRGetSize Computes the size of the context structure and work buffer for multi-rate FIR filtering.

FIRMR Performs multi-rate FIR filtering of a source vector.

SampleUp Up-samples a signal, conceptually increasing its sampling rate by an integer factor.

SampleDown Down-samples a signal, conceptually decreasing its sampling rate by an integer factor.

FIRMR

Performs multi-rate FIR filtering of a source vector.

Syntax

Case 1: Operation on a signal that has the same data type as the coefficients

```
IppStatus ippsFIRMR_32f(const Ipp32f* pSrc, Ipp32f* pDst, int numIters,
IppsFIRSpec_32f* pSpec, const Ipp32f* pDlySrc, Ipp32f* pDlyDst, Ipp8u* pBuf);

IppStatus ippsFIRMR_64f(const Ipp64f* pSrc, Ipp64f* pDst, int numIters,
IppsFIRSpec_64f* pSpec, const Ipp64f* pDlySrc, Ipp64f* pDlyDst, Ipp8u* pBuf);

IppStatus ippsFIRMR_32fc(const Ipp32fc* pSrc, Ipp32fc* pDst, int numIters,
IppsFIRSpec_32fc* pSpec, const Ipp32fc* pDlySrc, Ipp32fc* pDlyDst, Ipp8u* pBuf);

IppStatus ippsFIRMR_64fc(const Ipp64fc* pSrc, Ipp64fc* pDst, int numIters,
IppsFIRSpec_64fc* pSpec, const Ipp64fc* pDlySrc, Ipp64fc* pDlyDst, Ipp8u* pBuf);
```

```
IppStatus ippsFIRMR_16s(const Ipp16s* pSrc, Ipp16s* pDst, int numIters,
IppsFIRSpec_32f* pSpec, const Ipp16s* pDlySrc, Ipp16s* pDlyDst, Ipp8u* pBuf);

IppStatus ippsFIRMR_16sc(const Ipp16sc* pSrc, Ipp16sc* pDst, int numIters,
IppsFIRSpec_32fc* pSpec, const Ipp16sc* pDlySrc, Ipp16sc* pDlyDst, Ipp8u* pBuf);
```

Case 2: Operation on a signal that has a data type different from the data type of the coefficients

```
IppStatus ippsFIRMR32f_32fc(const Ipp32fc* pSrc, Ipp32fc* pDst, int numIters,
IppsFIRSpec32f_32fc* pSpec, const Ipp32fc* pDlySrc, Ipp32fc* pDlyDst, Ipp8u* pBuf);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pSrc`	Pointer to the source vector.
`pDst`	Pointer to the destination vector.
`numIters`	Number of iterations associated with the number of samples to be filtered by the function. The `(numIters*downFactor)` elements of the source vector are filtered and the resulting `(numIters*upFactor)` samples are stored in the destination array.
`pSpec`	Pointer to the internal specification structure.
`pDlySrc`	Pointer to the array containing values for the source delay lines. The value can be `NULL`. If not `NULL`, the array length is defined as `(tapsLen + upFactor - 1)/upFactor`.
`pDlyDst`	Pointer to the array containing values for the destination delay line. The value can be `NULL`. If not `NULL`, the array length is defined as `(tapsLen + upFactor - 1)/upFactor`.
`pBuf`	Pointer to the work buffer.

Description

Before using this function, you need to initialize the internal constant specification structure using the ippsFIRMR_Init function.

This function filters the source vector `pSrc` using the multi-rate FIR filter and stores the result in `pDst`. Filtering is performed by the following formula:

$$y(n) = \sum_{i=0}^{tapsLen-1} h(i) \cdot x(n-i), \quad 0 \le n < numIters$$
$$0 \le i < tapsLen - 1$$

where

- `x(0)...x(numIters)` is the source vector
- `h(0)...h(tapsLen-1)` are the FIR filter coefficients

The values of filter coefficients (taps) are specified in the `tapsLen - length` array `pTaps`, which is passed during initialization of the FIR filter specification structure. The `pDlySrc` and `pDlyDst` arrays specify the delay line values. The input array contains (`numIters * downFactor`) samples, and the output array stores the resulting (`numIters * upFactor`) samples. The multi-rate filtering is considered as a sequence of three operations: upsampling, filtering with a single-rate FIR filter, and downsampling. The algorithm is implemented as a single operation including the mentioned above three steps.

The parameter `upFactor` is the factor by which the filtered signal is internally upsampled (see the ippsSampleUp function description for more details). That is, `upFactor - 1` zeros are inserted between each sample of the input signal.

The parameter `upPhase` is the parameter that determines where a non-zero sample lies within the `upFactor` - length block of upsampled input signal.

The parameter `downFactor` is the factor by which the FIR response, which is obtained by filtering an upsampled input signal, is internally downsampled (see the ippsSampleDown function description for more details). That is, `downFactor - 1` output samples are discarded from each `downFactor - length` output block of the upsampled filter response.

The parameter `downPhase` is the parameter which determines where non-discarded sample lies within a block of upsampled filter response.

To compute the `y(0)...y(tapsLen-1)` destination vector, the function uses the `pDlySrc` array of the delay line.

The first `tapsLen - 1` elements of the function are:

```
y(0)=h(tapsLen-1)*d(0)+h(tapsLen-2)*d(1)+...+h(1)*d(tapsLen-2)+h(0)*x(0)
y(0)=h(tapsLen-1)*d(1)+h(tapsLen-2)*d(2)+...+h(1)*x(0)+h(0)*x(1)
y(tapsLen-1)=h(tapsLen-1)*x(0)+...+h(1)*x(tapsLen-2)+h(0)*x(tapsLen-1)
```

where

`d(0)`, `d(1)`, `d(2)`, and `d(tapsLen-2)` are the elements of the `pDlySrc` array

The last `tapsLen - 1` elements of the source vector are copied to the non-zero `pDlyDst` buffer for the next call of the FIR filter.

The arrays `pDlySrc` and `pDlyDst` support `NULL` values:

- if `pDlySrc` is `NULL`, the function uses the delay line with zero values
- if `pDlyDst` is `NULL`, the function does not copy any data to the destination delay line

Return Values

`ippStsNoErr`	Indicates no error.

ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.

Example

The code example below demonstrates how to use the FIRMRGetSize, FIRMRInit, and FIRMR functions.

```
int firmr()
{
    IppsFIRSpec_32f *pSpec;
    Ipp32f pTaps[8] = { 0.125,0.125,0.125,0.125,0.125,0.125,0.125,0.125};
    int i;
    int numIters = 33;
    int tapsLen = 8;
    int upFactor = 2;
    int upPhase = 0;
    int downFactor = 3;
    int downPhase = 0;
    int specSize, bufSize;
    Ipp32f pSrc[downFactor*numIters];
    Ipp32f pDst[upFactor *numIters];
    Ipp32f pDlySrc[(tapsLen + upFactor - 1  ) / upFactor];
    Ipp32f pDlyDst[(tapsLen + upFactor - 1  ) / upFactor];
    Ipp8u* pBuf;
    IppStatus status;
    status = ippsFIRMRGetSize(tapsLen, upFactor, downFactor, ipp32f,  &specSize,  &bufSize );
    printf("ippsFIRMRGetSize / status = %s\n", ippGetStatusString(status));
    pSpec = (IppsFIRSpec_32f*)ippsMalloc_8u(specSize);
    pBuf  = ippsMalloc_8u(bufSize);
    status = ippsFIRMRInit_32f ( pTaps, tapsLen, upFactor, upPhase, downFactor, downPhase,
pSpec);
    printf("ippsFIRMRInit_32f / status = %s\n", ippGetStatusString(status));
    if( ippStsNoErr != status){
        return -1;
    }
    for(i=0;i<downFactor*numIters;i++){
        pSrc[i] = 1;
    }
    status = ippsFIRMR_32f( pSrc, pDst, numIters, pSpec, NULL, pDlyDst, pBuf);
    printf("ippsFIRMR_32f / status = %s\n", ippGetStatusString(status));
    if( ippStsNoErr != status){
        return -1;
    }
    printf("src\n");
    for(i=0;i<numIters*downFactor;i++){
        printf("%6f ", pSrc[+i]);
    }
    printf("\ndst\n");
    for(i=0;i<numIters*upFactor;i++){
        printf("%6f ", pDst[i]);
    }
    printf("\n");
    return 0;
}
```

See Also

FIRMRGetSize Computes the size of the context structure and work buffer for multi-rate FIR filtering.

FIRMR_Init Initializes the context structure for multi-rate FIR filtering.

FIRSRGetSize

Computes the size of the constant structure and work buffer for single-rate FIR filtering.

Syntax

Case 1: Operation on a signal that has the same data type as the coefficients

```
IppStatus ippsFIRSRGetSize(int tapsLen, IppDataType tapsType, int* pSpecSize, int*
pBufSize);
```

Case 2: Operation on a signal that has a data type different from the data type of the coefficients

```
IppStatus ippsFIRSRGetSize32f_32fc(int tapsLen, int* pSpecSize, int* pBufSize);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`tapsLen`	Length of the FIR filter.
`tapsType`	Data type of the coefficients. The supported values are `ipp32f`, `ipp32fc`, `ipp64f`, and `ipp64fc`.
`pSpecSize`	Pointer to the size of the internal constant specification structure.
`pBufSize`	Pointer to the size of the work buffer required for FIR filtering.

Description

This function computes the following:

- Size of the internal constant specification structure for single-rate FIR filtering. The structure can be shared between all threads of the application.
- Size of the work buffer for each thread.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error when the `tapsLen` value is less than, or equal to zero.

ippStsDataTypeErr	Indicates an error when the specified taps type is not supported.

See Also

Examples of Using FIR Functions

FIRSRInit

Initializes the FIR constant structure for single-rate FIR filtering.

Syntax

```
IppStatus ippsFIRSRInit_32f(const Ipp32f* pTaps, int tapsLen, IppAlgType algType,
IppsFIRSpec_32f* pSpec);
IppStatus ippsFIRSRInit_64f(const Ipp64f* pTaps, int tapsLen, IppAlgType algType,
IppsFIRSpec_64f* pSpec);
IppStatus ippsFIRSRInit_32fc(const Ipp32fc* pTaps, int tapsLen, IppAlgType algType,
IppsFIRSpec_32fc* pSpec);
IppStatus ippsFIRSRInit_64fc(const Ipp64fc* pTaps, int tapsLen, IppAlgType algType,
IppsFIRSpec_64fc* pSpec);
IppStatus ippsFIRSRInit32f_32fc(const Ipp32f* pTaps, int tapsLen, IppAlgType algType,
IppsFIRSpec32f_32fc* pSpec);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pTaps`	Pointer to the filter coefficients.
`tapsLen`	Length of the FIR filter.
`algType`	Bit-field mask for the algorithm type definition. Possible values are: `ippAlgAuto`, `ippAlgDirect`, or `ippAlgFFT`.
`pSpec`	Pointer to the internal constant FIR specification structure.

Description

Before using this function, you need to compute the size of the specification structure using the `ippsFIRSRGetSize` function. This function initializes the constant specification structure for single-rate FIR filtering.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when the tapsLen value is less than, or equal to zero.
ippStsAlgTypeErr	Indicates an error when the specified algorithm type is not supported.

See Also

FIRSRGetSize Computes the size of the constant structure and work buffer for single-rate FIR filtering.

Examples of Using FIR Functions

FIRSR

Performs single-rate FIR filtering of a source vector.

Syntax

```
IppStatus ippsFIRSR_32f(const Ipp32f* pSrc, Ipp32f* pDst, int numIters,
IppsFIRSpec_32f* pSpec, const Ipp32f* pDlySrc, Ipp32f* pDlyDst, Ipp8u* pBuf);

IppStatus ippsFIRSR_64f(const Ipp64f* pSrc, Ipp64f* pDst, int numIters,
IppsFIRSpec_64f* pSpec, const Ipp64f* pDlySrc, Ipp64f* pDlyDst, Ipp8u* pBuf);

IppStatus ippsFIRSR_32fc(const Ipp32fc* pSrc, Ipp32fc* pDst, int numIters,
IppsFIRSpec_32fc* pSpec, const Ipp32fc* pDlySrc, Ipp32fc* pDlyDst, Ipp8u* pBuf);

IppStatus ippsFIRSR32f_32fc(const Ipp32fc* pSrc, Ipp32fc* pDst, int numIters,
IppsFIRSpec32f_32fc* pSpec, const Ipp32fc* pDlySrc, Ipp32fc* pDlyDst, Ipp8u* pBuf);

IppStatus ippsFIRSR_64fc(const Ipp64fc* pSrc, Ipp64fc* pDst, int numIters,
IppsFIRSpec_64fc* pSpec, const Ipp64fc* pDlySrc, Ipp64fc* pDlyDst, Ipp8u* pBuf);

IppStatus ippsFIRSR_16s(const Ipp16s* pSrc, Ipp16s* pDst, int numIters,
IppsFIRSpec_32f* pSpec, const Ipp16s* pDlySrc, Ipp16s* pDlyDst, Ipp8u* pBuf);

IppStatus ippsFIRSR_16sc(const Ipp16sc* pSrc, Ipp16sc* pDst, int numIters,
IppsFIRSpec_32fc* pSpec, const Ipp16sc* pDlySrc, Ipp16sc* pDlyDst, Ipp8u* pBuf);
```

Include Files

ipps.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

pSrc	Pointer to the source vector.

pDst	Pointer to the destination vector.
numIters	Number of elements in the destination vector.
pSpec	Pointer to the internal constant specification structure.
pDlySrc	Pointer to the array containing values for the source delay lines.
pDlyDst	Pointer to the array containing values for the destination delay line.
pBuf	Pointer to the work buffer.

Description

Before using this function, you need to initialize the internal constant specification structure using the ippsFIRSRInit function.

This function filters the source vector using the single-rate FIR filter. Filtering is performed by the following formula:

$$y(n) = \sum_{i=0}^{tapsLen-1} h(i) \cdot x(n-i), \quad \begin{matrix} 0 \le n < numIters \\ 0 \le i < tapsLen - 1 \end{matrix}$$

where

- x(0)...x(numIters) is the source vector
- h(0)...h(tapsLen-1) are the FIR filter coefficients

To compute the y(0)...y(tapsLen-1) destination vector, the function uses the pDlySrc array of the delay line. The length of the pDlySrc array is tapsLen - 1 elements.

The first tapsLen - 1 elements of the function are:

y(0)=h(tapsLen-1)*d(0)+h(tapsLen-2)*d(1)+...+h(1)*d(tapsLen-2)+h(0)*x(0)

y(1)=h(tapsLen-1)*d(1)+h(tapsLen-2)*d(2)+...+h(1)*x(0)+h(0)*x(1)

y(tapsLen-1)=h(tapsLen-1)*x(0)+...+h(1)*x(tapsLen-2)+h(0)*x(tapsLen-1)

where

d(0), d(1), d(2), and d(tapsLen-2) are the elements of the pDlySrc array

The last tapsLen - 1 elements of the source vector are copied to the non-zero pDlyDst buffer for the next call of the FIR filter.

The arrays pDlySrc and pDlyDst support NULL values:

- if pDlySrc is NULL, the function uses the delay line with zero values
- if pDlyDst is NULL, the function does not copy any data to the destination delay line

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.

See Also

FIRSRInit Initializes the FIR constant structure for single-rate FIR filtering.

Examples of Using FIR Functions

FIRSparseInit

Initializes a sparse FIR filter structure.

Syntax

```
IppStatus ippsFIRSparseInit_32f(IppsFIRSparseState_32f** ppState, const Ipp32f* pNZTaps, const Ipp32s* pNZTapPos, int nzTapsLen, const Ipp32f* pDlyLine, Ipp8u* pBuffer);
```

```
IppStatus ippsFIRSparseInit_32fc(IppsFIRSparseState_32fc** ppState, const Ipp32fc* pNZTaps, const Ipp32s* pNZTapPos, int nzTapsLen, const Ipp32fc* pDlyLine, Ipp8u* pBuffer);
```

Include Files

ipps.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

pNZTaps	Pointer to the array containing the non-zero tap values. The number of elements in the array is nzTapsLen.
pNZTapPos	Pointer to the array containing positions of the non-zero tap values. The number of elements in the array is nzTapsLen.
nzTapsLen	Number of elements in the array with non-zero tap values.
pDlyLine	Pointer to the array containing the delay line values.
ppState	Double pointer to the sparse FIR state structure.
pBuffer	Pointer to the external buffer for the sparse FIR state structure.

Description

This function initializes a sparse FIR filter state structure `ppState` in the external buffer `pBuffer`. The size of this buffer must be computed previously by calling the function FIRSparseGetStateSize. The initialization function copies the values of filter coefficients from the array `pNZTaps` containing `nzTapsLen` non-zero taps and their positions from the array `pNZTapPos` into the state structure `ppState`. The array `pDlyLine` specifies the delay line values. The number of elements in this array is `pNZTapPos[nzTapsLen-1]`. If the pointer to the array `pDlyLine` is not `NULL`, the array contents are copied into the state structure `ppState`, otherwise the delay line values in the state structure are initialized to 0.

NOTE The values of `nzTapsLen` and `pNZTapPos[nzTapsLen-1]` must be equal to those specified in the function FIRSparseGetStateSize.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if one of the pointers `ppState`, `pNZTaps`, `pNZTapPos`, or `pBuffer` is `NULL`.
`ippStsFIRLenErr`	Indicates an error if `nxTapsLen` is less than or equal to 0.
`ippStsSparseErr`	Indicates an error if positions of the non-zero taps are not in ascending order, or are negative or repetitive.

FIRSparseGetStateSize

Computes the size of the external buffer for the sparse FIR filter structure.

Syntax

```
IppStatus ippsFIRSparseGetStateSize_32f(int nzTapsLen, int order, int* pStateSize);
IppStatus ippsFIRSparseGetStateSize_32fc(int nzTapsLen, int order, int* pStateSize);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`nzTapsLen`	Number of elements in the array containing the non-zero tap values.
`order`	Order of the sparse FIR filter.

pStateSize	Pointer to the computed value of the external buffer.

Description

This function computes the size of the external buffer for a sparse FIR filter structure that is required for the function ippsFIRSparseInit . Computation is based on the specified number of non-zero filter coefficients nzTapsLen and filter order order that is equal to the number of elements in the delay line pNZTapPos[nzTapsLen-1] (see description of the function ippsFIRSparseInit). The result value is stored in the pStateSize.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if pStateSize pointer is NULL.
ippStsFIRLenErr	Indicates an error if nzTapsLen or order is less than or equal to 0; or nzTapsLen is more than order.

FIRSparseGetDlyLine

Retrieves the delay line contents from the sparse FIR filter state structure.

Syntax

```
IppStatus ippsFIRSparseGetDlyLine_32f(const IppsFIRSparseState_32f* pState, Ipp32f* pDlyLine);
```

```
IppStatus ippsFIRSparseGetDlyLine_32fc(const IppsFIRSparseState_32fc* pState, Ipp32fc* pDlyLine);
```

Include Files

ipps.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

pState	Pointer to the sparse FIR filter state structure.
pDlyLine	Pointer to the array holding the delay line values.

Description

This function copies the delay line values from the state structure pState and stores them into pDlyLine. The destination array pDlyLine contains samples in the reverse order as compared to the order of samples in the source vector.

Before calling `ippsFIRSparseGetDlyLine`, the corresponding filter state structure must be initialized with the FIRSparseInit function.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when the `pState` pointer is `NULL`.
`ippStsContextMatchErr`	Indicates an error when the state identifier is incorrect.

See Also

FIRSparseInit Initializes a sparse FIR filter structure.

FIRSparseSetDlyLine

Sets the delay line contents in the sparse FIR filter state structure.

Syntax

```
IppStatus ippsFIRSparseSetDlyLine_32f(IppsFIRSparseState_32f* pState, const Ipp32f* pDlyLine);
```

```
IppStatus ippsFIRSparseSetDlyLine_32fc(IppsFIRSparseState_32fc* pState, const Ipp32fc* pDlyLine);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pState`	Pointer to the FIR filter state structure.
`pDlyLine`	Pointer to the array holding the delay line values.

Description

This function copies the delay line values from `pDlyLine` and stores them into the state structure `pState`. The source array `pDlyLine` must contain samples in the reverse order as compared to the order of samples in the source vector.

Before calling `ippsFIRSparseSetDlyLine`, the corresponding filter state structure must be initialized with the FIRSparseInit function.

Return Values

`ippStsNoErr`	Indicates no error.

ippStsNullPtrErr	Indicates an error when the pState pointer is NULL.
ippStsContextMatchErr	Indicates an error when the state identifier is incorrect.

See Also

FIRSparseInit Initializes a sparse FIR filter structure.

FIRSparse

Filters a source vector through a sparse FIR filter.

Syntax

```
IppStatus ippsFIRSparse_32f(const Ipp32f* pSrc, Ipp32f* pDst, int len,
IppsFIRSparseState_32f* pState);
```

```
IppStatus ippsFIRSparse_32fc(const Ipp32fc* pSrc, Ipp32fc* pDst, int len,
IppsFIRSparseState_32fc* pState);
```

Include Files

ipps.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

pState	Pointer to the sparse FIR filter state structure.
pSrc	Pointer to the source vector.
pDst	Pointer to the destination vector.
len	Number of elements that are filtered.

Description

This function applies the sparse FIR filter to the len elements of the source vector pSrc, and stores the results in pDst. The filter parameters - the number of non-zero taps nzTapsLen, their values pNZTaps and their positions pNZTapPos, and the delay line values pDlyLine – are specified in the sparse FIR filter structure pState that should be previously initialized by calling the function ippsFIRSparseInit.

In the following definition of the sparse FIR filter, the sample to be filtered is denoted x(n), the non-zero taps are denoted pNZTaps(i), their positions are denoted pNZTapPos(i) and the return value is y(n).

The return value y(n) is defined by the formula for a sparse FIR filter:

$$y(n) = \sum_{i=0}^{nzTapsLen-1} pNZTaps(i) \cdot x(n - pNZTapPos(i)), \quad 0 \le n < len$$

After the function has performed calculations, it updates the delay line values stored in the state.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error if `len` is less or equal to 0.

Example

The example below shows how to use the sparse FIR filter functions.

```
int buflen;
Ipp8u *buf;
int nzTapsLen = 5; //number of non-zero taps
Ipp32f nzTaps [] = {0.5, 0.4, 0.3, 0.2, 0.1};  //non-zero taps values
Ipp32s nzTapsPos[] = {0, 10, 20, 30, 40};  //non-zero tap positions
IppsFIRSparseState_32f* firState;
Ipp32f *src, *dst;


/* ........................... */
ippsFIRSparseGetStateSize_32f(nzTapsLen, nzTapsPos [nzTapsLen - 1], &buflen);
buf = ippsMalloc_8u(buflen);
ippsFIRSparseInit_32f(&firState, nzTaps, nzTapsPos, nzTapsLen, NULL, buf);


/* .... initializing src somehow .... */
ippsFIRSparse_32f(src, dst, len, firState);


/*dst[i]=src[i]*0.5 + src[i-10]*0.4 + src[i-20]*0.3 + src[i-30]*0.2 + src[i-40]*0.1 */


/* ...........................*/
ippsFree(buf);
```

Examples of Using FIR Functions

The code examples below demonstrate how to use the `ippsFIRSR` function:

- Standard FIR Filtering with a Not-in-place Destination
- Standard FIR Filtering with an In-place Destination
- Stream FIR Filtering with a Not-in-place Destination
- Stream FIR Filtering with an In-place Destination
- Standard FIR Filtering with a Not-in-place Destination and Threading
- Standard FIR Filtering with an In-place Destination and Threading

Standard FIR Filtering with a Not-in-place Destination

Type of FIR Filter	Destination	Source Delay Line	Threading
standard	not-in-place	zero	none

```
#define LEN 1024
#define TAPS_LEN 8


IppsFIRSpec_32f *pSpec;
float           *src, *dst, *dly, *taps;
Ipp8u           *buf;
int             specSize, bufSize;
IppStatus status;
//get sizes of the spec structure and the work buffer
status = ippsFIRSRGetSize (TAPS_LEN,  ipp32f ,  &specSize,  &bufSize );


src   = ippsMalloc_32f(LEN);
dst   = ippsMalloc_32f(LEN);
dly   = ippsMalloc_32f(TAPS_LEN-1);
taps  = ippsMalloc_32f(TAPS_LEN);
pSpec = (IppsFIRSpec_32f*)ippsMalloc_8u(specSize);
buf   = ippsMalloc_8u(bufSize);


//initialize the spec structure
ippsFIRSRInit_32f( taps, TAPS_LEN, ippAlgDirect, pSpec );
//apply the FIR filter
ippsFIRSR_32f(src, dst, LEN,  pSpec,  NULL, dly,  buf);
```

Standard FIR Filtering with an In-place Destination

Type of FIR Filter	Destination	Source Delay Line	Threading
standard	in-place	zero	none

```
#define LEN 1024
#define TAPS_LEN 8


IppsFIRSpec_32f *pSpec;
float           *src, *dst, *dly, *taps;
Ipp8u           *buf;
int             specSize, bufSize;


//get sizes of the spec structure and the work buffer
ippsFIRSRGetSize(TAPS_LEN, ipp32f, &specSize, &bufSize );


src   = ippsMalloc_32f(LEN);
dst   = src;
dly   = ippsMalloc_32f(TAPS_LEN-1);
taps  = ippsMalloc_32f(TAPS_LEN);
pSpec = (IppsFIRSpec_32f*)ippsMalloc_8u(specSize);
buf   = ippsMalloc_8u(bufSize);
```

```
//initialize the spec structure
ippsFIRSRInit_32f( taps, TAPS_LEN, ippAlgDirect, pSpec );
//apply the FIR filter
ippsFIRSR_32f(src, dst, LEN,  pSpec,  NULL, dly,  buf);
```

Stream FIR Filtering with a Not-in-place Destination

Type of FIR Filter	Destination	Source Delay Line	Threading
stream	not-in-place	`src`	none

```
#define LEN 1024
#define TAPS_LEN 8


IppsFIRSpec_32f *pSpec;
float           *src, *dst,  *taps;
Ipp8u           *buf;
int             specSize, bufSize;


//get sizes of the spec structure and the work buffer
ippsFIRSRGetSize(TAPS_LEN, ipp32f, &specSize, &bufSize );


src   = ippsMalloc_32f(LEN+TAPS_LEN-1);
dst   = ippsMalloc_32f(LEN);
taps  = ippsMalloc_32f(TAPS_LEN);
pSpec = (IppsFIRSpec_32f*)ippsMalloc_8u(specSize);
buf   = ippsMalloc_8u(bufSize);


//initialize the spec structure
ippsFIRSRInit_32f( taps, TAPS_LEN, ippAlgDirect, pSpec );
//apply the FIR filter
ippsFIRSR_32f(src+TAPS_LEN-1, dst, LEN,  pSpec,  src, NULL,  buf);
```

Stream FIR Filtering with an In-place Destination

Type of FIR Filter	Destination	Source Delay Line	Threading
stream	in-place	`src`	none

```
#define LEN 1024
#define TAPS_LEN 8


IppsFIRSpec_32f *pSpec;
float           *src, *dst, *taps;
Ipp8u           *buf;
int             specSize, bufSize;


//get sizes of the spec structure and the work buffer
ippsFIRSRGetSize(TAPS_LEN, ipp32f, &specSize, &bufSize );
```

```
src   = ippsMalloc_32f(LEN+TAPS_LEN-1);
dst   = src;
taps  = ippsMalloc_32f(TAPS_LEN);
pSpec = (IppsFIRSpec_32f*)ippsMalloc_8u(specSize);
buf   = ippsMalloc_8u(bufSize);


//initialize the spec structure
ippsFIRSRInit_32f( taps, TAPS_LEN, ippAlgDirect, pSpec );
//apply the FIR filter
ippsFIRSR_32f(src+TAPS_LEN-1, dst, LEN,  pSpec,  src, NULL,  buf);
```

Standard FIR Filtering with a Not-in-place Destination and Threading

Type of FIR Filter	Destination	Source Delay Line	Threading
standard	not-in-place	zero	NTHREADS

```
#define LEN 1024
#define TAPS_LEN 8
#define DLY_LEN TAPS_LEN-1
#define NTH 4


float *src,*dst;
float *dlyOut, *taps;
unsigned char *buf;
IppsFIRSpec_32f* pSpec;
int specSize, bufSize;
int i,tlen, ttail;


//get sizes of the spec structure and the work buffer
ippsFIRSRGetSize(TAPS_LEN, ipp32f, &specSize, &bufSize );


src    = ippsMalloc_32f(LEN);
dst    = ippsMalloc_32f(LEN);
dlyOut = ippsMalloc_32f(TAPS_LEN-1);
taps   = ippsMalloc_32f(TAPS_LEN);
pSpec  = (IppsFIRSpec_32f*)ippsMalloc_8u(specSize);
buf    = ippsMalloc_8u(bufSize*NTH);
for(i=0;i<LEN;i++){
    src[i] = i;
}
for(i=0;i<TAPS_LEN;i++){
    taps[i] = 1;
}
//initialize the spec structure
ippsFIRSRInit_32f( taps, TAPS_LEN, ippAlgDirect, pSpec );
tlen  = LEN / NTH;
ttail = LEN % NTH;
for(i=0;i< NTH;i++) {//this cycle means parallel region
    Ipp32f* s   = src+i*tlen;
    Ipp32f* d   = dst+i*tlen;
    int     len = tlen+((i==(NTH - 1))?ttail:0);
    Ipp8u*  b   = buf+i*bufSize;
```

```
    if( i == 0)
        ippsFIRSR_32f(s, d, len, pSpec, NULL,           NULL  , b);
    else if (i == NTH - 1)
        ippsFIRSR_32f(s, d, len, pSpec, s-(TAPS_LEN-1), dlyOut, b);
    else
        ippsFIRSR_32f(s, d, len, pSpec, s-(TAPS_LEN-1), NULL  , b);
}
```

Standard FIR Filtering with an In-place Destination and Threading

Type of FIR Filter	Destination	Source Delay Line	Threading
standard	in-place	zero	NTHREADS

```
#define LEN 1024
#define TAPS_LEN 8
#define DLY_LEN TAPS_LEN-1
#define NTHREADS 4


float *src, *dst,  *dlyOut, *taps;
float* tdly[NTHREADS];
unsigned char *buf;
IppsFIRSpec_32f* pSpec;
int specSize, bufSize;
int i,tlen, ttail;


//get sizes of the spec structure and the work buffer
ippsFIRSRGetSize(TAPS_LEN, ipp32f, &specSize, &bufSize );


src    = ippsMalloc_32f(LEN);
dst    = ippsMalloc_32f(LEN);
dlyOut = ippsMalloc_32f(TAPS_LEN-1);
taps   = ippsMalloc_32f(TAPS_LEN);
pSpec  = (IppsFIRSpec_32f*)ippsMalloc_8u(specSize);
buf    = ippsMalloc_8u(bufSize*NTHREADS);


//initialize the spec structure
ippsFIRSRInit_32f( taps, TAPS_LEN, ippAlgDirect, pSpec );
tlen  = LEN / NTHREADS;
ttail = LEN % NTHREADS;
//tdly    = ippsMalloc_32f((TAPS_LEN-1)*(NTHREADS-1));


for(i=1;i<NTHREADS;i++){//cycle in main thread
    tdly[i]  = ippsMalloc_32f(TAPS_LEN-1);
    ippsCopy_32f(src+i*tlen-(TAPS_LEN-1), tdly[i], TAPS_LEN-1);
}
for(i=0;i< NTHREADS;i++) {//this cycle means parallel region
    Ipp32f* s   = src+i*tlen;
    Ipp32f* d   = dst+i*tlen;
    int     len = tlen+((i==NTHREADS - 1)?ttail:0);
    Ipp8u*  b   = buf+i*bufSize;
    if( i == 0)
        ippsFIRSR_32f(s, d, len, pSpec, NULL,    NULL  , b);
```

```
    else if (i == NTHREADS - 1)
        ippsFIRSR_32f(s, d, len, pSpec, tdly[i], dlyOut, b);
    else
        ippsFIRSR_32f(s, d, len, pSpec, tdly[i], NULL  , b);
}
```

FIR Filter Coefficient Generating Functions

The functions described in this section compute coefficients (tap values) for different FIR filters by windowing the ideal infinite filter coefficients.

- FIRGenGetBufferSize
- FIRGenLowpass
- FIRGenHighpass
- FIRGenBandpass
- FIRGenBandstop

FIRGenGetBufferSize

Computes the size of the internal buffer required for computation of FIR coefficients.

Syntax

```
IppStatus ippsFIRGenGetBufferSize(int tapsLen, int* pBufferSize);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`tapsLen`	Number of taps.
`pBufferSize`	Pointer to the calculated buffer size (in bytes).

Description

This function computes the size of the buffer that is required for ippsFIRGenBandpass, ippsFIRGenBandstop, ippsFIRGenHighpass, and ippsFIRGenLowpass internal calculations.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when any of the specified pointers is `NULL`.
`ippStsIIRGenOrderErr`	Indicates an error when the length of the coefficients array is less than 5.

See Also

FIRGenBandpass Computes bandpass FIR filter coefficients.

FIRGenBandstop Computes bandstop FIR filter coefficients.

FIRGenHighpass Computes highpass FIR filter coefficients.

FIRGenLowpass Computes lowpass FIR filter coefficients.

FIRGenLowpass

Computes lowpass FIR filter coefficients.

Syntax

```
IppStatus ippsFIRGenLowpass_64f(Ipp64f rFreq, Ipp64f* pTaps, int tapsLen, IppWinType winType, IppBool doNormal, Ipp8u* pBuffer);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`rFreq`	Normalized cutoff frequency, must be in the range (0, 0.5).
`pTaps`	Pointer to the array where computed tap values are stored. The number of elements in the array is `tapsLen`.
`tapsLen`	Number of elements in the array containing the tap values; must be equal or greater than 5.
`winType`	Specifies what type of window is used in computations. The `winType` must have one of the following values: • `ippWinBartlett` - Bartlett window • `ippWinBlackman` - Blackman window • `ippWinHamming` - Hamming window • `ippWinHann` - Hann window
`doNormal`	Specifies normalized or non-normalized sequence of the filter • `ippTrue` for normalized sequence of coefficients • `ippFalse` for non-normalized sequence of coefficients

	coefficients is computed. The `doNormal` must have one of the following values:
`pBuffer`	Pointer to the buffer for internal calculations. To get the size of the buffer, use the ippsFIRGenGetBufferSize function.

Description

This function computes `tapsLen` coefficients for lowpass FIR filter with the cutoff frequency `rFreq` by windowing the ideal infinite filter coefficients. The quality of filtering is defined by the number of coefficients. The parameter `winType` specifies the type of the window. For more information on window types used by the function, see Windowing Functions. The computed coefficients are stored in the array `pTaps`.

For more information about the used algorithm, see [MIT 93].

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when the `pTaps` pointer is `NULL`.
`ippStsSizeErr`	Indicates an error when the `tapsLen` is less than 5, or `rFreq` is out of range.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

#include <stdio.h>
#include "ipp.h"

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define
```

```
check_sts(st)
                                    \
    if ((st) !=
ippStsNoErr)
                            \
        goto exitLine; /* Go to Exit if Intel(R) Integrated Performance Primitives (Intel(R)
IPP) function returned status different from \
                           ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main()
{
    int len = 512;
    Ipp64f pDst[512];
    Ipp64f magn = 4095;
    Ipp64f rFreq = 0.2;
    int tapslen = 27;
    int numIters = 512;
    IppsFIRSpec_64f *pSpec = NULL;
    IppStatus status = ippStsNoErr;
    int i = 0, bufSize = 0, specSize = 0;
    Ipp8u *pBuffer = NULL;
    IppAlgType algType = ippAlgDirect;
    Ipp64f *pDlySrc = NULL;
    Ipp64f *pDlyDst = NULL;

    Ipp64f *FIRDst = ippsMalloc_64f(numIters * sizeof(Ipp64f));
    Ipp64f *taps = ippsMalloc_64f(tapslen * sizeof(Ipp64f));

    pDlySrc = ippsMalloc_64f(tapslen * sizeof(Ipp64f));
    check_sts(status = ippsZero_64f(pDlySrc, tapslen));

    /*generate source vector*/
    check_sts(status = ippsVectorJaehne_64f(pDst, len, magn)); /* create a Jaehne vector */
    printf("\nSource vector\n");
    for (i = 0; i < 32; i++)
        printf("%f,", pDst[i]);

    /*computes tapsLen coefficients for lowpass FIR filter*/
    check_sts(status = ippsFIRGenGetBufferSize(tapslen, &bufSize));

    pBuffer = ippsMalloc_8u(bufSize);

    check_sts(status = ippsFIRGenLowpass_64f(rFreq, taps, tapslen, ippWinBartlett, ippTrue,
pBuffer));

    printf("\nGenerated taps:\n");
    for (i = 0; i < tapslen; i++)
        printf("%f,", taps[i]);

    check_sts(status = ippsFIRSRGetSize(tapslen, ipp64f, &specSize, &bufSize));
    pSpec = (IppsFIRSpec_64f *)ippsMalloc_8u(specSize);
    pBuffer = ippsMalloc_8u(bufSize);

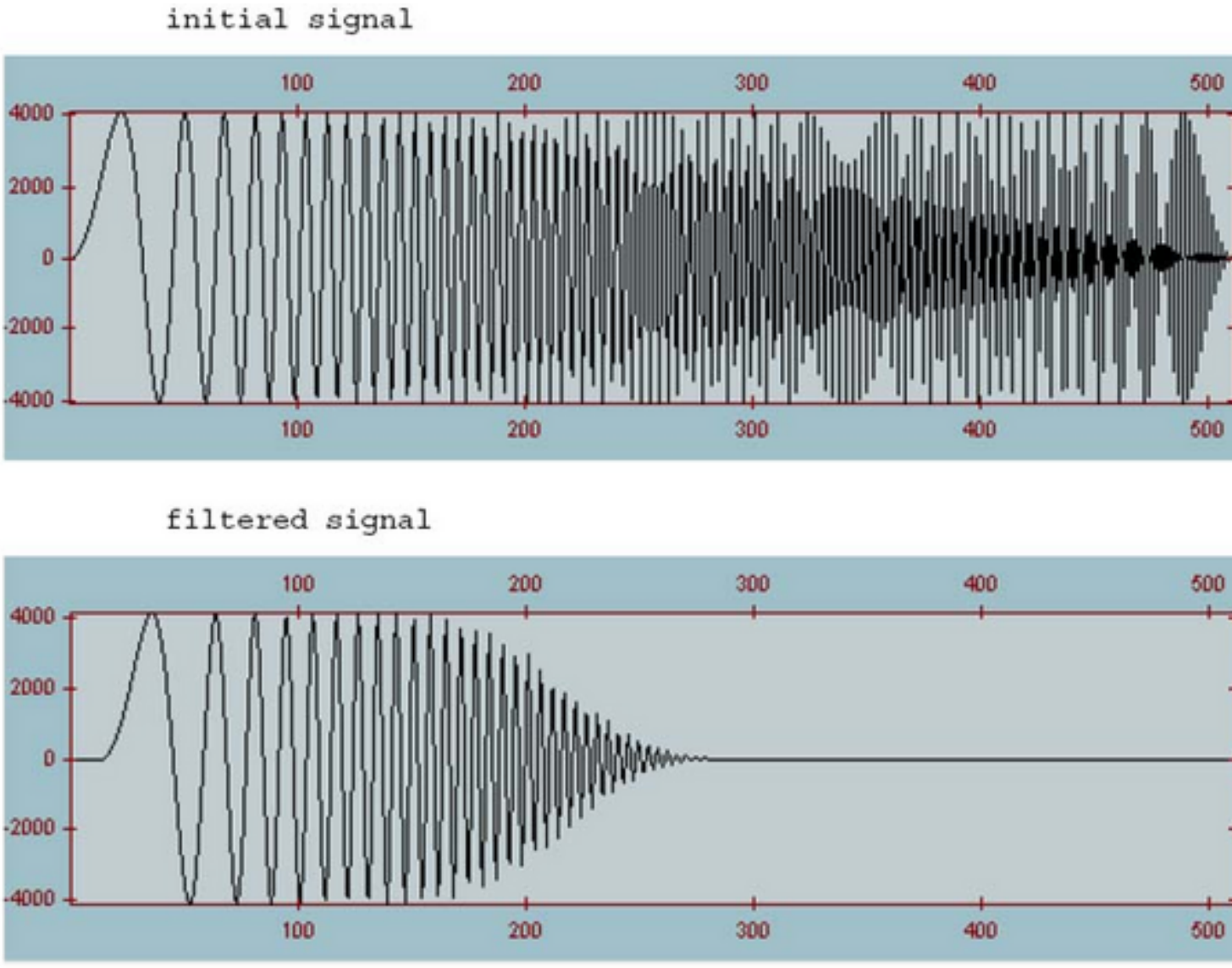
    check_sts(status = ippsFIRSRInit_64f(taps, tapslen, algType, pSpec));
```

```
    /* filter an input vector */
    check_sts(status = ippsFIRSR_64f(pDst, FIRDst, numIters, pSpec, pDlySrc, pDlyDst, pBuffer));

    printf("\nAfter LowPass filter\n");
    for (i = 0; i < 32; i++)
        printf("%f,", FIRDst[i]);
    printf("\n");

    EXIT_MAIN
    ippsFree(FIRDst);
    ippsFree(taps);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

Result:



FIRGenHighpass

Computes highpass FIR filter coefficients.

Syntax

```
IppStatus ippsFIRGenHighpass_64f(Ipp64f rFreq, Ipp64f* pTaps, int tapsLen, IppWinType winType, IppBool doNormal, Ipp8u* pBuffer);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`rFreq`	Normalized cutoff frequency, must be in the range (0, 0.5).
`pTaps`	Pointer to the array where computed tap values are stored. The number of elements in the array is `tapsLen`.
`tapsLen`	Number of elements in the array containing the tap values; must be equal or greater than 5.
`winType`	Specifies what type of window is used in computations. The `winType` must have one of the following values: • `ippWinBartlett` Bartlett window; • `ippWinBlackman` Blackman window; • `ippWinHamming` Hamming window; • `ippWinHann` Hann window.
`doNormal`	Specifies normalized or non-normalized sequence of the filter coefficients is computed. The `doNormal` must have • `ippTrue` The function computes normalized sequence of coefficients. • `ippFalse` The function computes non-normalized sequence of coefficients.

	one of the following values:
pBuffer	Pointer to the buffer for internal calculations. To get the size of the buffer, use the ippsFIRGenGetBufferSize function.

Description

This function computes tapsLen coefficients for highpass FIR filter the cutoff frequency rFreq by windowing the ideal infinite filter coefficients. The parameter winType specifies the type of the window. For more information on window types used by the function, see Windowing Functions.he computed coefficients are stored in the array pTaps.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when the pTaps pointer is NULL.
ippStsSizeErr	Indicates an error when the tapsLen is less than 5, or rFreq is out of the range.

FIRGenBandpass

Computes bandpass FIR filter coefficients.

Syntax

```
IppStatus ippsFIRGenBandpass_64f(Ipp64f rLowFreq, Ipp64f rHighFreq, Ipp64f* pTaps, int tapsLen, IppWinType winType, IppBool doNormal, Ipp8u* pBuffer);
```

Include Files

ipps.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

rLowFreq	Normalized low cutoff frequency, must be in the range (0, 0.5) and less than rHighFreq.
rHighFreq	Normalized high cutoff frequency, must be in the range (0, 0.5) and greater than rLowFreq.
pTaps	Pointer to the array where computed tap values are stored. The number of elements in the array is tapsLen.

tapsLen	Number of elements in the array containing the tap values; should be equal or greater than 5.
winType	Specifies what type of window is used in computations. The winType must have one of the following values: • ippWinBartlett Bartlett window; • ippWinBlackman Blackman window; • ippWinHamming Hamming window; • ippWinHann Hann window.
doNormal	Specifies normalized or non-normalized sequence of the filter coefficients is computed. The doNormal must have one of the following values: • ippTrue The function computes normalized sequence of coefficients. • ippFalse The function computes non-normalized sequence of coefficients.
pBuffer	Pointer to the buffer for internal calculations. To get the size of the buffer, use the ippsFIRGenGetBufferSize function.

Description

This function computes tapsLen coefficients for bandpass FIR filter with the cutoff frequencies rLowFreq and rHighFreq by windowing the ideal infinite filter coefficients. The parameter winType specifies the type of the window. For more information on window types used by the function, see Windowing Functions. The computed coefficients are stored in the array pTaps.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when the pTaps pointer is NULL.
ippStsSizeErr	Indicates an error when the tapsLen is less than 5, or rLowFreq is greater than or equal to rHighFreq, or one of the frequency parameters rLowFreq and rHighFreq is out of the range.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

#include <stdio.h>
#include "ipp.h"

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define
check_sts(st)
                                    \
    if ((st) !=
ippStsNoErr)
                           \
        goto exitLine; /* Go to Exit if Intel(R) Integrated Performance Primitives (Intel(R)
IPP) function returned status different from \
                            ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main()
{
    int len = 512;
    Ipp64f pDst[512];
    Ipp64f magn = 4095;
    Ipp64f rLowFreq = 0.2;
    Ipp64f rHighFreq = 0.3;
    int tapslen = 27;
    int numIters = 512;
    IppsFIRSpec_64f *pSpec = NULL;
    IppStatus status = ippStsNoErr;
    int i = 0, bufSize = 0, specSize = 0;
    Ipp8u *pBuffer = NULL;
    IppAlgType algType = ippAlgDirect;
    Ipp64f *pDlySrc = NULL;
    Ipp64f *pDlyDst = NULL;

    Ipp64f *FIRDst = ippsMalloc_64f(numIters * sizeof(Ipp64f));
    Ipp64f *taps = ippsMalloc_64f(tapslen * sizeof(Ipp64f));

    pDlySrc = ippsMalloc_64f(tapslen * sizeof(Ipp64f));
    check_sts(status = ippsZero_64f(pDlySrc, tapslen));
```

```
    /* generate source vector */
    check_sts(status = ippsVectorJaehne_64f(pDst, len, magn)); /* create a Jaehne vector */
    printf("\nSource vector\n");
    for (i = 0; i < 32; i++)
        printf("%f,", pDst[i]);

    /* computes tapsLen coefficients for lowpass FIR filter */
    check_sts(status = ippsFIRGenGetBufferSize(tapslen, &bufSize));

    pBuffer = ippsMalloc_8u(bufSize);

    check_sts(status = ippsFIRGenBandpass_64f(rLowFreq, rHighFreq, taps, tapslen,
ippWinBartlett, ippTrue, pBuffer));

    printf("\nGenerated taps:\n");
    for (i = 0; i < tapslen; i++)
        printf("%f,", taps[i]);

    check_sts(status = ippsFIRSRGetSize(tapslen, ipp64f, &specSize, &bufSize));
    pSpec = (IppsFIRSpec_64f *)ippsMalloc_8u(specSize);
    pBuffer = ippsMalloc_8u(bufSize);

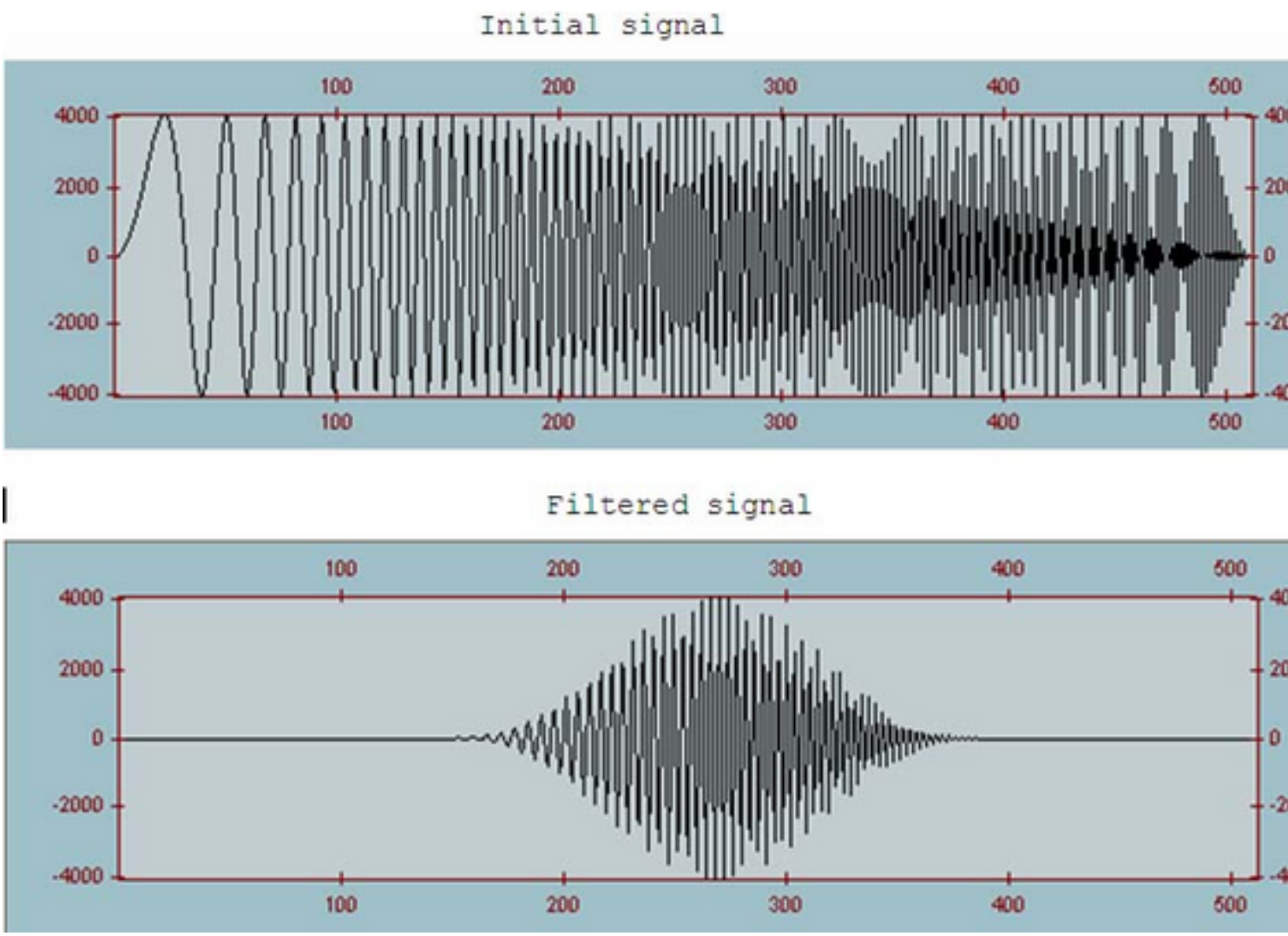
    check_sts(status = ippsFIRSRInit_64f(taps, tapslen, algType, pSpec));

    /*filter an input vector*/
    check_sts(status = ippsFIRSR_64f(pDst, FIRDst, numIters, pSpec, pDlySrc, pDlyDst, pBuffer));

    printf("\nAfter LowPass filter\n");
    for (i = 0; i < 32; i++)
        printf("%f,", FIRDst[i]);
    printf("\n\n");

    EXIT_MAIN
    ippsFree(FIRDst);
    ippsFree(taps);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return 0;
}
```

Result:



FIRGenBandstop

Computes bandstop FIR filter coefficients.

Syntax

```
IppStatus ippsFIRGenBandstop_64f(Ipp64f rLowFreq, Ipp64f rHighFreq, Ipp64f* pTaps, int tapsLen, IppWinType winType, IppBool doNormal, Ipp8u* pBuffer);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`rLowFreq`	Normalized low cutoff frequency, must be in the range (0, 0.5) and less than `rHighFreq`.
`rHighFreq`	Normalized high cutoff frequency, must be in the range (0, 0.5) and greater than `rLowFreq`.
`pTaps`	Pointer to the array where computed tap values are stored. The number of elements in the array is `tapsLen`.
`tapsLen`	Number of elements in the array containing the tap values, must be equal or greater than 5.
`winType`	Specifies what type of window is used in computations. The `winType` must have one of the following values: • `ippWinBartlett` Bartlett window; • `ippWinBlackman` Blackman window; • `ippWinHamming` Hamming window; • `ippWinHann` Hann window.
`doNormal`	Specifies normalized or non-normalized sequence of the filter coefficients is computed. The `doNormal` must have one of the following values: • `ippTrue` The function computes normalized sequence of coefficients. • `ippFalse` The function computes non-normalized sequence of coefficients.
`pBuffer`	Pointer to the buffer for internal calculations. To get the size of the buffer, use the ippsFIRGenGetBufferSize function.

Description

This function computes `tapsLen` coefficients for bandstop FIR filter with the cutoff frequencies `rLowFreq` and `rHighFreq` by windowing the ideal infinite filter coefficients. The parameter `winType` specifies the type of the window. For more information on window types used by the function, see Windowing Functions. The computed coefficients are stored in the array `pTaps`.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when the pTaps pointer is NULL.
ippStsSizeErr	Indicates an error when the tapsLen is less than 5, or rLowFreq is greater than or equal to rHighFreq, or one of the frequency parameters rLowFreq and rHighFreq is out of the range.

Single-Rate FIR LMS Filter Functions

The functions described in this section perform the following tasks:

- initialize a single-rate FIR least mean squares (LMS) filter
- get and set the delay line values
- get the filter coefficients (taps) values
- perform filtering

- FIRLMSGetTaps
- FIRLMSGetDlyLine
- FIRLMSSetDlyLine
- FIRLMSGetStateSize
- FIRLMSInit
- FIRLMS

FIRLMSGetTaps

Retrieves the tap values from the FIR LMS filter.

Syntax

```
IppStatus ippsFIRLMSGetTaps_32f(const IppsFIRLMSState_32f* pState, Ipp32f* pOutTaps);
```

```
IppStatus ippsFIRLMSGetTaps32f_16s(const IppsFIRLMSState32f_16s* pState, Ipp32f*
pOutTaps);
```

Include Files

ipps.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

pState	Pointer to the FIR LMS filter state structure.
pOutTaps	Pointer to the array holding copies of the taps.

Description

This function copies the taps from the state structure pState to the tapsLen - length array pOutTaps.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsContextMatchErr	Indicates an error when the state identifier is incorrect.

FIRLMSGetDlyLine

Retrieves the delay line contents from the FIR LMS filter.

Syntax

```
IppStatus ippsFIRLMSGetDlyLine_32f(const IppsFIRLMSState_32f* pState, Ipp32f* pDlyLine, int* pDlyLineIndex);
```

```
IppStatus ippsFIRLMSGetDlyLine32f_16s(const IppsFIRLMSState32f_16s* pState, Ipp16s* pDlyLine, int* pDlyLineIndex);
```

Include Files

ipps.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

pState	Pointer to the FIR LMS filter state structure.
pDlyLine	Pointer to the tapsLen - length array holding the delay line values.
pDlyLineIndex	Pointer to the array to store the current delay line index copied from the filter state structure.

Description

This function copies the delay line values and the current delay line index from the state structure pState, and stores them into pDlyLine and pDlyLineIndex, respectively.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsContextMatchErr	Indicates an error when the state identifier is incorrect.

FIRLMSSetDlyLine

Sets the delay line contents in the FIR LMS filter.

Syntax

```
IppStatus ippsFIRLMSGetDlyLine_32f(const IppsFIRLMSState_32f* pState, Ipp32f* pDlyLine, int* pDlyLineIndex);
IppStatus ippsFIRLMSGetDlyLine32f_16s(const IppsFIRLMSState32f_16s* pState, Ipp16s* pDlyLine, int* pDlyLineIndex);
IppStatus ippsFIRLMSSetDlyLine_32f(IppsFIRLMSState_32f* pState, const Ipp32f* pDlyLine, int dlyLineIndex);
IppStatus ippsFIRLMSSetDlyLine32f_16s(IppsFIRLMSState32f_16s* pState, const Ipp16s* pDlyLine, int dlyLineIndex);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pState`	Pointer to the FIR LMS filter state structure.
`pDlyLine`	Pointer to the `tapsLen` - length array holding the delay line values.
`pDlyLineIndex`	Pointer to the index of the delay line.
`dlyLineIndex`	Initial index of the delay line to be stored in the filter state structure `pState`.

Description

This function copies the delay line values from `pDlyLine`, and the current delay line index from `dlyLineIndex`, and stores them into the state structure `pState`.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when one of the specified pointers is `NULL`.
`ippStsContextMatchErr`	Indicates an error when the state identifier is incorrect.

FIRLMSGetStateSize

Computes the size of the external buffer for the FIR least mean squares (LMS) filter structure.

Syntax

```
IppStatus ippsFIRLMSGetStateSize32f_16s(int tapsLen, int dlyIndex, int* pBufferSize);
IppStatus ippsFIRLMSGetStateSize_32f(int tapsLen, int dlyIndex, int* pBufferSize);
```

Include Files

ipps.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

tapsLen	Number of elements in the array containing tap values.
dlyIndex	Current index of the delay line.
pBufferSize	Pointer to the computed buffer size value.

Description

This function computes the size of the external buffer for the FIR LMS filter state structure and stores the result in pBufferSize.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pBufferSize is NULL.
ippStsFIRLenErr	Indicates an error when tapsLen is less than, or equal to zero.

See Also

FIRLMS Filters a vector through the FIR least mean squares (LMS) filter.

FIRLMSInit

Initializes the adaptive FIR least mean squares (LMS) filter state structure.

Syntax

```
IppStatus ippsFIRLMSInit32f_16s(IppsFIRLMSState32f_16s** ppState, const Ipp32f* pTaps,
int tapsLen, const Ipp16s* pDlyLine, int dlyIndex, Ipp8u* pBuffer);
IppStatus ippsFIRLMSInit_32f(IppsFIRLMSState_32f** ppState, const Ipp32f* pTaps, int
tapsLen, const Ipp32f* pDlyLine, int dlyIndex, Ipp8u* pBuffer);
```

Include Files

ipps.h

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`ppState`	Double pointer to the state structure.
`pTaps`	Pointer to the array of tap values.
`tapsLen`	Number of elements in the array containing tap values.
`pDlyLine`	Pointer to the array containing delay line values. The number of elements in the array is 2* `tapsLen`.
`dlyIndex`	Current index of the delay line.
`pBuffer`	Pointer to the external buffer for the FIR LMS state structure.

Description

This function initializes the single-rate FIR LMS filter state structure. The `ippsFIRLMSInit` function copies the taps from the `pTaps` array of `tapsLen` length into the state structure `ppTaps`. The `pDlyLine` array of size 2* `tapsLen` specifies the delay line values. The current index of the delay line is defined by `dlyIndex`. If the pointer to `pDlyLine` or `pTaps` is `NULL`, the corresponding value of the state structure is initialized to zero.

To compute the size of the buffer required for the FIR LMS state structure, use the ippsFIRLMSGetStateSize function.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when either `ppState` or `pBuffer` is `NULL`.
`ippStsFIRLenErr`	Indicates an error when `tapsLen` is less than, or equal to zero.

See Also

FIRLMS Filters a vector through the FIR least mean squares (LMS) filter.

FIRLMSGetStateSize Computes the size of the external buffer for the FIR least mean squares (LMS) filter structure.

FIRLMS

Filters a vector through the FIR least mean squares (LMS) filter.

Syntax

```
IppStatus ippsFIRLMS_32f(const Ipp32f* pSrc, const Ipp32f* pRef, Ipp32f* pDst, int len, float mu, IppsFIRLMSState_32f* pState);
```

```
IppStatus ippsFIRLMS32f_16s(const Ipp16s* pSrc, const Ipp16s* pRef, Ipp16s* pDst, int len, float mu, IppsFIRLMSState32f_16s* pState);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pState`	Pointer to the FIR LMS filter state structure.
`pSrc`	Pointer to the source vector .
`pRef`	Pointer to the reference signal
`pDst`	Pointer to the output signal
`len`	Number of elements in the vector.
`mu`	Adaptation step.

Description

Before calling this function, compute the size of the buffer required for the `pState` structure using FIRLMSGetStateSize and initialize the structure using FIRLMSInit.

This function filters a source vector `pSrc` using an adaptive FIR LMS filter.

Each of `len` iterations performed by the function consists of two main procedures. First, `ippsLMS` filters the current element of the source vector `pSrc` and stores the result in `pDst`. Next, the function updates the current taps using the reference signal `pRef`, the computed result signal `pDst`, and the adaptation step `mu`.

The filtering procedure can be described as a FIR filter operation:

$$y(n) = \sum_{i=0}^{tapsLen-1} h(i) \cdot x(n-i)$$

Here the input sample to be filtered is denoted by `x(n)`, the taps are denoted by `h(i)`, and `y(n)` is the return value.

The function updates the filter coefficients that are stored in the filter state structure `pState`. Updated filter coefficients are defined as h_{n+1}`(i) =`h_n`(i) + 2*mu*errVal*x(n-i),`

where h_{n+1}`(i)` denotes new taps, h_n`(i)` denotes initial taps, `mu` and `errVal` are the adaptation step and adaptation error value, respectively. An adaptation error value `errVal` is computed inside the function as the difference between the output and reference signals.

Return Values

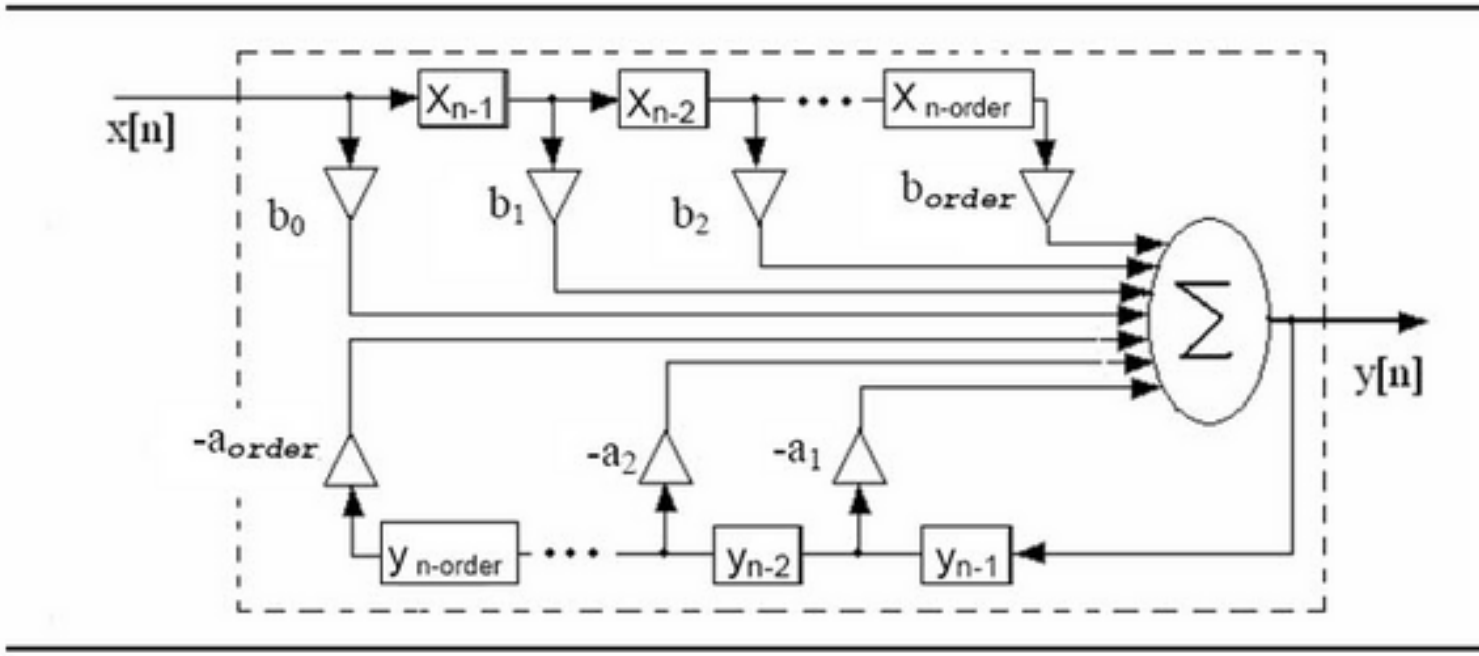
`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error when `len` is less than, or equal to 0.
`ippStsContextMatchErr`	Indicates an error when the state identifier is incorrect.

IIR Filter Functions

The functions described in this section initialize an infinite impulse response (IIR) filter and perform filtering. Intel IPP supports two types of filters: arbitrary order filter and biquad filter.

The figure below shows the structure of an arbitrary order IIR filter.

Structure of an Arbitrary Order Filter



Here `x[n]` is a sample of the input signal, `y[n]` is a sample of the output signal, `order` is the filter order, and b_0, b_1, …, b_{order}, a_1, …, a_{order} are the reduced filter coefficients.

The output signal is computed by the following formula:

$$y[n] = \sum_{k=0}^{order} b_k \cdot x(n-k) - \sum_{k=1}^{order} a_k \cdot y(n-k)$$

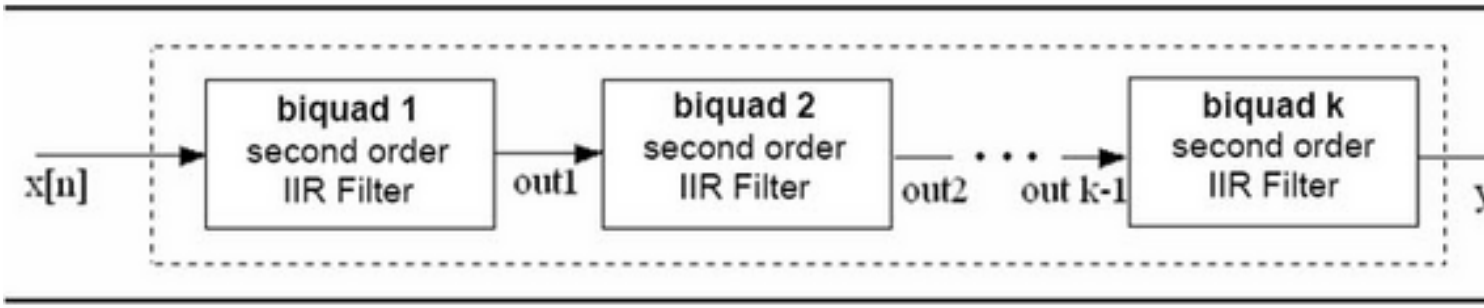
.

Reduced coefficients are calculated as $a_k = A_k/A_0$ and $b_k = B_k/A_0$

where A_0, A_1,…A_{order}, B_0, B_1,…B_{order} are initial filter coefficients (taps).

A biquad IIR filter is a cascade of second-order filters. The figure below illustrates the structure of the biquad filter with **k** cascades of second-order filters.

Structure of a BiQuad IIR Filter



By default, all Intel IPP IIR filter functions that do not have the `_DF1_` suffix in a name, use the direct form 2 (DF2) delay line. The difference between the direct form 1 (DF1) and DF2 representations of the delay line is that DF1 contains *delayed* values of the source and destination vectors, while DF2 is two times shorter and contains pre-calculated values based on the following code [Opp75]:

```
for( i = 0; i < order; i++ ){
    pDly[i] = 0;
    for( n = order - i; n > 0; n-- ){
        pDly[i] += pTaps[n+i] * pSrc[len-n]; /* b- coefficients */
    }
}
for( i = 0; i < order; i++ ){
    for( n = order - i; n > 0; n-- ){
        pDly[i] -= pTaps[order+n+i] * pDst[len-n]; /* a- coefficients */
    }
}
```

There is no way to transform DF2 back to DF1. Therefore, if you need DF1 output, copy the corresponding last order values of the source vector and last order values of the destination vector to DF1 buffer. Please note that the IIRSetDlyLine and IIRGetDlyLine functions get/return the delay line values also in DF2 form.

To initialize and use an IIR filter, follow this general scheme:

1. Call ippsIIRInit to initialize the filter as an arbitrary order IIR filter in the external buffer, or ippsIIRInit_BiQuad to initialize the filter as a cascade of biquads in the external buffer. Size of the buffer can be computed by calling the functions ippsIIRGetStateSize or ippsIIRGetStateSize_BiQuad, respectively.
2. Call ippsIIR to filter consecutive samples at once.
3. Call ippsIIRGetDlyLine and ippsIIRSetDlyLine to get and set the delay line values in the IIR state structure.

- IIRInit
- IIRInit_BiQuad
- IIRGetStateSize
- IIRGetStateSize_BiQuad
- IIRGetDlyLine
- IIRSetDlyLine
- IIR
- IIRSparseInit
- IIRSparseGetStateSize
- IIRSparse
- IIRGenGetBufferSize
- IIRGenLowpass, IIRGenHighpass

IIRInit

Initializes an arbitrary IIR filter state.

Syntax

Case 1: Operation on integer samples

```
IppStatus ippsIIRInit32f_16s(IppsIIRState32f_16s** ppState, const Ipp32f* pTaps, int order, const Ipp32f* pDlyLine, Ipp8u* pBuf);

IppStatus ippsIIRInit64f_16s(IppsIIRState64f_16s** ppState, const Ipp64f* pTaps, int order, const Ipp64f* pDlyLine, Ipp8u* pBuf);

IppStatus ippsIIRInit64f_32s(IppsIIRState64f_32s** ppState, const Ipp64f* pTaps, int order, const Ipp64f* pDlyLine, Ipp8u* pBuf);

IppStatus ippsIIRInit32fc_16sc(IppsIIRState32fc_16sc** ppState, const Ipp32fc* pTaps, int order, const Ipp32fc* pDlyLine, Ipp8u* pBuf);

IppStatus ippsIIRInit64fc_16sc(IppsIIRState64fc_16sc** ppState, const Ipp64fc* pTaps, int order, const Ipp64fc* pDlyLine, Ipp8u* pBuf);

IppStatus ippsIIRInit64fc_32sc(IppsIIRState64fc_32sc** ppState, const Ipp64fc* pTaps, int order, const Ipp64fc* pDlyLine, Ipp8u* pBuf);
```

Case 2: Operation on floating point samples

```
IppStatus ippsIIRInit_32f(IppsIIRState_32f** ppState, const Ipp32f* pTaps, int order, const Ipp32f* pDlyLine, Ipp8u* pBuf);

IppStatus ippsIIRInit64f_32f(IppsIIRState64f_32f** ppState, const Ipp64f* pTaps, int order, const Ipp64f* pDlyLine, Ipp8u* pBuf);

IppStatus ippsIIRInit_64f(IppsIIRState_64f** ppState, const Ipp64f* pTaps, int order, const Ipp64f* pDlyLine, Ipp8u* pBuf);

IppStatus ippsIIRInit_32fc(IppsIIRState_32fc** ppState, const Ipp32fc* pTaps, int order, const Ipp32fc* pDlyLine, Ipp8u* pBuf);

IppStatus ippsIIRInit64fc_32fc(IppsIIRState64fc_32fc** ppState, const Ipp64fc* pTaps, int order, const Ipp64fc* pDlyLine, Ipp8u* pBuf);

IppStatus ippsIIRInit_64fc(IppsIIRState_64fc** ppState, const Ipp64fc* pTaps, int order, const Ipp64fc* pDlyLine, Ipp8u* pBuf);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pTaps`	Pointer to the array containing the taps. The number of elements in the array is `2*(order+1)`.

order	Order of the IIR filter.
pDlyLine	Pointer to the array containing the delay line values. The number of elements in the array is order.
ppState	Pointer to the pointer to the arbitrary IIR state structure to be created.
pBuf	Pointer to the external buffer.

Description

This function initializes an arbitrary IIR filter state in the external buffer. The size of this buffer must be computed previously by calling the function IIRGetStateSize. The initialization functions copy the taps from the array pTaps into the state structure pState. The order - length array pDlyLine specifies the delay line values. If the pointer to the array pDlyLine is not NULL, the array content is copied into the context structure, otherwise the delay values of the state structure are set to 0.

The filter order is defined by the order value which is equal to 0 for zero-order filters. The 2*(order+ 1) - length array pTaps specifies the taps arranged in the array as follows:

$B_0, B_1, \ldots, B_{order}, A_0, A_1, \ldots, A_{order}$

$A_0 \neq 0$

If the state is not created, the initialization function returns an error status.

The initialization functions with the 32s_32f suffixes called with floating-point taps automatically convert the taps into integer data type.

In all cases the data is converted into integer type with scaling for better precision.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsDivByZeroErr	Indicates an error when A_0 is equal to 0.
ippStsIIROrderErr	Indicates an error when order is less than or equal to 0.

IIRInit_BiQuad

Initializes an IIR filter state.

Syntax

Case 1: Operation on integer samples

```
IppStatus ippsIIRInit32f_BiQuad_16s(IppsIIRState32f_16s** ppState, const Ipp32f* pTaps,
int numBq, const Ipp32f* pDlyLine, Ipp8u* pBuf);
```

```
IppStatus ippsIIRInit64f_BiQuad_16s(IppsIIRState64f_16s** ppState, const Ipp64f* pTaps,
int numBq, const Ipp64f* pDlyLine, Ipp8u* pBuf);
```

```
IppStatus ippsIIRInit64f_BiQuad_32s(IppsIIRState64f_32s** ppState, const Ipp64f* pTaps,
int numBq, const Ipp64f* pDlyLine, Ipp8u* pBuf);

IppStatus ippsIIRInit32fc_BiQuad_16sc(IppsIIRState32fc_16sc** ppState, const Ipp32fc*
pTaps, int numBq, const Ipp32fc* pDlyLine, Ipp8u* pBuf);

IppStatus ippsIIRInit64fc_BiQuad_16sc(IppsIIRState64fc_16sc** ppState, const Ipp64fc*
pTaps, int numBq, const Ipp64fc* pDlyLine, Ipp8u* pBuf);

IppStatus ippsIIRInit64fc_BiQuad_32sc(IppsIIRState64fc_32sc** ppState, const Ipp64fc*
pTaps, int numBq, const Ipp64fc* pDlyLine, Ipp8u* pBuf);

IppStatus ippsIIRInit64f_BiQuad_DF1_32s(IppsIIRState64f_32s** ppState, const Ipp64f*
pTaps, int numBq, const Ipp32s* pDlyLine, Ipp8u* pBuf);
```

Case 2: Operation on floating point samples

```
IppStatus ippsIIRInit_BiQuad_32f(IppsIIRState_32f** ppState, const Ipp32f* pTaps, int
numBq, const Ipp32f* pDlyLine, Ipp8u* pBuf);

IppStatus ippsIIRInit64f_BiQuad_32f(IppsIIRState64f_32f** ppState, const Ipp64f* pTaps,
int numBq, const Ipp64f* pDlyLine, Ipp8u* pBuf);

IppStatus ippsIIRInit_BiQuad_64f(IppsIIRState_64f** ppState, const Ipp64f* pTaps, int
numBq, const Ipp64f* pDlyLine, Ipp8u* pBuf);

IppStatus ippsIIRInit_BiQuad_32fc(IppsIIRState_32fc** ppState, const Ipp32fc* pTaps,
int numBq, const Ipp32fc* pDlyLine, Ipp8u* pBuf);

IppStatus ippsIIRInit64fc_BiQuad_32fc(IppsIIRState64fc_32fc** ppState, const Ipp64fc*
pTaps, int numBq, const Ipp64fc* pDlyLine, Ipp8u* pBuf);

IppStatus ippsIIRInit_BiQuad_64fc(IppsIIRState_64fc** ppState, const Ipp64fc* pTaps,
int numBq, const Ipp64fc* pDlyLine, Ipp8u* pBuf);

IppStatus ippsIIRInit_BiQuad_DF1_32f(IppsIIRState_32f** ppState, const Ipp32f* pTaps,
int numBq, const Ipp32f* pDlyLine, Ipp8u* pBuf);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pTaps`	Pointer to the array containing the taps. The number of elements in the array is 6* `numBq`.
`numBq`	Number of cascades of biquads.
`pDlyLine`	Pointer to the array containing the delay line values. The number of elements in the array is 2* `numBq`.
`ppState`	Pointer to the pointer to the biquad IIR state structure.

pBuf	Pointer to the external buffer.
ppState	Pointer to the pointer to the arbitrary IIR state structure to be created.

Description

This function initializes a biquad (BQ) IIR filter state in the external buffer. The size of this buffer must be computed previously by calling the corresponding function ippsIIRGetStateSize_BiQuad. The initialization function copies the taps from the array pTaps into the state structure ppState. The array pDlyLine specifies the delay line values. The number of elements in the array pDlyLine is 4* numBq for the function flavor ippsIIRInit_BiQuad_DF1, and 2* numBq for all other flavors.

If the pointer to the array pDlyLine is not NULL, the array content is copied into the context structure, otherwise the delay values of the state structure are set to 0.

The function flavor ippsIIRInit_BiQuad_DF1 operates with the delay line values that are arranged in the array as follows:

$x_{0,-2}, x_{0,-1}, y_{0,-2}, y_{0,-1}, x_{1,-2}, x_{1,-1}, y_{1,-2}, y_{1,-1}, \dots x_{numBq-1,-2}, x_{numBq-1,-1}, y_{numBq-1,-2}, y_{numBq-1,-1}.$

A biquad IIR filter is defined by a cascade of biquads. The number of cascades of biquads is specified by the numBq value. The 6 *numBq – length array pTaps specifies the taps arranged in the array as follows:

$B_{0,0}, B_{0,1}, B_{0,2}, A_{0,0}, A_{0,1}, A_{0,2}; B_{1,0}, B_{1,1}, B_{1,2}, A_{1,0}, A_{1,1}, A_{1,2}; \dots A_{numBq-1,2}$

$A_{n,0} \neq 0, B_{n,0} \neq 0$

If the state is not created, the initialization function returns an error status.

The initialization functions with the 32s_32f suffixes called with floating-point taps automatically convert the taps into integer data type.

In all cases the data is converted into integer type with scaling for better precision.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsDivByZeroErr	Indicates an error when A_0, $A_{n,0}$ or $B_{n,0}$ is equal to 0.
ippStsIIROrderErr	Indicates an error when numBq is less than or equal to 0.

IIRGetStateSize

Computes the length of the external buffer for the arbitrary IIR filter state structure.

Syntax

```
IppStatus ippsIIRGetStateSize32f_16s(int order, int* pBufferSize);
IppStatus ippsIIRGetStateSize64f_16s(int order, int* pBufferSize);
IppStatus ippsIIRGetStateSize64f_32s(int order, int* pBufferSize);
IppStatus ippsIIRGetStateSize32fc_16sc(int order, int* pBufferSize);
```

```
IppStatus ippsIIRGetStateSize64fc_16sc(int order, int* pBufferSize);
IppStatus ippsIIRGetStateSize64fc_32sc(int order, int* pBufferSize);
IppStatus ippsIIRGetStateSize_32f(int order, int* pBufferSize);
IppStatus ippsIIRGetStateSize64f_32f(int order, int* pBufferSize);
IppStatus ippsIIRGetStateSize_64f(int order, int* pBufferSize);
IppStatus ippsIIRGetStateSize_32fc(int order, int* pBufferSize);
IppStatus ippsIIRGetStateSize64fc_32fc(int order, int* pBufferSize);
IppStatus ippsIIRGetStateSize_64fc(int order, int* pBufferSize);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`order`	Order of the IIR filter.
`pBufferSize`	Pointer to the computed buffer size value.

Description

This function computes the size of the external buffer for an arbitrary IIR filter state, and stores the result in `pBufferSize`.

To compute a size of the buffer, the filter order parameter `order` must be specified.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when `pBufferSize` pointer is `NULL`.
`ippStsIIROrderErr`	Indicates an error when `order` is less than or equal to 0.

IIRGetStateSize_BiQuad

Computes the length of the external buffer for the biquad IIR filter state structure.

Syntax

```
IppStatus ippsIIRGetStateSize32f_BiQuad_16s(int numBq, int* pBufferSize);
IppStatus ippsIIRGetStateSize64f_BiQuad_16s(int numBq, int* pBufferSize);
IppStatus ippsIIRGetStateSize64f_BiQuad_32s(int numBq, int* pBufferSize);
```

```
IppStatus ippsIIRGetStateSize32fc_BiQuad_16sc(int numBq, int* pBufferSize);
IppStatus ippsIIRGetStateSize64fc_BiQuad_16sc(int numBq, int* pBufferSize);
IppStatus ippsIIRGetStateSize64fc_BiQuad_32sc(int numBq, int* pBufferSize);
IppStatus ippsIIRGetStateSize_BiQuad_32f(int numBq, int* pBufferSize);
IppStatus ippsIIRGetStateSize64f_BiQuad_32f(int numBq, int* pBufferSize);
IppStatus ippsIIRGetStateSize_BiQuad_64f(int numBq, int* pBufferSize);
IppStatus ippsIIRGetStateSize_BiQuad_32fc(int numBq, int* pBufferSize);
IppStatus ippsIIRGetStateSize64fc_BiQuad_32fc(int numBq, int* pBufferSize);
IppStatus ippsIIRGetStateSize_BiQuad_64fc(int numBq, int* pBufferSize);
IppStatus ippsIIRGetStateSize64f_BiQuad_DF1_32s(int numBq, int* pBufferSize);
IppStatus ippsIIRGetStateSize_BiQuad_DF1_32f(int numBq, int* pBufferSize);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`numBq`	Number of cascades of biquads.
`pBufferSize`	Pointer to the computed buffer size value.

Description

This function computes the size of the external buffer for a corresponding biquad IIR filter state, and stores the result in `pBufferSize`.

To compute a size of the buffer, the number of cascades of biquads `numBq` must be specified.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when `pBufferSize` pointer is `NULL`.
`ippStsIIROrderErr`	Indicates an error when `numBq` is less than or equal to 0.

IIRGetDlyLine

Retrieves the delay line contents from the IIR filter state.

Syntax

```
IppStatus ippsIIRGetDlyLine32f_16s(const IppsIIRState32f_16s* pState, Ipp32f*
pDlyLine);

IppStatus ippsIIRGetDlyLine64f_16s(const IppsIIRState64f_16s* pState, Ipp64f*
pDlyLine);

IppStatus ippsIIRGetDlyLine64f_32s(const IppsIIRState64f_32s* pState, Ipp64f*
pDlyLine);

IppStatus ippsIIRGetDlyLine32fc_16sc(const IppsIIRState32fc_16sc* pState, Ipp32fc*
pDlyLine);

IppStatus ippsIIRGetDlyLine64fc_16sc(const IppsIIRState64fc_16sc* pState, Ipp64fc*
pDlyLine);

IppStatus ippsIIRGetDlyLine64fc_32sc(const IppsIIRState64fc_32sc* pState, Ipp64fc*
pDlyLine);

IppStatus ippsIIRGetDlyLine_32f(const IppsIIRState_32f* pState, Ipp32f* pDlyLine);

IppStatus ippsIIRGetDlyLine64f_32f(const IppsIIRState64f_32f* pState, Ipp64f*
pDlyLine);

IppStatus ippsIIRGetDlyLine_64f(const IppsIIRState_64f* pState, Ipp64f* pDlyLine);

IppStatus ippsIIRGetDlyLine_32fc(const IppsIIRState_32fc* pState, Ipp32fc* pDlyLine);

IppStatus ippsIIRGetDlyLine64fc_32fc(const IppsIIRState64fc_32fc* pState, Ipp64fc*
pDlyLine);

IppStatus ippsIIRGetDlyLine_64fc(const IppsIIRState_64fc* pState, Ipp64fc* pDlyLine);

IppStatus ippsIIRGetDlyLine64f_DF1_32s(const IppsIIRState64f_32s* pState, Ipp32s*
pDlyLine);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pState`	Pointer to the IIR filter state structure.
`pDlyLine`	Pointer to the array containing the delay line values. The number of elements in the array is `order` for arbitrary filters and 2* `numBq` for BQ filters.

Description

This function copies the delay line values from the corresponding state structure `pState` and stores them into the `pDlyLine` array. If the pointer is `NULL`, then the delay line values in the state structure are initialized to zero.

The corresponding filter state must be initialized beforehand by one of the initialization functions.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when the pState pointer is NULL.
ippStsContextMatchErr	Indicates an error when the state identifier is incorrect.

IIRSetDlyLine

Sets the delay line contents in an IIR filter state.

Syntax

```
IppStatus ippsIIRSetDlyLine32f_16s(IppsIIRState32f_16s* pState, const Ipp32f*
pDlyLine);
IppStatus ippsIIRSetDlyLine64f_16s(IppsIIRState64f_16s* pState, const Ipp64f*
pDlyLine);
IppStatus ippsIIRSetDlyLine64f_32s(IppsIIRState64f_32s* pState, const Ipp64f*
pDlyLine);
IppStatus ippsIIRSetDlyLine32fc_16sc(IppsIIRState32fc_16sc* pState, const Ipp32fc*
pDlyLine);
IppStatus ippsIIRSetDlyLine64fc_16sc(IppsIIRState64fc_16sc* pState, const Ipp64fc*
pDlyLine);
IppStatus ippsIIRSetDlyLine64fc_32sc(IppsIIRState64fc_32sc* pState, const Ipp64fc*
pDlyLine);
IppStatus ippsIIRSetDlyLine_32f(IppsIIRState_32f* pState, const Ipp32f* pDlyLine);
IppStatus ippsIIRSetDlyLine64f_32f(IppsIIRState64f_32f* pState, const Ipp64f*
pDlyLine);
IppStatus ippsIIRSetDlyLine_64f(IppsIIRState_64f* pState, const Ipp64f* pDlyLine);
IppStatus ippsIIRSetDlyLine_32fc(IppsIIRState_32fc* pState, const Ipp32fc* pDlyLine);
IppStatus ippsIIRSetDlyLine64fc_32fc(IppsIIRState64fc_32fc* pState, const Ipp64fc*
pDlyLine);
IppStatus ippsIIRSetDlyLine_64fc(IppsIIRState_64fc* pState, const Ipp64fc* pDlyLine);
IppStatus ippsIIRSetDlyLine64f_DF1_32s(IppsIIRState64f_32s* pState, const Ipp32s*
pDlyLine);
```

Include Files

ipps.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

pState	Pointer to the IIR filter state structure.
pDlyLine	Pointer to the array holding the delay line values. The number of elements in the array is order for arbitrary filters and 2* numBq for BQ filters. If the pointer is NULL, then the delay line values in the state structure are initialized to zero.

Description

This function copies the delay line values from pDlyLine and stores them into the state structure pState. If the pointer is NULL, then the delay line values in the state structure are initialized to zero.

The filter state must be initialized beforehand by one of the initialization functions.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when the pState pointer is NULL.
ippStsContextMatchErr	Indicates an error when the state identifier is incorrect.

IIR

Filters a source vector through an IIR filter.

Syntax

Case 1: Not-in-place operation on integer samples

```
IppStatus ippsIIR32f_16s_Sfs(const Ipp16s* pSrc, Ipp16s* pDst, int len,
IppsIIRState32f_16s* pState, int scaleFactor);
```

```
IppStatus ippsIIR64f_16s_Sfs(const Ipp16s* pSrc, Ipp16s* pDst, int len,
IppsIIRState64f_16s* pState, int scaleFactor);
```

```
IppStatus ippsIIR64f_32s_Sfs(const Ipp32s* pSrc, Ipp32s* pDst, int len,
IppsIIRState64f_32s* pState, int scaleFactor);
```

```
IppStatus ippsIIR32fc_16sc_Sfs(const Ipp16sc* pSrc, Ipp16sc* pDst, int len,
IppsIIRState32fc_16sc* pState, int scaleFactor);
```

```
IppStatus ippsIIR64fc_16sc_Sfs(const Ipp16sc* pSrc, Ipp16sc* pDst, int len,
IppsIIRState64fc_16sc* pState, int scaleFactor);
```

```
IppStatus ippsIIR64fc_32sc_Sfs(const Ipp32sc* pSrc, Ipp32sc* pDst, int len,
IppsIIRState64fc_32sc* pState, int scaleFactor);
```

Case 2: Not-in-place operation on floating point samples

```
IppStatus ippsIIR_32f(const Ipp32f* pSrc, Ipp32f* pDst, int len, IppsIIRState_32f*
pState);
```

```
IppStatus ippsIIR_64f(const Ipp64f* pSrc, Ipp64f* pDst, int len, IppsIIRState_64f*
pState);

IppStatus ippsIIR64f_32f(const Ipp32f* pSrc, Ipp32f* pDst, int len,
IppsIIRState64f_32f* pState);

IppStatus ippsIIR_32fc(const Ipp32fc* pSrc, Ipp32fc* pDst, int len, IppsIIRState_32fc*
pState);

IppStatus ippsIIR_64fc(const Ipp64fc* pSrc, Ipp64fc* pDst, int len, IppsIIRState_64fc*
pState);

IppStatus ippsIIR64fc_32fc(const Ipp32fc* pSrc, Ipp32fc* pDst, int len,
IppsIIRState64fc_32fc* pState);
```

Case 3: In-place operation on integer samples

```
IppStatus ippsIIR32f_16s_ISfs(Ipp16s* pSrcDst, int len, IppsIIRState32f_16s* pState,
int scaleFactor);

IppStatus ippsIIR32fc_16sc_ISfs(Ipp16sc* pSrcDst, int len, IppsIIRState32fc_16sc*
pState, int scaleFactor);

IppStatus ippsIIR64f_16s_ISfs(Ipp16s* pSrcDst, int len, IppsIIRState64f_16s* pState,
int scaleFactor);

IppStatus ippsIIR64f_32s_ISfs(Ipp32s* pSrcDst, int len, IppsIIRState64f_32s* pState,
int scaleFactor);

IppStatus ippsIIR64fc_16sc_ISfs(Ipp16sc* pSrcDst, int len, IppsIIRState64fc_16sc*
pState, int scaleFactor);

IppStatus ippsIIR64fc_32sc_ISfs(Ipp32sc* pSrcDst, int len, IppsIIRState64fc_32sc*
pState, int scaleFactor);
```

Case 4: In-place operation on floating point samples

```
IppStatus ippsIIR_32f_I(Ipp32f* pSrcDst, int len, IppsIIRState_32f* pState);

IppStatus ippsIIR_64f_I(Ipp64f* pSrcDst, int len, IppsIIRState_64f* pState);

IppStatus ippsIIR64f_32f_I(Ipp32f* pSrcDst, int len, IppsIIRState64f_32f* pState);

IppStatus ippsIIR_32fc_I(Ipp32fc* pSrcDst, int len, IppsIIRState_32fc* pState);

IppStatus ippsIIR_64fc_I(Ipp64fc* pSrcDst, int len, IppsIIRState_64fc* pState);

IppStatus ippsIIR64fc_32fc_I(Ipp32fc* pSrcDst, int len, IppsIIRState64fc_32fc* pState);
```

Case 4: Operation with specified number of vector

```
IppStatus ippsIIR_32f_P(const Ipp32f** ppSrc, Ipp32f** ppDst, int len, int nChannels,
IppsIIRState_32f** ppState);

IppStatus ippsIIR64f_32s_PSfs(const Ipp32s** ppSrc, Ipp32s** ppDst, int len, int
nChannels, IppsIIRState64f_32s** ppState, int* pScaleFactor);

IppStatus ippsIIR_32f_IP(Ipp32f** ppSrcDst, int len, int nChannels, IppsIIRState_32f**
ppState);

IppStatus ippsIIR64f_32s_IPSfs(Ipp32s** ppSrcDst, int len, int nChannels,
IppsIIRState64f_32s** ppState, int* pScaleFactor);
```

Include Files

ipps.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

pState	Pointer to the IIR filter state structure.
ppState	Pointer to the array of the pointers to the IIR filter state structures.
pSrc	Pointer to the source vector.
ppSrc	Pointer to the array of pointers to the source vectors.
pDst	Pointer to the destination vector.
ppDst	Pointer to the array of pointers to the destination vector.
pSrcDst	Pointer to the source and destination vector for the in-place operations.
ppSrcDst	Pointer to the array of pointers to the source and destination vectors for the in-place operations.
len	Number of elements of the vector to be filtered.
nChannels	Number of vectors to be filtered.
scaleFactor	Scale factor, refer to Integer Scaling.
pScaleFactor	Pointer to the scale factor.

Description

This function filters len elements of the source vector pSrc or pSrcDst through an IIR filter, and stores the results in pDst or pSrcDst, respectively. The filter parameters are specified in pState.The output of the integer sample is scaled according to scaleFactor and can be saturated.

Do not modify the scaleFactor value unless the state structure is changed.

The filter state must be initialized before calling the function ippsIIR. Specify the number of taps tapsLen, the tap values in pTaps, the delay line values in pDlyLine, and the order or numBq value beforehand.

Function flavors described in the **Case 4** filter simultaneously the nChannels source vectors. Each vector must have the len elements and is filtered with its own state structure. These state structures must be initialized beforehand.

Example demonstrates how to use the function ippsIIR to filter a sample. The function ippsConvert_64f32s_Sfs converts floating-point taps into integer data type before calling ippsIIRInitAlloc_32s.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when len is less or equal to 0.
ippStsChannelErr	Indicates an error when nChannels is less or equal to 0.
ippStsContextMatchErr	Indicates an error when the state identifier is incorrect.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

#include <math.h>
#include <stdio.h>
#include "ipp.h"

#undef NUMITERS
#define NUMITERS 150

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define
check_sts(st)
                                  \
    if ((st) !=
ippStsNoErr)
                          \
        goto exitLine; /* Go to Exit if Intel(R) Integrated Performance Primitives (Intel(R)
IPP) function returned status different from \
                           ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main()
{
    int n = 0, i = 0, specSize = 0, bufferSize = 0;
```

```
    IppStatus status;
    IppsIIRState_32f *pIIRState = NULL; /* Pointer to Intel(R) IPP state structure */
    IppsFIRSpec_32f *pFIRSpec = NULL;   /* Pointer to FIR Spec structure */
    Ipp32f *x = ippsMalloc_32f(NUMITERS + 10), *y = ippsMalloc_32f(NUMITERS), *z =
ippsMalloc_32f(NUMITERS);
    const float taps[] = {0.0051f, 0.0180f, 0.0591f, 0.1245f, 0.1869f, 0.2127f, 0.1869f,
0.1245f, 0.0591f, 0.0180f, 0.0051f,
                          1,       0,       0,       0,       0,       0,       0,
0,       0,       0,       0};
    Ipp8u *pBuf = NULL; /* Pointer to the work buffers */

    for (n = 0; n < NUMITERS; ++n)
        x[n] = (float)sin(IPP_2PI * n * 0.2);

    check_sts(status = ippsIIRGetStateSize_32f(10, &bufferSize));

    pBuf = ippsMalloc_8u(bufferSize);

    check_sts(status = ippsIIRInit_32f(&pIIRState, taps, 10, NULL, pBuf));

    check_sts(status = ippsIIR_32f(x, y, NUMITERS, pIIRState));

    ippsFree(pBuf);
    pBuf = NULL;

    for (i = 120; i < 120 + 5; i++)
        printf("%f,", y[i]);
    printf("\n");

    check_sts(status = ippsFIRSRGetSize(11, ipp32f, &specSize, &bufferSize));
    pFIRSpec = (IppsFIRSpec_32f *)ippsMalloc_8u(specSize);
    pBuf = ippsMalloc_8u(bufferSize);

    check_sts(status = ippsFIRSRInit_32f(taps, 11, ippAlgDirect, pFIRSpec));
    check_sts(status = ippsFIRSR_32f(x, z, NUMITERS, pFIRSpec, NULL, NULL, pBuf));

    for (i = 120; i < 120 + 5; i++)
        printf("%f,", z[i]);

    printf("\n");
    EXIT_MAIN
    ippsFree(pBuf);
    ippsFree(z);
    ippsFree(y);
    ippsFree(x);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

IIRSparseInit

Initializes a sparse IIR filter structure.

Syntax

```
IppStatus ippsIIRSparseInit_32f(IppsIIRSparseState_32f** ppState, const Ipp32f*
pNZTaps, const Ipp32s* pNZTapPos, int nzTapsLen1, int nzTapsLen2, const Ipp32f*
pDlyLine, Ipp8u* pBuf);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pNZTaps`	Pointer to the array containing the non-zero tap values.
`pNZTapPos`	Pointer to the array containing positions of the non-zero tap values. The number of elements in the array is `nzTapsLen`.
`nzTapsLen1`, `nzTapsLen2`	Pointer to the destination vector.
`pDlyLine`	Pointer to the array containing the delay line values.
`ppState`	Double pointer to the sparse IIR state structure.
`pBuf`	Pointer to the external buffer for the sparse IIR state structure.

Description

This function initializes a sparse IIR filter state structure `ppState` in the external buffer `pBuf`. The size of this buffer must be computed previously by calling the function ippsIIRSparseGetStateSize.

The `(nzTapsLen1+nzTapsLen2) - length` array `pNZTaps` specifies the non-zero taps arranged in the array as follows:

B_0, B_1, …, $B_{nzTapsLen1-1}$, A_0, A_1, …, $A_{nzTapsLen2-1}$.

The `(nzTapsLen1+nzTapsLen2) - length` array `pNZTapPos` specifies the non-zero tap positions arranged in the array as follows:

BP_0, BP_1, …, $BP_{nzTapsLen1-1}$, AP_0, AP_1, …, $AP_{nzTapsLen2-1}$, $AP_0 \neq 0$

The initialization function copies the values of filter coefficients from the array `pNZTaps` containing non-zero taps and their positions from the array `pNZTapPos` into the state structure `ppState`. The array `pDlyLine` contains the delay line values.The number of elements in this array is `pNZTapPos[nzTapsLen1-1] +pNZTapPos[nzTapsLen1+nzTapsLen2- 1]`. If the pointer to the array `pDlyLine` is not `NULL`, the array contents is copied into the state structure `ppState`, otherwise the delay line values in the state structure are initialized to 0.

NOTE The values of `nzTapsLen1`, `nzTapsLen2`, `pNZTapPos[nzTapsLen -1]`, and `pNZTapPos[nzTapsLen1+nzTapsLen2- 1]` must be equal to those specified in the function ippsIIRSparseGetStateSize.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when one of the specified pointers is `NULL`.
`ippStsIIROrderErr`	Indicates an error if `nzTapsLen1` is less than or equal to 0, or `nzTapsLen2` is less than 0.
`ippStsSparseErr`	Indicates an error if positions of the non-zero taps are not in ascending order, or are negative or repetitive; or `pNZTapPos[nzTapsLen1]` is equal to 0.

IIRSparseGetStateSize

Computes the size of the external buffer for the sparse IIR filter structure.

Syntax

```
IppStatus ippsIIRSparseGetStateSize_32f(int nzTapsLen1, int nzTapsLen2, int order1, int order2, int* pStateSize);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`nzTapsLen1`, `nzTapsLen2`	Number of elements in the array containing the non-zero tap values.
`order1`, `order2`	Order of the sparse IIR filter.
`pStateSize`	Pointer to the computed value of the external buffer.

Description

This function computes the size in bytes of the external buffer for a sparse IIR filter state that is required for the function ippsIIRSparseInit. The computations are based on the specified number of non-zero filter coefficients `nzTapsLen1`, `nzTapsLen2` and filter orders `order1`, `order2`. `order1=pNZTapPos[nzTapsLen1-1]`, `order2=pNZTapPos[nzTapsLen1+nzTapsLen2- 1]` (see description of the function ippsIIRSparseInit for more details). The result value is stored in the `pStateSize`.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if pStateSize pointer is NULL.
ippStsIIROrderErr	Indicates an error if nzTapsLen1 is less than or equal to 0, or nzTapsLen2 is less than 0.
ippStsSparseErr	Indicates an error if order1 or order2 is less than 0.

IIRSparse

Filters a source vector through a sparse IIR filter.

Syntax

```
IppStatus ippsIIRSparse_32f(const Ipp32f* pSrc, Ipp32f* pDst, int len,
IppsIIRSparseState_32f* pState);
```

Include Files

ipps.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

pState	Pointer to the sparse IIR filter state structure.
pSrc	Pointer to the source vector.
pDst	Pointer to the destination vector.
len	Number of elements that will be filtered.

Description

This function applies the sparse IIR filter to the len elements of the source vector pSrc, and stores the results in pDst. The filter parameters - the number of non-zero taps nzTapsLen1, nzTapsLen2, their values pNZTaps and their positions pNZTapPos, and the delay line values pDlyLine – are specified in the sparse IIR filter structure pState that should be previously initialized the function ippsIIRSparseInit.

In the following definition of the sparse IIR filter, the sample to be filtered is denoted x(n), the non-zero taps are denoted B_i and A_i, their positions are denoted BP_i and AP_i.

The non-zero taps are arranged in the array as follows:

$B_0, B_1, \ldots, B_{nzTapsLen1 - 1}, A_0, A_1, \ldots, A_{nzTapsLen2 - 1}$.`

The non-zero tap positions are arranged in the array as follows:

$BP_0, BP_1, \ldots, BP_{nzTapsLen1 - 1}, AP_0, AP_1, \ldots, AP_{nzTapsLen2 - 1}$,` $AP_0 \neq 0$

The return value is y(n) is defined by the formula for a sparse IIR filter:

$$y(n) = \sum_{k=0}^{nzTapsLen1-1} B_k \cdot x(n - BP_k) + \sum_{k=1}^{nzTapsLen2-1} A_k \cdot y(n - AP_k) \quad 0 \le n < len$$

After the function has performed calculations, it updates the delay line values stored in the filter state structure.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error if `len` is less than or equal to 0.

Example

The example below shows how to use the sparse IIR filter functions.

```
int buflen;  Ipp8u *buf;  int nzTapsLen1 = 5; //number of non-zero taps in the FIR part of the
formula  int nzTapsLen2 = 3;   //number of non-zero taps in the IIR part of the formula  Ipp32f
nzTaps [] = {0.5, 0.4, 0.3, 0.2, 0.1, 0.8, 0.7,
0.6};                                          //non-zero taps values (FIR+IIR)  Ipp32s
nzTapsPos[] = {0, 10, 20, 30, 40, 1, 5, 15};                                          //non-
zero tap positions (FIR+IIR)  IppsIIRSparseState_32f* iirState;  Ipp32f *src,
*dst; /* ..................................... */
ippsIIRSparseGetStateSize_32f(nzTapsLen1, nzTapsLen2, nzTapsPos [nzTapsLen1 -
1],                                nzTapsPos [nzTapsLen1 + nzTapsLen2 - 1], &buflen);  buf =
ippsMalloc_8u(buflen);  ippsIIRSparseInit_32f(&iirState, nzTaps, nzTapsPos, nzTapsLen1,
nzTapsLen2,                        NULL, buf);  /* . . . . initializing src somehow . . . . */
ippsIIRSparse_32f(src, dst, len, iirState); /* dst[i] = src[i] * 0.5 + src[i-10] * 0.4 +
src[i-20] * 0.3 +  src[i-30] * 0.2            +  src[i-40] * 0.1 + dst[i-1] * 0.8 + dst[i-5] *
0.7 + dst[i-15] * 0.6  */ /* ..................................................... */
ippsFree(buf);
```

IIRGenGetBufferSize

Computes the size of the buffer required for `ippsIIRGenLowpass` and `ippsIIRGenHighpass` internal calculations.

Syntax

```
IppStatus ippsIIRGenGetBufferSize(int order, int* pBufferSize);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

order	Order of the filter [1, 12].
pBufferSize	Pointer to the calculated buffer size (in bytes).

Description

This function computes the size of the buffer that is required for **ippsIIRGenLowpass`/:ref:`ippsIIRGenHighpass <c_iirgenlowpass-iirgenhighpass>** internal calculations.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.
ippStsIIRGenOrderErr	Indicates an error when the order value is less than 1, or greater than 12.

See Also

IIRGenLowpass IIRGenHighpass Computes lowpass and highpass IIR filter coefficients.

IIRGenLowpass, IIRGenHighpass

Computes lowpass and highpass IIR filter coefficients.

Syntax

```
IppStatus ippsIIRGenLowpass_64f(Ipp64f rFreq, Ipp64f ripple, int order, Ipp64f* pTaps,
IppsIIRFilterType filterType, Ipp8u* pBuffer);
```

```
IppStatus ippsIIRGenHighpass_64f(Ipp64f rFreq, Ipp64f ripple, int order, Ipp64f* pTaps,
IppsIIRFilterType filterType, Ipp8u* pBuffer);
```

Include Files

ipps.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

rFreq	Cutoff frequency, should be in the range (0, 0.5).
ripple	Posible ripple in pass band for ippChebyshev1 type of filter.
order	Order of the filter [1, 12]

pTaps	Pointer to the array where computed tap values are stored.
filterType	Type of the IIR filter, possible values: ippButterworth, ippChebyshev1.
pBuffer	Pointer to the buffer for internal calculations. To get the size of the buffer, use the ippsIIRGenGetBufferSize function.

Description

These functions computes coefficients for lowpass or highpass IIR filters, respectively, with the cutoff frequency rFreq. The parameter filterType specifies the type of the filter. The computed coefficients are stored in the array pTaps. Its length must be at least 2*(order + 1) and the taps arranged in the array as follows:

$B_0, B_1, \ldots, B_{order}, A_0, A_1, \ldots, A_{order}$

Application Notes

Butterworth filters are characterized by a magnitude response that is at most flat in the passband and monotonic overall. Butterworth filters sacrifice rolloff steepness for monotonicity in the passband. Unless the smoothness of the Butterworth filter is needed, *Chebyshev1 filter* can generally provide steeper rolloff characteristics with a lower filter order. Chebyshev1 type filters are equiripple in the passband and monotonic in the stopband. For ippButterworth filter cutoff frequency is the frequency where the magnitude response of the filter is $2^{-1/2}$. For ippChebyshev1 filter cutoff frequency is the frequency at which the magnitude response of the filter is (-ripple) dB. For the functions ippsIIRGenLowpass and ippsIIRGenHighpass, the normalized cutoff frequency rFreq must be a number between 0 and 0.5, where 0.5 corresponds to the Nyquist frequency, π radians per sample. The correspondence between MATLAB's Wn and Intel IPP rFreq is very simple: Wn = 2*rFreq.

Examples:

1) For data sampled at 1000 Hz, create a 9th-order highpass Butterworth filter with cutoff frequency at 300 Hz.

Intel IPP:

```
Ipp64f pTaps[2*(9+1)];
```

```
status = ippsIIRGenHighpass( 300.0/1000.0, 0, 9, pTaps, ippButterworth );
```

MATLAB:

```
[b,a] = butter(9,300/500,'high');
```

2) For data sampled at 1000 Hz, create a 9th-order lowpass Chebyshev1 filter with ripple in the passband of 0.5 dB and a cutoff frequency at 300 Hz.

Intel IPP:

```
Ipp64f pTaps[2*(9+1)];
```

```
status = ippsIIRGenLowpass( 300.0/1000.0, 0.5, 9, pTaps, ippChebyshev1 );
```

MATLAB:

```
[b,a] = cheby1(9, 0.5, 300/500);
```

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when the pTaps pointer is NULL.
ippStsIIRGenOrderErr	Indicates an error when the order is less than 1 or greater than 12.
ippStsFilterFrequencyErr	Indicates an error when the rFreq is out of the range.

See Also

IIRGenGetBufferSize Computes the size of the buffer required for ippsIIRGenLowpass and ippsIIRGenHighpass internal calculations.

IIRIIR Filter Functions

The functions described in this section initialize an infinite impulse response (IIR) filter and perform a zero-phase digital filtering of input data in both forward and backward directions. The formulas below explain why the filtered signal has zero-phase distortion. Consider the following case in the frequency domain: if x(n) is the input sequence and h(n) is the IIR filter's impulse response, then the result of the forward filter pass is:

$$Y_1(e^{i\varphi}) = X(e^{i\varphi}) * H(e^{i\varphi})$$

where

- $X(e^{i\varphi})$ is the Fourier transform of x(n)
- $H(e^{i\varphi})$ is the Fourier transform of h(n)
- $Y_1(e^{i\varphi})$ is the Fourier transform of the forward filter pass

Backward filtering corresponds to filtering of time-reversed signal. Time reversal corresponds to replacing φ with -φ in the frequency domain, so the result of time reversal is:

$$Y_1(e^{-i\varphi}) = X(e^{-i\varphi}) * H(e^{-i\varphi})$$

When the filter is applied for the second time, the above formula is multiplied by the Fourier transform of the filter's impulse response function $H(e^{i\varphi})$:

$$Y_1(e^{-i\varphi}) = X(e^{-i\varphi}) * H(e^{-i\varphi}) * H(e^{i\varphi})$$

The final time reversal in the frequency domain results in:

$$Y_1(e^{-i\varphi}) = X(e^{i\varphi}) * H(e^{i\varphi}) * H(e^{-i\varphi}) = X(e^{i\varphi}) * |H(e^{i\varphi})|^2$$

You can see from the resulting equation that:

- The filtered signal has zero-phase distortion (as the filtering was done with $|H(e^{i\varphi})|^2$, which is purely real-valued)
- The filter transfer function has the squared magnitude of the original filter transfer function
- The filter order is double the order of the initialized IIR filter

To initialize and use an IIRIIR filter, follow this general scheme:

1. Call ippsIIRIIRInit to initialize the IIRIIR filter in the external buffer. To compute the size of the buffer, use the ippsIIRIIRGetStateSize function.
2. Call ippsIIRIIR to filter a vector.
3. Call ippsIIRIIRGetDlyLine and ippsIIRIIRSetDlyLine to get and set the delay line values in the IIRIIR state structure.

- IIRIIRGetStateSize
- IIRIIRInit
- IIRIIRGetDlyLine
- IIRIIRSetDlyLine
- IIRIIR

IIRIIRGetStateSize

Computes the length of the external buffer for the IIRIIR filter state structure.

Syntax

```
IppStatus ippsIIRIIRGetStateSize_32f(int order, int* pBufferSize);
IppStatus ippsIIRIIRGetStateSize64f_32f(int order, int* pBufferSize);
IppStatus ippsIIRIIRGetStateSize_64f(int order, int* pBufferSize);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`order`	Order of the IIRIIR filter.
`pBufferSize`	Pointer to the computed buffer size value.

Description

This function computes the size of the external buffer for the IIRIIR filter state structure, and stores the result in `pBufferSize`.

Use this function before using IIRIIRInit.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when `pBufferSize` pointer is `NULL`.
`ippStsIIROrderErr`	Indicates an error when `order` is less than or equal to zero.

See Also

IIRIIRInit Initializes the IIRIIR filter state structure.

IIRIIRInit

Initializes the IIRIIR filter state structure.

Syntax

```
IppStatus ippsIIRIIRInit_32f(IppsIIRState_32f** ppState, const Ipp32f* pTaps, int order, const Ipp32f* pDlyLine, Ipp8u* pBuf);
```

```
IppStatus ippsIIRIIRInit64f_32f(IppsIIRState64f_32f** ppState, const Ipp64f* pTaps, int order, const Ipp64f* pDlyLine, Ipp8u* pBuf);
```

```
IppStatus ippsIIRIIRInit_64f(IppsIIRState_64f** ppState, const Ipp64f* pTaps, int order, const Ipp64f* pDlyLine, Ipp8u* pBuf);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pTaps`	Pointer to the array containing the taps. The number of elements in the array is `2*(order+1)`.
`order`	Order of the IIRIIR filter.
`pDlyLine`	Pointer to the array containing the delay line values. The number of elements in the array is `order`. If `IIRIIRInit` is called with `pDlyLine==NULL`, then it automatically forms the delay line that minimizes the start-up and ending transients. The line is formed by matching the initial conditions to remove the DC offset at the beginning and the end of the input vector.
`ppState`	Pointer to the pointer to the arbitrary IIRIIR state structure to be created.
`pBuf`	Pointer to the external buffer.

Description

This function initializes the arbitrary IIRIIR filter state structure in the external buffer. Before using the `IIRIIRInit` function, compute the size of the external buffer by calling the IIRIIRGetStateSize function. The initialization functions copy the taps from the `pTaps` array into the `pState` structure . The `order – length` array `pDlyLine` specifies the delay line values. If the `pDlyLine` pointer to the array is not `NULL`, the array content is copied into the context structure, otherwise the delay values of the state structure are set to values that minimize the start-up and ending transients. These values are obtained by matching the initial conditions to remove the DC offset at the beginning and the end of the input vector.

The filter order is defined by the `order` value which is equal to 0 for zero-order filters. The `2*(order+ 1) – length` array `pTaps` specifies the taps arranged in the array as follows:

$B_0, B_1, \ldots, B_{order}, A_0, A_1, \ldots, A_{order}$

$A_0 \neq 0$

If the state structure is not created, the initialization function returns an error status.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when one of the specified pointers is `NULL`.
`ippStsDivByZeroErr`	Indicates an error when A_0 is equal to zero.
`ippStsIIROrderErr`	Indicates an error when `order` is less than or equal to zero.

See Also

IIRIIRGetStateSize Computes the length of the external buffer for the IIRIIR filter state structure.

IIRIIRGetDlyLine

Retrieves the delay line contents from the IIIRIR filter state structure.

Syntax

```
IppStatus ippsIIRIIRGetDlyLine_32f(const IppsIIRState_32f* pState, Ipp32f* pDlyLine);
IppStatus ippsIIRIIRGetDlyLine64f_32f(const IppsIIRState64f_32f* pState, Ipp64f* pDlyLine);
IppStatus ippsIIRIIRGetDlyLine_64f(const IppsIIRState_64f* pState, Ipp64f* pDlyLine);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pState`	Pointer to the IIRIIR filter state structure.
`pDlyLine`	Pointer to the array containing the delay line values. The number of elements in the array is `order`.

Description

This function copies the delay line values from the corresponding `pState` structure and stores them into the `pDlyLine` array.

The corresponding filter state structure must be initialized beforehand by the initialization function.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when the pState pointer is NULL.
ippStsContextMatchErr	Indicates an error when the state identifier is incorrect.

IIRIIRSetDlyLine

Sets the delay line contents in the IIRIIR filter state structure.

Syntax

```
IppStatus ippsIIRIIRSetDlyLine_32f(IppsIIRState_32f* pState, const Ipp32f* pDlyLine);
IppStatus ippsIIRIIRSetDlyLine64f_32f(IppsIIRState64f_32f* pState, const Ipp64f* pDlyLine);
IppStatus ippsIIRIIRSetDlyLine_64f(IppsIIRState_64f* pState, const Ipp64f* pDlyLine);
```

Include Files

ipps.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

pState	Pointer to the IIRIIR filter state structure.
pDlyLine	Pointer to the array holding the delay line values. The number of elements in the array is order. If the pointer is NULL, then the delay line values in the state structure are formed internally to minimize the start-up and ending transients. These values are obtained by matching the initial conditions to remove the DC offset at the beginning and the end of the input vector.

Description

This function copies the delay line values from pDlyLine and stores them into the pState structure . If the pointer is NULL, then the delay line values in the state structure are initialized to values that minimize the start-up and ending transients. These values are obtained by matching the initial conditions to remove the DC offset at the beginning and the end of the input vector.

The filter state must be initialized beforehand by the initialization function.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when the pState pointer is NULL.
ippStsContextMatchErr	Indicates an error when the state identifier is incorrect.

IIRIIR

Filters a source vector through an IIRIIR filter.

Syntax

Case 1: Not-in-place operation

```
IppStatus ippsIIRIIR_32f(const Ipp32f* pSrc, Ipp32f* pDst, int len, IppsIIRState_32f* pState);
IppStatus ippsIIRIIR_64f(const Ipp64f* pSrc, Ipp64f* pDst, int len, IppsIIRState_64f* pState);
IppStatus ippsIIRIIR64f_32f(const Ipp32f* pSrc, Ipp32f* pDst, int len, IppsIIRState64f_32f* pState);
```

Case 2: In-place operation

```
IppStatus ippsIIRIIR_32f_I(Ipp32f* pSrcDst, int len, IppsIIRState_32f* pState);
IppStatus ippsIIRIIR_64f_I(Ipp64f* pSrcDst, int len, IppsIIRState_64f* pState);
IppStatus ippsIIRIIR64f_32f_I(Ipp32f* pSrcDst, int len, IppsIIRState64f_32f* pState);
```

Include Files

ipps.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

pState	Pointer to the IIRIIR filter state structure.
pSrc	Pointer to the source vector.
pDst	Pointer to the destination vector.
pSrcDst	Pointer to the source and destination vector for the in-place operations.
len	Number of elements of the vector to be filtered.

Description

This function filters `len` elements of the source vector `pSrc` or `pSrcDst` through an IIRIIR filter, and stores the results in `pDst` or `pSrcDst`, respectively. The filter parameters are specified in `pState`.

Before calling the `ippsIIRIIR` function, initialize the filter state structure by using the IIRIIRInit function and specify the number of taps `tapsLen`, the tap values in `pTaps`, the delay line values in `pDlyLine`, and the `order` value.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when one of the specified pointers is `NULL`.
`ippStsLengthErr`	Indicates an error when `length of the vectors < 3*(IIR order).`
`ippStsContextMatchErr`	Indicates an error when the state identifier is incorrect.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

#include <stdio.h>
#include <ipp/ipps.h>

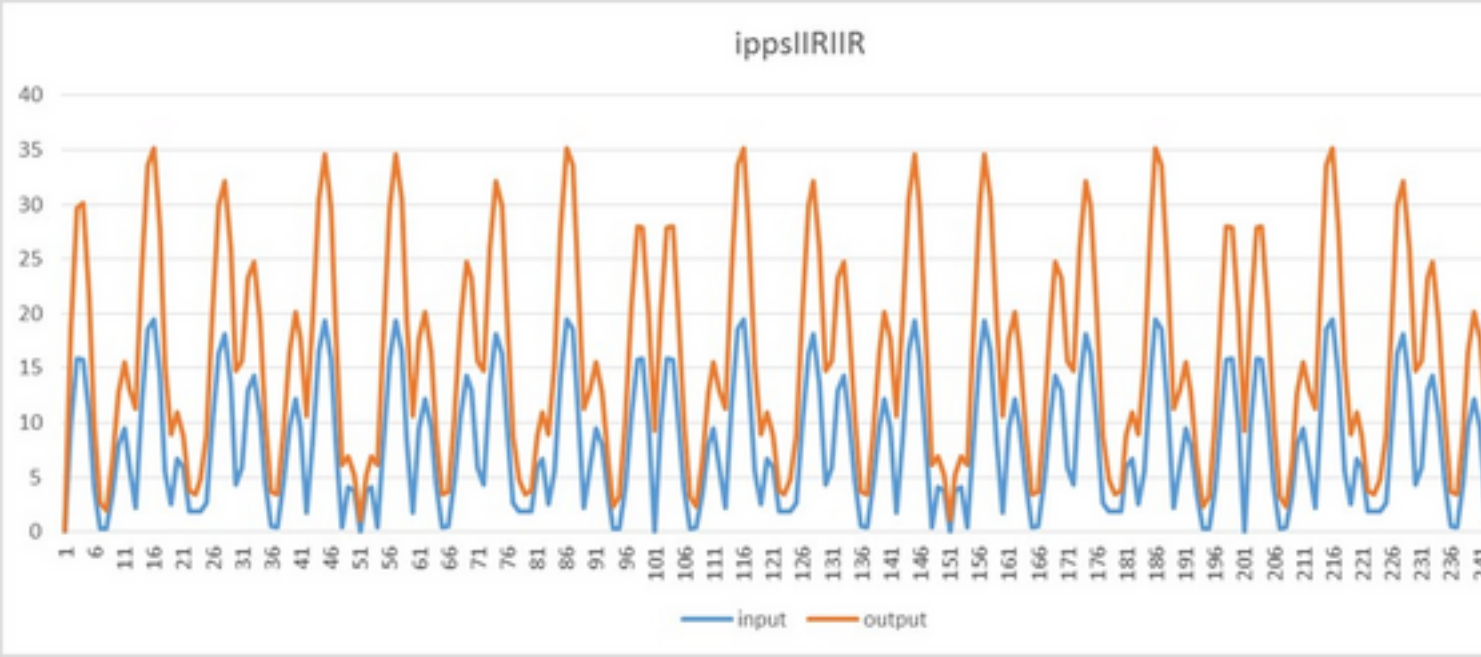
#define SMP_RATE 1000.f
#define F1       50.f
#define F2       120.f
#define AMPL     10
#define CUT_OFF  0.3
#define LEN      256
#define ORDER    3

int main()
{
    Ipp32f *x, *y, *f1, *f2;
    Ipp32f phase;
    int i, bufferSizeGen, bufferSizeIIRIIR;
    Ipp64f *pTaps;
    Ipp8u *pBufIIRIIR;
    IppsIIRState64f_32f *pStateIIRIIR;
```

```
    IppStatus st;

    /* generate test input signal - sum of 2 sinusoids with 50 & 120 Hz */
    f1 = ippsMalloc_32f(LEN);
    f2 = ippsMalloc_32f(LEN);
    x = ippsMalloc_32f(LEN);
    phase = 3 * (Ipp32f)IPP_PI2;                                                    /* ippsTone
generates cos, we want sin */
    st = ippsTone_32f(f1, LEN, AMPL, F1 / SMP_RATE, &phase, ippAlgHintAccurate);  /* generate 50
Hz sin */
    phase = 3 * (Ipp32f)IPP_PI2;                                                    /* restore
phase*/
    st |= ippsTone_32f(f2, LEN, AMPL, F2 / SMP_RATE, &phase, ippAlgHintAccurate); /* generate
120 Hz sin */
    st |= ippsAdd_32f(f1, f2, x, LEN);                                              /* add 50 &
120 Hz sin to form input signal */
    st |= ippsAbs_32f_I(x, LEN); /* make positive, x contains input signal to be filtered with
IIRIIR */
    ippsFree(f1);
    ippsFree(f2);
    /* get size of mem buffers for IIR and IIRGen functions */
    st |= ippsIIRIIRGetStateSize64f_32f(ORDER, &bufferSizeIIRIIR);
    st |= ippsIIRGenGetBufferSize(ORDER, &bufferSizeGen);
    /* we can allocate MAX size from two - operations are sequential and therefore buffer can be
reused */
    pBufIIRIIR = ippsMalloc_8u(IPP_MAX(bufferSizeGen, bufferSizeIIRIIR));
    pTaps = ippsMalloc_64f(2 * (ORDER + 1));
    /* gen LowPass filter coefficients */
    st |= ippsIIRGenLowpass_64f(CUT_OFF / 2, 0.0, ORDER, pTaps, ippButterworth, pBufIIRIIR);
    /* init state structure for IIRIIR */
    st |= ippsIIRIIRInit64f_32f(&pStateIIRIIR, pTaps, ORDER, NULL, pBufIIRIIR);
    y = ippsMalloc_32f(LEN); /* output vector */
    st |= ippsIIRIIR64f_32f(x, y, LEN, pStateIIRIIR);
    printf("idx,\tinput,\toutput\n");
    for (i = 0; i < LEN; i++) {
        printf("%d,\t%f,\t%f\n", i, (double)x[i], (double)y[i]);
    }
    if (ippStsNoErr != st)
        printf("\nCertain problems in Intel(R) Integrated Performance Primitives (Intel(R) IPP) !
\n");
    ippsFree(y);
    ippsFree(pTaps);
    ippsFree(pBufIIRIIR);
    ippsFree(x);
    return 0;
}
```

The figure below shows the result of the example, where the `X – axis` is index of the input/output vector/ signal to see that there is no phase distortion, `Y – axis – amplitude` of input/output signals.



Median Filter Functions

Median filters are nonlinear rank-order filters based on replacing each element of the source vector with the median value, taken over the fixed neighborhood (mask) of the processed element. These filters are extensively used in image and signal processing applications. Median filtering removes impulsive noise, while keeping the signal blurring to the minimum. Typically mask size (or window width) is set to odd value which ensures simple function implementation and low output signal bias. You can use an even mask size in function calls as well, but internally it will be changed to odd by subtracting 1.

Another specific feature of the median filter implementation in Intel IPP is that elements outside the source vector, which are needed to determine the median value for "border" elements, are located in a delay line. If the delay line is absent, then they are set to be equal to the corresponding edge element of the source vector.

- FilterMedianGetBufferSize
- FilterMedian

FilterMedianGetBufferSize

Computes the size of the work buffer for the `ippsFilterMedian` function.

Syntax

```
IppStatus ippsFilterMedianGetBufferSize (int maskSize, IppDataType dataType, int*
pBufferSize);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

maskSize	Size of the median mask.
dataType	Data type of the source and destination vectors. Possible values are ipp8u, ipp16s, ipp32s, ipp32f, or ipp64f.
pBufferSize	Pointer to the computed size of the external work buffer, in bytes.

Description

The ippsFilterMedianGetBufferSize function computes the size, in bytes, of the external work buffer needed for the ippsFilterMedian function. The result is stored in the pBufferSize parameter.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pBufferSize is NULL.
ippStsMaskSizeErr	Indicates an error when maskSize is less than, or equal to zero.
ippStsDataTypeErr	Indicates an error when dataType has an illegal value.
ippStsEvenMedianMaskSize	Indicates a warning when maskSize has an even value.

See Also

FilterMedian MODIFIED API. Computes median values for each source vector element.

FilterMedian

MODIFIED API. Computes median values for each source vector element.

Syntax

```
IppStatus ippsFilterMedian_8u(const Ipp8u* pSrc, Ipp8u* pDst, int len, int maskSize,
const Ipp8u* pDlySrc, Ipp8u* pDlyDst, Ipp8u* pBuffer);

IppStatus ippsFilterMedian_16s(const Ipp16s* pSrc, Ipp16s* pDst, int len, int maskSize,
const Ipp16s* pDlySrc, Ipp16s* pDlyDst, Ipp8u* pBuffer);

IppStatus ippsFilterMedian_32s(const Ipp32s* pSrc, Ipp32s* pDst, int len, int maskSize,
const Ipp32s* pDlySrc, Ipp32s* pDlyDst, Ipp8u* pBuffer);

IppStatus ippsFilterMedian_32f(const Ipp32f* pSrc, Ipp32f* pDst, int len, int maskSize,
const Ipp32f* pDlySrc, Ipp32f* pDlyDst, Ipp8u* pBuffer);

IppStatus ippsFilterMedian_64f(const Ipp64f* pSrc, Ipp64f* pDst, int len, int maskSize,
const Ipp64f* pDlySrc, Ipp64f* pDlyDst, Ipp8u* pBuffer);

IppStatus ippsFilterMedian_8u_I(Ipp8u* pSrcDst, int len, int maskSize, const Ipp8u*
pDlySrc, Ipp8u* pDlyDst, Ipp8u* pBuffer);
```

```
IppStatus ippsFilterMedian_16s_I(Ipp16s* pSrcDst, int len, int maskSize, const Ipp16s*
pDlySrc, Ipp16s* pDlyDst, Ipp8u* pBuffer);

IppStatus ippsFilterMedian_32s_I(Ipp32s* pSrcDst, int len, int maskSize, const Ipp32s*
pDlySrc, Ipp32s* pDlyDst, Ipp8u* pBuffer);

IppStatus ippsFilterMedian_32f_I(Ipp32f* pSrcDst, int len, int maskSize, const Ipp32f*
pDlySrc, Ipp32f* pDlyDst, Ipp8u* pBuffer);

IppStatus ippsFilterMedian_64f_I(Ipp64f* pSrcDst, int len, int maskSize, const Ipp64f*
pDlySrc, Ipp64f* pDlyDst, Ipp8u* pBuffer);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pSrcDst`	Pointer to the source and destination vector (for the in-place operation).
`pSrc`	Pointer to the source vector.
`pDst`	Pointer to the destination vector.
`len`	Number of elements in the vector.
`maskSize`	Median mask size, must be a positive integer. If an even value is specified, the function subtracts 1 and uses the odd value of the filter mask for median filtering.
`pDlySrc`	Pointer to the array containing values for the source delay lines.
`pDlyDst`	Pointer to the array containing values for the destination delay lines.
`pBuffer`	Pointer to the work buffer. To compute the size of the buffer, use the FilterMedianGetBufferSize function.

Description

NOTE
Important
The API of this function has been modified in Intel IPP 9.0 release.

This function computes median values for each element of the source vector `pSrc` or `pSrcDst`, and stores the result in `pDst` or `pSrcDst`, respectively.

NOTE The values for non-existent elements are stored in `pDlySrc` (if it is not NULL). The last (`maskSize - 1`) elements of vectors are stored in `pDlyDst` (if it is not NULL). For example, if `maskSize` is equal to 3, then:

```
pDst[0] = median(pDlySrc[0], pDlySrc[1], pSrc[0]);
pDst[1] = median(pDlySrc[1], pSrc[0], pSrc[1]);
pDst[2] = median(pSrc[0], pSrc[1], pSrc[2]);
...
pDlyDst[0] = pSrc[len-2];
pDlyDst[1] = pSrc[len-1]
```

If `pDlySrc` is NULL, the value of a non-existent element is equal to the value of the first vector element:

```
pDst[0] = median(pSrc[0], pSrc[0], pSrc[0]);
pDst[1] = median(pSrc[0], pSrc[0], pSrc[1]);
pDst[2] = median(pSrc[0], pSrc[1], pSrc[2]);
...
```

If `pDlyDst` is NULL, the operation of storing the last (`maskSize - 1`) elements of vectors is not performed.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when one of the `pSrcDst`, `pSrc`, `pDst`, `pBuffer` pointers is `NULL`.
`ippStsSizeErr`	Indicates an error when `len` is less than or equal to 0.
`ippStsEvenMedianMaskSize`	Indicates a warning when the median mask length is even.

Example

The example below illustrates using `ippsFilterMedian_16s_I` for single-rate filtering.

```
void median(void) {
      Ipp16s x[8] = {1,2,127,4,5,0,7,8};
      IppStatus status = ippsFilterMedian_16s_I(x, 8, 3);
      printf_16s("median =", x,8, status);
}
```

Output:

```
    median =  1 1 2 4 5 4 5 7
Matlab* Analog:
    >> x = [1 2 127 4 5 0 7 8]; medfilt1(x)
```

Polyphase Resampling Functions

The Intel® IPP functions described in this section build, apply, and free Kaizer-windowed polyphase filters for data resampling. Functions with the `Fixed` suffix are intended for fixed rational resampling factor and can provide faster speed. Functions without the suffix build universal resampling filter with linear interpolation of filter coefficients and enable a variable factor.

For general description of the polyphase resampling algorithm, see *"Multirate Digital Signal Processing"* by R. Crochire and L. Rabiner, [Cro83].

- ResamplePolyphaseGetSize, ResamplePolyphaseFixedGetSize
- ResamplePolyphaseInit, ResamplePolyphaseFixedInit
- ResamplePolyphaseSetFixedFilter
- ResamplePolyphaseGetFixedFilter
- ResamplePolyphase, ResamplePolyphaseFixed

ResamplePolyphaseGetSize, ResamplePolyphaseFixedGetSize

Get the size of the polyphase resampling structure.

Syntax

```
IppStatus ippsResamplePolyphaseGetSize_16s(Ipp32f window, int nStep, int* pSize,
IppHintAlgorithm hint);
```

```
IppStatus ippsResamplePolyphaseGetSize_32f(Ipp32f window, int nStep, int* pSize,
IppHintAlgorithm hint);
```

```
IppStatus ippsResamplePolyphaseFixedGetSize_16s(int inRate, int outRate, int len, int*
pSize, int* pLen, int* pHeight, IppHintAlgorithm hint);
```

```
IppStatus ippsResamplePolyphaseFixedGetSize_32f(int inRate, int outRate, int len, int*
pSize, int* pLen, int* pHeight, IppHintAlgorithm hint);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`window`	The size of the ideal lowpass filter window.
`nStep`	The discretization step for filter coefficients.
`inRate`	The input rate for fixed factor resampling.
`outRate`	The output rate for fixed factor resampling.
`len`	The filter length for fixed factor resampling.
`pSize`	The pointer to the variable that contains the size of the polyphase resampling structure.
`pLen`	The pointer to the variable that contains the real filter length.
`pHeight`	The pointer to the variable that contains the number of filters.

hint	Suggests using specific code (must be equal to ippAlgHintFast). Possible values for the hint parameter are described in IppHintAlgorithm enum Type.

Description

These functions determine the size required for the fixed rate polyphase resampling structure and associated storage, the filter length, and the number of filters in the filter bank. The returned length of the filter is equal to min{l≥len, l%4}, the subfilter length for zero phase is greater by 1. These values can be used for export and import of fixed polyphase resampling filter.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	For ippsResamplePolyphaseGetSize function, indicates an error when inRate, outRate,nStep``or ``len is less than or equal to 0. For ippsResamplePolyphaseFixedGetSize function, indicates an error when inRate, outRate, nStep or len is less than or equal to 0.
ippStsBadArgErr	Indicates an error when window is less than 2/nStep.

ResamplePolyphaseInit, ResamplePolyphaseFixedInit

Initialize the structure for polyphase resampling with calculating the filter coefficients.

Syntax

```
IppStatus ippsResamplePolyphaseInit_16s( Ipp32f window, int nStep, Ipp32f rollf, Ipp32f alpha, IppsResamplingPolyphase_16s* pSpec, IppHintAlgorithm hint);
```

```
IppStatus ippsResamplePolyphaseInit_32f( Ipp32f window, int nStep, Ipp32f rollf, Ipp32f alpha, IppsResamplingPolyphase_32f* pSpec, IppHintAlgorithm hint);
```

```
IppStatus ippsResamplePolyphaseFixedInit_16s( int inRate, int outRate, int len, Ipp32f rollf, Ipp32f alpha, IppsResamplingPolyphaseFixed_16s* pSpec, IppHintAlgorithm hint);
```

```
IppStatus ippsResamplePolyphaseFixedInit_32f( int inRate, int outRate, int len, Ipp32f rollf, Ipp32f alpha, IppsResamplingPolyphaseFixed_32f* pSpec, IppHintAlgorithm hint);
```

Include Files

ipps.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

window	The size of the ideal lowpass filter window.
nStep	The discretization step for filter coefficients.
rollf	The roll-off frequency of the filter.
alpha	The parameter of the Kaiser window.
inRate	The input rate for fixed factor resampling.
outRate	The output rate for fixed factor resampling.
len	The filter length for fixed factor resampling.
pSpec	The pointer to the resampling state structure.
hint	Suggests using specific code (must be equal to ippAlgHintFast). The possible values for the parameter hint are described in IppHintAlgorithm enum Type.

Description

The function ippsResamplePolyphaseInit initializes structures for data resampling using the ideal lowpass filter. The function ippsResamplePolyphaseInit applies the Kaiser window with alpha parameter and window width to the lowpass filter. This means that the values of the ideal lowpass filtering function are calculated for all i values such that | i/nStep |≤ window.

Use the pSpec structure to resample input samples with the ippsResample function with arbitrary resampling factor. In this case, filter coefficients for each output sample are calculated using linear interpolation between two nearest values. The size of the filter depends on the resampling factor.

The function ippsResamplePolyphaseFixedInit initializes structures for data resampling with the factor equal to inRate/outRate. If you denote the number of filters created in the IppsResamplingPolyphaseStructure structure for input and output frequencies by fnum, then

```
fnum = outRate/GCD (inRate, outRate)
```

where

GCD (a, b) is the greatest common divisor of a and b. For example, if inRate = 8000 and outRate = 11025, then the number of filters will be fnum = 11025/ GCD (8000, 11025) = 441.

Functions with the Fixed suffix pre-calculate filter coefficients for each phase and store them in the data structure for better performance. Use these functions when the ratio inRate/outRate is rational only. These functions can be considerably faster but may require large data structures for some input and output rates.

Before calling these functions, you need to allocate memory for the resampling state structure. To calculate the memory size, filter length, and the number of filters, use the ippsResamplePolyphaseGetSize or ippsResamplePolyphaseFixedGetSize functions.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.

ippStsSizeErr	For ippsResamplePolyphaseInit function, indicates an error when inRate, outRate,nStep``or ``len is less than or equal to 0. For ippsResamplePolyphaseFixedInit function, indicates an error when inRate, outRate, nStep or len is less than or equal to 0.
ippStsBadArgErr	Indicates an error when rollf is less than or equal to 0 or is greater than 1, or if alpha is less than 1, or if window is less than 2/ nStep.

ResamplePolyphaseSetFixedFilter

Sets polyphase resampling filter coefficients.

Syntax

```
IppStatus ippsResamplePolyphaseSetFixedFilter_16s(const Ipp16s* pSrc, int step, int height, IppsResamplingPolyphaseFixed_16s* pSpec);
```

```
IppStatus ippsResamplePolyphaseSetFixedFilter_32f(const Ipp32f* pSrc, int step, int height, IppsResamplingPolyphaseFixed_32f* pSpec);
```

Include Files

ipps.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

pSrc	The pointer to the input vector of filter coefficients.
step	The row step in pSrc vector.
height	The number of filters (the number of rows in pSrc vector).
pSpec	The pointer to the resampling state structure.

Description

This function imports pre-calculated filter coefficients into the polyphase resampling structure. If the step value is less than the filter length, trailing filter coefficients are zeroed.

When allocating memory keep in mind that actual structure size of the filter is height * step + 1, because zero-phase subfilter has step + 1 coefficients. This function only works with structures obtained by using ResamplePolyphaseFixedInit function. The behavior for custom-created structures is undefined.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when step or height is less than or equal to 0.
ippStsBadArgErr	Indicates that height is greater than the number of filters in pSpec structure.

ResamplePolyphaseGetFixedFilter

Gets polyphase resampling filter coefficients.

Syntax

```
IppStatus ippsResamplePolyphaseGetFixedFilter_16s(Ipp16s* pDst, int step, int height,
const IppsResamplingPolyphaseFixed_16s* pSpec);
```

```
IppStatus ippsResamplePolyphaseGetFixedFilter_32f(Ipp32f* pDst, int step, int height,
const IppsResamplingPolyphaseFixed_32f* pSpec);
```

Include Files

ipps.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

pDst	The pointer to the output vector of filter coefficients.
step	The row step in pDst vector.
height	The number of filters (the number of rows in pDst vector).
pSpec	The pointer to the resampling state structure.

Description

This function exports filter coefficients from the polyphase resampling structure. If the step value is less than the filter length, only first step coefficients are exported.

Return Values

ippStsNoErr	Indicates no error.

ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when step or height is less than or equal to 0.
ippStsBadArgErr	Indicates an error when height is greater than the number of filters in pSpec structure.

Example

The code example below demonstrates export and import of the Polyphase Resampling Filter Bank.

```
// Export filter

int inRate=16000;  // input frequency
int outRate=8000;  // output frequency
int history;       // half of filter length
char fname[]="filter.flt\0"; // coefficient file name

{
   int size,len,height;
   FILE *file;
   short *pFilter;
   IppsResamplingPolyphaseFixed_16s *state;

   history=(int)(64.0f*0.5*IPP_MAX(1.0,1.0/(double)outRate/(double)inRate))+1;
   ippsResamplePolyphaseFixedGetSize_16s(inRate, outRate, 2*(history-1),
                                         &size, &len, &height, ippAlgHintFast);
   state = (IppsResamplingPolyphaseFixed_16s*)ippsMalloc_8u(size);
   ippsResamplePolyphaseFixedInit_16s(inRate, outRate, 2*(history-1),
                                      0.95f, 9.0f, state, ippAlgHintFast);
   pFilter=ippsMalloc_16s(len*height);
   ippsResamplePolyphaseGetFixedFilter_16s(pFilter, len, height, state);

   file=fopen(fname,"wb");
   fwrite(&size, sizeof(int), 1, file);
   fwrite(&len, sizeof(int), 1, file);
   fwrite(&height, sizeof(int), 1, file);
   fwrite(pFilter, sizeof(short), len*height, file);
   fclose(file);

   ippsFree(pFilter);
   ippsFree(state);
}

// Import filter
{
   int size,len,height;
   FILE *file;
   short *pFilter;
   IppsResamplingPolyphaseFixed_16s *state;

   history=(int)(64.0f*0.5*IPP_MAX(1.0,1.0/(double)outRate/(double)inRate))+1;

   file=fopen(fname,"rb");
   fread(&size, sizeof(int), 1, file);
   fread(&len, sizeof(int), 1, file);
```

```
   fread(&height, sizeof(int), 1, file);
   pFilter=ippsMalloc_16s(len*height);
   fread(pFilter, sizeof(short), len*height, file);
   fclose(file);

   state=(IppsResamplingPolyphaseFixed_16s*)ippsMalloc_8u(size);
   ippsResamplePolyphaseFixedInit_16s(inRate, outRate, 2*(history-1),
                                      0.95f, 9.0f, state, ippAlgHintFast);
   ippsResamplePolyphaseSetFixedFilter_16s((const Ipp16s*)pFilter, len, height,
                                           (IppsResamplingPolyphaseFixed_16s*)state);
   ippsFree(pFilter);

   // use of polyphase filter
   // ...

   ippsFree(state);
}
```

ResamplePolyphase, ResamplePolyphaseFixed

Resample input data using polyphase filters.

Syntax

```
IppStatus ippsResamplePolyphase_16s(const Ipp16s* pSrc, int len, Ipp16s* pDst, Ipp64f
factor, Ipp32f norm, Ipp64f* pTime, int* pOutlen, const IppsResamplingPolyphase_16s*
pSpec);
```

```
IppStatus ippsResamplePolyphase_32f(const Ipp32f* pSrc, int len, Ipp32f* pDst, Ipp64f
factor, Ipp32f norm, Ipp64f* pTime, int* pOutlen, const IppsResamplingPolyphase_32f*
pSpec);
```

```
IppStatus ippsResamplePolyphaseFixed_16s(const Ipp16s* pSrc, int len, Ipp16s* pDst,
Ipp32f norm, Ipp64f* pTime, int* pOutlen, const IppsResamplingPolyphaseFixed_16s*
pSpec);
```

```
IppStatus ippsResamplePolyphaseFixed_32f(const Ipp32f* pSrc, int len, Ipp32f* pDst,
Ipp32f norm, Ipp64f* pTime, int* pOutlen, const IppsResamplingPolyphaseFixed_32f*
pSpec);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pSrc`	The pointer to the input vector.
`pDst`	The pointer to the output vector.
`len`	The number of input vector elements to resample.

norm	The norm factor for output samples.
factor	The resampling factor.
pTime	The pointer to the start time of resampling (in input vector elements). Keeps the input sample number and the phase for the first output sample from the next input data portion.
pOutlen	The number of calculated output vector elements.
pSpec	The pointer to the resampling state structure.

Description

These functions convert data from the input vector changing their frequency and compute all output samples that can be correctly calculated for the given input and the filter length. For the `ippsResamplePolyphase` function, the ratio of output and input frequencies is defined by the `factor` argument. For the `ippsResamplePolyphaseFixed` function, this ratio is defined during creation of the resampling structure. The value for `pTime` [0] defines the time value for which the first output sample is calculated.

Input vector with indices less than `pTime[0]` contains the history data of filters. The history length is equal to `flen / 2` for `ippsResamplePolyphaseFixed` function , and [1/2 `window` *max(1, `1/ factor`)]+1 for `ippsResamplePolyphase` function. Here `flen` is the filter length and `window` is the size of the ideal lowpass filter window. The input vector must contain the same number of elements with indices greater than `pTime` [0] + `len` for the right filter wing for the last element.

After function execution, the time value is updated and `pOutlen` [0] contains the number of calculated output samples.

The output samples are multiplied by `norm` * min (1, `factor`) before saturation.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when `pSpec`, `pSrc`, `pDst`, `pTime` or `pOutlen` pointer is `NULL`.
ippStsSizeErr	Indicates an error when `len` is less than or equal to 0.
ippStsBadArgErr	Indicates an error when `factor` is less than or equal to 0.

Example

The code example below demonstrates resampling of the input mono pcm file.

```
void resampleIPP(
    int      inRate,    // input frequency
    int      outRate,   // output frequency
    FILE    *infd,      // input pcm file
    FILE    *outfd)     // output pcm file
 {  short *inBuf,*outBuf;
    int bufsize=4096;
    int history=128;
    double time=history;
```

```
    int lastread=history;
    int inCount=0,outCount=0,inLen,outLen;
    int size,len,height;
    IppsResamplingPolyphaseFixed_16s *state;
    ippsResamplePolyphaseFixedGetSize_16s(inRate,outRate,2*(history-1),&size,&len,
&height,ippAlgHintAccurate);
    state=(IppsResamlingPolyphaseFixed_16s*)ippsMalloc_8u(size);
    ippsResamplePolyphaseFixedInit_16s(inRate,outRate,2*(history-1),0.95f,9.0f,state,
ippAlgHintAccurate);
    inBuf=ippsMalloc_16s(bufsize+history+2);
    outBuf=ippsMalloc_16s((int)((bufsize-history)*outRate/(float)inRate+2));
    ippsZero_16s(inBuf,history);
    while ((inLen=fread(inBuf+lastread,sizeof(short),bufsize-lastread,infd))>0) {
       inCount+=inLen;
       lastread+=inLen;
       ippsResamplePolyphaseFixed_16s(inBuf,lastread-history-(int)time,
                           outBuf,0.98f,&time,&outLen,state);
       fwrite(outBuf,outLen,sizeof(short),outfd);
       outCount+=outLen;
       ippsMove_16s(inBuf+(int)time-history,inBuf,lastread+history-(int)time);
       lastread-=(int)time-history;
       time-=(int)time-history;
    }
                ippsZero_16s(inBuf+lastread,history);
    ippsResamplePolyphaseFixed_16s(inBuf,lastread-(int)time,
                         outBuf,0.98f,&time,&outLen,state);
    fwrite(outBuf,outLen,sizeof(short),outfd);
    outCount+=outLen;
    printf("%d inputs resampled to %d outputs\n",inCount,outCount);
    ippsFree(outBuf);
    ippsFree(inBuf);
    ippsFree (state);
 }
```

Transform Functions

This chapter describes the Intel ® IPP functions that perform Fourier and discrete cosine transforms (DCT), as well as Hartley, Hilbert, Walsh-Hadamard and wavelet transforms of signals.

- Fourier Transform Functions
- Discrete Cosine Transform Functions
- Hilbert Transform Functions
- Wavelet Transform Functions

Fourier Transform Functions

The functions described in this section perform the fast Fourier transform (FFT), the discrete Fourier transform (DFT) of signal samples. It also includes variations of the basic functions to support different application requirements.

- Special Arguments
- Packed Formats
- Format Conversion Functions
- Functions for Packed Data Multiplication
- Fast Fourier Transform Functions
- Discrete Fourier Transform Functions

- DFT for a Given Frequency (Goertzel) Functions

Special Arguments

This section describes the flag and hint arguments used by the Fourier transform functions.

The Fourier transform functions require you to specify the `flag` and `hint` arguments.

The `flag` argument specifies the result normalization method. The following table lists the possible values for the `flag` argument. Specify one and only one of the represented values in the `flag` argument. The **A** and **B** factors are multipliers used in the DFT computation.

Value	A	B	Description
`IPP_FFT_DIV_FWD_BY_N`	`1/ N`	1	Forward transform is done with the `1/ N` normalization.
`IPP_FFT_DIV_INV_BY_N`	1	`1/ N`	Inverse transform is done with the `1/ N` normalization.
`IPP_FFT_DIV_BY_SQRTN`	$1/ N^{1/2}$	$1/ N^{1/2}$	Forward and inverse transform is done with the $1/ N^{1/2}$ normalization.
`IPP_FFT_NODIV_BY_ANY`	1	1	Forward or inverse transform is done without the `1/ N` or $1/ N^{1/2}$ normalization.

Packed Formats

This section describes the main packed formats `Perm`, `Pack`, and `CCS` used by the Fourier transform functions.

Pack Format

The `Pack` format is a convenient, compact representation of a complex conjugate-symmetric sequence. The disadvantage of this format is that it is not the natural format used by the real Fourier transform algorithms (“natural” in the sense that bit-reversed order is natural for radix-2 complex Fourier transforms). In `Pack` format, the output samples of the Fourier transform are arranged as shown in the tables below. The output signal can be unpacked to a complex signal using the function `ippsConjPack`.

Perm Format

The `Perm` format stores the values in the order in which the Fourier transform algorithms use them. This is the most natural way of storing values for the Fourier transform algorithms. The `Perm` format is an arbitrary permutation of the `Pack` format. An important characteristic of the `Perm` format is that the real and imaginary parts of a given sample need not be adjacent.

In `Perm` format, the output samples of the Fourier transform are arranged as shown in the tables below. The output signal can be unpacked to a complex signal using the function `ippsConjPerm`.

NOTE For input signal of odd length the perm and pack format are identical.

CCS Format

The CCS format stores the values of the first half of the output complex signal resulted from the forward Fourier transform.

In CCS format, the output samples of the Fourier transform are arranged as shown in the tables below. Note that the signal stored in CCS format is one complex element longer. The output signal can be unpacked to a complex signal using the function ippsConjCcs.

Index	0	1	2	3	…	N-2	N-1	N	N+1
Pack	R_0	R_1	I_1	R_2	…	$I_{(N-1)/2}$	$R_{N/2}$		
Perm	R_0	$R_{N/2}$	R_1	I_1	…	$R_{N/2-1}$	$I_{N/2-1}$		
CCS	R_0	0	R_1	I_1	…	$R_{N/2-1}$	$I_{N/2-1}$	$R_{N/2}$	0

Index	0	1	2	3	…	N-2	N-1	N
Pack	R_0	R_1	I_1	R_2	…	$R_{(N-1)/2}$	$I_{(N-1)/2}$	
Perm	R_0	R_1	I_1	R_2	…	$R_{(N-1)/2}$	$I_{(N-1)/2}$	
CCS	R_0	0	R_1	I_1	…	$I_{(N-1)/2-1}$	$R_{(N-1)/2}$	$I_{(N-1)/2}$

Format Conversion Functions

The following functions ippsConjPack, ippsConjPerm, and ippsConjCcs convert data from the packed formats to a usual complex data format using the FFT symmetry property for transforming real data. The output data is complex, the output array length is defined by the number of complex elements in the output vector. Note that the output array size is two times as big as the input array size. The data stored in CCS format require a bigger array than the other formats. Even and odd length arrays have some specific features discussed for each function separately.

- ConjPack
- ConjPerm
- ConjCcs

ConjPack

Converts the data in Pack format to complex data format.

Syntax

```
IppStatus ippsConjPack_32fc(const Ipp32f* pSrc, Ipp32fc* pDst, int lenDst);
IppStatus ippsConjPack_64fc(const Ipp64f* pSrc, Ipp64fc* pDst, int lenDst);
IppStatus ippsConjPack_32fc_I(Ipp32fc* pSrcDst, int lenDst);
IppStatus ippsConjPack_64fc_I(Ipp64fc* pSrcDst, int lenDst);
```

Platform-aware functions:

```
IppStatus ippsConjPack_32fc_L(const Ipp32f* pSrc, Ipp32fc* pDst, IppSizeL lenDst);
IppStatus ippsConjPack_64fc_L(const Ipp64f* pSrc, Ipp64fc* pDst, IppSizeL lenDst);
IppStatus ippsConjPack_32fc_I_L(Ipp32fc* pSrcDst, IppSizeL lenDst);
IppStatus ippsConjPack_64fc_I_L(Ipp64fc* pSrcDst, IppSizeL lenDst);
```

Include Files

ipps.h

Flavors with the _L suffix: ipps_l.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

pSrc	Pointer to the source vector.
pDst	Pointer to the destination vector.
pSrcDst	Pointer to the source and destination vector (for the in-place operation).
lenDst	Number of elements in the vector.

Description

This function converts the data in Pack format in the vector pSrc to complex data format and stores the results in pDst.

The in-place function ippsConjPack converts the data in Pack format in the vector pSrcDst to complex data format and stores the results in pSrcDst.

The table below shows the examples of unpack from the Pack format. The Data column contains the real input data to be converted by the forward FFT transform to the packed data. The packed real data is in the Packed column. The output result is the complex data vector in the Extended column. The number of vector elements is in the Length column.

Data	Packed	Extended	Length
FFT([1])	1	{1, 0}	1
FFT([1 2])	3, -1	{3, 0}, {-1, 0}	2
FFT([1 2 3])	6, -1.5, 0.86	{6, 0}, {-1.5, 0.86}, {-1.5, -0.86}	3
FFT([1 2 3 9])	15, -2, 7, -7	{15, 0}, {-2, 7}, {-7, 0}, {-2, -7}	4

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when the pSrcDst, pDst, or pSrc pointer is NULL.
ippStsSizeErr	Indicates an error when lenDst is less than or equal to 0.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

#include <stdio.h>
#include "ipp.h"

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define
check_sts(st)
                                  \
    if ((st) !=
ippStsNoErr)
                          \
        goto exitLine; /* Go to Exit if Intel(R) Integrated Performance Primitives (Intel(R)
IPP) function returned status different from \
                           ippStsNoErr */

int main(void)
{
    IppStatus status;
    Ipp32f x[8] = {1, 2, 3, 5, 6, 7, 8, 9};
    Ipp32fc zero = {0, 0}, y[6] = {0};
    int i = 0;
    check_sts(status = ippsSet_32fc(zero, y, 6));
    check_sts(status = ippsConjPack_32fc(x, y, 6));
    for (i = 0; i < 6; i++) {
        printf("{%.3f %.3f} ", y[i].re, y[i].im);
    }
    printf("\n");
    check_sts(status = ippsSet_32fc(zero, y, 6));
    check_sts(status = ippsConjPack_32fc(x, y, 5));
    for (i = 0; i < 5; i++) {
        printf("{%.3f %.3f} ", y[i].re, y[i].im);
    }
    printf("\n");

    EXIT_MAIN
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

ConjPerm

Converts the data in `Perm` format to complex data format.

Syntax

```
IppStatus ippsConjPerm_32fc(const Ipp32f* pSrc, Ipp32fc* pDst, int lenDst);
IppStatus ippsConjPerm_64fc(const Ipp64f* pSrc, Ipp64fc* pDst, int lenDst);
IppStatus ippsConjPerm_32fc_I(Ipp32fc* pSrcDst, int lenDst);
IppStatus ippsConjPerm_64fc_I(Ipp64fc* pSrcDst, int lenDst);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pSrc`	Pointer to the source vector.
`pDst`	Pointer to the destination vector.
`pSrcDst`	Pointer to the source and destination vector (for the in-place operation).
`lenDst`	Number of elements in the vector.

Description

This function converts the data in Perm format in the vector `pSrc` to complex data format and stores the results in `pDst`.

The in-place function `ippsConjPerm` converts the data in `Perm` format in the vector `pSrcDst` to complex data format and stores the results in `pSrcDst`.

The following table shows the examples of unpack from the `Perm` format. The `Data` column contains the real input data to be converted by the forward FFT transform to the packed data. The packed real data are in the `Packed` column. The output result is the complex data vector in the `Extended` column. The number of vector elements is in the `Length` column.

Data	Packed	Extended	Length
FFT([1])	1	{1, 0}	1
FFT([1 2])	3, -1	{3, 0}, {-1, 0}	2
FFT([1 2 3])	6, -1.5, 0.86	{6, 0}, {-1.5, 0.86}, {-1.5, -0.86}	3
FFT([1 2 3 9])	15, -7, -2, 7	{15, 0}, {-2, 7}, {-7, 0}, {-2, -7}	4

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when the pSrcDst, pDst, or pSrc pointer is NULL.
ippStsSizeErr	Indicates an error when lenDst is less than or equal to 0.

ConjCcs

Converts the data in CCS format to complex data format.

Syntax

```
IppStatus ippsConjCcs_32fc(const Ipp32f* pSrc, Ipp32fc* pDst, int lenDst);
IppStatus ippsConjCcs_64fc(const Ipp64f* pSrc, Ipp64fc* pDst, int lenDst);
IppStatus ippsConjCcs_32fc_I(Ipp32fc* pSrcDst, int lenDst);
IppStatus ippsConjCcs_64fc_I(Ipp64fc* pSrcDst, int lenDst);
```

Platform-aware functions:

```
IppStatus ippsConjCcs_32fc_L(const Ipp32f* pSrc, Ipp32fc* pDst, IppSizeL lenDst);
IppStatus ippsConjCcs_64fc_L(const Ipp64f* pSrc, Ipp64fc* pDst, IppSizeL lenDst);
IppStatus ippsConjCcs_32fc_I_L(Ipp32fc* pSrcDst, IppSizeL lenDst);
IppStatus ippsConjCcs_64fc_I_L(Ipp64fc* pSrcDst, IppSizeL lenDst);
```

Include Files

ipps.h

Flavors with the _L suffix: ipps_l.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

pSrc	Pointer to the source vector.
pDst	Pointer to the destination vector.
pSrcDst	Pointer to the source and destination vector (for the in-place operation).
lenDst	Number of elements in the vector.

Description

This function converts the data in CCS format in the vector pSrc to complex data format and stores the results in pDst.

The in-place function ippsConjCcs converts the data in CCS format in the vector pSrcDst to complex data format and stores the results in pSrcDst.

The following table shows the examples of unpack from the CCS format. The Data column contains the real input data to be converted by the forward FFT transform to the packed data. The packed real data are in the Packed column. The output result is the complex data vector in the Extended column. The number of vector elements is in the Length column. The data stored in CCS format are two real elements longer.

Data	Packed	Extended	Length
FFT([1])	1, 0	{1, 0}	1
FFT([1 2])	3, 0, -1, 0	{3, 0}, {-1, 0}	2
FFT([1 2 3])	6, 0, -1.5, 0.86	{6, 0}, {-1.5, 0.86}, {-1.5, -0.86}	3
FFT([1 2 3 9])	15, 0, -2, 7, -7, 0	{15, 0}, {-2, 7}, {-7, 0}, {-2, -7}	4

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when the pSrcDst, pDst, or pSrc pointer is NULL.
ippStsSizeErr	Indicates an error when lenDst is less than or equal to 0.

Functions for Packed Data Multiplication

The functions described in this section perform the element-wise complex multiplication of vectors stored in Pack or Perm formats. These functions are used with the function ippsFFTFwd and ippsFFTInv to perform fast convolution on real signals.

The standard vector multiplication function ippsMul can not be used to multiply Pack or Perm format vectors because:

- Two real samples are stored in Pack format.
- The Perm format might not pair the real parts of a signal with their corresponding imaginary parts.

> **NOTE** The vectors stored in CCS format can be multiplied using the standard function for complex data multiplication.

- MulPack
- MulPerm
- MulPackConj

MulPack

Multiply the elements of two vectors stored in Pack format.

Syntax

Case 1: Not-in-place operation

```
IppStatus ippsMulPack_32f(const Ipp32f* pSrc1, const Ipp32f* pSrc2, Ipp32f* pDst, int len);

IppStatus ippsMulPack_64f(const Ipp64f* pSrc1, const Ipp64f* pSrc2, Ipp64f* pDst, int len);
```

Case 2: In-place operation

```
IppStatus ippsMulPack_32f_I(const Ipp32f* pSrc, Ipp32f* pSrcDst, int len);

IppStatus ippsMulPack_64f_I(const Ipp64f* pSrc, Ipp64f* pSrcDst, int len);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pSrc1`, `pSrc2`	Pointers to the vectors whose elements are to be multiplied together.
`pDst`	Pointer to the destination vector which stores the result of the multiplication `pSrc1[n] * pSrc2[n]`.
`pSrc`	Pointer to the vector whose elements are to be multiplied by the elements of `pSrcDst` in-place.
`pSrcDst`	Pointer to the source and destination vector (for the in-place operation).
`len`	Number of elements in the vector.

Description

This function multiplies the elements of the vector `pSrc1` by the elements of the vector `pSrc2`, and stores the result in `pDst`.

The in-place flavors `ippsMulPack` multiply the elements of the vector `pSrc` by the elements of the vector `pSrcDst`, and store the result in `pSrcDst`.

The functions multiply the packed data according to their packed format. The data in Pack packed format include several real values, the rest are complex. Thus, the function performs several real multiplication operations on real elements and complex multiplication operations on complex data. Such kind of packed data multiplication is usually used for signals filtering with the FFT transform when the element-wise multiplication is performed in the frequency domain.

Return Values

`ippStsNoErr`	Indicates no error
`ippStsNullPtrErr`	Indicates an error when the `pSrcDst`, `pDst`, `pSrc1`, `pSrc2`, or `pSrc` pointer is `NULL`.

ippStsSizeErr	Indicates an error when `len` is less than or equal to 0.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

//    The code example below illustrates how to use the function ippsMulPack_32f_I.
// implemented with Intel(R) Integrated Performance Primitives (Intel(R) IPP) functions:
//     ippsFFTGetSize_R_32f
//     ippsFFTInit_R_32f
//     ippsSet_32f
//     ippsFFTFwd_RToPack_32f
//     ippsMulPack_32f_I
//     ippsFFTInv_PackToR_32f

#include <stdio.h>
#include "ipp.h"

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define check_sts(st)        \
    if ((st) != ippStsNoErr) \
        goto exitLine; /* Go to Exit if Intel(R) IPP function returned status different from
ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main(void)
{
    int i = 0;
    IppStatus status = ippStsNoErr;
    Ipp32f pFlt[8] = {1.0f / 3, 1.0f / 3, 1.0f / 3, 0, 0, 0, 0, 0}; /* Pointers to filter signal
*/
    Ipp32f pSrc[8], pDst[8];                                        /* Pointers to source/filter/
destination signals */
    Ipp32f pSrcFr[8], pFltFr[8];                                    /* Pointers to source/filter/
destination signals into the frequency domain */

    IppsFFTSpec_R_32f *pSpec = NULL;                           /* Pointer to FFT pSpec structure
*/
    Ipp8u *pMemInit = NULL, *pBuffer = NULL, *pSpecMem = NULL; /* Pointer to the work buffers */
```

```
    int sizeSpec = 0, sizeInit = 0, sizeBuf = 0;              /* size of FFT pSpec structure,
Init and work buffers */

    check_sts(status = ippsFFTGetSize_R_32f(3, IPP_FFT_DIV_INV_BY_N, ippAlgHintNone, &sizeSpec,
&sizeInit, &sizeBuf));

    /* memory allocation */
    pSpecMem = (Ipp8u *)ippMalloc(sizeSpec);
    pBuffer = (Ipp8u *)ippMalloc(sizeBuf);
    pMemInit = (Ipp8u *)ippMalloc(sizeInit);

    check_sts(status = ippsFFTInit_R_32f(&pSpec, 3, IPP_FFT_DIV_INV_BY_N, ippAlgHintNone,
pSpecMem, pMemInit));

    ippsSet_32f(3, pSrc, 8);
    pSrc[3] = 5;

    /* forward FFT transform : the source signal is transformed into the frequency domain*/
    check_sts(status = ippsFFTFwd_RToPack_32f(pSrc, pSrcFr, pSpec, NULL));

    /* forward FFT transform: the filter signal is transformed into the frequency domain */
    check_sts(status = ippsFFTFwd_RToPack_32f(pFlt, pFltFr, pSpec, NULL));

    /* multiplying the packed data */
    check_sts(status = ippsMulPack_32f_I(pFltFr, pSrcFr, 8));

    /* inverse FFT transform : the filtered data is transformed to the time domain */
    check_sts(status = ippsFFTInv_PackToR_32f(pSrcFr, pDst, pSpec, NULL));

    printf("filtered =");

    for (i = 0; i < 8; ++i)
        printf(" %f", pDst[i]);
    printf("\n");

    EXIT_MAIN
    ippFree(pMemInit);
    ippFree(pSpec);
    ippFree(pBuffer);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return status;
}
```

MulPerm

Multiply the elements of two vectors stored in `Perm` format.

Syntax

Case 1: Not-in-place operation

```
IppStatus ippsMulPerm_32f(const Ipp32f* pSrc1, const Ipp32f* pSrc2, Ipp32f* pDst, int
len);
```

```
IppStatus ippsMulPerm_64f(const Ipp64f* pSrc1, const Ipp64f* pSrc2, Ipp64f* pDst, int
len);
```

Case 2: In-place operation

```
IppStatus ippsMulPerm_32f_I(const Ipp32f* pSrc, Ipp32f* pSrcDst, int len);
IppStatus ippsMulPerm_64f_I(const Ipp64f* pSrc, Ipp64f* pSrcDst, int len);
```

Include Files

ipps.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

pSrc1, pSrc2	Pointers to the vectors whose elements are to be multiplied together.
pDst	Pointer to the destination vector which stores the result of the multiplication pSrc1[n] * pSrc2[n].
pSrc	Pointer to the vector whose elements are to be multiplied by the elements of pSrcDst in-place.
pSrcDst	Pointer to the source and destination vector (for the in-place operation).
len	Number of elements in the vector.

Description

This function multiplies the elements of the vector pSrc1 by the elements of the vector pSrc2, and stores the result in pDst.

The in-place flavors of ippsMulPerm multiply the elements of the vector pSrc by the elements of the vector pSrcDst, and store the result in pSrcDst.

The function multiplies the packed data according to their packed format. The data in Perm packed formats include several real values, the rest are complex. Thus, the function performs several real multiplication operations on real elements and complex multiplication operations on complex data. Such kind of packed data multiplication is usually used for signals filtering with the FFT transform when the element-wise multiplication is performed in the frequency domain.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when the pSrcDst, pDst, pSrc1, pSrc2, or pSrc pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.

MulPackConj

Multiplies elements of a vector by the elements of a complex conjugate vector stored in Pack format.

Syntax

```
IppStatus ippsMulPackConj_32f_I(const Ipp32f* pSrc, Ipp32f* pSrcDst, int len);
IppStatus ippsMulPackConj_64f_I(const Ipp64f* pSrc, Ipp64f* pSrcDst, int len);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pSrc`	Pointer to the first source vector.
`pSrcDst`	Pointer to the second source and destination vector.
`len`	Number of elements in the vector.

Description

This function multiplies the elements of a source vector `pSrc` by elements of the vector that is complex conjugate to the source vector `pSrcDst` and stores the results in `pSrcDst`. The function performs only in-place operations on data stored in Pack format.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when the `pSrc` or `pSrcDst` pointer is `NULL`.
`ippStsSizeErr`	Indicates an error when `len` is less than or equal to 0.

Fast Fourier Transform Functions

The functions described in this section compute the forward and inverse fast Fourier transform of real and complex signals. The FFT is similar to the discrete Fourier transform (DFT) but is significantly faster. The length of the vector transformed by the FFT must be a power of 2.

To use the FFT functions, initialize the specification structure which contains such data as tables of twiddle factors. The initialization functions create the specifications for both forward and inverse transforms. The amount of prior calculations is thus reduced and the overall performance increased.

The `hint` argument, passed to the initialization functions, suggests using special algorithm, faster or more accurate. The `flag` argument specifies the result normalization method.

To initialize the FFT specification structure, use the ippsFFTInit_R and ippsFFTInit_C functions. Before using these functions, you need to compute the size of the specification structure using ippsFFTGetSize_R and ippsFFTGetSize_C, respectively.

The complex signal can be represented as a single array containing complex elements, or two separate arrays containing real and imaginary parts. The output result of the FFT can be packed in `Perm`, `Pack`, or `CCS` format.

You can speed up the FFT by using an external buffer. The use of external buffer can improve performance by avoiding allocation and deallocation of internal buffers and storing data in cache. The size of the external buffer is returned by the ippsFFTInit_R and ippsFFTInit_C functions.

- FFTInit_R, FFTInit_C
- FFTGetSize_R, FFTGetSize_C
- FFTFwd_CToC
- FFTInv_CToC
- FFTFwd_RToPack, FFTFwd_RToPerm, FFTFwd_RToCCS
- FFTInv_PackToR, FFTInv_PermToR, FFTInv_CCSToR

FFTInit_R, FFTInit_C

Initializes the FFT specification structure for real and complex signals.

Syntax

Case 1: Operation on real signal

```
IppStatus ippsFFTInit_R_32f(IppsFFTSpec_R_32f** ppFFTSpec, int order, int flag,
IppHintAlgorithm hint, Ipp8u* pSpec, Ipp8u* pSpecBuffer);
```

```
IppStatus ippsFFTInit_R_64f(IppsFFTSpec_R_64f** ppFFTSpec, int order, int flag,
IppHintAlgorithm hint, Ipp8u* pSpec, Ipp8u* pSpecBuffer);
```

Case 2: Operation on complex signal

```
IppStatus ippsFFTInit_C_32f(IppsFFTSpec_C_32f** ppFFTSpec, int order, int flag,
IppHintAlgorithm hint, Ipp8u* pSpec, Ipp8u* pSpecBuffer);
```

```
IppStatus ippsFFTInit_C_64f(IppsFFTSpec_C_64f** ppFFTSpec, int order, int flag,
IppHintAlgorithm hint, Ipp8u* pSpec, Ipp8u* pSpecBuffer);
```

```
IppStatus ippsFFTInit_C_32fc(IppsFFTSpec_C_32fc** ppFFTSpec, int order, int flag,
IppHintAlgorithm hint, Ipp8u* pSpec, Ipp8u* pSpecBuffer);
```

```
IppStatus ippsFFTInit_C_64fc(IppsFFTSpec_C_64fc** ppFFTSpec, int order, int flag,
IppHintAlgorithm hint, Ipp8u* pSpec, Ipp8u* pSpecBuffer);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`order`	FFT order. The input signal length is $N = 2^{order}$.
`flag`	Specifies the result normalization method. The values for the `flag` argument are described in the section Flag and Hint Arguments.
`hint`	This parameter is deprecated. Set the value to `ippAlgHintNone`.

ppFFTSpec	Double pointer to the FFT specification structure to be created.
pSpec	Pointer to the area for the FFT specification structure.
pSpecBuffer	Pointer to the work buffer.

Description

These functions initialize the FFT specification structure ppFFTSpec with the following parameters:

- the transform order. This parameter defines the transform length. Input and output signals are arrays of 2^{order} length.
- the normalization flag
- the specific code hint

Before calling these functions, you need to compute the size of the specification structure and the work buffer (if it is required) using the ippsFFTGetSize_R and ippsFFTGetSize_C functions.

If pSpecBufferSize returned by the ippsFFTGetSize function is equal to zero, the parameter pSpecBuffer can be NULL.

The suffix after the function name indicates the flavors of the FFT functions: ippsFFTInit_C is for complex flavors and ippsFFTInit_R is for real flavors.

Application Notes

The maximum values for signal length are:

Function Flavor	Max Length
C_32fc	268435456 (2^28)
C_64fc	67108864 (2^27)

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsFftOrderErr	Indicates an error when the order value is incorrect.
ippStsFftFlagErr	Indicates an error when the flag value is incorrect.

See Also

FFTGetSize_R FFTGetSize_C Computes sizes of the FFT specification structure and required working buffers.

FFTGetSize_R, FFTGetSize_C

Computes sizes of the FFT specification structure and required working buffers.

Syntax

Case 1: Operation on real signal

```
IppStatus ippsFFTGetSize_R_32f(int order, int flag, IppHintAlgorithm hint, int*
pSpecSize, int* pSpecBufferSize, int* pBufferSize);

IppStatus ippsFFTGetSize_R_64f(int order, int flag, IppHintAlgorithm hint, int*
pSpecSize, int* pSpecBufferSize, int* pBufferSize);
```

Case 2: Operation on complex signal

```
IppStatus ippsFFTGetSize_C_32f(int order, int flag, IppHintAlgorithm hint, int*
pSpecSize, int* pSpecBufferSize, int* pBufferSize);

IppStatus ippsFFTGetSize_C_64f(int order, int flag, IppHintAlgorithm hint, int*
pSpecSize, int* pSpecBufferSize, int* pBufferSize);

IppStatus ippsFFTGetSize_C_32fc(int order, int flag, IppHintAlgorithm hint, int*
pSpecSize, int* pSpecBufferSize, int* pBufferSize);

IppStatus ippsFFTGetSize_C_64fc(int order, int flag, IppHintAlgorithm hint, int*
pSpecSize, int* pSpecBufferSize, int* pBufferSize);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`order`	FFT order. The input signal length is $N = 2^{order}$.
`flag`	Specifies the result normalization method. The values for the `flag` argument are described in Flag and Hint Arguments.
`hint`	This parameter is deprecated. Set the value to `ippAlgHintNone`.
`pSpecSize`	Pointer to the FFT specification structure size value.
`pSpecBufferSize`	Pointer to the buffer size value for FFT initialization function.
`pBufferSize`	Pointer to the size value of the FFT external work buffer.

Description

These functions compute the following:

- the size of the FFT specification structure. Computed value stored in `pSpecSize`.
- the work buffer size for the FFT structure initialization functions `ippsFFTInit_R` and `ippsFFTInit_C`. Computed value is stored in `pSpecBufferSize`.

- the size of the FFT work buffer for the different flavors of `ippsFFTFwd` and `ippsFFTInv`. Computed value is stored in `pBufferSize`.

The suffix after the function name indicates the flavors of the FFT functions: `ippsFFTGetSize_C` is for complex flavors and `ippsFFTGetSize_R` is for real flavors.

Application Notes

The maximum values for signal length are:

Function Flavor	Max Length	Order
`R_32f`	536870912 (2^29)	0..29
`R_64f`	268435456 (2^28)	0..28
`C_32f`	268435456 (2^28)	0..28
`C_64f`	134217728 (2^27)	0..27
`C_32fc`	268435456 (2^28)	0..28
`C_64fc`	134217728 (2^27)	0..27

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when one of the specified pointers is `NULL`.
`ippStsFftOrderErr`	Indicates an error when the `order` value is incorrect.
`ippStsFftFlagErr`	Indicates an error when the `flag` value is incorrect.

See Also

Special Arguments

FFTInit_R FFTInit_C Initializes the FFT specification structure for real and complex signals.

FFTFwd_CToC

Computes the forward fast Fourier transform (FFT) of a complex signal.

Syntax

Case 1: Not-in-place operation on real data type

```
IppStatus ippsFFTFwd_CToC_32f(const Ipp32f* pSrcRe, const Ipp32f* pSrcIm, Ipp32f*
pDstRe, Ipp32f* pDstIm, const IppsFFTSpec_C_32f* pFFTSpec, Ipp8u* pBuffer);
```

```
IppStatus ippsFFTFwd_CToC_64f(const Ipp64f* pSrcRe, const Ipp64f* pSrcIm, Ipp64f*
pDstRe, Ipp64f* pDstIm, const IppsFFTSpec_C_64f* pFFTSpec, Ipp8u* pBuffer);
```

Case 2: Not-in-place operation on complex data type

```
IppStatus ippsFFTFwd_CToC_32fc(const Ipp32fc* pSrc, Ipp32fc* pDst, const
IppsFFTSpec_C_32fc* pFFTSpec, Ipp8u* pBuffer);
```

```
IppStatus ippsFFTFwd_CToC_64fc(const Ipp64fc* pSrc, Ipp64fc* pDst, const
IppsFFTSpec_C_64fc* pFFTSpec, Ipp8u* pBuffer);
```

Case 3: In-place operation on real data type.

```
IppStatus ippsFFTFwd_CToC_32f_I(Ipp32f* pSrcDstRe, Ipp32f* pSrcDstIm, const
IppsFFTSpec_C_32f* pFFTSpec, Ipp8u* pBuffer);
```

```
IppStatus ippsFFTFwd_CToC_64f_I(Ipp64f* pSrcDstRe, Ipp64f* pSrcDstIm, const
IppsFFTSpec_C_64f* pFFTSpec, Ipp8u* pBuffer);
```

Case 4: In-place operation on complex data type.

```
IppStatus ippsFFTFwd_CToC_32fc_I(Ipp32fc* pSrcDst, const IppsFFTSpec_C_32fc* pFFTSpec,
Ipp8u* pBuffer);
```

```
IppStatus ippsFFTFwd_CToC_64fc_I(Ipp64fc* pSrcDst, const IppsFFTSpec_C_64fc* pFFTSpec,
Ipp8u* pBuffer);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pFFTSpec`	Pointer to the FFT specification structure.
`pSrc`	Pointer to the input array containing complex values.
`pDst`	Pointer to the output array containing complex values.
`pSrcRe`	Pointer to the input array containing real parts of the signal.
`pSrcIm`	Pointer to the input array containing imaginary parts of the signal.
`pDstRe`	Pointer to the output array containing real parts of the signal.
`pDstIm`	Pointer to the output array containing imaginary parts of the signal.
`pSrcDst`	Pointer to the input and output array containing complex values (for the in-place operation).
`pSrcDstRe`	Pointer to the input and output array containing real parts of the signal (for the in-place operation).
`pSrcDstIm`	Pointer to the input and output array containing imaginary parts of the signal (for the in-place operation).

pBuffer	Pointer to the external work buffer.

Description

This function computes the forward FFT of a complex signal according to the following pFFTSpec specification parameters: the transform order, the normalization flag, and the specific code hint. Before calling these functions, you need to initialize the FFT specification structure using the ippsFFTInit_C function.

The functions using the complex data type, for example with the 32fc suffixes, process the input complex array pSrc and store the result in pDst. Their in-place flavors use the complex array pSrcDst.

The functions using the real data type and processing complex signals represented by separate real pSrcRe and imaginary pSrcIm parts, for example, with the 32f suffixes, store the result separately in pDstRe and pDstIm, respectively. Their in-place flavors use separate real and imaginary arrays pSrcDstRe and pSrcDstIm, respectively.

Use this function with the external work buffer pBuffer. Once the work buffer is allocated, it can be used for all following calls of the functions computing FFT. The use of an external buffer improves performance significantly, especially for the small size transforms.

The size of the external buffer must be previously computed by the function ippsFFTGetBufSize_C or ippsFFTGetSize_C.

The length of the FFT must be a power of 2.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers with exception of pBuffer is NULL.
ippStsContextMatchErr	Indicates an error when the specification identifier pFFTSpec is incorrect.
ippStsMemAllocErr	Indicates an error when no memory is allocated.

Example

The code example below demonstrates how to use the ippsFFTGetSize, ippsFFTInit, and ippsFFTFwd_CToC functions.

```
void ippsFFT_32fc_example()
{
    Ipp32fc Src[32] = {
        {0.0, 1.0}, {2.0, 3.0}, {4.0, 5.0}, {6.0, 7.0},
        {0.0, 1.0}, {2.0, 3.0}, {4.0, 5.0}, {6.0, 7.0},
        {0.0, 1.0}, {2.0, 3.0}, {4.0, 5.0}, {6.0, 7.0},
        {0.0, 1.0}, {2.0, 3.0}, {4.0, 5.0}, {6.0, 7.0},
        {0.0, 1.0}, {2.0, 3.0}, {4.0, 5.0}, {6.0, 7.0},
        {0.0, 1.0}, {2.0, 3.0}, {4.0, 5.0}, {6.0, 7.0},
        {0.0, 1.0}, {2.0, 3.0}, {4.0, 5.0}, {6.0, 7.0},
        {0.0, 1.0}, {2.0, 3.0}, {4.0, 5.0}, {6.0, 7.0}
    };
    Ipp32fc Dst[32];


    int FFTOrder = 5;
    IppsFFTSpec_C_32fc *pSpec = 0;
```

```
Ipp8u *pMemSpec = 0;
Ipp8u *pMemInit = 0;
Ipp8u *pMemBuffer = 0;


int sizeSpec = 0;
int sizeInit = 0;
int sizeBuffer = 0;


int flag = IPP_FFT_NODIV_BY_ANY;


/// get sizes for required buffers
ippsFFTGetSize_C_32fc(FFTOrder, flag, ippAlgHintNone, &sizeSpec, &sizeInit, &sizeBuffer);


/// allocate memory for required buffers
pMemSpec = (Ipp8u*) ippMalloc(sizeSpec);


if (sizeInit > 0)
{
    pMemInit = (Ipp8u*) ippMalloc(sizeInit);
}


if (sizeBuffer > 0)
{
    pMemBuffer = (Ipp8u*) ippMalloc(sizeBuffer);
}


/// initialize FFT specification structure
ippsFFTInit_C_32fc(&pSpec, FFTOrder, flag, ippAlgHintNone, pMemSpec, pMemInit);


/// free initialization buffer
if (sizeInit > 0)
{
    ippFree(pMemInit);
}


/// perform forward FFT
ippsFFTFwd_CToC_32fc(Src, Dst, pSpec, pMemBuffer);


/// ...


/// free buffers
if (sizeBuffer > 0)
{
    ippFree(pMemBuffer);
}
```

```
    ippFree(pMemSpec);
}
```

Result:

```
Dst ->      {  96.0, 128.0 }, { 0.0, 0.0 }, { 0.0, 0.0 }, { 0.0, 0.0 },
           {   0.0,   0.0 }, { 0.0, 0.0 }, { 0.0, 0.0 }, { 0.0, 0.0 },
            { -64.0,   0.0 }, { 0.0, 0.0 }, { 0.0, 0.0 }, { 0.0, 0.0 },
            {   0.0,   0.0 }, { 0.0, 0.0 }, { 0.0, 0.0 }, { 0.0, 0.0 },
            { -32.0, -32.0 }, { 0.0, 0.0 }, { 0.0, 0.0 }, { 0.0, 0.0 },
            {   0.0,   0.0 }, { 0.0, 0.0 }, { 0.0, 0.0 }, { 0.0, 0.0 },
            {   0.0, -64.0 }, { 0.0, 0.0 }, { 0.0, 0.0 }, { 0.0, 0.0 },
            {   0.0,   0.0 }, { 0.0, 0.0 }, { 0.0, 0.0 }, { 0.0, 0.0 }
```

See Also

Integer Scaling

FFTInit_R FFTInit_C Initializes the FFT specification structure for real and complex signals.

FFTGetSize_R FFTGetSize_C Computes sizes of the FFT specification structure and required working buffers.

FFTInv_CToC

Computes the inverse fast Fourier transform (FFT) of a complex signal.

Syntax

Case 1: Not-in-place operation on real data type

```
IppStatus ippsFFTInv_CToC_32f(const Ipp32f* pSrcRe, const Ipp32f* pSrcIm, Ipp32f*
pDstRe, Ipp32f* pDstIm, const IppsFFTSpec_C_32f* pFFTSpec, Ipp8u* pBuffer);
```

```
IppStatus ippsFFTInv_CToC_64f(const Ipp64f* pSrcRe, const Ipp64f* pSrcIm, Ipp64f*
pDstRe, Ipp64f* pDstIm, const IppsFFTSpec_C_64f* pFFTSpec, Ipp8u* pBuffer);
```

Case 2: Not-in-place operation on complex data type

```
IppStatus ippsFFTInv_CToC_32fc(const Ipp32fc* pSrc, Ipp32fc* pDst, const
IppsFFTSpec_C_32fc* pFFTSpec, Ipp8u* pBuffer);
```

```
IppStatus ippsFFTInv_CToC_64fc(const Ipp64fc* pSrc, Ipp64fc* pDst, const
IppsFFTSpec_C_64fc* pFFTSpec, Ipp8u* pBuffer);
```

Case 3: In-place operation on real data type

```
IppStatus ippsFFTInv_CToC_32f_I(Ipp32f* pSrcDstRe, Ipp32f* pSrcDstIm, const
IppsFFTSpec_C_32f* pFFTSpec, Ipp8u* pBuffer);
```

```
IppStatus ippsFFTInv_CToC_64f_I(Ipp64f* pSrcDstRe, Ipp64f* pSrcDstIm, const
IppsFFTSpec_C_64f* pFFTSpec, Ipp8u* pBuffer);
```

Case 4: In-place operation on complex data type

```
IppStatus ippsFFTInv_CToC_32fc_I(Ipp32fc* pSrcDst, const IppsFFTSpec_C_32fc* pFFTSpec,
Ipp8u* pBuffer);
```

```
IppStatus ippsFFTInv_CToC_64fc_I(Ipp64fc* pSrcDst, const IppsFFTSpec_C_64fc* pFFTSpec,
Ipp8u* pBuffer);
```

Include Files

ipps.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

pFFTSpec	Pointer to the FFT specification structure.
pSrc	Pointer to the input array containing complex values.
pDst	Pointer to the output array containing complex values.
pSrcRe	Pointer to the input array containing real parts of the signal.
pSrcIm	Pointer to the input array containing imaginary parts of the signal.
pDstRe	Pointer to the output array containing real parts of the signal.
pDstIm	Pointer to the output array containing imaginary parts of the signal.
pSrcDst	Pointer to the input and output array containing complex values (for the in-place operation).
pSrcDstRe	Pointer to the input and output array containing real parts of the signal(for the in-place operation).
pSrcDstIm	Pointer to the input and output array containing imaginary parts of the signal(for the in-place operation).
pBuffer	Pointer to the external work buffer.

Description

This function computes the inverse FFT of a complex signal according to the pFFTSpec specification parameters: the transform order, the normalization flag, and the specific code hint. The FFT specification structure must be initialized by the ippsFFTInit_C function beforehand.

The function flavors using the complex data type, for example with the 32fc suffixes, process the input complex array pSrc and store the result in pDst. Their in-place flavors use the complex array pSrcDst.

The function flavors using the real data type and processing complex signals represented by separate real pSrcRe and imaginary pSrcIm parts, for example with the 32f suffixes, store the result separately in pDstRe and pDstIm, respectively. Their in-place flavors uses separate real and imaginary arrays pSrcDstRe and pSrcDstIm, respectively.

Use this function with the external work buffer `pBuffer`. Once the work buffer is allocated, it can be used for all following calls to the functions computing FFT. The use of an external buffer improves performance significantly, especially for the small size transforms.

The size of the external buffer must be previously computed by the ippsFFTGetSize_C function.

The length of the FFT must be a power of 2.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when one of the specified pointers with exception of `pBuffer` is `NULL`.
`ippStsContextMatchErr`	Indicates an error when the specification identifier `pFFTSpec` is incorrect.
`ippStsMemAllocErr`	Indicates an error when no memory is allocated.

FFTFwd_RToPack, *FFTFwd_RToPerm*, *FFTFwd_RToCCS*

Computes the forward or inverse fast Fourier transform (FFT) of a real signal.

Syntax

Case 1: Not-in-place operation, result in Pack Format

```
IppStatus ippsFFTFwd_RToPack_32f(const Ipp32f* pSrc, Ipp32f* pDst, const
IppsFFTSpec_R_32f* pFFTSpec, Ipp8u* pBuffer);
```

```
IppStatus ippsFFTFwd_RToPack_64f(const Ipp64f* pSrc, Ipp64f* pDst, const
IppsFFTSpec_R_64f* pFFTSpec, Ipp8u* pBuffer);
```

Case 2: In-place operation, result in Pack Format.

```
IppStatus ippsFFTFwd_RToPack_32f_I(Ipp32f* pSrcDst, const IppsFFTSpec_R_32f* pFFTSpec,
Ipp8u* pBuffer);
```

```
IppStatus ippsFFTFwd_RToPack_64f_I(Ipp64f* pSrcDst, const IppsFFTSpec_R_64f* pFFTSpec,
Ipp8u* pBuffer);
```

Case 3: Not-in-place operation, result in Perm Format

```
IppStatus ippsFFTFwd_RToPerm_32f(const Ipp32f* pSrc, Ipp32f* pDst, const
IppsFFTSpec_R_32f* pFFTSpec, Ipp8u* pBuffer);
```

```
IppStatus ippsFFTFwd_RToPerm_64f(const Ipp64f* pSrc, Ipp64f* pDst, const
IppsFFTSpec_R_64f* pFFTSpec, Ipp8u* pBuffer);
```

Case 4: In-place operation, result in Perm Format.

```
IppStatus ippsFFTFwd_RToPerm_32f_I(Ipp32f* pSrcDst, const IppsFFTSpec_R_32f* pFFTSpec,
Ipp8u* pBuffer);
```

```
IppStatus ippsFFTFwd_RToPerm_64f_I(Ipp64f* pSrcDst, const IppsFFTSpec_R_64f* pFFTSpec,
Ipp8u* pBuffer);
```

Case 5: Not-in-place operation, result in CCS Format

```
IppStatus ippsFFTFwd_RToCCS_32f(const Ipp32f* pSrc, Ipp32f* pDst, const
IppsFFTSpec_R_32f* pFFTSpec, Ipp8u* pBuffer);
```

```
IppStatus ippsFFTFwd_RToCCS_64f(const Ipp64f* pSrc, Ipp64f* pDst, const
IppsFFTSpec_R_64f* pFFTSpec, Ipp8u* pBuffer);
```

Case 6: In-place operation, result in CCS Format.

```
IppStatus ippsFFTFwd_RToCCS_64f_I(Ipp64f* pSrcDst, const IppsFFTSpec_R_64f* pFFTSpec,
Ipp8u* pBuffer);
```

```
IppStatus ippsFFTFwd_RToCCS_32f_I(Ipp32f* pSrcDst, const IppsFFTSpec_R_32f* pFFTSpec,
Ipp8u* pBuffer);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pFFTSpec`	Pointer to the FFT specification structure.
`pSrc`	Pointer to the input array.
`pDst`	Pointer to the output array containing packed complex values.
`pSrcDst`	Pointer to the input and output arrays for the in-place operation.
`pBuffer`	Pointer to the external work buffer.

Description

These functions compute the forward FFT of a real signal and store the result in Pack, Perm, or CCS packed formats respectively. The transform is performed in accordance with the `pFFTSpec` specification parameters: the transform `order`, the normalization `flag`, and the specific code `hint`. Before calling these functions the FFT specification structure must be initialized by the corresponding flavors of ippsFFTInit_R. The length of the FFT must be a power of 2.

Use these functions with the external work buffer `pBuffer`. Once the work buffer is allocated, it can be used for all following calls to the functions computing FFT. The use of an external buffer improves performance significantly, especially for the small size transforms.

The size of the external buffer must be previously computed by the ippsFFTGetSize_R function.

``ippsFFTFwd_RToPack``. This function computes the forward FFT and stores the result in `Pack` format.

``ippsFFTFwd_RToPerm``. This function computes the forward FFT and stores the result in `Perm` format.

``ippsFFTFwd_RToCCS``. This function computes the forward FFT and stores the result in `CCS` format.

Tables “Arrangement of Forward Fourier Transform Results in Packed Formats - Even Length” and “Forward Fourier transform Result Representation in Packed Formats - Odd Length” show how the output results are arranged in the packed formats.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers with exception of pBuffer is NULL.
ippStsContextMatchErr	Indicates an error when the specification identifier pFFTSpec is incorrect.
ippStsMemAllocErr	Indicates an error when no memory is allocated.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

//   A simple example of performing the forward FFT for processing real data
//     for CCS packed formats and computing magnitude
// implemented with Intel(R) Integrated Performance Primitives (Intel(R) IPP) functions:
//     ippiFFTGetSize_R_32f
//     ippiFFTInit_R_32f
//     ippiFFTFwd_RToPack_32f_C1R
//     ippsMagnitude_32fc

#include <stdio.h>
#include <math.h>
#include "ipp.h"

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define check_sts(st)        \
    if ((st) != ippStsNoErr) \
        goto exitLine; /* Go to Exit if Intel(R) IPP function returned status different from
ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main(void)
{
    int i;
    IppStatus status = ippStsNoErr;
    Ipp32f pSrc[8], pDst[10];        /* Pointers to source/destination images */
    Ipp32f pMagn[4];                 /* Pointer to the computed magnitude values */
```

```
    IppsFFTSpec_R_32f *pSpec = NULL; /* Pointer to FFT pSpec structure */

    Ipp8u *pMemInit = NULL, *pBuffer = NULL, *pSpecMem = NULL; /* Pointer to the work buffers */
    int sizeSpec = 0, sizeInit = 0, sizeBuf = 0;               /* size of FFT pSpec structure,
Init and work buffers */

    check_sts(status = ippsFFTGetSize_R_32f(3, IPP_FFT_DIV_INV_BY_N, ippAlgHintNone, &sizeSpec,
&sizeInit, &sizeBuf));

    /* memory allocation */
    pSpecMem = (Ipp8u *)ippMalloc(sizeSpec);
    pBuffer = (Ipp8u *)ippMalloc(sizeBuf);
    pMemInit = (Ipp8u *)ippMalloc(sizeInit);

    check_sts(status = ippsFFTInit_R_32f(&pSpec, 3, IPP_FFT_DIV_INV_BY_N, ippAlgHintNone,
pSpecMem, pMemInit));

    for (i = 0; i < 8; ++i) {
        pSrc[i] = (float)cos(IPP_2PI * i * 16.0 / 64);
    }

    check_sts(status = ippsFFTFwd_RToCCS_32f(pSrc, pDst, pSpec, pBuffer));

    check_sts(status = ippsMagnitude_32fc((Ipp32fc *)pDst, pMagn, 4));

    printf("FFT Magnitude =  %f  %f  %f  %f\n", pMagn[0], pMagn[1], pMagn[2], pMagn[3]);

    EXIT_MAIN
    ippFree(pMemInit);
    ippFree(pSpec);
    ippFree(pBuffer);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

FFTInv_PackToR, FFTInv_PermToR, FFTInv_CCSToR

Computes the inverse fast Fourier transform (FFT) of a real signal.

Syntax

Case 1: Not-in-place operation on input data in Pack format

```
IppStatus ippsFFTInv_PackToR_32f(const Ipp32f* pSrc, Ipp32f* pDst, const
IppsFFTSpec_R_32f* pFFTSpec, Ipp8u* pBuffer);
```

```
IppStatus ippsFFTInv_PackToR_64f(const Ipp64f* pSrc, Ipp64f* pDst, const
IppsFFTSpec_R_64f* pFFTSpec, Ipp8u* pBuffer);
```

Case 2: In-place operation on input data in Pack format

```
IppStatus ippsFFTInv_PackToR_32f_I(Ipp32f* pSrcDst, const IppsFFTSpec_R_32f* pFFTSpec,
Ipp8u* pBuffer);
```

```
IppStatus ippsFFTInv_PackToR_64f_I(Ipp64f* pSrcDst, const IppsFFTSpec_R_64f* pFFTSpec,
Ipp8u* pBuffer);
```

Case 3: Not-in-place operation on input data in Perm format

```
IppStatus ippsFFTInv_PermToR_32f(const Ipp32f* pSrc, Ipp32f* pDst, const
IppsFFTSpec_R_32f* pFFTSpec, Ipp8u* pBuffer);

IppStatus ippsFFTInv_PermToR_64f(const Ipp64f* pSrc, Ipp64f* pDst, const
IppsFFTSpec_R_64f* pFFTSpec, Ipp8u* pBuffer);
```

Case 4: In-place operation on input data in Perm format

```
IppStatus ippsFFTInv_PermToR_32f_I(Ipp32f* pSrcDst, const IppsFFTSpec_R_32f* pFFTSpec,
Ipp8u* pBuffer);

IppStatus ippsFFTInv_PermToR_64f_I(Ipp64f* pSrcDst, const IppsFFTSpec_R_64f* pFFTSpec,
Ipp8u* pBuffer);
```

Case 5 Not-in-place operation on input data in CCS format

```
IppStatus ippsFFTInv_CCSToR_32f(const Ipp32f* pSrc, Ipp32f* pDst, const
IppsFFTSpec_R_32f* pFFTSpec, Ipp8u* pBuffer);

IppStatus ippsFFTInv_CCSToR_64f(const Ipp64f* pSrc, Ipp64f* pDst, const
IppsFFTSpec_R_64f* pFFTSpec, Ipp8u* pBuffer);
```

Case 6: In-place operation on input data in CCS format

```
IppStatus ippsFFTInv_CCSToR_32f_I(Ipp32f* pSrcDst, const IppsFFTSpec_R_32f* pFFTSpec,
Ipp8u* pBuffer);

IppStatus ippsFFTInv_CCSToR_64f_I(Ipp64f* pSrcDst, const IppsFFTSpec_R_64f* pFFTSpec,
Ipp8u* pBuffer);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pFFTSpec`	Pointer to the FFT specification structure.
`pSrc`	Pointer to the input array.
`pDst`	Pointer to the output array containing real values.
`pSrcDst`	Pointer to the input and output for the in-place operation.
`pBuffer`	Pointer to the external work buffer.

Description

Use these functions with the external work buffer `pBuffer`. Once the work buffer is allocated, it can be used for all following calls to the functions computing FFT. The use of an external buffer improves performance significantly, especially for the small size transforms.

The size of the external buffer must be previously computed by the function ippsFFTGetSize_R.

``ippsFFTInv_PackToR``. This function computes the inverse FFT of input data in `Pack` format.

``**ippsFFTInv_PermToR**``. This function computes the inverse FFT of input data in `Perm` format.

``**ippsFFTInv_CCSToR**``. This function computes the inverse FFT of input data in `CCS` format.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when one of the specified pointers with exception of `pBuffer` is `NULL`.
`ippStsContextMatchErr`	Indicates an error when the specification identifier `pFFTSpec` is incorrect.
`ippStsMemAllocErr`	Indicates an error when no memory is allocated.

Discrete Fourier Transform Functions

The functions described in this section compute the forward and inverse discrete Fourier transform of real and complex signals. The DFT is less efficient than the fast Fourier transform, however the length of the vector transformed by the DFT can be arbitrary.

The `hint` argument, passed to the initialization functions, suggests using special algorithm, faster or more accurate. The `flag` argument specifies the result normalization method. The complex signal can be represented as a single array containing complex elements, or two separate arrays containing real and imaginary parts. The output result of the FFT can be packed in Pack, Perm, or CCS formats.

To use the DFT functions, you should initialize the specification structure which contains such data as tables of twiddle factors. Use the ippsDFTInit_R and ippsDFTInit_C functions to initialize the specification structure both for forward and inverse transforms. Before using these functions, compute the size of the DFT specification structure using the ippsDFTGetSize_R or ippsDFTGetSize_C functions and allocate memory for the structure beforehand.

Speed up the DFT by using an external buffer. The size of the external buffer is returned by the ippsDFTInit_R and ippsDFTInit_C functions.

For more information about the fast computation of the discrete Fourier transform, see [Mit93], section 8-2, *Fast Computation of the DFT*.

A special set of Intel IPP functions provides the so called "out-of-order" DFT of the complex signal. In this case, the elements in frequency domain for both forward and inverse transforms can be re-ordered to speed-up the computation of the transforms. This re-ordering is hidden from the user and can be different in different implementations of the functions. However, reversibility of each pair of functions for forward/inverse transforms is ensured.

- DFTInit_R, DFTInit_C
- DFTGetSize_R, DFTGetSize_C
- DFTFwd_CToC
- DFTFwd_Direct_CToC
- DFTInv_CToC
- DFTInv_Direct_CToC
- DFTFwd_RToPack, DFTFwd_RToPerm, DFTFwd_RToCCS
- DFTInv_PackToR, DFTInv_PermToR, DFTInv_CCSToR

DFTInit_R, DFTInit_C

Initializes the DFT specification structure for real and complex signals.

Syntax

Case 1: Operation on real signal

```
IppStatus ippsDFTInit_R_32f(int length, int flag, IppHintAlgorithm hint,
IppsDFTSpec_R_32f* pDFTSpec, Ipp8u* pMemInit);

IppStatus ippsDFTInit_R_64f(int length, int flag, IppHintAlgorithm hint,
IppsDFTSpec_R_64f* pDFTSpec, Ipp8u* pMemInit);
```

Case 2: Operation on complex signal

```
IppStatus ippsDFTInit_C_32fc(int length, int flag, IppHintAlgorithm hint,
IppsDFTSpec_C_32fc* pDFTSpec, Ipp8u* pMemInit);

IppStatus ippsDFTInit_C_32f(int length, int flag, IppHintAlgorithm hint,
IppsDFTSpec_C_32f* pDFTSpec, Ipp8u* pMemInit);

IppStatus ippsDFTInit_C_64fc(int length, int flag, IppHintAlgorithm hint,
IppsDFTSpec_C_64fc* pDFTSpec, Ipp8u* pMemInit);

IppStatus ippsDFTInit_C_64f(int length, int flag, IppHintAlgorithm hint,
IppsDFTSpec_C_64f* pDFTSpec, Ipp8u* pMemInit);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`length`	Length of the DFT transform.
`flag`	Specifies the result normalization method. The values for the `flag` argument are described in the section Flag and Hint Arguments.
`hint`	This parameter is deprecated. Set the value to `ippAlgHintNone`.
`pDFTSpec`	Pointer to the DFT specification structure to be initialized.
`pMemInit`	Pointer to the temporary work buffer.

Description

These functions initialize the DFT specification structure `pDFTSpec` with the following parameters: the transform `length`, the normalization `flag`, and the specific code `hint`. The `length` argument defines the transform length.

Before calling these functions the memory must be allocated for the DFT specification structure and the temporary work buffer (if it is required). The size of the DFT specification structure and the work buffer must be computed by the functions ippsDFTGetSize_R or ippsDFTGetSize_C.

If the work buffer is not used, the parameter `pMemInit` can be `NULL`. If the work buffer is used, the parameter `pMemInit` cannot be `NULL`. After initialization is done, the temporary work buffer can be freed.

``ippsDFTInit_R`` function initializes the real DFT specification structure.

``ippsDFTInit_C`` function initializes the complex DFT specification structure.

Application Notes

The maximum values for `length` are:

Function Flavor	Max `length`
`C_32fc`	134217727 (2^27 - 1)
`C_64fc`	67108863 (2^26 - 1)

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when one of the specified pointers is `NULL`.
`ippStsFftFlagErr`	Indicates an error when the `flag` value is incorrect.
`ippStsFftOrderErr`	Indicates an error when the memory needed to calculate the `length` value of the DFT transform exceeds the limit.
`ippStsSizeErr`	Indicates an error when `length` is less than, or equal to 0.

DFTGetSize_R, DFTGetSize_C

DFTGetSize_R, DFTGetSize_C

Computes sizes of the DFT work buffer and required working buffers.

Syntax

Case 1: Operation on real signal

```
IppStatus ippsDFTGetSize_R_32f(int length, int flag, IppHintAlgorithm hint, int*
pSizeSpec, int* pSizeInit, int* pSizeBuf);
```

```
IppStatus ippsDFTGetSize_R_64f(int length, int flag, IppHintAlgorithm hint, int*
pSizeSpec, int* pSizeInit, int* pSizeBuf);
```

Case 2: Operation on complex signal

```
IppStatus ippsDFTGetSize_C_32fc(int length, int flag, IppHintAlgorithm hint, int*
pSizeSpec, int* pSizeInit, int* pSizeBuf);
```

```
IppStatus ippsDFTGetSize_C_32f(int length, int flag, IppHintAlgorithm hint, int*
pSizeSpec, int* pSizeInit, int* pSizeBuf);
```

```
IppStatus ippsDFTGetSize_C_64fc(int length, int flag, IppHintAlgorithm hint, int*
pSizeSpec, int* pSizeInit, int* pSizeBuf);
```

```
IppStatus ippsDFTGetSize_C_64f(int length, int flag, IppHintAlgorithm hint, int*
pSizeSpec, int* pSizeInit, int* pSizeBuf);
```

Include Files

ipps.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

length	Length of the DFT transform.
flag	Specifies the result normalization method. The values for the flag argument are described in the section Flag and Hint Arguments.
hint	This parameter is deprecated. Set the value to ippAlgHintNone.
pSizeSpec	Pointer to the DFT specification structure size value.
pSizeInit	Pointer to the buffer size value for DFT initialization functions.
pSizeBuf	Pointer to the size value of the DFT external work buffer.

Description

These functions compute the size of DFT specification structure, the work buffer size for the DFT structure initialization functions ippsDFTInit_R and ippsDFTInit_C, and size of the DFT work buffer for different flavors of ippsDFTFwd and ippsDFTInv. Their values in bytes are stored in pSpecSize, pSizeInit, and pSizeBuf respectively.

ippsDFTGetSize_R function is used for real flavors of the DFT functions.

ippsDFTGetSize_C function is used for complex flavors of the DFT functions.

Application Notes

The maximum values for length are:

Function Flavor	Max length
C_32fc	134217727 (2^27 - 1)
C_64fc	67108863 (2^26 - 1)

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsFftFlagErr	Indicates an error when the flag value is incorrect.

ippStsFftOrderErr	Indicates an error when the memory needed to calculate the length value of the DFT transform exceeds the limit.
ippStsSizeErr	Indicates an error when length is less than, or equal to 0.

DFTInit_R, DFTInit_C

DFTFwd_CToC

Computes the forward discrete Fourier transform of a complex signal.

Syntax

Case 1: Operation on real data type

```
IppStatus ippsDFTFwd_CToC_32f(const Ipp32f* pSrcRe, const Ipp32f* pSrcIm, Ipp32f*
pDstRe, Ipp32f* pDstIm, const IppsDFTSpec_C_32f* pDFTSpec, Ipp8u* pBuffer);
```

```
IppStatus ippsDFTFwd_CToC_64f(const Ipp64f* pSrcRe, const Ipp64f* pSrcIm, Ipp64f*
pDstRe, Ipp64f* pDstIm, const IppsDFTSpec_C_64f* pDFTSpec, Ipp8u* pBuffer);
```

Case 2: Operation on complex data type

```
IppStatus ippsDFTFwd_CToC_32fc(const Ipp32fc* pSrc, Ipp32fc* pDst, const
IppsDFTSpec_C_32fc* pDFTSpec, Ipp8u* pBuffer);
```

```
IppStatus ippsDFTFwd_CToC_64fc(const Ipp64fc* pSrc, Ipp64fc* pDst, const
IppsDFTSpec_C_64fc* pDFTSpec, Ipp8u* pBuffer);
```

Include Files

ipps.h

Domain Dependencies

Flavors declared in ipps.h:

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

pDFTSpec	Pointer to the DFT specification structure.
pSrc	Pointer to the input array containing complex values.
pDst	Pointer to the output array containing complex values.
pSrcRe	Pointer to the input array containing real parts of the signal.
pSrcIm	Pointer to the input array containing imaginary parts of the signal.

pDstRe	Pointer to the output array containing real parts of the signal.
pDstIm	Pointer to the output array containing imaginary parts of the signal.
pBuffer	Pointer to the work buffer.

Description

These functions compute the forward DFT according to the pDFTSpec specification parameters: the transform len, the normalization flag, and the specific code hint.

The functions operating on the complex data type process the input complex array pSrc and store the result in pDst.

The functions operating on the real data type (processing complex signals represented by separate real pSrcRe and imaginary pSrcIm parts) store the result separately in pDstRe and pDstIm, respectively.

Use this function with the external work buffer pBuffer.

Required buffer size must be computed by the corresponding function ippsDFTGetSize_R or ippsDFTGetSize_C prior to using DFT computation functions.

NOTE Data vectors for these functions must be aligned to an appropriate number of bytes that is determined by the SIMD width that is supported by the customer's platform - use ippMalloc function for such alignment.

The forward DFT functionality can be described as follows:

$$X(k) = A \sum_{n=0}^{N-1} x(n) \cdot \exp\left(-j2\pi\frac{kn}{N}\right),$$

where k is the index of elements in the frequency domain, n is the index of elements in the time domain, N is the input signal len, and A is a multiplier defined by flag. Also, x(n) is pSrc[n] and X(k) is pDst[k].

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers with exception of pBuffer is NULL.
ippStsContextMatchErr	Indicates an error when the specification identifier pDFTSpec is incorrect.
ippStsMemAllocErr	Indicates an error when no memory is allocated.
ippStsFftFlagErr	Indicates an error when the flag value is incorrect.

DFTFwd_Direct_CToC

Computes the forward discrete Fourier transform of a complex signal.

Syntax

```
IppStatus ippsDFTFwd_Direct_CToC_16fc(const Ipp16fc *pSrc, Ipp16fc *pDst, int length);
```

Include Files

`ipps.h`

Domain Dependencies

Flavors declared in `ipps.h`:

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pSrc`	Pointer to the input array containing complex values.
`pDst`	Pointer to the output array containing complex values.
`length`	Length of transform

Description

This function computes the forward DFT according to the transform length.

The function operating on the complex data type processes the input complex array `pSrc` and stores the result in `pDst`.

> **NOTE** Data vectors for this function must be aligned to an appropriate number of bytes that is determined by the SIMD width that is supported by the customer's platform - use `ippMalloc` function for such alignment.
> The direct form of DFT uses the static twiddle tables and therefore supports only the limited set of transform lengths. 47 DFT lengths and 11 FFT lengths: 12, 24, 36, 48, 60, 72, 96, 108, 120, 144, 180, 192, 216, 240, 288, 300, 324, 360, 384, 432, 480, 540, 576, 600, 648, 720, 768, 864, 900, 960, 972, 1080, 1152, 1200, 1296, 1440, 1500, 1536, 1620, 1920, 1944, 2160, 2400, 2700, 2916, 3000, 3240 - **DFT**, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096 - **FFT**

The forward DFT functionality can be described as follows:

$$X(k) = A \sum_{n=0}^{N-1} x(n) \cdot \exp\left(-j2\pi\frac{kn}{N}\right),$$

where `k` is the index of elements in the frequency domain, `n` is the index of elements in the time domain, `N` is the input signal `len`, and `A` is a multiplier equal to `1.0` (no normalization). Also, `x(n)` is `pSrc[n]` and `X(k)` is `pDst[k]`.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers with exception of pBuffer is NULL.
ippStsSizeErr	This length is not supported by direct function.

DFTInv_CToC

Computes the inverse discrete Fourier transform of a complex signal.

Syntax

Case 1: Operation on real data type

```
IppStatus ippsDFTInv_CToC_32f(const Ipp32f* pSrcRe, const Ipp32f* pSrcIm, Ipp32f*
pDstRe, Ipp32f* pDstIm, const IppsDFTSpec_C_32f* pDFTSpec, Ipp8u* pBuffer);
```

```
IppStatus ippsDFTInv_CToC_64f(const Ipp64f* pSrcRe, const Ipp64f* pSrcIm, Ipp64f*
pDstRe, Ipp64f* pDstIm, const IppsDFTSpec_C_64f* pDFTSpec, Ipp8u* pBuffer);
```

Case 2: Operation on complex data type

```
IppStatus ippsDFTInv_CToC_32fc(const Ipp32fc* pSrc, Ipp32fc* pDst, const
IppsDFTSpec_C_32fc* pDFTSpec, Ipp8u* pBuffer);
```

```
IppStatus ippsDFTInv_CToC_64fc(const Ipp64fc* pSrc, Ipp64fc* pDst, const
IppsDFTSpec_C_64fc* pDFTSpec, Ipp8u* pBuffer);
```

Include Files

ipps.h

Domain Dependencies

Flavors declared in ipps.h:

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

pDFTSpec	Pointer to the DFT specification structure.
pSrc	Pointer to the input array containing complex values.
pDst	Pointer to the output array containing complex values.
pSrcRe	Pointer to the input array containing real parts of the signal.
pSrcIm	Pointer to the input array containing imaginary parts of the signal.

pDstRe	Pointer to the output array containing real parts of the signal.
pDstIm	Pointer to the output array containing imaginary parts of the signal.
pBuffer	Pointer to the work buffer.

Description

These functions compute the inverse DFT according to the pDFTSpec specification parameters: the transform len, the normalization flag, and the specific code hint.

The functions using the complex data type, for example with 32fc suffixes, process the input complex array pSrc and store the result in pDst.

The functions using the real data type and processing complex signals represented by separate real pSrcRe and imaginary pSrcIm parts, for example with 32f suffixes, store the result separately in pDstRe and pDstIm, respectively.

Use this function with the external work buffer pBuffer. Once the work buffer is allocated, it can be used for all following calls to the functions computing DFT. The use of an external buffer improves performance significantly, especially for the small size transforms.

Required buffer size must be computed by the corresponding function ippsDFTGet BufSize_C prior to using DFT computation functions.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers with exception of pBuffer is NULL.
ippStsContextMatchErr	Indicates an error when the specification identifier pDFTSpec is incorrect.
ippStsMemAllocErr	Indicates an error when no memory is allocated.
ippStsFftFlagErr	Indicates an error when the flag value is incorrect.

DFTInv_Direct_CToC

Computes the inverse discrete Fourier transform of a complex signal.

Syntax

```
IppStatus ippsDFTInv_Direct_CToC_16fc(const Ipp16fc *pSrc, Ipp16fc *pDst, int length);
```

Include Files

ipps.h

Domain Dependencies

Flavors declared in ipps.h:

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

pSrc	Pointer to the input array containing complex values.
pDst	Pointer to the output array containing complex values.
length	Length of transform

Description

This function computes the inverse DFT according to the transform length.

The function operating on the complex data type processes the input complex array pSrc and stores the result in pDst.

NOTE Data vectors for this function must be aligned to an appropriate number of bytes that is determined by the SIMD width that is supported by the customer's platform - use ippMalloc function for such alignment.
The direct form of DFT uses the static twiddle tables and therefore supports only the limited set of transform lengths. 47 DFT lengths and 11 FFT lengths: 12, 24, 36, 48, 60, 72, 96, 108, 120, 144, 180, 192, 216, 240, 288, 300, 324, 360, 384, 432, 480, 540, 576, 600, 648, 720, 768, 864, 900, 960, 972, 1080, 1152, 1200, 1296, 1440, 1500, 1536, 1620, 1920, 1944, 2160, 2400, 2700, 2916, 3000, 3240 - **DFT**, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096 - **FFT**

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers with exception of pBuffer is NULL.
ippStsSizeErr	This length is not supported by direct function.

DFTFwd_RToPack, *DFTFwd_RToPerm*, *DFTFwd_RToCCS*

Computes the forward discrete Fourier transform of a real signal.

Syntax

Case 1: Result in Pack format

```
IppStatus ippsDFTFwd_RToPack_32f(const Ipp32f* pSrc, Ipp32f* pDst, const
IppsDFTSpec_R_32f* pDFTSpec, Ipp8u* pBuffer);
```

```
IppStatus ippsDFTFwd_RToPack_64f(const Ipp64f* pSrc, Ipp64f* pDst, const
IppsDFTSpec_R_64f* pDFTSpec, Ipp8u* pBuffer);
```

supported value for<len>: integer in the range [2, 64].

Case 2: Result in Perm format

```
IppStatus ippsDFTFwd_RToPerm_32f(const Ipp32f* pSrc, Ipp32f* pDst, const
IppsDFTSpec_R_32f* pDFTSpec, Ipp8u* pBuffer);

IppStatus ippsDFTFwd_RToPerm_64f(const Ipp64f* pSrc, Ipp64f* pDst, const
IppsDFTSpec_R_64f* pDFTSpec, Ipp8u* pBuffer);
```

Case 3: Result in CCS format

```
IppStatus ippsDFTFwd_RToCCS_32f(const Ipp32f* pSrc, Ipp32f* pDst, const
IppsDFTSpec_R_32f* pDFTSpec, Ipp8u* pBuffer);

IppStatus ippsDFTFwd_RToCCS_64f(const Ipp64f* pSrc, Ipp64f* pDst, const
IppsDFTSpec_R_64f* pDFTSpec, Ipp8u* pBuffer);
```

Include Files

`ipps.h`

Domain Dependencies

Flavors declared in `ipps.h`:

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pDFTSpec`	Pointer to the DFT specification structure.
`pSrc`	Pointer to the input array containing real values .
`pDst`	Pointer to the output array containing packed complex values.
`pBuffer`	Pointer to the work buffer.

Description

These functions compute the forward DFT of a real signal. The result of the forward transform (that is in the frequency-domain) of real signals is represented in several possible packed formats: Pack, Perm, or CCS. The data can be packed due to the symmetry property of the DFT transform of a real signal. Tables show how the output results are arranged in the packed formats.

These functions compute the forward DFT according to the `pDFTSpec` specification parameters: the transform `len`, the normalization `flag`, and the specific code `hint`.

Use these functions with the external work buffer `pBuffer`. Once the work buffer is allocated, it can be used for all following calls to the functions computing DFT. The use of an external buffer improves performance significantly, especially for the small size transforms.

``ippsDFTFwd_RToPack``. These functions compute the forward DFT and stores the result in `Pack` format.

``ippsDFTFwd_RToPerm``. These functions compute the forward DFT and stores the result in `Perm` format.

``ippsDFTFwd_RToCCS``. These functions compute the forward DFT and stores the result in `CCS` format.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers with exception of pBuffer is NULL.
ippStsContextMatchErr	Indicates an error when the specification identifier pDFTSpec is incorrect.
ippStsMemAllocErr	Indicates an error when no memory is allocated.
ippStsFftFlagErr	Indicates an error when the flag value is incorrect.

DFTInv_PackToR, DFTInv_PermToR, DFTInv_CCSToR

Computes the inverse discrete Fourier transform of a real signal.

Syntax

Case 1: Input data in Pack format

```
IppStatus ippsDFTInv_PackToR_32f(const Ipp32f* pSrc, Ipp32f* pDst, const
IppsDFTSpec_R_32f* pDFTSpec, Ipp8u* pBuffer);
```

```
IppStatus ippsDFTInv_PackToR_64f(const Ipp64f* pSrc, Ipp64f* pDst, const
IppsDFTSpec_R_64f* pDFTSpec, Ipp8u* pBuffer);
```

Case 2: Input data in Perm format

```
IppStatus ippsDFTInv_PermToR_32f(const Ipp32f* pSrc, Ipp32f* pDst, const
IppsDFTSpec_R_32f* pDFTSpec, Ipp8u* pBuffer);
```

```
IppStatus ippsDFTInv_PermToR_64f(const Ipp64f* pSrc, Ipp64f* pDst, const
IppsDFTSpec_R_64f* pDFTSpec, Ipp8u* pBuffer);
```

Case 3: Input data in CCS format

```
IppStatus ippsDFTInv_CCSToR_32f(const Ipp32f* pSrc, Ipp32f* pDst, const
IppsDFTSpec_R_32f* pDFTSpec, Ipp8u* pBuffer);
```

```
IppStatus ippsDFTInv_CCSToR_64f(const Ipp64f* pSrc, Ipp64f* pDst, const
IppsDFTSpec_R_64f* pDFTSpec, Ipp8u* pBuffer);
```

Include Files

16s_Sfs flavors: ipps.h

Domain Dependencies

16s_Sfs:

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

pDFTSpec	Pointer to the DFT specification structure.

pSrc	Pointer to the input array containing packed complex values.
pDst	Pointer to the output array containing real values.
pBuffer	Pointer to the work buffer.

Description

These functions compute the inverse DFT of a real signal. The input data (that is in the frequency-domain) are represented in several possible packed formats: Pack, Perm, or CCS. Tables show how the input data can be represented in the packed formats.

Use these functions with the external work buffer pBuffer. Once the work buffer is allocated, it can be used for all following calls to the functions computing DFT. The use of an external buffer improves performance significantly, especially for the small size transforms.

``ippsDFTInv_PackToR``. This function computes the inverse DFT for input data in Pack format.

``ippsDFTInv_PermToR``. This function computes the inverse DFT for input data in Perm format.

``ippsDFTInv_CCSToR``. This function computes the inverse DFT for input data in CCS format.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers with exception of pBuffer is NULL.
ippStsContextMatchErr	Indicates an error when the specification identifier pDFTSpec is incorrect.
ippStsMemAllocErr	Indicates an error when no memory is allocated.
ippStsFftFlagErr	Indicates an error when the flag value is incorrect.

DFT for a Given Frequency (Goertzel) Functions

The functions described in this section compute a single or a number of the discrete Fourier transforms for a given frequency. Note that the DFT exists only for the following normalized frequencies: 0, 1/ N, 2/ N, … (N – 1)/N, where N is the number of time domain samples. Therefore you must select the frequency value from the above set.

These Intel IPP functions use a Goertzel algorithm [Mit98] and are more efficient when a small number of DFT values is needed.

Some of the functions compute two values, not one. The applications computing several values, for example the dual-tone multi frequency signal detection, work faster with such functions.

- Goertz

Goertz

Computes the discrete Fourier transform for a given frequency for a single signal.

Syntax

```
IppStatus ippsGoertz_32f(const Ipp32f* pSrc, int len, Ipp32fc* pVal, Ipp32f rFreq);
```

```
IppStatus ippsGoertz_64f(const Ipp64f* pSrc, int len, Ipp64fc* pVal, Ipp64f rFreq);

IppStatus ippsGoertz_32fc(const Ipp32fc* pSrc, int len, Ipp32fc* pVal, Ipp32f rFreq);

IppStatus ippsGoertz_64fc(const Ipp64fc* pSrc, int len, Ipp64fc* pVal, Ipp64f rFreq);

IppStatus ippsGoertz_16s_Sfs(const Ipp16s* pSrc, int len, Ipp16sc* pVal, Ipp32f rFreq,
int scaleFactor);

IppStatus ippsGoertz_16sc_Sfs(const Ipp16sc* pSrc, int len, Ipp16sc* pVal, Ipp32f
rFreq, int scaleFactor);
```

Platform-aware functions:

```
IppStatus ippsGoertz_32f_L(const Ipp32f* pSrc, IppSizeL len, Ipp32fc* pVal, Ipp32f
rFreq);

IppStatus ippsGoertz_64f_L(const Ipp64f* pSrc, IppSizeL len, Ipp64fc* pVal, Ipp64f
rFreq);

IppStatus ippsGoertz_32fc_L(const Ipp32fc* pSrc, IppSizeL len, Ipp32fc* pVal, Ipp32f
rFreq);

IppStatus ippsGoertz_64fc_L(const Ipp64fc* pSrc, IppSizeL len, Ipp64fc* pVal, Ipp64f
rFreq);
```

Include Files

`ipps.h`

Flavors with the `_L` suffix: `ipps_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pSrc`	Pointer to the input data vector.
`len`	Number of elements in the vector.
`pVal`	Pointer to the output DFT value.
`rFreq`	Single relative frequency value [0, 1.0).
`scaleFactor`	Scale factor, refer to Integer Scaling.

Description

This function computes a DFT for an input `len – length` vector `pSrc` for a given frequency `rFreq`, and stores the result in `pVal`.

`ippsGoertzQ15`. This function operates with relative frequency in Q15 format. Data in Q15 format are converted to the corresponding float data type that lay in the range [0, 1.0).

The functionality of the Goertzel algorithm can be described as follows:

$$y(k) = \sum_{n=0}^{N-1} x(n) \cdot \exp\left(-j2\pi\frac{kn}{N}\right),$$

where `k/N` is the normalized `rFreq` value for which the DFT is computed.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when one of the specified pointers is `NULL`.
`ippStsRelFreqErr`	Indicates an error when `rFreq` is out of range.
`ippStsSizeErr`	Indicates an error when `len` is less than or equal to 0.

Example

The example below illustrates the use of Goertzel functions for selecting the magnitudes of a given frequency when computing DFTs.

```
IppStatus
goertzel( void ) {
#undef  LEN
#define LEN 100
     IppStatus status;
     Ipp32fc *x = ippsMalloc_32fc( LEN ), y;
     int n;
     ///generate a signal of 60 Hz freq that
     /// is sampled with 400 Hz freq
     for( n=0; n<LEN; ++n) {
        x[n].re =(Ipp32f)sin(IPP_2PI * n * 60 / 400);
        x[n].im = 0;
     }
     status = ippsGoertz_32fc( x, LEN, &y, 60.0f / 400 );
     printf_32fc("goertz =", &y, 1, status );
     ippsFree( x );
     return status;
}
```

Output:

```
goertz =  {0.000090,-50.000008}
Matlab* Analog
    >> N=100;F=60/400;n=0:N-1;x=sin(2*pi*n*F);y=fft(x);n=N*F;y(n+1)
```

Discrete Cosine Transform Functions

This section describes the functions that compute the discrete cosine transform (DCT) of a signal. DCT functions used in the Intel IPP signal processing data-domain implement the modified computation algorithm proposed in [Rao90].

- DCTFwdInit
- DCTInvInit

- DCTFwdGetSize
- DCTInvGetSize
- DCTFwd
- DCTInv

DCTFwdInit

Initializes the forward discrete cosine transform structure.

Syntax

```
IppStatus ippsDCTFwdInit_32f(IppsDCTFwdSpec_32f** ppDCTSpec, int len, IppHintAlgorithm hint, Ipp8u* pSpec, Ipp8u* pSpecBuffer);
```

```
IppStatus ippsDCTFwdInit_64f(IppsDCTFwdSpec_64f** ppDCTSpec, int len, IppHintAlgorithm hint, Ipp8u* pSpec, Ipp8u* pSpecBuffer);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`ppDCTSpec`	Double pointer to the forward DCT specification structure to be created.
`len`	Number of samples in the DCT.
`hint`	This parameter is deprecated. Set the value to `ippAlgHintNone`.
`pSpec`	Pointer to the area for the DCT specification structure.
`pSpecBuffer`	Pointer to the additional work buffer, can be `NULL`.

Description

This function initializes the forward DCT specification structure `ppDCTSpec` with the following parameters: the transform `len`, and the specific code `hint`.

Before calling this function the memory must be allocated for the DCT specification structure and the work buffer. The size of the DCT specification structure and the work buffer must be computed by the function ippsDCTFwdGetSize beforehand.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when one of the specified pointers with exception of `pSpecBuffer` is `NULL`.

ippStsSizeErr	Indicates an error when len is less than or equal to 0.

DCTInvInit

Initializes the inverse discrete cosine transform structure.

Syntax

```
IppStatus ippsDCTInvInit_32f(IppsDCTInvSpec_32f** ppDCTSpec, int len, IppHintAlgorithm hint, Ipp8u* pSpec, Ipp8u* pSpecBuffer);
```

```
IppStatus ippsDCTInvInit_64f(IppsDCTInvSpec_64f** ppDCTSpec, int len, IppHintAlgorithm hint, Ipp8u* pSpec, Ipp8u* pSpecBuffer);
```

Include Files

ipps.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

ppDCTSpec	Double pointer to the inverse DCT specification structure to be created.
len	Number of samples in the DCT.
hint	This parameter is deprecated. Set the value to ippAlgHintNone.
pSpec	Pointer to the area for the DCT specification structure.
pSpecBuffer	Pointer to the work buffer, can be NULL.

Description

This function initializes in the buffer pSpec the inverse DCT specification structure ppDCTSpec with the following parameters: the transform len, and the specific code hint.

Before calling this function the memory must be allocated for the DCT specification structure and the work buffer. The size of the DFT specification structure and the work buffer must be computed by the function ippsDCTInvGetSize.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers with exception of pSpecBuffer is NULL.

ippStsSizeErr	Indicates an error when len is less than or equal to 0.

DCTFwdGetSize

Computes the size of all buffers required for the forward DCT.

Syntax

```
IppStatus ippsDCTFwdGetSize_32f(int len, IppHintAlgorithm hint, int* pSpecSize, int* pSpecBufferSize, int* pBufferSize);
```

```
IppStatus ippsDCTFwdGetSize_64f(int len, IppHintAlgorithm hint, int* pSpecSize, int* pSpecBufferSize, int* pBufferSize);
```

Include Files

ipps.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

len	Number of samples in the DCT.
hint	This parameter is deprecated. Set the value to ippAlgHintNone.
pSpecSize	Pointer to the size of the forward DCT specification structure.
pSpecBufferSize	Pointer to the size of the work buffer for the initialization function.
pBufferSize	Pointer to the size of the forward DCT work buffer.

Description

This function computes the size pSpecSize for the forward DCT structure with the following parameters: the transform len, and the specific code hint. Additionally the function computes the size pSpecBufferSize of the work buffer for the initialization function ippsDCTFwdInit , and the size pBufferSize of the work buffer for the function ippsDCTFwd .

The function ippsDCTFwdGetSize should be called prior to them.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if one of the specified pointers is NULL.

ippStsSizeErr	Indicates an error when len is less than or equal to 0.

DCTInvGetSize

Computes the size of all buffers required for the inverse DCT.

Syntax

```
IppStatus ippsDCTInvGetSize_32f(int len, IppHintAlgorithm hint, int* pSpecSize, int* pSpecBufferSize, int* pBufferSize);
```

```
IppStatus ippsDCTInvGetSize_64f(int len, IppHintAlgorithm hint, int* pSpecSize, int* pSpecBufferSize, int* pBufferSize);
```

Include Files

ipps.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

len	Number of samples in the DCT.
hint	This parameter is deprecated. Set the value to ippAlgHintNone.
pSpecSize	Pointer to the size of the forward DCT specification structure.
pSpecBufferSize	Pointer to the size of the work buffer for the initialization function.
pBufferSize	Pointer to the size of the forward DCT work buffer.

Description

This function computes in bytes the size pSpecSize of the external buffer for the inverse DCT structure with the following parameters: the transform len, and the specific code hint. Additionally the function computes the size pSpecBufferSize of the work buffer for the initialization function ippsDCTInvInit and the size pBufferSize of the work buffer for the function ippsDCTInv.

The function ippsDCTInvGetSize must be called prior to them.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if one of the specified pointers is NULL.

ippStsSizeErr	Indicates an error when len is less than or equal to 0.

DCTFwd

Computes the forward discrete cosine transform of a signal.

Syntax

Case 1: Not-in-place operation

```
IppStatus ippsDCTFwd_32f(const Ipp32f* pSrc, Ipp32f* pDst, const IppsDCTFwdSpec_32f* pDCTSpec, Ipp8u* pBuffer);
IppStatus ippsDCTFwd_64f(const Ipp64f* pSrc, Ipp64f* pDst, const IppsDCTFwdSpec_64f* pDCTSpec, Ipp8u* pBuffer);
```

Case 2: In-place operation

```
IppStatus ippsDCTFwd_32f_I(Ipp32f* pSrcDst, const IppsDCTFwdSpec_32f* pDCTSpec, Ipp8u* pBuffer);
IppStatus ippsDCTFwd_64f_I(Ipp64f* pSrcDst, const IppsDCTFwdSpec_64f* pDCTSpec, Ipp8u* pBuffer);
```

Include Files

ipps.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

pDCTSpec	Pointer to the forward DCT specification structure.
pSrc	Pointer to the source vector.
pDst	Pointer to the destination vector.
pSrcDst	Pointer to the source and destination vector for in-place operations.
pBuffer	Pointer to the external work buffer.

Description

This function computes the forward discrete cosine transform (DCT) of the source signal pSrc (pSrcDst for in-place operations) in accordance with the specification structure pDCTSpec that must be initialized by calling ippsDCTFwdInit beforehand. The result in stored in the pDst (pSrcDst for in-place operations).

If len is a power of 2, the function use an efficient algorithm that is significantly faster than the direct computation of DCT. For other values of len, these functions use the direct formulas given below; however, the symmetry of the cosine function is taken into account, which allows to perform about half of the multiplication operations in the formulas.

In the following definition of DCT, `N=len`,

$$C(k) = \frac{1}{\sqrt{N}} \text{ for } k = 0, \quad C(k) = \frac{\sqrt{2}}{\sqrt{N}} \text{ for } k > 0;$$

`x(n)` is `pSrc[n]` and `y(k)` is `pDst[k]`.

The forward DCT is defined by the formula:

$$y(k) = C(k) \sum_{n=0}^{N-1} x(n) \cdot \cos\frac{(2n+1)\pi k}{2N}$$

Use this function with the external work buffer `pBuffer`. Once the work buffer is allocated, it can be used for all following calls to the functions computing DCT. The use of an external buffer improves performance significantly, especially for the small size transforms.

The size of this buffer must be computed previously using ippsDCTFwdGetSize.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if one of the `pDCTSpec`, `pSrc`, `pDst`, `pSrcDst` pointers is `NULL`.
`ippStsContextMatchErr`	Indicates an error if the specification identifier `pDCTSpec` is incorrect.
`ippStsMemAllocErr`	Indicates an error if memory allocation fails.

DCTInv

Computes the inverse discrete cosine transform of a signal.

Syntax

Case 1: Not-in-place operation

```
IppStatus ippsDCTInv_32f(const Ipp32f* pSrc, Ipp32f* pDst, const IppsDCTInvSpec_32f* pDCTSpec, Ipp8u* pBuffer);
```

```
IppStatus ippsDCTInv_64f(const Ipp64f* pSrc, Ipp64f* pDst, const IppsDCTInvSpec_64f* pDCTSpec, Ipp8u* pBuffer);
```

Case 2: In-place operation

```
IppStatus ippsDCTInv_32f_I(Ipp32f* pSrcDst, const IppsDCTInvSpec_32f* pDCTSpec, Ipp8u* pBuffer);
```

```
IppStatus ippsDCTInv_64f_I(Ipp64f* pSrcDst, const IppsDCTInvSpec_64f* pDCTSpec, Ipp8u* pBuffer);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pDCTSpec`	Pointer to the inverse DCT specification structure.
`pSrc`	Pointer to the source vector.
`pDst`	Pointer to the destination vector.
`pSrcDst`	Pointer to the source and destination vector for in-place operations.
`pBuffer`	Pointer to the external work buffer.

Description

This function computes the inverse discrete cosine transform (DCT) of the source signal `pSrc` (`pSrcDst` for in-place operations) in accordance with the specification structure `pDCTSpec` that must be initialized by calling ippsDCTInvInit beforehand. The result in stored in the `pDst` (`pSrcDst` for in-place operations).

If `len` is a power of 2, the functions use an efficient algorithm that is significantly faster than the direct computation of DCT. For other values of `len`, these functions use the direct formulas given below; however, the symmetry of the cosine function is taken into account, which allows to perform about half of the multiplication operations in the formulas.

In the following definition of DCT, `N=len`,

$$C(k) = \frac{1}{\sqrt{N}} \text{ for } k = 0, \quad C(k) = \frac{\sqrt{2}}{\sqrt{N}} \text{ for } k > 0;$$

`x(n)` is `pDst[n]` and `y(k)` is `pSrc[k]`.

The inverse DCT is defined by the formula:

$$x(n) = \sum_{k=0}^{N-1} C(k)y(k) \cdot \cos\frac{(2n+1)\pi k}{2N}$$

Use this function with the external work buffer `pBuffer`. Once the work buffer is allocated, it can be used for all following calls to the functions computing DCT. The use of an external buffer improves performance significantly, especially for the small size transforms.

The size of this buffer must be computed previously using ippsDCTInvGetSize.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if one of the pDCTSpec, pSrc, pDst, pSrcDst pointers is NULL.
ippStsContextMatchErr	Indicates an error when the specification identifier pDCTSpec is incorrect.
ippStsMemAllocErr	Indicates an error if memory allocation fails.

Hilbert Transform Functions

The functions described in this section compute a discrete-time analytic signal from a real data sequence using the Hilbert transform. The analytic signal is a complex signal whose real part is a replica of the original data, and imaginary part contains the Hilbert transform. That is, the imaginary part is a version of the original real data with a 90 degrees phase shift. The Hilbert transformed data have the same amplitude and frequency content as the original real data, plus the additional phase information.

- HilbertGetSize
- HilbertInit
- Hilbert

HilbertGetSize

Computes the size of the Hilbert transform structure and temporary work buffer.

Syntax

```
IppStatus ippsHilbertGetSize_32f32fc(int length, IppHintAlgorithm hint, int* pSpecSize, int* pBufferSize);
IppStatus ippsHilbertGetSize_64f64fc(int length, IppHintAlgorithm hint, int* pSpecSize, int* pBufferSize);
```

Include Files

ipps.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

length	Number of samples in the Hilbert transform.
hint	Option to select the algorithmic implementation of the transform function (DFT). Possible values for the hint parameter are described in IppHintAlgorithm enum Type.

pSpecSize	Pointer to the size, in bytes, of the Hilbert context structure.
pBufferSize	Pointer to the size, in bytes, of the work buffer.

Description

This function computes the size of the Hilbert specification structure and temporary work buffer for the ippsHilbert function.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when length is less than 1.

See Also

Hilbert MODIFIED API. Computes an analytic signal using the Hilbert transform.

HilbertInit

Initializes the Hilbert transform structure.

Syntax

```
IppStatus ippsHilbertInit_32f32fc(int length, IppHintAlgorithm hint, IppsHilbertSpec* pSpec, Ipp8u* pBuffer);
```

```
IppStatus ippsHilbertInit_64f64fc(int length, IppHintAlgorithm hint, IppsHilbertSpec* pSpec, Ipp8u* pBuffer);
```

Include Files

ipps.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

length	Number of samples in the Hilbert transform.
hint	Option to select the algorithmic implementation of the transform function (DFT). The values for the hint argument are described in IppHintAlgorithm enum Type.
pSpec	Pointer to the Hilbert context structure.
pBuffer	Pointer to the work buffer.

Description

This function initializes the Hilbert specification structure `pSpec` with the following parameters: the length of the transform `length`, and the specific code indicator `hint`. Call this function before using the Hilbert transform function ippsHilbert.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error when `length` is less than, or equal to 0.

See Also

Hilbert MODIFIED API. Computes an analytic signal using the Hilbert transform.

Hilbert

MODIFIED API. Computes an analytic signal using the Hilbert transform.

Syntax

```
IppStatus ippsHilbert_32f32fc(const Ipp32f* pSrc, Ipp32fc* pDst, IppsHilbertSpec* pSpec, Ipp8u* pBuffer);
```

```
IppStatus ippsHilbert_64f64fc(const Ipp64f* pSrc, Ipp64fc* pDst, IppsHilbertSpec* pSpec, Ipp8u* pBuffer);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pSpec`	Pointer to the Hilbert specification structure.
`pSrc`	Pointer to the vector containing original real data.
`pDst`	Pointer to the output array containing complex data.
`pBuffer`	Pointer to the work buffer.

Description

NOTE
Important
The API of this function has been modified in Intel IPP 9.0 release.

The `ippsHilbert` function computes a complex analytic signal `pDst`, which contains the original real signal `pSrc` as its real part and computed Hilbert transform as its imaginary part. The Hilbert transform is performed according to the `pSpec` specification parameters: the number of samples `len`, and the specific code `hint`. The input data is zero-padded or truncated to the size of `len` as appropriate.

Before using this function, you need to compute the size of the work buffer and specification structure using the HilbertGetSize function and initialize the structure using HilbertInit.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when one of the specified pointers is `NULL`.
`ippStsContextMatchErr`	Indicates an error when the specification identifier `pSpec` is incorrect.

Example

The example below shows how to initialize the specification structure and use the function `ippsHilbert_32f32fc`.

```
IppStatus hilbert( )
{
   Ipp32f x[10];
   Ipp32fc y[10];
   int n;
   IppStatus status;
   IppsHilbertSpec* pSpec;
   Ipp8u* pBuffer;
   int sizeSpec, sizeBuf;


   status = ippsHilbertGetSize_32f32fc(10, ippAlgHintNone, &sizeSpec, &sizeBuf);


   pSpec = (IppsHilbertSpec*)ippMalloc(sizeSpec);
   pBuffer = (Ipp8u*)ippMalloc(sizeBuf);


   status = ippsHilbertInit_32f32fc(10, ippAlgHintNone, pSpec, pBuffer);


   for (n = 0; n < 10; n ++) {
      x[n] = (Ipp32f)cos(IPP_2PI * n * 2 / 9);
   }


   status = ippsHilbert_32f32fc(x, y, pSpec, pBuffer);
```

```
    ippsMagnitude_32fc((Ipp32fc*)y, x, 5);


    ippFree(pSpec);
    ippFree(pBuffer);


    printf_32f("hilbert magn =", x, 5, status);
    return status;
}
```

Output:

```
hilbert magn = 1.0944 1.1214 1.0413 0.9707 0.9839
Matlab* Analog:
>> n=0:9; x=cos(2*pi*n*2/9); y=abs(hilbert(x)); y(1:5)
```

See Also

HilbertGetSize Computes the size of the Hilbert transform structure and temporary work buffer.

HilbertInit Initializes the Hilbert transform structure.

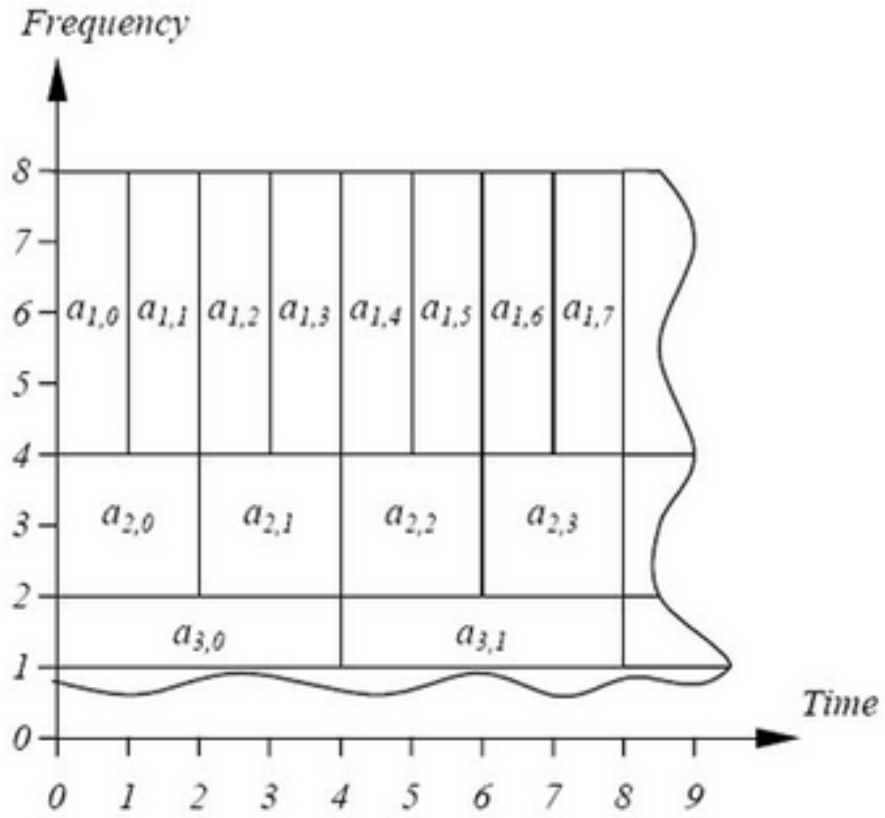
Wavelet Transform Functions

This section describes the wavelet transform functions implemented in Intel IPP.

In signal processing, signals can represented in both frequency and time-frequency domains. In many cases the wavelet transforms become an alternative to short time Fourier transforms.

The discrete wavelet signal can be considered as a set of the coefficients $a_{i,k}$ with two indices, one of which is a “frequency” characteristic and the other is a time localization. The coefficient value corresponds to the localized wave amplitude or to one of basis transform functions. The “frequency” index shows the time scale of the localized wave. Function bases originated from one local wave by decreasing the wave by 2^n in time are the most widely used. Such transforms can be used for building very efficient implementations called fast wavelet transforms by analogy with fast Fourier transforms. Figure “Wavelet Decomposition Coefficients in Time-Frequency Domain” shows how the time and frequency plane is divided into areas that correspond to the local wave amplitudes. This kind of transforms is implemented in Intel IPP and referred to as the discrete wavelet transform (DWT).

Wavelet Decomposition Coefficients in Time-Frequency Domain



The DWT is one of the wavelet analysis methods that stem from the basis functions related to the scale factor 2. Thus, there is a basic common element shared by the DWT and the other packet analysis methods.

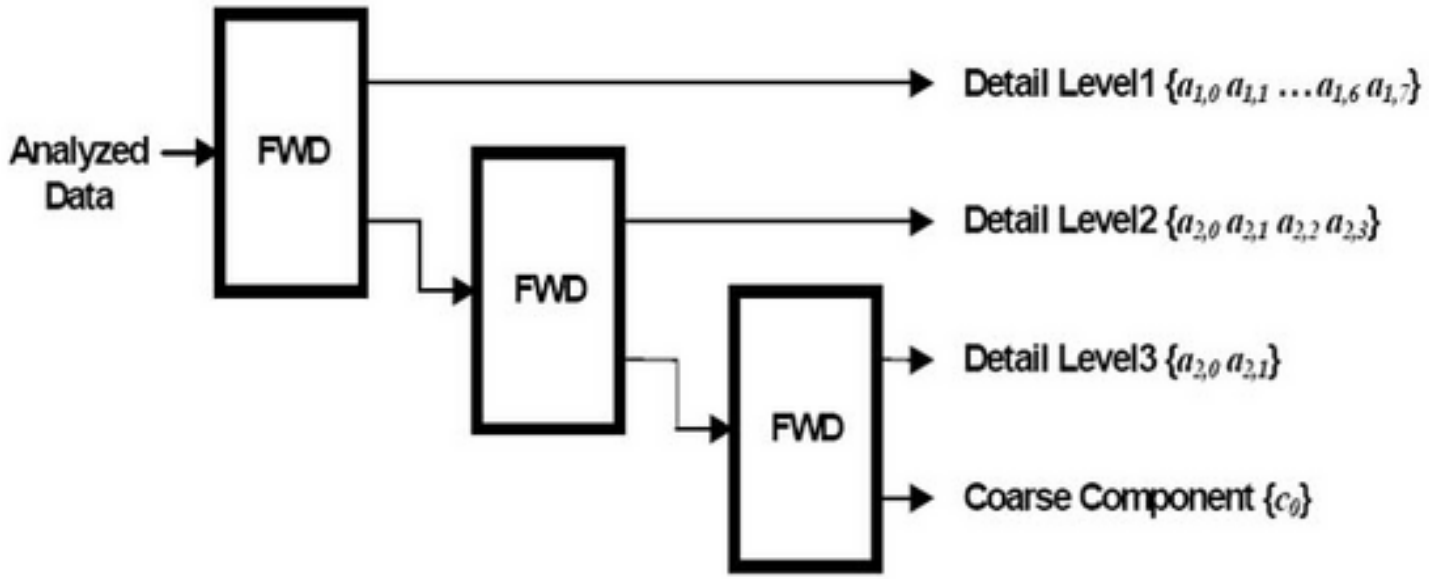
Likewise another basic element for signal reconstruction or synthesis can be defined, called the one-level inverse DWT. Figure “Three-Level Discrete Wavelet Decomposition” shows the diagram of the forward DWT which allows to switch to time-frequency representation shown in Figure above. The diagram includes three levels of decomposition. Figure “Three-Level Discrete Wavelet Reconstruction” shows the corresponding procedure of signal reconstruction based on the elementary one-level inverse transform.

The implementation of discrete multi-scale transforms is based on the use of interpolation and decimation filters with the resampling factor 2. The basis of the multi-scale signal decomposition and reconstruction functions uniquely defines the filter parameters. The Intel IPP multi-scale transform functions use filters with finite impulse response.

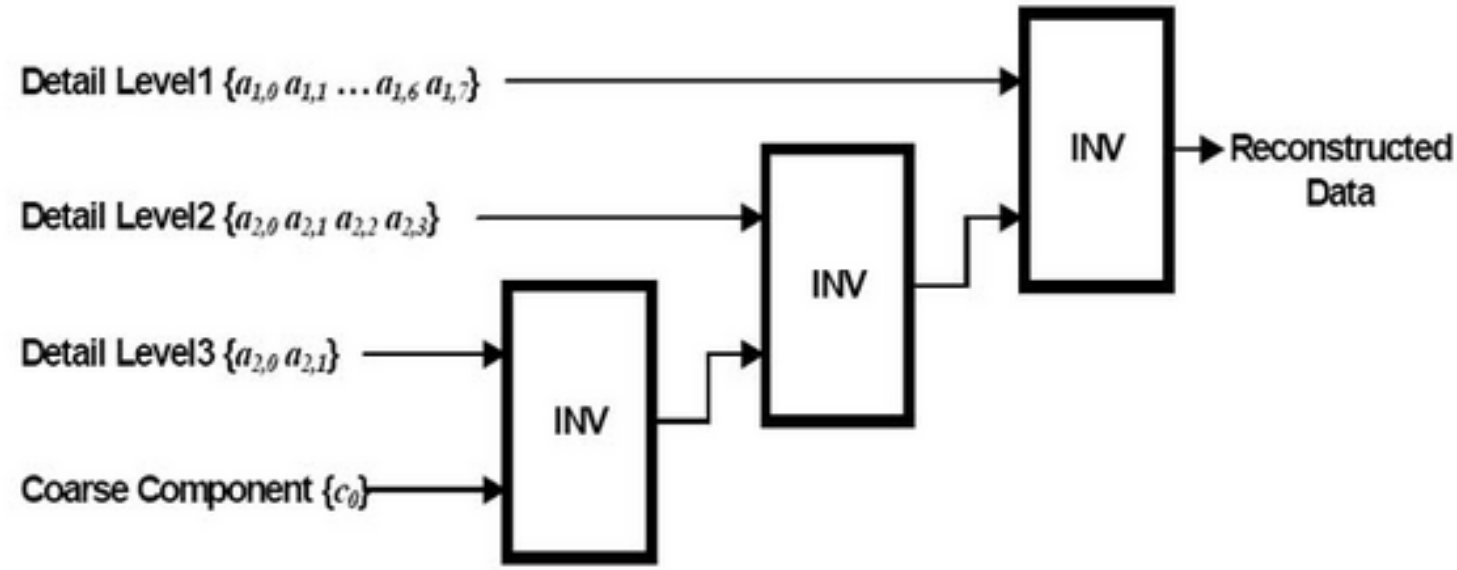
The Primitives contains two sets of functions.

- Transforms designed for fixed filter banks. These transforms yield the highest performance.
- Transforms that enable the user to work with arbitrary filters. These functions use effective polyphase filtration algorithms. The transform interface gives the option of processing the data in blocks, including in real-time applications.

Three-Level Discrete Wavelet Decomposition



Three-Level Discrete Wavelet Reconstruction



- Transforms for Fixed Filter Banks
- Transforms for User Filter Banks
- Wavelet Transforms Example

Transforms for Fixed Filter Banks

This section describes the functions that perform forward or inverse wavelet transforms for fixed filter banks.

- WTHaarFwd, WTHaarInv

WTHaarFwd, WTHaarInv

Performs forward or inverse single-level discrete wavelet Haar transforms.

Syntax

Case 1: Forward transform

```
IppStatus ippsWTHaarFwd_32f(const Ipp32f* pSrc, int len, Ipp32f* pDstLow, Ipp32f* pDstHigh);
IppStatus ippsWTHaarFwd_64f(const Ipp64f* pSrc, int len, Ipp64f* pDstLow, Ipp64f* pDstHigh);
IppStatus ippsWTHaarFwd_16s_Sfs(const Ipp16s* pSrc, int len, Ipp16s* pDstLow, Ipp16s* pDstHigh, int scaleFactor);
```

Case 2: Inverse transform

```
IppStatus ippsWTHaarInv_32f(const Ipp32f* pSrcLow, const Ipp32f* pSrcHigh, Ipp32f* pDst, int len);
IppStatus ippsWTHaarInv_64f(const Ipp64f* pSrcLow, const Ipp64f* pSrcHigh, Ipp64f* pDst, int len);
IppStatus ippsWTHaarInv_16s_Sfs(const Ipp16s* pSrcLow, const Ipp16s* pSrcHigh, Ipp16s* pDst, int len, int scaleFactor);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pSrc`	Pointer to the source vector for forward transform.
`len`	Number of elements in the vector.
`pDstLow`	Pointer to the array with the coarse “low frequency” components of the output for forward transform.
`pDstHigh`	Pointer to the array with the detail “high frequency” components of the output for forward transform.
`pSrcLow`	Pointer to the array with the coarse “low frequency” components of the input for inverse transform.
`pSrcHigh`	Pointer to the array with the detail “high frequency” components of the input for inverse transform.
`pDst`	Pointer to the array with the output signal for inverse transform.
`scaleFactor`	Scale factor, refer to Integer Scaling.

Description

These functions perform forward and inverse single-level discrete Haar transforms. These transforms are orthogonal and reconstruct the original signal perfectly.

The forward transform can be considered as wavelet signal decomposition with lowpass decimation filter coefficients {1/2, 1/2} and highpass decimation filter coefficients {1/2, -1/2}.

The inverse transform is represented as a wavelet signal reconstruction with lowpass interpolation filter coefficients {1, 1} and highpass interpolation filter coefficients {-1, 1}.

The decomposition filter coefficients are frequency response normalized to provide the same value range for both input and output signals. Thus, the amplitude of the low pass filter frequency response is 1 for zero-valued frequency, and the amplitude of the high pass filter frequency response is also 1 for the frequency value near to 0.5.

As the absolute values of the interpolation filter coefficients are equal to 1, the reconstruction of the signal requires few operations. It is well suited for usage in data compression applications. As the decomposition filter coefficients are powers of 2, the integer functions perform lossless decomposition with the `scaleFactor` value equal to -1. To avoid saturation, use higher-precision data types.

Note that the filter coefficients can be power spectral response normalized, see [Strang96] for more information. Thus, the decomposition filter coefficients are $\{2^{-1/2}, 2^{-1/2}\}$ and $\{2^{-1/2}, -2^{-1/2}\}$; accordingly; the reconstruction filter coefficients are $\{2^{-1/2}, 2^{-1/2}\}$ and $\{-2^{-1/2}, 2^{-1/2}\}$.

In the following definition of the forward single-level discrete Haar transform, `N = len`. The coarse "low-frequency" component `c(k)` is `pDstLow[k]` and the detail "high-frequency" component `d(k)` is `pDstHigh[k]`; also `x (2 k)` and `x (2 k + 1)` are even and odd values of the input signal `pSrc`, respectively.

`c(k) = (x (2 k) + x (2 k + 1 ))/2`

`d(k) = (x (2 k + 1) - x (2 k))/2`

In the inverse direction, `N=len`. The coarse "low-frequency" component `c(k)` is `pSrcLow[k]` and the detail "high-frequency" component `d(k)` is `pSrcHigh[k]`; also `y (2 i)` and `y (2 i + 1)` are even and odd values of the output signal `pDst`, respectively.

`y (2 i) = c(i)-d(i)`

`y (2 i + 1) = c(i)+d(i)`

For even length `N`, `0≤ k < N / 2` and `0≤ i < N / 2`. Also, "low-frequency" and "high-frequency" components are of size `N / 2` for both original and reconstructed signals. The total length of components is equal to the signal length `N`.

In case of odd length `N`, the vector is considered as a vector of the extended length `N + 1` whose two last elements are equal to each other `x[N] = x[N – 1]`. The last elements of the coarse and detail components of the decomposed signal are defined as follows:

`c((N + 1 )/2 - 1) = x(N – 1)`

`d((N + 1)/2 - 1) = 0`

Correspondingly, the last element of the reconstructed signal is defined as:

`y(N) = y(N – 1) = c((N + 1)/2 - 1)`

For odd length `N`, `0≤ k < N – 1)/2` and `0≤ i < N - 1)/2` , assuming that `c((N + 1)/2 - 1) = x(N – 1)` and `y(N – 1) = c((N + 1)/2 - 1)`. The "low-frequency" component is of size `(N + 1)/2`. The "high-frequency" component is of size `(N – 1)/2`, because the last element `d((N + 1)/2 - 1)` is always equal to 0. The total length of components is also `N`.

Such an approach applies continuation of boundaries for filters having the symmetry properties, see [Bris94].

When performing block mode transforms, take into consideration that for decomposition and reconstruction of even-length signals no extrapolations at the boundaries is used. In case of odd-length signals, a symmetric continuation of the signal boundary with the last point replica is applied.

When it is necessary to have a continuous set of output blocks, all the input blocks are to be of even length, besides the last one (which can be either of odd or even length). Thus, if the whole amount of elements is odd, only the last block can be of odd length.

``ippsWTHaarFwd``. Thise function performs the forward single-level discrete Haar transform of a `len – length` signal `pSrc` and stores the decomposed coarse "low-frequency" components in `pDstLow`, and the detail "high-frequency" components in `pDstHigh`.

``ippsWTHaarInv``. This function performs the inverse single-level discrete Haar transform of the coarse "low-frequency" components `pSrcLow` and detail "high-frequency" components `pSrcHigh`, and stores the reconstructed signal in the `len – length` vector `pDst`.

For more information on wavelet transforms see [Strang96] and [Bris94].

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when the `pDst` or `pSrc` pointer is `NULL`.
`ippStsSizeErr`	Indicates an error when `len` is less than 4 for the function `ippsWinBlackmanOpt` and less than 3 for all other functions of the family.

Example

The example below illustrates the use of the function `ippsWTHaarFwd_32f`.

```
IppStatus wthaar(void)
 {     Ipp32f x[8], lo[4], hi[4];
       IppStatus status;
       ippsSet_32f(7, x, 8);
                   --x[4];
       status = ippsWTHaarFwd_32f(x, 8, lo, hi);
       printf_32f("WT Haar low  =", lo, 4, status);
       printf_32f("WT Haar high =", hi, 4, status);
       return status;  }
```

Output:

```
WT Haar low  =  7.000000 7.000000 6.500000 7.000000
     WT Haar high =  0.000000 0.000000 0.500000 0.000000
```

Transforms for User Filter Banks

This section describes the functions that perform forward or inverse wavelet transforms for user filter banks.

- WTFwdGetSize, WTInvGetSize
- WTFwdInit, WTInvInit
- WTFwd
- WTFwdSetDlyLine, WTFwdGetDlyLine
- WTInv
- WTInvSetDlyLine, WTInvGetDlyLine

WTFwdGetSize, WTInvGetSize

Compute the size of the wavelet transform state structures.

Syntax

```
IppStatus ippsWTFwdGetSize(IppDataType srcType, int lenLow, int offsLow, int lenHigh,
int offsHigh, int* pStateSize);
```

```
IppStatus ippsWTInvGetSize(IppDataType dstType, int lenLow, int offsLow, int lenHigh,
int offsHigh, int* pStateSize);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`srcType`, `dstType`	Data type of the transformed vector.
`lenLow`	Length of the low-pass filter.
`offsLow`	Input delay of the low-pass filter.
`lenHigh`	Length of the high-pass filter.
`offsHigh`	Input delay of the high-pass filter.
`pStateSize`	Pointer to the size of the `ippsWTFwd` or `ippsWTInv` state structure, in bytes.

Description

The `ippsWTFwd` and `ippsWTInv` functions compute the size of the `ippsWTFwd` and `ippsWTInv` state structures, in bytes, for the ippsWTFwdInit and ippsWTInvInit functions, respectively.

For an example on how to use these functions, refer to Wavelet Transforms Example.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error when `lenLow` or `lenHigh` is less than, or equal to zero.
`ippStsWtOffsetErr`	Indicates an error when the filter delay `offsLow` or `offsHigh` is less than -1.

See Also

WTFwdInit, WTInvInt Initialize the wavelet transform state structures.

WTFwd Computes the forward wavelet transform.

WTInv Computes the inverse wavelet transform.

Wavelet Transforms Example

WTFwdInit, WTInvInit

Initialize the wavelet transform state structures.

Syntax

Case 1: Forward transform

```
IppStatus ippsWTFwdInit_32f(IppsWTFwdState_32f* pState, const Ipp32f* pTapsLow, int lenLow, int offsLow, const Ipp32f* pTapsHigh, int lenHigh, int offsHigh);
IppStatus ippsWTFwdInit_8u32f(IppsWTFwdState_8u32f* pState, const Ipp32f* pTapsLow, int lenLow, int offsLow, const Ipp32f* pTapsHigh, int lenHigh, int offsHigh);
IppStatus ippsWTFwdInit_16s32f(IppsWTFwdState_16s32f* pState, const Ipp32f* pTapsLow, int lenLow, int offsLow, const Ipp32f* pTapsHigh, int lenHigh, int offsHigh);
IppStatus ippsWTFwdInit_16u32f(IppsWTFwdState_16u32f* pState, const Ipp32f* pTapsLow, int lenLow, int offsLow, const Ipp32f* pTapsHigh, int lenHigh, int offsHigh);
```

Case 2: Inverse transform

```
IppStatus ippsWTInvInit_32f(IppsWTInvState_32f* pState, const Ipp32f* pTapsLow, int lenLow, int offsLow, const Ipp32f* pTapsHigh, int lenHigh, int offsHigh);
IppStatus ippsWTInvInit_32f8u(IppsWTInvState_32f8u* pState, const Ipp32f* pTapsLow, int lenLow, int offsLow, const Ipp32f* pTapsHigh, int lenHigh, int offsHigh);
IppStatus ippsWTInvInit_32f16s(IppsWTInvState_32f16s* pState, const Ipp32f* pTapsLow, int lenLow, int offsLow, const Ipp32f* pTapsHigh, int lenHigh, int offsHigh);
IppStatus ippsWTInvInit_32f16u(IppsWTInvState_32f16u* pState, const Ipp32f* pTapsLow, int lenLow, int offsLow, const Ipp32f* pTapsHigh, int lenHigh, int offsHigh);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pState`	Pointer to the initialized forward or inverse wavelet transform state structure.
`pTapsLow`	Pointer to the vector of low-pass filter taps.
`lenLow`	Number of taps in the low-pass filter.
`offsLow`	Input delay (offset) of the low-pass filter.
`pTapsHigh`	Pointer to the vector of high-pass filter taps.

lenHigh	Number of taps in the high-pass filter.
offsHigh	Input delay (offset) of the high-pass filter.

Description

The ippsWTFwdInit and ippsWTInvInit functions initialize the forward and inverse wavelet transform state structures, respectively, with the following parameters: the low-pass and high-pass filter taps pTapsLow and pTapsHigh, lengths lenLow and lenHigh, input additional delays offsLow and offsHigh.

Application Notes

These functions initialize the wavelet state structure and return the pState pointer to it. The initialization procedures are implemented separately for forward and inverse transforms. To perform both forward and inverse wavelet transforms, create two separate state structures. In general, the meanings of initialization parameters of forward and inverse transforms are similar. Each function has parameters describing of a pair of filters. The forward transform uses the taps pTapsHigh and pTapsLow, and the lengths lenHigh and lenLow of a pair of analysis filters. The inverse transform uses the taps pTapsHigh and pTapsLow, and the lengths lenHigh and lenLow of a pair of synthesis filters. You can also specify an additional delay offsLow and offsHigh for each filter. With the adjustable values of delays you can synchronize:

- Group of delays for high-pass and low-pass filters
- Delays between data of different levels in multilevel decomposition and reconstruction algorithms

For more information about using these parameters, see descriptions of the ippsWTFwd and ippsWTInv functions. The minimum allowed value of the additional delay for the forward transform is -1. For the inverse transform the delay values must be greater than, or equal to 0. See descriptions of the ippsWTFwd and ippsWTInv functions for an example showing how to choose additional delay values. The initialization functions copy filter taps into the state structure pState. So all the memory referred to with the pointers can be freed or modified after the functions finished operating. In case of the memory shortage, the function sets a zero pointer to the structure.

Boundaries extrapolation. Typically, reversible wavelet transforms of a bounded signal require data extrapolation towards one or both sides. All internal delay lines are set to zero at the initialization stage. To set a non-zero signal prehistory, call the function ippsWTFwdSetDlyLine. When processed an entire limited data set, data extrapolation may be performed both towards the start and the end of the data vector. For that, the source data and their initial extrapolation are used to form the delay line, the rest of the signal is subdivided into the main block and the signal end. The signal end data and their extrapolation are used to form the last block.

For an example on how to use these functions, refer to Wavelet Transforms Example.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when lenLow or lenHigh is less than, or equal to 0.
ippStsWtOffsetErr	Indicates an error when the filter delay offsLow or offsHigh is less than -1.

See Also

WTFwd Computes the forward wavelet transform.

Wavelet Transforms Example

WTFwd

Computes the forward wavelet transform.

Syntax

```
IppStatus ippsWTFwd_32f(const Ipp32f* pSrc, Ipp32f* pDstLow, Ipp32f* pDstHigh, int dstLen, IppsWTFwdState_32f* pState);
IppStatus ippsWTFwd_8u32f(const Ipp8u* pSrc, Ipp32f* pDstLow, Ipp32f* pDstHigh, int dstLen, IppsWTFwdState_8u32f* pState);
IppStatus ippsWTFwd_16s32f(const Ipp16s* pSrc, Ipp32f* pDstLow, Ipp32f* pDstHigh, int dstLen, IppsWTFwdState_16s32f* pState);
IppStatus ippsWTFwd_16u32f(const Ipp16u* pSrc, Ipp32f* pDstLow, Ipp32f* pDstHigh, int dstLen, IppsWTFwdState_16u32f* pState);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pSrc`	Pointer to the vector which holds the input signal for decomposition.
`pDstLow`	Pointer to the vector which holds output coarse “low frequency” components.
`pDstHigh`	Pointer to the vector which holds output detail “high frequency” components.
`dstLen`	Number of elements in the vectors `pDstHigh` and `pDstLow`.
`pState`	Pointer to the state structure.

Description

This function computes the forward wavelet transform. The function transforms the (2* `dstLen`)-length source data block `pSrc` into “low frequency” components `pDstLow` and “high frequency” components `pDstHigh`. The transform parameters are specified in the state structure `pState`.

Before using this function, you need to compute the size of the state structure and work buffer using the WTFwdGetSize_ WTInvGetSize function, and initialize the structure using WTFwdInit WTInvInit.

For an example on how to use this function, refer to Wavelet Transforms Example.

Application Notes

These functions perform the one-level forward discrete multi-scale transform. An equivalent transform diagram is shown in the Figure below. The input signal is divided into the "low frequency" and "high frequency" components. The transfer characteristics of filters are defined by the coefficients set at the initialization stage. The functions are designed for the block processing of data; the transform state structure `pState` contains all needed filter delay lines. Besides these main delay lines each function has an additional delay line for each filter. Adjustable extra delay lines help synchronize group delay times of both highpass and lowpass filters. Moreover, in multilevel systems of signal decomposition delays between different decomposition levels may also be synchronized.

Input and output data block lengths. The functions are designed to decompose signal blocks of even length, therefore, these functions have one parameter only, that is the length of input components. The length of the input block must be double the size of each component.

Filter group delays synchronization. Some applications may require synchronization of highpass and lowpass filter time responses. A typical example of this synchronization is synchronizing symmetrical filters of different length.

Below follows an example of bi orthogonal set of spline filters of respective length of 6 and 2:

```
static const float decLow[6] = {     -6.25000000e-002f,      6.25000000e-002f,
5.00000000e-001f,      5.00000000e-001f,      6.25000000e-002f,      -6.25000000e-002f };
```

```
static const float decHigh[2] = {     -5.00000000e-001f,      5.00000000e-001f };
```

In this case the lowpass filter gives a delay two samples longer than the highpass filter, which is exactly what the difference between additional initialization function delays should be. The following values must be selected to ensure minimum common signal delay, `offsLow=-1`, `offsHigh=-1+ 2 = 1`. In this case the group times of filter delays are balanced by additional delays. The total delay time is equal to the lowpass filter group delay which has the value of two samples in the decomposition stage in the original signal time frame.

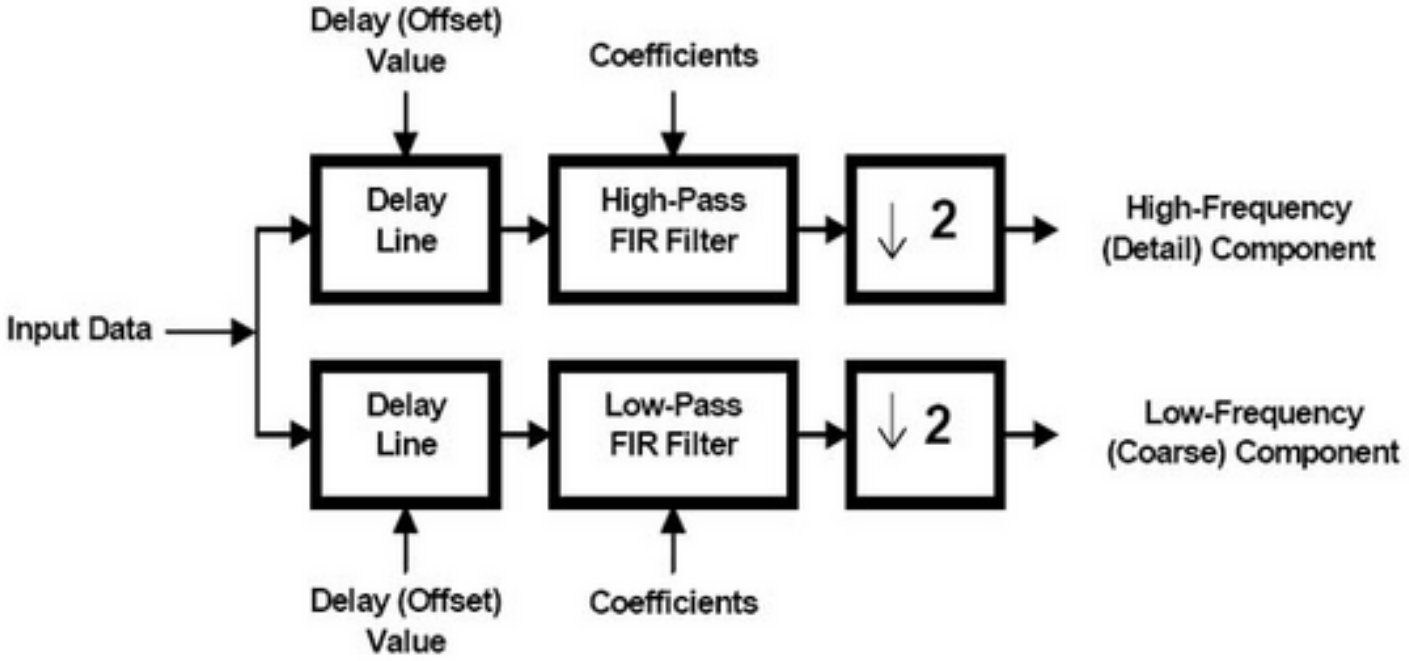
> **NOTE** Biorthogonal and orthogonal filter banks are distinguished by one specific peculiarity, that is, forward transform additional delays must be uniformly even for faultless signal reconstruction.

Multilevel decomposition algorithm. The implementation of multilevel decomposition algorithms may require synchronization of signal delays across components of different levels.

This is illustrated in the example of the three-level decomposition shown in Figure "Three-Level Discrete Wavelet Decomposition". Assume that for transformation the biorthogonal set of spline filters with respective filter length of 6 and 2 is used. Since group delay definitely needs to be synchronized, for the last level select additional filter delays `offsLow3 = -1`, `offsHigh3 = 1`. Total delay at the last stage of decomposition for this set of filters is two samples. This value corresponds to the time scale of the input of the last stage of decomposition. In order to ensure an equivalent delay of the "detail" part on the second level, the delay must be increased by 2*2 samples. Respective values of additional delays for the second level is equal to `offsLow2 = -1`, `offsHigh2 = offsHigh3 + 4 = 5`. A greater value of the "high frequency" component delay needs to be selected for the first level of decomposition, `offsLow1 = -1`, `offsHigh1 = offsHigh2 + 2 * 4 = 13`.

Total delay for three levels of decomposition is equal to 12 samples.

One Level Forward Transform



Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when one of the specified pointers is `NULL`.
`ippStsContextMatchErr`	Indicates an error when the state identifier `pState` is incorrect.
`ippStsSizeErr`	Indicates an error when `dstLen` or `srcLen` is less than or equal to 0.

See Also

WTFwdGetSize Compute the size of the wavelet transform state structures.

WTFwdInit Initialize the wavelet transform state structures.

WTInv Computes the inverse wavelet transform.

Wavelet Transforms Example

WTFwdSetDlyLine, WTFwdGetDlyLine

Sets and gets the delay lines of the forward wavelet transform.

Syntax

```
IppStatus ippsWTFwdSetDlyLine_32f(IppsWTFwdState_32f* pState, const Ipp32f* pDlyLow,
const Ipp32f* pDlyHigh);
```

```
IppStatus ippsWTFwdSetDlyLine_8u32f(IppsWTFwdState_8u32f* pState, const Ipp32f*
pDlyLow, const Ipp32f* pDlyHigh);

IppStatus ippsWTFwdSetDlyLine_16s32f(IppsWTFwdState_16s32f* pState, const Ipp32f*
pDlyLow, const Ipp32f* pDlyHigh);

IppStatus ippsWTFwdSetDlyLine_16u32f(IppsWTFwdState_16u32f* pState, const Ipp32f*
pDlyLow, const Ipp32f* pDlyHigh);

IppStatus ippsWTFwdGetDlyLine_32f(IppsWTFwdState_32f* pState, Ipp32f* pDlyLow, Ipp32f*
pDlyHigh);

IppStatus ippsWTFwdGetDlyLine_8u32f(IppsWTFwdState_8u32f* pState, Ipp32f* pDlyLow,
Ipp32f* pDlyHigh);

IppStatus ippsWTFwdGetDlyLine_16s32f(IppsWTFwdState_16s32f* pState, Ipp32f* pDlyLow,
Ipp32f* pDlyHigh);

IppStatus ippsWTFwdGetDlyLine_16u32f(IppsWTFwdState_16u32f* pState, Ipp32f* pDlyLow,
Ipp32f* pDlyHigh);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

`pState`	Pointer to the state structure.
`pDlyLow`	Pointer to the vector which holds the delay lines for “low frequency” components.
`pDlyHigh`	Pointer to the vector which holds the delay lines for “high frequency” components.

Description

These functions copy the delay line values from `pDlyHigh` and `pDlyLow`, and stores them into the state structure `pState`.

`ippsWTFwdSetDlyLine`. This function sets the delay line values of the forward WT state.

``ippsWTFwdGetDlyLine``. This function gets the delay line values of the forward WT state.

Application Notes

These functions are designed to shape the signal prehistory, save and reconstruct delay lines. Delay lines are implemented separately for highpass and lowpass filters, which gives the option of getting independent signal prehistories for each filter.

Delay line data format. Despite that any delay line formats could be used inside transformations, the functions provide the simplest format of received and returned vectors. Data either transferred to or returned from the delay lines have the same format as the initial signal fed into the forward transform functions, i.e., delay line vectors must be made up of a succession of the signal prehistory counts in the same time frame as the initial signal.

Delay line lengths. The length of the vectors that are transferred to or received by the delay line installation or reading functions is uniquely defined by the filter length and the value of additional filter delay.

The following expression defines the length of the delay line vector of the "low frequency" component filter:

`dlyLowLen=lenLow+offsLow- 1,`

where `lenLow` and `offsLow` are respectively the length and additional delay of the "low frequency" component filter.

The following expression defines the length of the delay line vector of the "high frequency" component filter:
`dlyHighLen=lenHigh+offsHigh- 1,`

where `lenHigh` and `offsHigh` are respectively the length and additional delay of the "high frequency" component filter.

The `lenLow`, `offsLow`, `lenHigh`, and `offsHigh` parameters are specified by the function ippsWTFwdInit.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when `pDlyLow` or `pDlyHigh` is `NULL`.
`ippStsStateMatchErr`	Indicates an error when the state identifier `pState` is incorrect.

WTInv

Computes the inverse wavelet transform.

Syntax

```
IppStatus ippsWTInv_32f(const Ipp32f* pSrcLow, const Ipp32f* pSrcHigh, int srcLen,
Ipp32f* pDst, IppsWTInvState_32f* pState);

IppStatus ippsWTInv_32f8u(const Ipp32f* pSrcLow, const Ipp32f* pSrcHigh, int srcLen,
Ipp8u* pDst, IppsWTInvState_32f8u* pState);

IppStatus ippsWTInv_32f16s(const Ipp32f* pSrcLow, const Ipp32f* pSrcHigh, int srcLen,
Ipp16s* pDst, IppsWTInvState_32f16s* pState);

IppStatus ippsWTInv_32f16u(const Ipp32f* pSrcLow, const Ipp32f* pSrcHigh, int srcLen,
Ipp16u* pDst, IppsWTInvState_32f16u* pState);
```

Include Files

`ipps.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`

Libraries: `ippcore.lib`, `ippvm.lib`

Parameters

pSrcLow	Pointer to the vector which holds input coarse "low frequency" components.
pSrcHigh	Pointer to the vector which holds detail "high frequency" components.
srcLen	Number of elements in the vectors pSrcHigh and pSrcLow.
pDst	Pointer to the vector which holds the output reconstructed signal.
pState	Pointer to the state structure.

Description

This function computes the inverse wavelet transform. The function transforms the "low frequency" components pSrcLow and "high frequency" components pSrcHigh into the (2* srcLen)-length destination data block pDst. The transform parameters are specified in the state structure pState.

Before using this function, you need to compute the size of the state structure and work buffer using the WTFwdGetSize_ WTInvGetSize function, and initialize the structure using WTFwdInit_ WTInvInit.

For an example on how to use this function, refer to Wavelet Transforms Example.

Application Notes

These functions are used for one level of inverse multiscale transformation which results in reconstructing the original signal from the two "low frequency" and "high frequency" components. The Figure below shows an equivalent transform algorithm. Two interpolation filters are used for signal reconstruction; their coefficients are set at the initialization stage. The inverse transform implementation, similar to forward transform implementation, contains additional delay lines needed to synchronize the group time of filter delays and delays across different levels of data reconstruction.

Input and output data block lengths. These functions are designed to reconstruct the blocks of the even length signal. The signal component length must be the input data. The length of the output block of the reconstructed signal must be double the length of each of the components.

Filter group delay synchronization. In this example consider a biorthogonal set of spline filters of length 2 and 6:

```
static const float recLow[2] =
{
     1.00000000e+000f,
     1.00000000e+000f
};
static const float recHigh[6] =
{
    -1.25000000e-001f,
    -1.25000000e-001f,
     1.00000000e+000f,
    -1.00000000e+000f,
     1.25000000e-001f,
     1.25000000e-001f
};
```

This set of filters corresponds to the set of filters considered in a similar section of the description of the forward transform function ippsWTFwd.

Unlike the case described above, this time the high-pass filter generates a delay greater by two samples compared against the low frequency filter. The two sample difference should also exist between initialization function additional delays. The following parameters of additional delays need to be selected in order to ensure the minimum total delay, `offsLow = 2`, `offsHigh = 0`. In this case the total delay is equal to the high-pass filter group delay, which at the decomposition stage is equal to two samples in the original signal time frame.

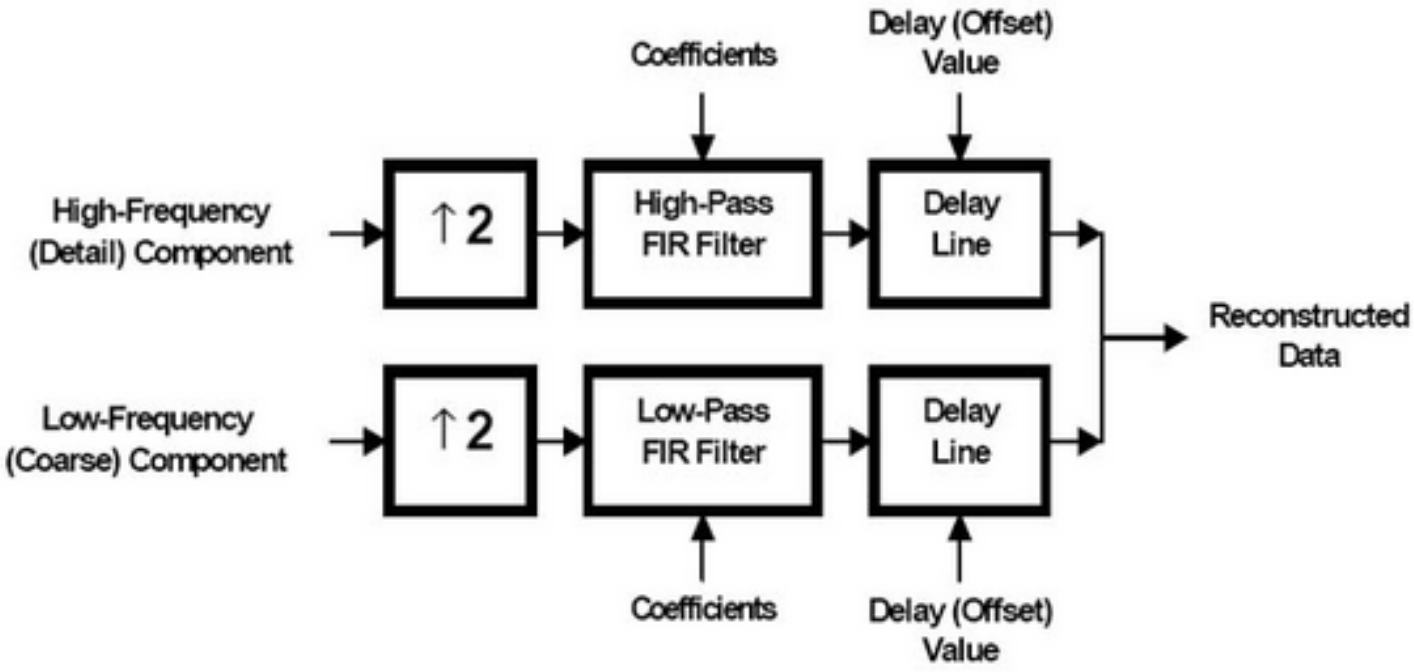
Total delay of one level of decomposition and reconstruction is equal to 4 samples, considering the decomposition stage delay.

> **NOTE** Biorthogonal and orthogonal filter banks are distinguished by one specific peculiarity, that is, inverse transform additional delays must be uniformly even and opposite to the evenness of the decomposition delays for faultless signal reconstruction.

Multilevel reconstruction algorithms. An example of a three-level signal reconstruction algorithm is shown in Figure "Three-Level Discrete Wavelet Reconstruction". The scheme corresponds to the decomposition scheme described in the section of the description of the forward transform function ippsWTFwd. Therefore, for the inverse transform the biorthogonal set of spline filters with respective filter length of 6 and 2 is used. The lowest level filter delays are set to `offsLow3 = 2`, `offsHigh3 = 0`. The total delay at this stage of reconstruction is equal to two samples. In order to ensure an equivalent delay of the "detail" part in the middle level, the delay must be increased. Respective values of additional delays for the second level are equal to `offsLow2 = 2`, `offsHigh2` = `offsHigh3 + 2 * 2 = 4`. A greater value of high frequency component delay needs to be selected for the last level of reconstruction, `offsLow1` = -1, `offsHigh1 = offsHigh2 + 2 * 4 = 12`.

The total delay for three levels of reconstruction is equal to 12 samples. The total delay of the three-level decomposition and reconstruction cycle is equal to 24 samples.

One Level Inverse Wavelet Transform



Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsStateMatchErr	Indicates an error when the state identifier pState is incorrect.
ippStsSizeErr	Indicates an error when dstLen or srcLen is less than, or equal to 0.

See Also

WTFwdGetSize, WTInvGetSize Compute the size of the wavelet transform state structures.

WTFwdInit, WTInvInit Initialize the wavelet transform state structures.

WTFwd Computes the forward wavelet transform.

Wavelet Transforms Example

WTInvSetDlyLine, WTInvGetDlyLine

Sets and gets the delay lines of the inverse wavelet transform.

Syntax

```
IppStatus ippsWTInvSetDlyLine_32f(IppsWTInvState_32f* pState, const Ipp32f* pDlyLow,
const Ipp32f* pDlyHigh);
```

```
IppStatus ippsWTInvSetDlyLine_32f8u(IppsWTInvState_32f8u* pState, const Ipp32f*
pDlyLow, const Ipp32f* pDlyHigh);
```

```
IppStatus ippsWTInvSetDlyLine_32f16s(IppsWTInvState_32f16s* pState, const Ipp32f*
pDlyLow, const Ipp32f* pDlyHigh);
```

```
IppStatus ippsWTInvSetDlyLine_32f16u(IppsWTInvState_32f16u* pState, const Ipp32f*
pDlyLow, const Ipp32f* pDlyHigh);
```

```
IppStatus ippsWTInvGetDlyLine_32f(IppsWTInvState_32f* pState, Ipp32f* pDlyLow, Ipp32f*
pDlyHigh);
```

```
IppStatus ippsWTInvGetDlyLine_32f8u(IppsWTInvState_32f8u* pState, Ipp32f* pDlyLow,
Ipp32f* pDlyHigh);
```

```
IppStatus ippsWTInvGetDlyLine_32f16s(IppsWTInvState_32f16s* pState, Ipp32f* pDlyLow,
Ipp32f* pDlyHigh);
```

```
IppStatus ippsWTInvGetDlyLine_32f16u(IppsWTInvState_32f16u* pState, Ipp32f* pDlyLow,
Ipp32f* pDlyHigh);
```

Include Files

ipps.h

Domain Dependencies

Headers: ippcore.h, ippvm.h

Libraries: ippcore.lib, ippvm.lib

Parameters

pState	Pointer to the state structure.
pDlyLow	Pointer to the vector which holds delay lines for "low frequency" components.
pDlyHigh	Pointer to the vector which holds delay lines for "high frequency" components.

Description

These functions copy the delay line values from pDlyHigh and pDlyLow, and store them into the state structure pState.

``ippsWTInvSetDlyLine``. This function sets the delay line values of the inverse WT state.

``ippsWTInvGetDlyLine``. This function gets the delay line values of the inverse WT state.

Application Notes

These functions set and read delay lines of inverse multiscale transformation. The functions receive or return filter low and high frequency component delay line vectors. The functions may be used to shape previous history of each of the components. Installation functions and read functions together ensure that delay lines from each filter are saved and reconstructed.

Delay line data format. Despite that any delay line formats could be used inside transformations, the functions provide the simplest format of received and returned vectors. Data either transferred to or returned from the delay lines have the same format as the low and high frequency components at the input of the inverse transform functions. Thus, delay line vectors must be made up of a succession of signal prehistory counts in the same time frame as the input components.

Delay line lengths. The length of the vectors that are transferred to or received by the delay line installation or reading functions is uniquely defined by the filter length and the value of additional filter delay.

The following expression defines the length of the delay line vector of the "low frequency" component filter in terms of the C language (integer division by two is used here for simplicity):

```
dlyLowLen= (lenLow+offsLow- 1) / 2,
```

where lenLow and offsLow are respectively the length and additional delay of the "low frequency" component filter.

The following expression defines the length of the delay line vector of the "high frequency" component filter in terms of the C language:

```
dlyHighLen= (lenHigh+offsHigh- 1) / 2,
```

where lenHigh and offsHigh are respectively the length and additional delay of the "high frequency" component filter.

The lenLow, offsLow, lenHigh, and offsHigh parameters are specified by the function ippsWTInvInit.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pDlyLow or pDlyHigh is NULL.

ippStsStateMatchErr	Indicates an error when the state identifier pState is incorrect.

Wavelet Transforms Example

The delay line paradigm is well-known interface solution for functions that require some pre-history in the streaming processing. In such application the use of the Intel IPP wavelet transform functions is similar to the use of the FIR, IIR, or multi-rate filters. (See also the discussion on the synchronization of low-pass and high-pass filter delays in this chapter.) But very often the wavelet transforms are used to process entire non-streaming data by extending with borders that are suitable for filter bank type that are used in transforms.

The following code example demonstrates how to implement this approach using the Intel IPP functions. It performs forward and inverse wavelet transforms of a short vector containing 12 elements. It uses Daubechies filter bank of the order 2 (that allows the perfect reconstruction) and periodical data extension by wrapping.

It is also may be useful as an illustration of how to fill delay line, if you need non-zero pre-history of signal in streaming applications.

Example

```
//   Filter bank for Daubechies, order 2
static const int fwdFltLenL = 4;
static const int fwdFltLenH = 4;
static const Ipp32f pFwdFltL[4] =
{    -1.294095225509215e-001f,     2.241438680418574e-001f,     8.365163037374690e-001f,
4.829629131446903e-001f  };
static const Ipp32f pFwdFltH[4] =
{    -4.829629131446903e-001f,     8.365163037374690e-001f,    -2.241438680418574e-001f,
-1.294095225509215e-001f  };
static const int invFltLenL = 4;
static const int invFltLenH = 4;
static const Ipp32f pInvFltL[4] =
{     4.829629131446903e-001f,     8.365163037374690e-001f,     2.241438680418574e-001f,
-1.294095225509215e-001f  };
static const Ipp32f pInvFltH[4] =
{    -1.294095225509215e-001f,    -2.241438680418574e-001f,     8.365163037374690e-001f,
-4.829629131446903e-001f  };
// minimal values
static const int fwdFltOffsL = -1;
static const int fwdFltOffsH = -1;
// minimal values, that corresponds to perfect reconstruction
static const int invFltOffsL = 0;
static const int invFltOffsH = 0;


void func_wavelet()
{
   IppStatus status=ippStsNoErr;
   Ipp32f pSrc[] = {1, -10, 324, 48, -483, 4, 7, -5532, 34, 8889, -57, 54};
   Ipp32f pDst[12];
   Ipp32f pLow[6];
   Ipp32f pHigh[6];
   IppsWTFwdState_32f* pFwdState;
   IppsWTInvState_32f* pInvState;
   int i, szState;


   printf("original:\n");
```

```
   for(i = 0; i < 12; i++)
      printf("%.0f; ", pSrc[i]);
   printf("\n");


   // Forward transform
   ippsWTFwdGetSize( ipp32f, fwdFltLenL, fwdFltOffsL, fwdFltLenH, fwdFltOffsH, &szState );
   pFwdState = (IppsWTFwdState_32f*)ippMalloc( szState );
   ippsWTFwdInit_32f( pFwdState, pFwdFltL, fwdFltLenL, fwdFltOffsL, pFwdFltH, fwdFltLenH,
fwdFltOffsH);
   // We substitute wrapping extension in "the beginning of stream"
   // Here should be the same pointers for this offsets,
   // but in the general case it may be different
   ippsWTFwdSetDlyLine_32f( pFwdState, &pSrc[10], &pSrc[10] );
   ippsWTFwd_32f( pSrc, pLow, pHigh, 6, pFwdState );


   printf("approx:\n");
   for(i = 0; i < 6; i++)
      printf("%.4f; ", pLow[i]);
   printf("\n");
   printf("details:\n");
   for(i = 0; i < 6; i++)
      printf("%.4f; ", pHigh[i]);
   printf("\n");


   // Inverse transform
   ippsWTInvGetSize( ipp32f, invFltLenL, invFltOffsL, invFltLenH, invFltOffsH, &szState );
   pInvState = (IppsWTInvState_32f*)ippMalloc( szState );
   ippsWTInvInit_32f( pInvState, pInvFltL, invFltLenL, invFltOffsL, pInvFltH, invFltLenH,
invFltOffsH );
   // For this particular case (non-shifted reconstruction)
   // here is first data itself,
   // that we need to place to delay line
   // [(invFltLenL + invFltOffsL - 1) / 2] elements for l. filtering
   // [(invFltLenH + invFltOffsH - 1) / 2] elements for h. filtering
   ippsWTInvSetDlyLine_32f( pInvState, pLow, pHigh);
   ippsWTInv_32f( &pLow[1], &pHigh[1], 5, pDst, pInvState );
   // Here are the substitution of the wrapping extension
   // at the "end of stream" and calculation of last samples of reconstruction
   // We do not use additional buffer and do not copy any data externally,
   // just substitute beginning of input data itself to simulate wrapping
   ippsWTInv_32f( pLow, pHigh, 1, &pDst[10], pInvState );


   printf("reconstruction:\n");
   for(i = 0; i < 12; i++)
      printf("%.0f; ", pDst[i]);
   printf("\n");


   ippFree(pFwdState);
   ippFree(pInvState);
}
```

After compiling and running it gives the following console output:

```
original:
1; -10; 324; 48; -483; 4; 7; -5532; 34; 8889; -57; 54;
approx:
19.1612; 58.5288; 87.8536; 487.5375; -5766.9277; 7432.4497;
details:
0.9387; 249.9611; -458.6568; 2739.2146; -3025.5576; -2070.5762;
reconstruction:
1; -10; 324; 48; -483; 4; 7; -5532; 34; 8889; -57; 54;
```

The program prints on console the original data, approximation, and details components after forward transform and perfect reconstruction of original data after inverse transform.

Fixed-Accuracy Arithmetic Functions

This chapter describes Intel® IPP fixed-accuracy transcendental mathematical real and complex functions of vector arguments. These functions take an input vector as argument, compute values of the respective elementary function element-wise, and return the results in an output vector.

Function specifications comply with the common API agreement of Intel IPP, but include some new features essential to scientific arithmetic functions. The main feature is a more elaborate specification of accuracy that differs from the common definition in adding several new levels of accuracy, besides original levels introduced by single precision and double precision data formats.

Fixed-accuracy vector functions implementation supports the IEEE-754 standard in all flavors, which means that:

- All functions have a precisely determined and guaranteed level of accuracy for all argument values.
- All special value processing and exceptions handling requirements are met, which implies that when accuracy is below the standard level, the function meets the IEEE-754 requirements in all other respects.

The choice of accuracy levels should be based on practical experience and identified application demands. Available options are specified in the function name suffix and include `A11`, `A21`, or `A24` for the single precision, and `A26`, `A50`, or `A53` for the double precision data format. Flavors `A11`, `A21`, `A26`, and `A50` provide approximately 3, 6, 8, and 15 exact decimal digits, respectively. For flavors `A24` and `A53`, the maximum guaranteed error is within 1 ulp and in most cases does not exceed 0.55 ulp.

Fixed-accuracy arithmetic functions subset of Intel IPP has the similar functionality as the respective part of the Intel® Math Kernel Library (Intel® MKL).

However, Intel IPP provides lower-level transcendental functions that have separate flavors for each mode of operations and data type and are better suitable for multimedia and signal processing in real time applications.

> **NOTE** Do not confuse fixed-accuracy arithmetic functions described here with common arithmetic functions that have similar functionality but follow different accuracy specifications.

Intel IPP fixed-accuracy arithmetic functions may return status codes of the specific warnings listed in the table below. In this case, the value returned is positive and the computation is continued.

	Value	Message
IppStsOverflow	12	Overflow occurred in the operation.
IppStsUnderflow	17	Underflow occurred in the operation.

IppStsSingularity	18	Singularity occurred in the operation.
IppStsDomain	19	Argument is out of the function domain.

See appendix A "Handling of Special Cases" for more information on function operation in cases when their arguments take on specific values that are outside the range of function definition.

NOTE All functions described in this chapter support in-place operation.

- Arithmetic Functions
- Power and Root Functions
- Exponential and Logarithmic Functions
- Trigonometric Functions
- Hyperbolic Functions
- Special Functions
- Rounding Functions

Arithmetic Functions

- Add
- Sub
- Sqr
- Mul
- MulByConj
- Conj
- Abs
- Arg

Add

Performs element by element addition of two vectors.

Syntax

```
IppStatus ippsAdd_32f_A24 (const Ipp32f* pSrc1, const Ipp32f* pSrc2, Ipp32f* pDst,
Ipp32s len);

IppStatus ippsAdd_64f_A53 (const Ipp64f* pSrc1, const Ipp64f* pSrc2, Ipp64f* pDst,
Ipp32s len);

IppStatus ippsAdd_32fc_A24 (const Ipp32fc* pSrc1, const Ipp32fc* pSrc2, Ipp32fc* pDst,
Ipp32s len);

IppStatus ippsAdd_64fc_A53 (const Ipp64fc* pSrc1, const Ipp64fc* pSrc2, Ipp64fc* pDst,
Ipp32s len);
```

Include Files

`ippvm.h`

Domain Dependencies

Headers: `ippcore.h`

Libraries: `ippcore.lib`

Parameters

pSrc1	Pointer to the first source vector.
pSrc2	Pointer to the second source vector.
pDst	Pointer to the destination vector.
len	Number of elements in the vectors.

Description

This function performs element by element addition of the vectors pSrc1 and pSrc2, and stores the result in the corresponding element of the vector pDst.

For single precision data:

function flavors ippsAdd_32f_A24 and ippsAdd_32fc_A24 guarantee 24 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

For double precision data:

function flavors ippsAdd_64f_A53 and ippsAdd_64fc_A53 guarantee 53 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

The computation is performed as follows:

pDst[n] = (pSrc1[n]) + (pSrc2[n]),0 ≤ n <len.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pSrc1, pSrc2 or pDst pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.

Example

The example below shows how to use the function ippsAdd.

```
IppStatus ippsAdd_32f_A24_sample(void) {
    const Ipp32f x1[4] = {+4.885, -0.543, -3.809, -4.953};
    const Ipp32f x2[4] = {-0.543, -3.809, -4.953, +4.885};
    Ipp32f  y[4];
    IppStatus st = ippsAdd_32f_A24( x1, x2, y, 4 );


    printf(" ippsAdd_32f_A24:\n");
    printf(" x1 = %+.3f %+.3f %+.3f %+.3f \n", x1[0], x1[1], x1[2], x1[3]);
    printf(" x2 = %+.3f %+.3f %+.3f %+.3f \n", x2[0], x2[1], x2[2], x2[3]);
    printf(" y  = %+.3f %+.3f %+.3f %+.3f \n",  y[0],  y[1],  y[2],  y[3]);
    return st;
}
```

Output:

```
ippsAdd_32f_A24:
x1 = +4.885 -0.543 -3.809 -4.953
x2 = -0.543 -3.809 -4.953 +4.885
y = +4.342 -4.352 -8.762 -0.068
```

Sub

Performs element by element subtraction of one vector from another.

Syntax

```
IppStatus ippsSub_32f_A24 (const Ipp32f* pSrc1, const Ipp32f* pSrc2, Ipp32f* pDst,
Ipp32s len);
```

```
IppStatus ippsSub_64f_A53 (const Ipp64f* pSrc1, const Ipp64f* pSrc2, Ipp64f* pDst,
Ipp32s len);
```

```
IppStatus ippsSub_32fc_A24 (const Ipp32fc* pSrc1, const Ipp32fc* pSrc2, Ipp32fc* pDst,
Ipp32s len);
```

```
IppStatus ippsSub_64fc_A53 (const Ipp64fc* pSrc1, const Ipp64fc* pSrc2, Ipp64fc* pDst,
Ipp32s len);
```

Include Files

`ippvm.h`

Domain Dependencies

Headers: `ippcore.h`

Libraries: `ippcore.lib`

Parameters

`pSrc1`	Pointer to the first source vector.
`pSrc2`	Pointer to the second source vector.
`pDst`	Pointer to the destination vector.
`len`	Number of elements in the vectors.

Description

This function performs element by element subtraction of the vector `pSrc2` from the vector `pSrc1`, and stores the result in the corresponding element of the vector `pDst`.

For single precision data:

function flavors `ippsSub_32f_A24` and `ippsSub_32fc_A24` guarantee 24 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

For double precision data:

function flavors `ippsSub_64f_A53` and `ippsSub_64fc_A53` guarantee 53 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

The computation is performed as follows:

pDst[n] = (pSrc1[n]) - (pSrc2[n]),0 ≤ n <len.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pSrc1, pSrc2 or pDst pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.

Example

The example below shows how to use the function ippsSub.

```
IppStatus ippsSub_32f_A24_sample(void) {
    const Ipp32f x1[4] = {+4.885, -0.543, -3.809, -4.953};
    const Ipp32f x2[4] = {-0.543, -3.809, -4.953, +4.885};


    Ipp32f  y[4];
    IppStatus st = ippsSub_32f_A24( x1, x2, y, 4 );


    printf(" ippsSub_32f_A24:\n");
    printf(" x1 = %+.3f %+.3f %+.3f %+.3f \n", x1[0], x1[1], x1[2], x1[3]);
    printf(" x2 = %+.3f %+.3f %+.3f %+.3f \n", x2[0], x2[1], x2[2], x2[3]);
    printf(" y  = %+.3f %+.3f %+.3f %+.3f \n",  y[0],  y[1],  y[2],  y[3]);
    return st;
}
```

Output:

```
ippsSub_32f_A24:
x1 = +4.885 -0.543 -3.809 -4.953
x2 = -0.543 -3.809 -4.953 +4.885
y = +5.428 +3.266 +1.144 -9.838
```

Sqr

Performs element by element squaring of the vector.

Syntax

```
IppStatus ippsSqr_32f_A24 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsSqr_64f_A53 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
```

Include Files

ippvm.h

Domain Dependencies

Headers: ippcore.h

Libraries: ippcore.lib

Parameters

pSrc	Pointer to the source vector.
pDst	Pointer to the destination vector.
len	Number of elements in the vectors.

Description

This function performs element by element squaring of the vector pSrc, and stores the result in the corresponding element of pDst.

For single precision data:

function flavor ippsSqr_32f_A24 guarantees 24 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

For double precision data:

function flavor ippsSqr_64f_A53 guarantees 53 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

The computation is performed as follows:

$pDst[n] = (pSrc[n])^2, 0 \le n < len.$

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pSrc or pDst pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.

Example

The example below shows how to use the function ippsSqr.

```
IppStatus ippsSqr_32f_A24_sample(void) {
    const Ipp32f x[4] = {+4.885, -0.543, -3.809, -4.953};
    Ipp32f  y[4];
    IppStatus st = ippsSqr_32f_A24( x, y, 4 );


    printf(" ippsSqr_32f_A24:\n");
    printf(" x = %+.3f %+.3f %+.3f %+.3f \n", x[0], x[1], x[2], x[3]);
    printf(" y = %+.3f %+.3f %+.3f %+.3f \n", y[0], y[1], y[2], y[3]);
    return st;
}
```

Output:

```
ippsSqr_32f_A24:
x = +4.885 -0.543 -3.809 -4.953
y = +23.863 +0.295 +14.508 +24.532
```

Mul

Performs element by element multiplication of two vectors.

Syntax

```
IppStatus ippsMul_32f_A24 (const Ipp32f* pSrc1, const Ipp32f* pSrc2, Ipp32f* pDst,
Ipp32s len);

IppStatus ippsMul_64f_A53 (const Ipp64f* pSrc1, const Ipp64f* pSrc2, Ipp64f* pDst,
Ipp32s len);

IppStatus ippsMul_32fc_A11 (const Ipp32fc* pSrc1, const Ipp32fc* pSrc2, Ipp32fc* pDst,
Ipp32s len);

IppStatus ippsMul_32fc_A21 (const Ipp32fc* pSrc1, const Ipp32fc* pSrc2, Ipp32fc* pDst,
Ipp32s len);

IppStatus ippsMul_32fc_A24 (const Ipp32fc* pSrc1, const Ipp32fc* pSrc2, Ipp32fc* pDst,
Ipp32s len);

IppStatus ippsMul_64fc_A26 (const Ipp64fc* pSrc1, const Ipp64fc* pSrc2, Ipp64fc* pDst,
Ipp32s len);

IppStatus ippsMul_64fc_A50 (const Ipp64fc* pSrc1, const Ipp64fc* pSrc2, Ipp64fc* pDst,
Ipp32s len);

IppStatus ippsMul_64fc_A53 (const Ipp64fc* pSrc1, const Ipp64fc* pSrc2, Ipp64fc* pDst,
Ipp32s len);
```

Include Files

`ippvm.h`

Domain Dependencies

Headers: `ippcore.h`

Libraries: `ippcore.lib`

Parameters

`pSrc1`	Pointer to the first source vector.
`pSrc2`	Pointer to the second source vector.
`pDst`	Pointer to the destination vector.
`len`	Number of elements in the vectors.

Description

This function performs element by element multiplication of the vectors `pSrc1` and `pSrc2`, and stores the result in the corresponding element of the vector `pDst`.

For single precision data:

function flavor `ippsMul_32fc_A11` guarantees 11 correctly rounded bits of significand, or at least 3 exact decimal digits;

function flavor `ippsMul_32fc_A21` guarantees 21 correctly rounded bits of significand, or 4 ulps, or about 6 exact decimal digits;

function flavors ippsMul_32f_A24 and ippsMul_32fc_A24 guarantee 24 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

For double precision data:

function flavor ippsMul_64fc_A26 guarantees 26 correctly rounded bits of significand, or 6.7E+7 ulps, or approximately 8 exact decimal digits;

function flavor ippsMul_64fc_A50 guarantees 50 correctly rounded bits of significand, or 4 ulps, or approximately 15 exact decimal digits;

function flavors ippsMul_64f_A53 and ippsMul_64fc_A53 guarantee 53 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

The computation is performed as follows:

pDst[n] = (pSrc1[n]) x (pSrc2[n]),0 ≤ n <len.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pSrc1, pSrc2 or pDst pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.

Example

The example below shows how to use the function ippsMul.

```
IppStatus ippsMul_32f_A24_sample(void) {
    const Ipp32f x1[4] = {+4.885, -0.543, -3.809, -4.953};
    const Ipp32f x2[4] = {-0.543, -3.809, -4.953, +4.885};
    Ipp32f  y[4];
    IppStatus st = ippsMul_32f_A24( x1, x2, y, 4 );


    printf(" ippsMul_32f_A24:\n");
    printf(" x1 = %+.3f %+.3f %+.3f %+.3f \n", x1[0], x1[1], x1[2], x1[3]);
    printf(" x2 = %+.3f %+.3f %+.3f %+.3f \n", x2[0], x2[1], x2[2], x2[3]);
    printf(" y  = %+.3f %+.3f %+.3f %+.3f \n",  y[0],  y[1],  y[2],  y[3]);
    return st;
}
```

Output:

```
ippsMul_32f_A24:
x1 = +4.885 -0.543 -3.809 -4.953
x2 = -0.543 -3.809 -4.953 +4.885
y = -2.653 +2.068 +18.866 -24.195
```

MulByConj

Performs element by element multiplication of a vector a element and a conjugated vector b element.

Syntax

```
IppStatus ippsMulByConj_32fc_A11 (const Ipp32fc* pSrc1, const Ipp32fc* pSrc2, Ipp32fc*
pDst, Ipp32s len);
```

```
IppStatus ippsMulByConj_32fc_A21 (const Ipp32fc* pSrc1, const Ipp32fc* pSrc2, Ipp32fc*
pDst, Ipp32s len);

IppStatus ippsMulByConj_32fc_A24 (const Ipp32fc* pSrc1, const Ipp32fc* pSrc2, Ipp32fc*
pDst, Ipp32s len);

IppStatus ippsMulByConj_64fc_A26 (const Ipp64fc* pSrc1, const Ipp64fc* pSrc2, Ipp64fc*
pDst, Ipp32s len);

IppStatus ippsMulByConj_64fc_A50 (const Ipp64fc* pSrc1, const Ipp64fc* pSrc2, Ipp64fc*
pDst, Ipp32s len);

IppStatus ippsMulByConj_64fc_A53 (const Ipp64fc* pSrc1, const Ipp64fc* pSrc2, Ipp64fc*
pDst, Ipp32s len);
```

Include Files

`ippvm.h`

Domain Dependencies

Headers: `ippcore.h`

Libraries: `ippcore.lib`

Parameters

`pSrc1`	Pointer to the first source vector.
`pSrc2`	Pointer to the second source vector.
`pDst`	Pointer to the destination vector.
`len`	Number of elements in the vectors.

Description

This function performs element by element multiplication of the vector `pSrc1` and the conjugated vector `pSrc2`, and stores the result in the corresponding element of the vector `pDst`.

For single precision data:

function flavor `ippsMulByConj_32fc_A11` guarantees 11 correctly rounded bits of significand, or at least 3 exact decimal digits;

function flavor `ippsMulByConj_32fc_A21` guarantees 21 correctly rounded bits of significand, or 4 ulps, or about 6 exact decimal digits;

function flavor `ippsMulByConj_32fc_A24` guarantees 24 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

For double precision data:

function flavor `ippsMulByConj_64fc_A26` guarantees 26 correctly rounded bits of significand, or 6.7E+7 ulps, or approximately 8 exact decimal digits;

function flavor `ippsMulByConj_64fc_A50` guarantees 50 correctly rounded bits of significand, or 4 ulps, or approximately 15 exact decimal digits;

function flavor `ippsMulByConj_64fc_A53` guarantees 53 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

The computation is performed as follows:

pDst[n] =pSrc1[n] x CONJ(pSrc2[n]),0 ≤ n <len.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pSrc1, pSrc2 or pDst pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.

Example

The example below shows how to use the function ippsMulByConj.

```
IppStatus ippsMulByConj_32fc_A24_sample(void) {
    const Ipp32fc x1[4] = {{+2.885,-1.809}, {-0.543,-2.809}};
    const Ipp32fc x2[4] = {{-0.543,-2.809}, {-1.809,-2.809}};
    Ipp32fc  y[2];
    IppStatus st = ippsMulByConj_32fc_A24( x1, x2, y, 2 );


    printf(" ippsMulByConj_32fc_A24:\n");
    printf(" x1 = %+.3f%+.3f*i   %+.3f%+.3f*i \n", x1[0].re, x1[0].im, x1[1].re, x1[1].im);
    printf(" x2 = %+.3f%+.3f*i   %+.3f%+.3f*i \n", x2[0].re, x2[0].im, x2[1].re, x2[1].im);
    printf(" y  = %+.3f%+.3f*i   %+.3f%+.3f*i \n",  y[0].re,  y[0].im,  y[1].re,  y[1].im);
    return st;
}
```

Output:

```
ippsMulByConj_32fc_A24:
x1 = +2.885-1.809*i   -0.543-2.809*i
x2 = -0.543-2.809*i   -1.809-2.809*i
y = +3.515+9.086*i   +8.873+3.556*i
```

Conj

Performs element by element conjugation of the vector.

Syntax

```
IppStatus ippsConj_32fc_A24 (const Ipp32fc* pSrc, Ipp32fc* pDst, Ipp32s len);
IppStatus ippsConj_64fc_A53 (const Ipp64fc* pSrc, Ipp64fc* pDst, Ipp32s len);
```

Include Files

ippvm.h

Domain Dependencies

Headers: ippcore.h

Libraries: ippcore.lib

Parameters

pSrc	Pointer to the source vector.
pDst	Pointer to the destination vector.
len	Number of elements in the vectors.

Description

This function performs element by element conjugation of the vector pSrc and stores the result in the corresponding element of the vector pDst.

For single precision data:

function flavor ippsConj_32fc_A24 guarantees 24 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

For double precision data:

function flavor ippsConj_64fc_A53 guarantees 53 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

The computation is performed as follows:

pDst[n] = CONJ(pSrc[n]), 0 ≤ n <len.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pSrc1, pSrc2 or pDst pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.

Example

The example below shows how to use the function ippsConj.

```
IppStatus ippsConj_32fc_A24_sample(void) {
    const Ipp32fc x[2] = {{+2.885,-1.809}, {-0.543,-2.809}};
    Ipp32fc  y[2];
    IppStatus st = ippsConj_32fc_A24( x, y, 2 );


    printf(" ippsConj_32fc_A24:\n");
    printf(" x = %+.3f%+.3f*i   %+.3f%+.3f*i \n", x[0].re, x[0].im, x[1].re, x[1].im);
    printf(" y = %+.3f%+.3f*i   %+.3f%+.3f*i \n", y[0].re, y[0].im, y[1].re, y[1].im);
    return st;
}
```

Output:

```
ippsConj_32fc_A24:
x = +2.885-1.809*i   -0.543-2.809*i
y = +2.885+1.809*i   -0.543+2.809*i
```

Abs

Computes the absolute value of vector elements.

Syntax

```
IppStatus ippsAbs_32f_A24 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsAbs_64f_A53 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsAbs_32fc_A11 (const Ipp32fc* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsAbs_32fc_A21 (const Ipp32fc* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsAbs_32fc_A24 (const Ipp32fc* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsAbs_64fc_A26 (const Ipp64fc* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsAbs_64fc_A50 (const Ipp64fc* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsAbs_64fc_A53 (const Ipp64fc* pSrc, Ipp64f* pDst, Ipp32s len);
```

Include Files

`ippvm.h`

Domain Dependencies

Headers: `ippcore.h`

Libraries: `ippcore.lib`

Parameters

`pSrc`	Pointer to the source vector.
`pDst`	Pointer to the destination vector.
`len`	Number of elements in the vectors.

Description

This function computes the absolute value of the vector `pSrc` elements and stores the result in the corresponding element of the vector `pDst`.

For single precision data:

function flavor `ippsAbs_32fc_A11` guarantees 11 correctly rounded bits of significand, or at least 3 exact decimal digits;

function flavor `ippsAbs_32fc_A21` guarantees 21 correctly rounded bits of significand, or 4 ulps, or about 6 exact decimal digits;

function flavors `ippsAbs_32f_A24` and `ippsAbs_32fc_A24` guarantee 24 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

For double precision data:

function flavor `ippsAbs_64fc_A26` guarantees 26 correctly rounded bits of significand, or 6.7E+7 ulps, or approximately 8 exact decimal digits;

function flavor `ippsAbs_64fc_A50` guarantees 50 correctly rounded bits of significand, or 4 ulps, or approximately 15 exact decimal digits;

function flavors `ippsAbs_64f_A53` and `ippsAbs_64fc_A53` guarantee 53 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

The computation is performed as follows:

`pDst[n] = |pSrc1[n]|, 0 ≤ n <len.`

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when `pSrc1`, `pSrc2` or `pDst` pointer is `NULL`.
`ippStsSizeErr`	Indicates an error when `len` is less than or equal to 0.

Example

The example below shows how to use the function `ippsAbs`.

```
IppStatus ippsAbs_32fc_A24_sample(void) {
    const Ipp32fc x[2] = {{+2.885,-1.809}, {-0.543,-2.809}};
    Ipp32fc  y[2];
    IppStatus st = ippsAbs_32fc_A24( x, y, 2 );


    printf(" ippsAbs_32fc_A24:\n");
    printf(" x = %+.3f%+.3f*i   %+.3f%+.3f*i \n", x[0].re, x[0].im, x[1].re, x[1].im);
    printf(" y = %+.3f          %+.3f%       \n", y[0],             y[1]);
    return st;
}
```

Output:

```
ippsAbs_32fc_A24:
x = +2.885-1.809*i   -0.543-2.809*i
y = +3.405           +2.861
```

Arg

Computes the argument of vector elements.

Syntax

```
IppStatus ippsArg_32fc_A11(const Ipp32fc* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsArg_32fc_A21(const Ipp32fc* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsArg_32fc_A24(const Ipp32fc* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsArg_64fc_A26(const Ipp64fc* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsArg_64fc_A50(const Ipp64fc* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsArg_64fc_A53(const Ipp64fc* pSrc, Ipp64f* pDst, Ipp32s len);
```

Include Files

`ippvm.h`

Domain Dependencies

Headers: `ippcore.h`

Libraries: ippcore.lib

Parameters

pSrc	Pointer to the source vector.
pDst	Pointer to the destination vector.
len	Number of elements in the vectors.

Description

This function computes the argument of the vector pSrc elements and stores the result in the corresponding element of the vector pDst.

For single precision data:

function flavor ippsArg_32fc_A11 guarantees 11 correctly rounded bits of significand, or at least 3 exact decimal digits;

function flavor ippsArg_32fc_A21 guarantees 21 correctly rounded bits of significand, or 4 ulps, or about 6 exact decimal digits;

function flavor ippsArg_32fc_A24 guarantees 24 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

For double precision data:

function flavor ippsArg_64fc_A26 guarantees 26 correctly rounded bits of significand, or 6.7E+7 ulps, or approximately 8 exact decimal digits;

function flavor ippsArg_64fc_A50 guarantees 50 correctly rounded bits of significand,or 4 ulps, or approximately 15 exact decimal digits;

function flavor ippsArg_64fc_A53 guarantees 53 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

The computation is performed as follows:

pDst[n] =φ(pSrc1[n]),0 ≤ n <len.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pSrc1, pSrc2 or pDst pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.

Example

The example below shows how to use the function ippsArg.

```
IppStatus ippsArg_32fc_A24_sample(void) {
    const Ipp32fc x[2] = {{+2.885,-1.809}, {-0.543,-2.809}};
    Ipp32fc  y[2];
    IppStatus st = ippsArg_32fc_A24( x, y, 2 );


    printf(" ippsArg_32fc_A24:\n");
```

```
    printf(" x = %+.3f%+.3f*i    %+.3f%+.3f*i \n", x[0].re, x[0].im, x[1].re, x[1].im);
    printf(" y = %+.3f           %+.3f%       \n", y[0],             y[1]);
    return st;
}
```

Output:

```
ippsArg_32fc_A24:
x = +2.885-1.809*i    -0.543-2.809*i
y = -0.560            -1.762
```

Power and Root Functions

- Inv
- Div
- Sqrt
- InvSqrt
- Cbrt
- InvCbrt
- Pow2o3
- Pow3o2
- Pow
- Powx
- Hypot

Inv

Computes inverse value of each vector element.

Syntax

```
IppStatus ippsInv_32f_A11 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsInv_32f_A21 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsInv_32f_A24 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsInv_64f_A26 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsInv_64f_A50 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsInv_64f_A53 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
```

Include Files

`ippvm.h`

Domain Dependencies

Headers: `ippcore.h`

Libraries: `ippcore.lib`

Parameters

`pSrc`	Pointer to the source vector.
`pDst`	Pointer to the destination vector.

len	Number of elements in the vectors.

Description

This function computes the inverse value of each element of the vector pSrc, and stores the result in the corresponding element of the vector pDst.

For single precision data:

function flavor ippsInv_32f_A11 guarantees 11 correctly rounded bits of significand, or at least 3 exact decimal digits;

function flavor ippsInv_32f_A21 guarantees 21 correctly rounded bits of significand, or 4 ulps, or about 6 exact decimal digits;

function flavor ippsInv_32f_A24 guarantees 24 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

For double precision data:

function flavor ippsInv_64f_A26 guarantees 26 correctly rounded bits of significand, or 6.7E+7 ulps, or approximately 8 exact decimal digits;

function flavor ippsInv_64f_A50 guarantees 50 correctly rounded bits of significand, or 4 ulps, or approximately 15 exact decimal digits;

function flavor ippsInv_64f_A53 guarantees 53 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

The computation is performed as follows:

pDst[n] = 1/(pSrc[n]), 0 ≤ n <len.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pSrc or pDst pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.
IppStsSingularity	Indicates a warning that the argument is the singularity point, that is, at least one of the elements of pSrc is equal to 0.

Example

The example below shows how to use the function ippsInv.

```
IppStatus ippsInv_32f_A21_sample(void) {
    const Ipp32f x[4] = {-9.975, 1.272, -6.134, 6.175};
    Ipp32f y[4];
    IppStatus st = ippsInv_32f_A21( x, y, 4 );


    printf(" ippsInv_32f_A21:\n");
    printf(" x = %.3f %.3f %.3f %.3f \n", x[0], x[1], x[2], x[3]);
```

```
    printf(" y = %.3f %.3f %.3f %.3f \n", y[0], y[1], y[2], y[3]);
    return st;
}
```

Output:

```
ippsInv_32f_A21:
x = -9.975 1.272 -6.134 6.175
y = -0.100 0.786 -0.163 0.162
```

Div

Divides each element of the first vector by corresponding element of the second vector.

Syntax

```
IppStatus ippsDiv_32f_A11 (const Ipp32f* pSrc1, const Ipp32f* pSrc2, Ipp32f* pDst,
Ipp32s len);

IppStatus ippsDiv_32f_A21 (const Ipp32f* pSrc1, const Ipp32f* pSrc2, Ipp32f* pDst,
Ipp32s len);

IppStatus ippsDiv_32f_A24 (const Ipp32f* pSrc1, const Ipp32f* pSrc2, Ipp32f* pDst,
Ipp32s len);

IppStatus ippsDiv_64f_A26 (const Ipp64f* pSrc1, const Ipp64f* pSrc2, Ipp64f* pDst,
Ipp32s len);

IppStatus ippsDiv_64f_A50 (const Ipp64f* pSrc1, const Ipp64f* pSrc2, Ipp64f* pDst,
Ipp32s len);

IppStatus ippsDiv_64f_A53 (const Ipp64f* pSrc1, const Ipp64f* pSrc2, Ipp64f* pDst,
Ipp32s len);

IppStatus ippsDiv_32fc_A11 (const Ipp32fc* pSrc1, const Ipp32fc* pSrc2, Ipp32fc* pDst,
Ipp32s len);

IppStatus ippsDiv_32fc_A21 (const Ipp32fc* pSrc1, const Ipp32fc* pSrc2, Ipp32fc* pDst,
Ipp32s len);

IppStatus ippsDiv_32fc_A24 (const Ipp32fc* pSrc1, const Ipp32fc* pSrc2, Ipp32fc* pDst,
Ipp32s len);

IppStatus ippsDiv_64fc_A26 (const Ipp64fc* pSrc1, const Ipp64fc* pSrc2, Ipp64fc* pDst,
Ipp32s len);

IppStatus ippsDiv_64fc_A50 (const Ipp64fc* pSrc1, const Ipp64fc* pSrc2, Ipp64fc* pDst,
Ipp32s len);

IppStatus ippsDiv_64fc_A53 (const Ipp64fc* pSrc1, const Ipp64fc* pSrc2, Ipp64fc* pDst,
Ipp32s len);
```

Include Files

`ippvm.h`

Domain Dependencies

Headers: `ippcore.h`

Libraries: `ippcore.lib`

Parameters

pSrc1	Pointer to the first source vector.
pSrc2	Pointer to the second source vector.
pDst	Pointer to the destination vector.
len	Number of elements in the vectors.

Description

This function divides each element of the vector pSrc1 by the corresponding element of the vector pSrc2 and stores the result in the corresponding element of pDst.

For single precision data:

function flavors ippsDiv_32f_A11 and ippsDiv_32cf_A11 guarantee 11 correctly rounded bits of significand, or at least 3 exact decimal digits;

function flavors ippsDiv_32f_A21 and ippsDiv_32fc_A21 guarantee 21 correctly rounded bits of significand, or 4 ulps, or about 6 exact decimal digits;

function flavors ippsDiv_32f_A24 and ippsDiv_32fc_A24 guarantee 24 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

For double precision data:

function flavors ippsDiv_64f_A26 and ippsDiv_64fc_A26 guarantee 26 correctly rounded bits of significand, or 6.7E+7 ulps, or approximately 8 exact decimal digits;

function flavors ippsDiv_64f_A50 and ippsDiv_64fc_A50 guarantee 50 correctly rounded bits of significand, or 4 ulps, or approximately 15 exact decimal digits;

function flavors ippsDiv_64f_A53 and ippsDiv_64fc_A53 guarantee 53 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

The computation is performed as follows:

pDst[n] = (pSrc1[n]) /(pSrc2[n]),0 ≤ n <len.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pSrc1 or pSrc2 or pDst pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.
IppStsSingularity	In real functions, indicates a warning that the argument is the singularity point, that is, at least one of the elements of pSrc2 is equal to 0.

Example

The example below shows how to use the function ippsDiv.

```
IppStatus ippsDiv_32f_A21_sample(void) {
    const Ipp32f x1[4] = {599.088, 735.034, 572.448, 151.640};
    const Ipp32f x2[4] = {385.297, 609.005, 361.403, 225.182};
```

```
    Ipp32f y[4];
    IppStatus st = ippsDiv_32f_A21( x1, x2, y, 4 );


    printf(" ippsDiv_32f_A21:\n");
    printf(" x1 = %.3f %.3f %.3f %.3f \n", x1[0], x1[1], x1[2], x1[3]);
    printf(" x2 = %.3f %.3f %.3f %.3f \n", x2[0], x2[1], x2[2], x2[3]);
    printf(" y  = %.3f %.3f %.3f %.3f \n", y[0],  y[1],  y[2],  y[3]);
    return st;
}
```

Output:

```
ippsDiv_32f_A21:
x1 = 599.088 735.034 572.448 151.640
x2 = 385.297 609.005 361.403 225.182
y  = 1.555 1.207 1.584 0.673
```

Sqrt

Computes square root of each vector element.

Syntax

```
IppStatus ippsSqrt_32f_A11 ( const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsSqrt_32f_A21 ( const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsSqrt_32f_A24 ( const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsSqrt_64f_A26 ( const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsSqrt_64f_A50 ( const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsSqrt_64f_A53 ( const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsSqrt_32fc_A11 ( const Ipp32fc* pSrc, Ipp32fc* pDst, Ipp32s len);
IppStatus ippsSqrt_32fc_A21 ( const Ipp32fc* pSrc, Ipp32fc* pDst, Ipp32s len);
IppStatus ippsSqrt_32fc_A24 ( const Ipp32fc* pSrc, Ipp32fc* pDst, Ipp32s len);
IppStatus ippsSqrt_64fc_A26 ( const Ipp64fc* pSrc, Ipp64fc* pDst, Ipp32s len);
IppStatus ippsSqrt_64fc_A50 ( const Ipp64fc* pSrc, Ipp64fc* pDst, Ipp32s len);
IppStatus ippsSqrt_64fc_A53 ( const Ipp64fc* pSrc, Ipp64fc* pDst, Ipp32s len);
```

Include Files

`ippvm.h`

Domain Dependencies

Headers: `ippcore.h`

Libraries: `ippcore.lib`

Parameters

`pSrc`	Pointer to the source vector.

pDst	Pointer to the destination vector.
len	Number of elements in the vectors.

Description

This function computes square root of each element of pSrc and stores the result in the corresponding element of pDst.

For single precision data:

function flavors ippsSqrt_32f_A11 and ippsSqrt_32cf_A11 guarantee 11 correctly rounded bits of significand, or at least 3 exact decimal digits;

function flavors ippsSqrt_32f_A21 and ippsSqrt_32fc_A21 guarantee 21 correctly rounded bits of significand, or 4 ulps, or about 6 exact decimal digits;

function flavors ippsSqrt_32f_A24 and ippsSqrt_32fc_A24 guarantee 24 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

For double precision data:

function flavors ippsSqrt_64f_A26 and ippsSqrt_64fc_A26 guarantee 26 correctly rounded bits of significand, or 6.7E+7 ulps, or approximately 8 exact decimal digits;

function flavors ippsSqrt_64f_A50 and ippsSqrt_64fc_A50 guarantee 50 correctly rounded bits of significand, or 4 ulps, or approximately 15 exact decimal digits;

function flavors ippsSqrt_64f_A53 and ippsSqrt_64fc_A53 guarantee 53 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

The computation is performed as follows:

$pDst[n] = (pSrc[n])^{1/2}, 0 \le n < len.$

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pSrc or pDst pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.
IppStsDomain	In real functions, indicates a warning that the argument is out of the function domain, that is, at least one of the elements of pSrc is less than 0.

Example

The example below shows how to use the function ippsSqrt.

```
IppStatus ippsSqrt_32f_A21_sample(void) {
    const Ipp32f x[4] = {5850.093, 4798.730, 3502.915, 8959.624};
    Ipp32f y[4];
    IppStatus st = ippsSqrt_32f_A21( x, y, 4 );


    printf(" ippsSqrt_32f_A21:\n");
    printf(" x = %.3f %.3f %.3f %.3f \n", x[0], x[1], x[2], x[3]);
```

```
    printf(" y = %.3f %.3f %.3f %.3f \n", y[0], y[1], y[2], y[3]);
    return st;
}
```

Output:

```
ippsSqrt_32f_A21:
x = 5850.093 4798.730 3502.915 8959.624
y = 76.486 69.273 59.185 94.655
```

InvSqrt

Computes inverse square root of each vector element.

Syntax

```
IppStatus ippsInvSqrt_32f_A11 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsInvSqrt_32f_A21 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsInvSqrt_32f_A24 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsInvSqrt_64f_A26 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsInvSqrt_64f_A50 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsInvSqrt_64f_A53 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
```

Include Files

`ippvm.h`

Domain Dependencies

Headers: `ippcore.h`

Libraries: `ippcore.lib`

Parameters

`pSrc`	Pointer to the source vector.
`pDst`	Pointer to the destination vector.
`len`	Number of elements in the vectors.

Description

This function computes inverse square root of each element of `pSrc` and stores the result in the corresponding element of `pDst`.

For single precision data:

function flavor `ippsInvSqrt_32f_A11` guarantees 11 correctly rounded bits of significand, or at least 3 exact decimal digits;

function flavor `ippsInvSqrt_32f_A21` guarantees 21 correctly rounded bits of significand, or 4 ulps, or about 6 exact decimal digits;

function flavor `ippsInvSqrt_32f_A24` guarantees 24 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

For double precision data:

function flavor `ippsInvSqrt_64f_A26` guarantees 26 correctly rounded bits of significand, or 6.7E+7 ulps, or approximately 8 exact decimal digits;

function flavor `ippsInvSqrt_64f_A50` guarantees 50 correctly rounded bits of significand, or 4 ulps, or approximately 15 exact decimal digits;

function flavor `ippsInvSqrt_64f_A53` guarantees 53 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

The computation is performed as follows:

`pDst[n] = (pSrc[n])`$^{-1/2}$, `0 ≤ n <len.`

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when `pSrc` or `pDst` pointer is `NULL`.
`ippStsSizeErr`	Indicates an error when `len` is less than or equal to 0.
`IppStsDomain`	Indicates a warning that the argument is out of the function domain, that is, at least one of the elements of `pSrc` is less than 0.
`IppStsSingularity`	Indicates a warning that the argument is the singularity point, that is, at least one of the elements of `pSrc` is equal to 0.

Example

The example below shows how to use the function `ippsInvSqrt`.

```
IppStatus ippsInvSqrt_32f_A21_sample(void) {
    const Ipp32f x[4] = {7105.043, 5135.398, 3040.018, 149.944};
    Ipp32f y[4];


    IppStatus st = ippsInvSqrt_32f_A21( x, y, 4 );
    printf(" ippsInvSqrt_32f_A21:\n");
    printf(" x = %.3f %.3f %.3f %.3f \n", x[0], x[1], x[2], x[3]);
    printf(" y = %.3f %.3f %.3f %.3f \n", y[0], y[1], y[2], y[3]);
    return st;
}
```

Output:

```
ippsInvSqrt_32f_A21:
x = 7105.043 5135.398 3040.018 149.944
y = 0.012 0.014 0.018 0.082
```

Cbrt

Computes cube root of each vector element.

Syntax

```
IppStatus ippsCbrt_32f_A11 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsCbrt_32f_A21 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsCbrt_32f_A24 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsCbrt_64f_A26 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsCbrt_64f_A50 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsCbrt_64f_A53 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
```

Include Files

`ippvm.h`

Domain Dependencies

Headers: `ippcore.h`

Libraries: `ippcore.lib`

Parameters

`pSrc`	Pointer to the source vector.
`pDst`	Pointer to the destination vector.
`len`	Number of elements in the vectors.

Description

This function computes cube root of each element of `pSrc`, and stores the result in the corresponding element of `pDst`.

For single precision data:

function flavor `ippsCbrt_32f_A11` guarantees 11 correctly rounded bits of significand, or at least 3 exact decimal digits;

function flavor `ippsCbrt_32f_A21` guarantees 21 correctly rounded bits of significand, or 4 ulps, or about 6 exact decimal digits;

function flavor `ippsCbrt_32f_A24` guarantees 24 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

For double precision data:

function flavor `ippsCbrt_64f_A26` guarantees 26 correctly rounded bits of significand, or 6.7E+7 ulps, or approximately 8 exact decimal digits;

function flavor `ippsCbrt_64f_A50` guarantees 50 correctly rounded bits of significand, or 4 ulps, or approximately 15 exact decimal digits;

function flavor `ippsCbrt_64f_A53` guarantees 53 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

The computation is performed as follows:

$pDst[n] = (pSrc[n])^{1/3}, 0 \le n < len.$

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pSrc or pDst pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.

Example

The example below shows how to use the function ippsCbrt.

```
IppStatus ippsCbrt_32f_A21_sample(void) {
    const Ipp32f x[4] = {6456.801, 4932.096, -6517.838, 7178.869};
    Ipp32f y[4];
    IppStatus st = ippsCbrt_32f_A21( x, y, 4 );


    printf(" ippsCbrt_32f_A21:\n");
    printf(" x = %.3f %.3f %.3f %.3f \n", x[0], x[1], x[2], x[3]);
    printf(" y = %.3f %.3f %.3f %.3f \n", y[0], y[1], y[2], y[3]);
    return st;
}
```

Output:

```
ippsCbrt_32f_A21:
x = 6456.801 4932.096 -6517.838 7178.869
y = 18.621 17.022 -18.680 19.291
```

InvCbrt

Computes inverse cube root of each vector element.

Syntax

```
IppStatus ippsInvCbrt_32f_A11 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsInvCbrt_32f_A21 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsInvCbrt_32f_A24 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsInvCbrt_64f_A26 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsInvCbrt_64f_A50 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsInvCbrt_64f_A53 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
```

Include Files

ippvm.h

Domain Dependencies

Headers: ippcore.h

Libraries: ippcore.lib

Parameters

pSrc	Pointer to the source vector.
pDst	Pointer to the destination vector.
len	Number of elements in the vectors.

Description

This function computes inverse cube root of each element of pSrc and stores the result in the corresponding element of pDst.

For single precision data:

function flavor ippsInvCbrt_32f_A11 guarantees 11 correctly rounded bits of significand, or at least 3 exact decimal digits;

function flavor ippsInvCbrt_32f_A21 guarantees 21 correctly rounded bits of significand, or 4 ulps, or about 6 exact decimal digits;

function flavor ippsInvCbrt_32f_A24 guarantees 24 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

For double precision data:

function flavor ippsInvCbrt_64f_A26 guarantees 26 correctly rounded bits of significand, or 6.7E+7 ulps, or approximately 8 exact decimal digits;

function flavor ippsInvCbrt_64f_A50 guarantees 50 correctly rounded bits of significand, or 4 ulps, or approximately 15 exact decimal digits;

function flavor ippsInvCbrt_64f_A53 guarantees 53 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

The computation is performed as follows:

$pDst[n] = (pSrc[n])^{-1/3}$, $0 \le n < len$.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pSrc or pDst pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.
IppStsSingularity	Indicates a warning that the argument is the singularity point, that is, at least one of the elements of pSrc is equal to 0.

Example

The example below shows how to use the function ippsInvCbrt.

```
IppStatus ippsInvCbrt_32f_A21_sample(void) {
    const Ipp32f x[4] = {914.120, 3644.584, 1473.214, 1659.070};
    Ipp32f y[4];
    IppStatus st = ippsInvCbrt_32f_A21( x, y, 4 );
```

```
    printf(" ippsInvCbrt_32f_A21:\n");
    printf(" x = %.3f %.3f %.3f %.3f \n", x[0], x[1], x[2], x[3]);
    printf(" y = %.3f %.3f %.3f %.3f \n", y[0], y[1], y[2], y[3]);
    return st;
}
```

Output:

```
ippsInvCbrt_32f_A21:
x = 914.120 3644.584 1473.214 1659.070
y = 0.103 0.065 0.088 0.084
```

Pow2o3

Computes the value of each vector element raised to the power of 2/3.

Syntax

```
IppStatus ippsPow2o3_32f_A11 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsPow2o3_32f_A21 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsPow2o3_32f_A24 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsPow2o3_64f_A26 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsPow2o3_64f_A50 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsPow2o3_64f_A53 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
```

Include Files

`ippvm.h`

Domain Dependencies

Headers: `ippcore.h`

Libraries: `ippcore.lib`

Parameters

`pSrc`	Pointer to the source vector.
`pDst`	Pointer to the destination vector.
`len`	Number of elements in the vectors.

Description

This function computes the value of each vector element of the vector `pSrc` raised to 2/3 power and stores the result in the corresponding element of the vector `pDst`.

For single precision data:

function flavor `ippsPow2o3_32f_A11` guarantees 11 correctly rounded bits of significand, or at least 3 exact decimal digits;

function flavor `ippsPow2o3_32f_A21` guarantees 21 correctly rounded bits of significand, or 4 ulps, or about 6 exact decimal digits;

function flavor ippsPow2o3_32f_A24 guarantees 24 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

For double precision data:

function flavor ippsPow2o3_64f_A26 guarantees 26 correctly rounded bits of significand, or 6.7E+7 ulps, or approximately 8 exact decimal digits;

function flavor ippsPow2o3_64f_A50 guarantees 50 correctly rounded bits of significand, or 4 ulps, or approximately 15 exact decimal digits;

function flavor ippsPow2o3_64f_A53 guarantees 53 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pSrc or pDst pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.

Pow3o2

Computes the value of each vector element raised to the power of 3/2.

Syntax

```
IppStatus ippsPow3o2_32f_A11 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsPow3o2_32f_A21 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsPow3o2_32f_A24 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsPow3o2_64f_A26 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsPow3o2_64f_A50 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsPow3o2_64f_A53 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
```

Include Files

ippvm.h

Domain Dependencies

Headers: ippcore.h

Libraries: ippcore.lib

Parameters

pSrc	Pointer to the source vector.
pDst	Pointer to the destination vector.
len	Number of elements in the vectors.

Description

This function computes the value of each vector element of the vector `pSrc` raised to 3/2 power and stores the result in the corresponding element of the vector `pDst`.

For single precision data:

function flavor `ippsPow3o2_32f_A11` guarantees 11 correctly rounded bits of significand, or at least 3 exact decimal digits;

function flavor `ippsPow3o2_32f_A21` guarantees 21 correctly rounded bits of significand, or 4 ulps, or about 6 exact decimal digits;

function flavor `ippsPow3o2_32f_A24` guarantees 24 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

For double precision data:

function flavor `ippsPow3o2_64f_A26` guarantees 26 correctly rounded bits of significand, or 6.7E+7 ulps, or approximately 8 exact decimal digits;

function flavor `ippsPow3o2_64f_A50` guarantees 50 correctly rounded bits of significand, or 4 ulps, or approximately 15 exact decimal digits;

function flavor `ippsPow3o2_64f_A53` guarantees 53 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when `pSrc` or `pDst` pointer is `NULL`.
`ippStsSizeErr`	Indicates an error when `len` is less than or equal to 0.
`IppStsDomain`	Indicates a warning that the argument is out of the function domain, that is, at least one of the elements of `pSrc` is less than 0.

Pow

Raises each element of the first vector to the power of corresponding element of the second vector.

Syntax

```
IppStatus ippsPow_32f_A11 (const Ipp32f* pSrc1, const Ipp32f* pSrc2, Ipp32f* pDst,
Ipp32s len);
IppStatus ippsPow_32f_A21 (const Ipp32f* pSrc1, const Ipp32f* pSrc2, Ipp32f* pDst,
Ipp32s len);
IppStatus ippsPow_32f_A24 (const Ipp32f* pSrc1, const Ipp32f* pSrc2, Ipp32f* pDst,
Ipp32s len);
IppStatus ippsPow_64f_A26 (const Ipp64f* pSrc1, const Ipp64f* pSrc2, Ipp64f* pDst,
Ipp32s len);
IppStatus ippsPow_64f_A50 (const Ipp64f* pSrc1, const Ipp64f* pSrc2, Ipp64f* pDst,
Ipp32s len);
```

```
IppStatus ippsPow_64f_A53 (const Ipp64f* pSrc1, const Ipp64f* pSrc2, Ipp64f* pDst,
Ipp32s len);

IppStatus ippsPow_32fc_A11 (const Ipp32fc* pSrc1, const Ipp32fc* pSrc2, Ipp32fc* pDst,
Ipp32s len);

IppStatus ippsPow_32fc_A21 (const Ipp32fc* pSrc1, const Ipp32fc* pSrc2, Ipp32fc* pDst,
Ipp32s len);

IppStatus ippsPow_32fc_A24 (const Ipp32fc* pSrc1, const Ipp32fc* pSrc2, Ipp32fc* pDst,
Ipp32s len);

IppStatus ippsPow_64fc_A26 (const Ipp64fc* pSrc1, const Ipp64fc* pSrc2, Ipp64fc* pDst,
Ipp32s len);

IppStatus ippsPow_64fc_A50 (const Ipp64fc* pSrc1, const Ipp64fc* pSrc2, Ipp64fc* pDst,
Ipp32s len);

IppStatus ippsPow_64fc_A53 (const Ipp64fc* pSrc1, const Ipp64fc* pSrc2, Ipp64fc* pDst,
Ipp32s len);
```

Include Files

`ippvm.h`

Domain Dependencies

Headers: `ippcore.h`

Libraries: `ippcore.lib`

Parameters

`pSrc1`	Pointer to the first source vector.
`pSrc2`	Pointer to the second source vector.
`pDst`	Pointer to the destination vector.
`len`	Number of elements in the vectors.

Description

This function raises each element of vector `pSrc1` to the power of the corresponding element of the vector `pSrc2` and stores the result in the corresponding element of `pDst`.

For single precision data:

function flavors `ippsPow_32f_A11` and `ippsPow_32cf_A11` guarantee 11 correctly rounded bits of significand, or at least 3 exact decimal digits;

function flavors `ippsPow_32f_A21` and `ippsPow_32fc_A21` guarantee 21 correctly rounded bits of significand, or 4 ulps, or about 6 exact decimal digits;

function flavors `ippsPow_32f_A24` and `ippsPow_32fc_A24` guarantee 24 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

For double precision data:

function flavors `ippsPow_64f_A26` and `ippsPow_64fc_A26` guarantee 26 correctly rounded bits of significand, or 6.7E+7 ulps, or approximately 8 exact decimal digits;

function flavors ippsPow_64f_A50 and ippsPow_64fc_A50 guarantee 50 correctly rounded bits of significand, or 4 ulps, or approximately 15 exact decimal digits;

function flavors ippsPow_64f_A53 and ippsPow_64fc_A53 guarantee 53 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

Note that for ippsPow complex functions there may be argument ranges where the accuracy specification does not hold.

The computation is performed as follows:

pDst[n] = (pSrc1[n])$^{pSrc2[n]}$, 0 ≤ n <len.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pSrc1 or pSrc2 or pDst pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.
IppStsDomain	In real functions, indicates a warning that the argument is out of the function domain, that is, at least one pair of the source elements meets the following condition: element of pSrc1 is finite, less than 0, and element of pSrc2 is finite, non-integer.
IppStsSingularity	In real functions, indicates a warning that the argument is the singularity point, that is, at least one pair of the elements is as follows: element of pSrc1 is equal to 0, and element of pSrc2 is integer and less than 0.

Example

The example below shows how to use the function ippsPow.

```
IppStatus ippsPow_32f_A21_sample(void) {
    const Ipp32f x1[4] = {0.483, 0.565, 0.776, 0.252};
    const Ipp32f x2[4] = {0.823, 0.991, 0.411, 0.692};
    Ipp32f y[4];
    IppStatus st = ippsPow_32f_A21( x1, x2, y, 4 );


    printf(" ippsPow_32f_A21:\n");
    printf(" x1 = %.3f %.3f %.3f %.3f \n", x1[0], x1[1], x1[2], x1[3]);
    printf(" x2 = %.3f %.3f %.3f %.3f \n", x2[0], x2[1], x2[2], x2[3]);
    printf(" y  = %.3f %.3f %.3f %.3f \n", y[0],  y[1],  y[2],  y[3]);
    return st;
}
```

Output:

```
ippsPow_32f_A21:
x1 = 0.483 0.565 0.776 0.252
x2 = 0.823 0.991 0.411 0.692
y  = 0.549 0.568 0.901 0.386
```

Powx

Raises each element of a vector to a constant power.

Syntax

```
IppStatus ippsPowx_32f_A11 (const Ipp32f* pSrc1, const Ipp32f ConstValue, Ipp32f* pDst,
Ipp32s len);

IppStatus ippsPowx_32f_A21 (const Ipp32f* pSrc1, const Ipp32f ConstValue, Ipp32f* pDst,
Ipp32s len);

IppStatus ippsPowx_32f_A24 (const Ipp32f* pSrc1, const Ipp32f ConstValue, Ipp32f* pDst,
Ipp32s len);

IppStatus ippsPowx_64f_A26 (const Ipp64f* pSrc1, const Ipp64f ConstValue, Ipp64f* pDst,
Ipp32s len);

IppStatus ippsPowx_64f_A50 (const Ipp64f* pSrc1, const Ipp64f ConstValue, Ipp64f* pDst,
Ipp32s len);

IppStatus ippsPowx_64f_A53 (const Ipp64f* pSrc1, const Ipp64f ConstValue, Ipp64f* pDst,
Ipp32s len);

IppStatus ippsPowx_32fc_A11 (const Ipp32fc* pSrc1, const Ipp32fc ConstValue, Ipp32fc*
pDst, Ipp32s len);

IppStatus ippsPowx_32fc_A21 (const Ipp32fc* pSrc1, const Ipp32fc ConstValue, Ipp32fc*
pDst, Ipp32s len);

IppStatus ippsPowx_32fc_A24 (const Ipp32fc* pSrc1, const Ipp32fc ConstValue, Ipp32fc*
pDst, Ipp32s len);

IppStatus ippsPowx_64fc_A26 (const Ipp64fc* pSrc1, const Ipp64fc ConstValue, Ipp64fc*
pDst, Ipp32s len);

IppStatus ippsPowx_64fc_A50 (const Ipp64fc* pSrc1, const Ipp64fc ConstValue, Ipp64fc*
pDst, Ipp32s len);

IppStatus ippsPowx_64fc_A53 (const Ipp64fc* pSrc1, const Ipp64fc ConstValue, Ipp64fc*
pDst, Ipp32s len);
```

Include Files

`ippvm.h`

Domain Dependencies

Headers: `ippcore.h`

Libraries: `ippcore.lib`

Parameters

`pSrc1`	Pointer to the source vector.
`ConstValue`	Constant value.
`pDst`	Pointer to the destination vector.

len	Number of elements in the vectors.

Description

This function raises each element of the vector pSrc1 to the constant power ConstValue and stores the result in the corresponding element of pDst.

For single precision data:

function flavors ippsPowx_32f_A11 and ippsPowx_32cf_A11 guarantee 11 correctly rounded bits of significand, or at least 3 exact decimal digits;

function flavors ippsPowx_32f_A21 and ippsPowx_32fc_A21 guarantee 21 correctly rounded bits of significand, or 4 ulps, or about 6 exact decimal digits;

function flavors ippsPowx_32f_A24 and ippsPowx_32fc_A24 guarantee 24 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

For double precision data:

function flavors ippsPowx_64f_A26 and ippsPowx_64fc_A26 guarantee 26 correctly rounded bits of significand, or 6.7E+7 ulps, or approximately 8 exact decimal digits;

function flavors ippsPowx_64f_A50 and ippsPowx_64fc_A50 guarantee 50 correctly rounded bits of significand, or 4 ulps, or approximately 15 exact decimal digits;

function flavors ippsPowx_64f_A53 and ippsPowx_64fc_A53 guarantee 53 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

Note that for ippsPowx complex functions there may be argument ranges where the accuracy specification does not hold.

The computation is performed as follows:

pDst[n] = (pSrc[n])$^{\text{ConstValue}}$, 0 ≤ n <len.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pSrc1 or pDst pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.
ippStsDomain	In real functions, indicates a warning that the argument is out of the function domain, that is, at least one pair of the elements meets the following condition: element of pSrc1 is finite, less than 0, and ConstValue is finite, non-integer.
IppStsSingularity	In real functions, indicates a warning that the argument is the singularity point, that is, at least one pair of the elements is as follows: element of pSrc1 is equal to 0, and ConstValue is integer and less than 0.

Example

The example below shows how to use the function `ippsPowx`.

```
IppStatus ippsPowx_32f_A21_sample(void) {
    const Ipp32f x1[4] = {0.483, 0.565, 0.776, 0.252};
    const Ipp32f x2 = 0.823;
    Ipp32f y[4];
    IppStatus st = ippsPowx_32f_A21( x1, x2, y, 4 );


    printf(" ippsPowx_32f_A21:\n");
    printf(" x1 = %.3f %.3f %.3f %.3f \n", x1[0], x1[1], x1[2], x1[3]);
    printf(" x2 = %.3f \n", x2);
    printf(" y  = %.3f %.3f %.3f %.3f \n", y[0],  y[1],  y[2],  y[3]);
    return st;
}
```

Output results:

```
ippsPowx_32f_A21:
x1 = 0.483 0.565 0.776 0.252
x2 = 0.823
y  = 0.549 0.625 0.812 0.322
```

Hypot

Computes a square root of sum of two squared elements.

Syntax

```
IppStatus ippsHypot_32f_A11 (const Ipp32f* pSrc1, const Ipp32f pSrc2, Ipp32f* pDst,
Ipp32s len);

IppStatus ippsHypot_32f_A21 (const Ipp32f* pSrc1, const Ipp32f* pSrc2, Ipp32f* pDst,
Ipp32s len);

IppStatus ippsHypot_32f_A24 (const Ipp32f* pSrc1, const Ipp32f* pSrc2, Ipp32f* pDst,
Ipp32s len);

IppStatus ippsHypot_64f_A26 (const Ipp64f* pSrc1, const Ipp64f* pSrc2, Ipp64f* pDst,
Ipp32s len);

IppStatus ippsHypot_64f_A50 (const Ipp64f* pSrc1, const Ipp64f* pSrc2, Ipp64f* pDst,
Ipp32s len);

IppStatus ippsHypot_64f_A53 (const Ipp64f* pSrc1, const Ipp64f* pSrc2, Ipp64f* pDst,
Ipp32s len);
```

Include Files

`ippvm.h`

Domain Dependencies

Headers: `ippcore.h`

Libraries: `ippcore.lib`

Parameters

pSrc1	Pointer to the first source vector.
pSrc2	Pointer to the second source vector.
pDst	Pointer to the destination vector.
len	Number of elements in the vectors.

Description

This function computes square of each element of the pSrc1 and pSrc2 vectors, sums corresponding elements, computes square roots of each sum and stores the result in the corresponding element of pDst.

For single precision data:

function flavor ippsHypot_32f_A11 guarantees 11 correctly rounded bits of significand, or at least 3 exact decimal digits;

function flavor ippsHypot_32f_A21 guarantees 21 correctly rounded bits of significand, or 4 ulps, or about 6 exact decimal digits;

function flavor ippsHypot_32f_A24 guarantees 24 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

For double precision data:

function flavor ippsHypot_64f_A26 guarantees 26 correctly rounded bits of significand, or 6.7E+7 ulps, or approximately 8 exact decimal digits;

function flavor ippsHypot_64f_A50 guarantees 50 correctly rounded bits of significand, or 4 ulps, or approximately 15 exact decimal digits;

function flavor ippsHypot_64f_A53 guarantees 53 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

The computation is performed as follows:

$pDst[n] = ((pSrc1[n])^2 + (pSrc2[n])^2)^{1/2}, 0 \le n < len.$

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pSrc1 or pSrc2 or pDst pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.

Example

The example below shows how to use the function ippsHypot.

```
IppStatus ippsHypot_32f_A21_sample(void) {
    const Ipp32f x1[4] = {0.483, 0.565, 0.776, 0.252}
    const Ipp32f x2[4] = {0.823, 0.991, 0.411, 0.692};
    Ipp32f y[4];
    IppStatus st = ippsHypot_32f_A21( x1, x2, y, 4 );
```

```
    printf(" ippsHypot_32f_A21:\n");
    printf(" x1 = %.3f %.3f %.3f %.3f \n", x1[0], x1[1], x1[2], x1[3]);
    printf(" x2 = %.3f %.3f %.3f %.3f \n", x2[0], x2[1], x2[2], x2[3]);
    printf(" y  = %.3f %.3f %.3f %.3f \n", y[0],  y[1],  y[2],  y[3]);
    return st;
}
```

Output:

```
ippsHypot_32f_A21:
x1 = 0.483 0.565 0.776 0.252
x2 = 0.823 0.991 0.411 0.692
y  = 0.954 1.141 0.878 0.736
```

Exponential and Logarithmic Functions

- Exp
- Expm1
- Ln
- Log10
- Log1p

Exp

Raises `e` to the power of each vector element.

Syntax

```
IppStatus ippsExp_32f_A11 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsExp_32f_A21 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsExp_32f_A24 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsExp_64f_A26 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsExp_64f_A50 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsExp_64f_A53 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsExp_32fc_A11 (const Ipp32fc* pSrc, Ipp32fc* pDst, Ipp32s len);
IppStatus ippsExp_32fc_A21 (const Ipp32fc* pSrc, Ipp32fc* pDst, Ipp32s len);
IppStatus ippsExp_32fc_A24 (const Ipp32fc* pSrc, Ipp32fc* pDst, Ipp32s len);
IppStatus ippsExp_64fc_A26 (const Ipp64fc* pSrc, Ipp64fc* pDst, Ipp32s len);
IppStatus ippsExp_64fc_A50 (const Ipp64fc* pSrc, Ipp64fc* pDst, Ipp32s len);
IppStatus ippsExp_64fc_A53 (const Ipp64fc* pSrc, Ipp64fc* pDst, Ipp32s len);
```

Include Files

`ippvm.h`

Domain Dependencies

Headers: `ippcore.h`

Libraries: `ippcore.lib`

Parameters

pSrc	Pointer to the source vector.
pDst	Pointer to the destination vector.
len	Number of elements in the vectors.

Description

This function raises e to the power of each element of pSrc and stores the result in the corresponding element of pDst.

For single precision data:

function flavors ippsExp_32f_A11 and ippsExp_32cf_A11 guarantee 11 correctly rounded bits of significand, or at least 3 exact decimal digits;

function flavors ippsExp_32f_A21 and ippsExp_32fc_A21 guarantee 21 correctly rounded bits of significand, or 4 ulps, or about 6 exact decimal digits;

function flavors ippsExp_32f_A24 and ippsExp_32fc_A24 guarantee 24 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

For double precision data:

function flavors ippsExp_64f_A26 and ippsExp_64fc_A26 guarantee 26 correctly rounded bits of significand, or 6.7E+7 ulps, or approximately 8 exact decimal digits;

function flavors ippsExp_64f_A50 and ippsExp_64fc_A50 guarantee 50 correctly rounded bits of significand, or 4 ulps, or approximately 15 exact decimal digits;

function flavors ippsExp_64f_A53 and ippsExp_64fc_A53 guarantee 53 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

The computation is performed as follows:

$pDst[n] = e^{pSrc[n]}, 0 \le n < len.$

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pSrc or pDst pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.
IppStsOverflow	In real functions, indicates a warning that the function overflows, that is, at least one of elements of pSrc is greater than Ln(FPMAX), where FPMAX is the maximum representable floating-point number.
IppStsUnderflow	In real functions, indicates a warning that the function underflows, that is, at least one of elements of pSrc is less than Ln(FPMIN), where FPMIN is the minimum positive floating-point value.

Example

The example below shows how to use the function `ippsExp`.

```
IppStatus ippsExp_32f_A21_sample(void) {
    const Ipp32f x[4] = {4.885, -0.543, -3.809, -4.953};
    Ipp32f y[4];
    IppStatus st = ippsExp_32f_A21( x, y, 4 );


    printf(" ippsExp_32f_A21:\n");
    printf(" x = %.3f %.3f %.3f %.3f \n", x[0], x[1], x[2], x[3]);
    printf(" y = %.3f %.3f %.3f %.3f \n", y[0], y[1], y[2], y[3]);
    return st;
}
```

Output:

```
ippsExp_32f_A21:
x = 4.885 -0.543 -3.809 -4.953
y = 132.324 0.581 0.022 0.007
```

Expm1

Computes `e` raised to the power of each vector element and decreased by 1.

Syntax

```
IppStatus ippsExpm1_32f_A11 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsExpm1_32f_A21 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsExpm1_32f_A24 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsExpm1_64f_A26 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsExpm1_64f_A50 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsExpm1_64f_A53 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
```

Include Files

`ippvm.h`

Domain Dependencies

Headers: `ippcore.h`

Libraries: `ippcore.lib`

Parameters

`pSrc`	Pointer to the source vector.
`pDst`	Pointer to the destination vector.
`len`	Number of elements in the vectors.

Description

This function computes `e` raised to the power of each vector element of `pSrc` and decreased by 1, and stores the result in the corresponding element of the vector `pDst`.

For single precision data:

function flavor `ippsExpm1_32f_A11` guarantees 11 correctly rounded bits of significand, or at least 3 exact decimal digits;

function flavor `ippsExpm1_32f_A21` guarantees 21 correctly rounded bits of significand, or 4 ulps, or about 6 exact decimal digits;

function flavor `ippsExpm1_32f_A24` guarantees 24 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

For double precision data:

function flavor `ippsExpm1_64f_A26` guarantees 26 correctly rounded bits of significand, or 6.7E+7 ulps, or approximately 8 exact decimal digits;

function flavor `ippsExpm1_64f_A50` guarantees 50 correctly rounded bits of significand, or 4 ulps, or approximately 15 exact decimal digits;

function flavor `ippsExpm1_64f_A53` guarantees 53 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when `pSrc` or `pDst` pointer is `NULL`.
`ippStsSizeErr`	Indicates an error when `len` is less than or equal to 0.
`IppStsOverflow`	Indicates a warning that the function overflows, that is, at least one of elements of `pSrc` is greater than `Ln(FPMAX)`, where `FPMAX` is the maximum representable floating-point number.

Ln

Computes natural logarithm of each vector element.

Syntax

```
IppStatus ippsLn_32f_A11 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsLn_32f_A21 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsLn_32f_A24 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsLn_64f_A26 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsLn_64f_A50 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsLn_64f_A53 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsLn_32fc_A11 (const Ipp32fc* pSrc, Ipp32fc* pDst, Ipp32s len);
IppStatus ippsLn_32fc_A21 (const Ipp32fc* pSrc, Ipp32fc* pDst, Ipp32s len);
IppStatus ippsLn_32fc_A24 (const Ipp32fc* pSrc, Ipp32fc* pDst, Ipp32s len);
IppStatus ippsLn_64fc_A26 (const Ipp64fc* pSrc, Ipp64fc* pDst, Ipp32s len);
```

```
IppStatus ippsLn_64fc_A50 (const Ipp64fc* pSrc, Ipp64fc* pDst, Ipp32s len);
IppStatus ippsLn_64fc_A53 (const Ipp64fc* pSrc, Ipp64fc* pDst, Ipp32s len);
```

Include Files

`ippvm.h`

Domain Dependencies

Headers: `ippcore.h`

Libraries: `ippcore.lib`

Parameters

`pSrc`	Pointer to the source vector.
`pDst`	Pointer to the destination vector.
`len`	Number of elements in the vectors.

Description

This function computes a natural logarithm of each element of `pSrc` and stores the result in the corresponding element of `pDst`.

For single precision data:

function flavors `ippsLn_32f_A11` and `ippsLn_32cf_A11` guarantee 11 correctly rounded bits of significand, or at least 3 exact decimal digits;

function flavors `ippsLn_32f_A21` and `ippsLn_32fc_A21` guarantee 21 correctly rounded bits of significand, or 4 ulps, or about 6 exact decimal digits;

function flavors `ippsLn_32f_A24` and `ippsLn_32fc_A24` guarantee 24 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

For double precision data:

function flavors `ippsLn_64f_A26` and `ippsLn_64fc_A26` guarantee 26 correctly rounded bits of significand, or 6.7E+7 ulps, or approximately 8 exact decimal digits;

function flavors `ippsLn_64f_A50` and `ippsLn_64fc_A50` guarantee 50 correctly rounded bits of significand, or 4 ulps, or approximately 15 exact decimal digits;

function flavors `ippsLn_64f_A53` and `ippsLn_64fc_A53` guarantee 53 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

The computation is performed as follows:

$pDst[n] = \log_e(pSrc[n]), 0 \le n < len$.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when `pSrc` or `pDst` pointer is `NULL`.

ippStsSizeErr	Indicates an error when len is less than or equal to 0.
IppStsDomain	In real functions, indicates a warning that the argument is out of the function domain, that is, at least one of the elements of pSrc is less than 0.
IppStsSingularity	In real functions, indicates a warning that the argument is the singularity point, that is, at least one of the elements of pSrc is equal to 0.

Example

The example below shows how to use the function ippsLn.

```
IppStatus ippsLn_32f_A21_sample(void) {
    const Ipp32f x[4] = {0.188, 3.841, 5.363, 5.755};
    Ipp32f y[4];
    IppStatus st = ippsLn_32f_A21( x, y, 4 );


    printf(" ippsLn_32f_A21:\n");
    printf(" x = %.3f %.3f %.3f %.3f \n", x[0], x[1], x[2], x[3]);
    printf(" y = %.3f %.3f %.3f %.3f \n", y[0], y[1], y[2], y[3]);
    return st;
}
```

Output:

```
ippsLn_32f_A21:
x = 0.188 3.841 5.363 5.755
y = -1.670 1.346 1.680 1.750
```

Log10

Computes common logarithm of each vector element.

Syntax

```
IppStatus ippsLog10_32f_A11 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsLog10_32f_A21 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsLog10_32f_A24 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsLog10_64f_A26 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsLog10_64f_A50 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsLog10_64f_A53 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsLog10_32fc_A11 (const Ipp32fc* pSrc, Ipp32fc* pDst, Ipp32s len);
IppStatus ippsLog10_32fc_A21 (const Ipp32fc* pSrc, Ipp32fc* pDst, Ipp32s len);
IppStatus ippsLog10_32fc_A24 (const Ipp32fc* pSrc, Ipp32fc* pDst, Ipp32s len);
IppStatus ippsLog10_64fc_A26 (const Ipp64fc* pSrc, Ipp64fc* pDst, Ipp32s len);
IppStatus ippsLog10_64fc_A50 (const Ipp64fc* pSrc, Ipp64fc* pDst, Ipp32s len);
IppStatus ippsLog10_64fc_A53 (const Ipp64fc* pSrc, Ipp64fc* pDst, Ipp32s len);
```

Include Files

ippvm.h

Domain Dependencies

Headers: ippcore.h

Libraries: ippcore.lib

Parameters

pSrc	Pointer to the source vector.
pDst	Pointer to the destination vector.
len	Number of elements in the vectors.

Description

This function computes a natural logarithm of each element of pSrc and stores the result in the corresponding element of pDst.

For single precision data:

function flavors ippsLog10_32f_A11 and ippsLog10_32cf_A11 guarantee 11 correctly rounded bits of significand, or at least 3 exact decimal digits;

function flavors ippsLog10_32f_A21 and ippsLog10_32fc_A21 guarantee 21 correctly rounded bits of significand, or 4 ulps, or about 6 exact decimal digits;

function flavors ippsLog10_32f_A24 and ippsLog10_32fc_A24 guarantee 24 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

For double precision data:

function flavors ippsLog10_64f_A26 and ippsLog10_64fc_A26 guarantee 26 correctly rounded bits of significand, or 6.7E+7 ulps, or approximately 8 exact decimal digits;

function flavors ippsLog10_64f_A50 and ippsLog10_64fc_A50 guarantee 50 correctly rounded bits of significand, or 4 ulps, or approximately 15 exact decimal digits;

function flavors ippsLog10_64f_A53 and ippsLog10_64fc_A53 guarantee 53 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

The computation is performed as follows:

$pDst[n] = \log_{10}(pSrc[n]), 0 \le n < len.$

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pSrc or pDst pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.

ippStsDomain	In real functions, indicates a warning that the argument is out of the function domain, that is, at least one of the elements of pSrc is less than 0.
ippStsSingularity	In real functions, indicates a warning that the argument is the singularity point, that is, at least one of the elements of pSrc is equal to 0.

Example

The example below shows how to use the function ippsLog10.

```
IppStatus ippsLog10_32f_A21_sample(void) {
    const Ipp32f x[4] = {6.057, 6.111, 1.746, 6.664};
    Ipp32f y[4];
    IppStatus st = ippsLog10_32f_A21( x, y, 4 );


    printf(" ippsLog10_32f_A21:\n");
    printf(" x = %.3f %.3f %.3f %.3f \n", x[0], x[1], x[2], x[3]);
    printf(" y = %.3f %.3f %.3f %.3f \n", y[0], y[1], y[2], y[3]);
    return st;
}
```

Output:

```
ippsLog10_32f_A21:
x = 6.057 6.111 1.746 6.664
y = 0.782 0.786 0.242 0.824
```

Log1p

Computes natural logarithm of each vector element decreased by 1.

Syntax

```
IppStatus ippsLog1p_32f_A11 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsLog1p_32f_A21 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsLog1p_32f_A24 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsLog1p_64f_A26 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsLog1p_64f_A50 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsLog1p_64f_A53 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
```

Include Files

ippvm.h

Domain Dependencies

Headers: ippcore.h

Libraries: ippcore.lib

Parameters

pSrc	Pointer to the source vector.
pDst	Pointer to the destination vector.
len	Number of elements in the vectors.

Description

This function computes a natural logarithm of each vector element of pSrc decreased by 1, and stores the result in the corresponding element of the vector pDst.

For single precision data:

function flavor ippsLog1p_32f_A11 guarantees 11 correctly rounded bits of significand, or at least 3 exact decimal digits;

function flavor ippsLog1p_32f_A21 guarantees 21 correctly rounded bits of significand, or 4 ulps, or about 6 exact decimal digits;

function flavor ippsLog1p_32f_A24 guarantees 24 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

For double precision data:

function flavor ippsLog1p_64f_A26 guarantees 26 correctly rounded bits of significand, or 6.7E+7 ulps, or approximately 8 exact decimal digits;

function flavor ippsLog1p_64f_A50 guarantees 50 correctly rounded bits of significand, or 4 ulps, or approximately 15 exact decimal digits;

function flavor ippsLog1p_64f_A53 guarantees 53 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pSrc or pDst pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.
IppStsDomain	Indicates a warning that the argument is out of the function domain, that is, at least one of the elements of pSrc is less than -1.
IppStsSingularity	Indicates a warning that the argument is the singularity point, that is, at least one of the elements of pSrc is equal to -1.

Trigonometric Functions

- Cos
- Sin
- SinCos
- CIS

- Tan
- Acos
- Asin
- Atan
- Atan2

Cos

Computes cosine of each vector element.

Syntax

```
IppStatus ippsCos_32f_A11 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsCos_32f_A21 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsCos_32f_A24 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsCos_64f_A26 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsCos_64f_A50 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsCos_64f_A53 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsCos_32fc_A11 (const Ipp32fc* pSrc, Ipp32fc* pDst, Ipp32s len);
IppStatus ippsCos_32fc_A21 (const Ipp32fc* pSrc, Ipp32fc* pDst, Ipp32s len);
IppStatus ippsCos_32fc_A24 (const Ipp32fc* pSrc, Ipp32fc* pDst, Ipp32s len);
IppStatus ippsCos_64fc_A26 (const Ipp64fc* pSrc, Ipp64fc* pDst, Ipp32s len);
IppStatus ippsCos_64fc_A50 (const Ipp64fc* pSrc, Ipp64fc* pDst, Ipp32s len);
IppStatus ippsCos_64fc_A53 (const Ipp64fc* pSrc, Ipp64fc* pDst, Ipp32s len);
```

Include Files

`ippvm.h`

Domain Dependencies

Headers: `ippcore.h`

Libraries: `ippcore.lib`

Parameters

`pSrc`	Pointer to the source vector.
`pDst`	Pointer to the destination vector.
`len`	Number of elements in the vectors.

Description

This function computes a cosine of each element of `pSrc`, and stores the result in the corresponding element of `pDst`.

For single precision data:

function flavors `ippsCos_32f_A11` and `ippsCos_32cf_A11` guarantee 11 correctly rounded bits of significand, or at least 3 exact decimal digits;

function flavors ippsCos_32f_A21 and ippsCos_32fc_A21 guarantee 21 correctly rounded bits of significand, or 4 ulps, or about 6 exact decimal digits;

function flavors ippsCos_32f_A24 and ippsCos_32fc_A24 guarantee 24 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

For double precision data:

function flavors ippsCos_64f_A26 and ippsCos_64fc_A26 guarantee 26 correctly rounded bits of significand, or 6.7E+7 ulps, or approximately 8 exact decimal digits;

function flavors ippsCos_64f_A50 and ippsCos_64fc_A50 guarantee 50 correctly rounded bits of significand, or 4 ulps, or approximately 15 exact decimal digits;

function flavors ippsCos_64f_A53 and ippsCos_64fc_A53 guarantee 53 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

The computation is performed as follows:

pDst[n] = cos(pSrc[n]), 0 ≤ n <len.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pSrc or pDst pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.
IppStsDomain	In real functions, indicates a warning that the argument is out of the function domain, that is, at least one of the elements of pSrc is equal to ± INF.

Example

The example below shows how to use the function ippsCos.

```
IppStatus ippsCos_32f_A21_sample(void) {
    const Ipp32f x[4] = {-984.222, -2957.549, -8859.218, 2153.691};
    Ipp32f y[4];
    IppStatus st = ippsCos_32f_A21( x, y, 4 );


    printf(" ippsCos_32f_A21:\n");
    printf(" x = %.3f %.3f %.3f %.3f \n", x[0], x[1], x[2], x[3]);
    printf(" y = %.3f %.3f %.3f %.3f \n", y[0], y[1], y[2], y[3]);
    return st;
}
```

Output:

```
ippsCos_32f_A21:
x = -984.222 -2957.549 -8859.218 2153.691
y = -0.619 -0.258 0.997 0.129
```

Sin

Computes sine of each vector element.

Syntax

```
IppStatus ippsSin_32f_A11 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsSin_32f_A21 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsSin_32f_A24 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsSin_64f_A26 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsSin_64f_A50 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsSin_64f_A53 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsSin_32fc_A11 (const Ipp32fc* pSrc, Ipp32fc* pDst, Ipp32s len);
IppStatus ippsSin_32fc_A21 (const Ipp32fc* pSrc, Ipp32fc* pDst, Ipp32s len);
IppStatus ippsSin_32fc_A24 (const Ipp32fc* pSrc, Ipp32fc* pDst, Ipp32s len);
IppStatus ippsSin_64fc_A26 (const Ipp64fc* pSrc, Ipp64fc* pDst, Ipp32s len);
IppStatus ippsSin_64fc_A50 (const Ipp64fc* pSrc, Ipp64fc* pDst, Ipp32s len);
IppStatus ippsSin_64fc_A53 (const Ipp64fc* pSrc, Ipp64fc* pDst, Ipp32s len);
```

Include Files

`ippvm.h`

Domain Dependencies

Headers: `ippcore.h`

Libraries: `ippcore.lib`

Parameters

`pSrc`	Pointer to the source vector.
`pDst`	Pointer to the destination vector.
`len`	Number of elements in the vectors.

Description

This function computes a sine of each element of `pSrc`, and stores the result in the corresponding element of `pDst`.

For single precision data:

function flavors `ippsSin_32f_A11` and `ippsSin_32fc_A11` guarantee 11 correctly rounded bits of significand, or at least 3 exact decimal digits;

function flavors `ippsSin_32f_A21` and `ippsSin_32fc_A21` guarantee 21 correctly rounded bits of significand, or 4 ulps, or about 6 exact decimal digits;

function flavors `ippsSin_32f_A24` and `ippsSin_32fc_A24` guarantee 24 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

For double precision data:

function flavors `ippsSin_64f_A26` and `ippsSin_64fc_A26` guarantee 26 correctly rounded bits of significand, or 6.7E+7 ulps, or approximately 8 exact decimal digits;

function flavors ippsSin_64f_A50 and ippsSin_64fc_A50 guarantee 50 correctly rounded bits of significand, or 4 ulps, or approximately 15 exact decimal digits;

function flavors ippsSin_64f_A53 and ippsSin_64fc_A53 guarantee 53 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

The computation is performed as follows:

pDst[n] = sin(pSrc[n]), 0 ≤ n <len.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pSrc or pDst pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.
IppStsDomain	In real functions, indicates a warning that the argument is out of the function domain, that is, at least one of the elements of pSrc is equal to ± INF.

Example

The example below shows how to use the function ippsSin.

```
IppStatus ippsSin_32f_A21_sample(void) {
    const Ipp32f x[4] = {5666.372, 6052.125, 397.656, -3960.997};
    Ipp32f y[4];
    IppStatus st = ippsSin_32f_A21( x, y, 4 );


    printf(" ippsSin_32f_A21:\n");
    printf(" x = %.3f %.3f %.3f %.3f \n", x[0], x[1], x[2], x[3]);
    printf(" y = %.3f %.3f %.3f %.3f \n", y[0], y[1], y[2], y[3]);
    return st;
}
```

Output:

```
ippsSin_32f_A21:
x = 5666.372 6052.125 397.656 -3960.997
y = -0.873 0.988 0.970 -0.524
```

SinCos

Computes sine and cosine of each vector element.

Syntax

```
IppStatus ippsSinCos_32f_A11 (const Ipp32f* pSrc, Ipp32f* pDst1, Ipp32f* pDst2, Ipp32s len);
```

```
IppStatus ippsSinCos_32f_A21 (const Ipp32f* pSrc, Ipp32f* pDst1, Ipp32f* pDst2, Ipp32s len);
```

```
IppStatus ippsSinCos_32f_A24 (const Ipp32f* pSrc, Ipp32f* pDst1, Ipp32f* pDst2, Ipp32s
len);
IppStatus ippsSinCos_64f_A26 (const Ipp64f* pSrc, Ipp64f* pDst1, Ipp64f* pDst2, Ipp32s
len);
IppStatus ippsSinCos_64f_A50 (const Ipp64f* pSrc, Ipp64f* pDst1, Ipp64f* pDst2, Ipp32s
len);
IppStatus ippsSinCos_64f_A53 (const Ipp64f* pSrc, Ipp64f* pDst1, Ipp64f* pDst2, Ipp32s
len);
```

Include Files

`ippvm.h`

Domain Dependencies

Headers: `ippcore.h`

Libraries: `ippcore.lib`

Parameters

`pSrc`	Pointer to the first source vector.
`pDst1`	Pointer to the destination vector for sine values.
`pDst2`	Pointer to the destination vector for cosine values.
`len`	Number of elements in the vectors.

Description

This function computes sine of each element of `pSrc` and stores the result in the corresponding element of `pDst1`; computes cosine of each element of `pSrc` and stores the result in the corresponding element of `pDst2`.

For single precision data:

function flavor `ippsSinCos_32f_A11` guarantees 11 correctly rounded bits of significand, or at least 3 exact decimal digits;

function flavor `ippsSinCos_32f_A21` guarantees 21 correctly rounded bits of significand, or 4 ulps, or about 6 exact decimal digits;

function flavor `ippsSinCos_32f_A24` guarantees 24 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

For double precision data:

function flavor `ippsSinCos_64f_A26` guarantees 26 correctly rounded bits of significand, or 6.7E+7 ulps, or approximately 8 exact decimal digits;

function flavor `ippsSinCos_64f_A50` guarantees 50 correctly rounded bits of significand, or 4 ulps, or approximately 15 exact decimal digits;

function flavor `ippsSinCos_64f_A53` guarantees 53 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

The computation is performed as follows:

`pDst1[n] = sin(pSrc[n]),pDst2[n] = cos(pSrc[n]), 0 ≤ n <len.`

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pDst1 or pDst2 or pSrc pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.
IppStsDomain	In real functions, indicates a warning that the argument is out of the function domain, that is, at least one of the pSrc elements is equal to ± INF.

Example

The example below shows how to use the function ippsSinCos.

```
IppStatus ippsSinCos_32f_A21_sample(void) {
    const Ipp32f x[4] = {3857.845, -3939.024, -1468.856, -8592.486};
    Ipp32f y1[4];
    Ipp32f y2[4];
    IppStatus st = ippsSinCos_32f_A21( x, y1, y2, 4 );
    printf(" ippsSinCos_32f_A21:\n");
    printf(" x  = %.3f %.3f %.3f %.3f \n", x[0],  x[1],  x[2],  x[3]);
    printf(" y1 = %.3f %.3f %.3f %.3f \n", y1[0], y1[1], y1[2], y1[3]);
    printf(" y2 = %.3f %.3f %.3f %.3f \n", y2[0], y2[1], y2[2], y2[3]);
    return st;
}
```

Output results:

```
ippsSinCos_32f_A21:
x  = 3857.845 -3939.024 -1468.856 -8592.486
y1 = -0.031 0.508 0.987 0.228
y2 = 1.000 0.861 0.161 -0.974
```

CIS

Computes complex exponent of each vector element.

Syntax

```
IppStatus ippsCIS_32fc_A11 (const Ipp32f* pSrc, Ipp32fc* pDst, Ipp32s len);
IppStatus ippsCIS_32fc_A21 (const Ipp32f* pSrc, Ipp32fc* pDst, Ipp32s len);
IppStatus ippsCIS_32fc_A24 (const Ipp32f* pSrc, Ipp32fc* pDst, Ipp32s len);
IppStatus ippsCIS_64fc_A26 (const Ipp64f* pSrc, Ipp64fc* pDst, Ipp32s len);
IppStatus ippsCIS_64fc_A50 (const Ipp64f* pSrc, Ipp64fc* pDst, Ipp32s len);
IppStatus ippsCIS_64fc_A53 (const Ipp64f* pSrc, Ipp64fc* pDst, Ipp32s len);
```

Include Files

ippvm.h

Domain Dependencies

Headers: `ippcore.h`

Libraries: `ippcore.lib`

Parameters

`pSrc`	Pointer to the source vector.
`pDst`	Pointer to the destination vector.
`len`	Number of elements in the vectors.

Description

This function computes a complex exponent of each vector element of `pSrc` and stores the result in the corresponding element of the vector `pDst`.

For single precision data:

function flavor `ippsCIS_32fc_A11` guarantees 11 correctly rounded bits of significand, or at least 3 exact decimal digits;

function flavor `ippsCIS_32fc_A21` guarantees 21 correctly rounded bits of significand, or 4 ulps, or about 6 exact decimal digits;

function flavor `ippsCIS_32fc_A24` guarantees 24 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

For double precision data:

function flavor `ippsCIS_64fc_A26` guarantees 26 correctly rounded bits of significand, or 6.7E+7 ulps, or approximately 8 exact decimal digits;

function flavor `ippsCIS_64fc_A50` guarantees 50 correctly rounded bits of significand, or 4 ulps, or approximately 15 exact decimal digits;

function flavor `ippsCIS_64fc_A53` guarantees 53 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when `pSrc` or `pDst` pointer is `NULL`.
`ippStsSizeErr`	Indicates an error when `len` is less than or equal to 0.
`ippStsDomain`	Indicates a warning that the argument is out of the function domain, that is, at least one of the `pSrc` elements is equal to ± `INF`.

Tan

Computes tangent of each vector element.

Syntax

```
IppStatus ippsTan_32f_A11 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsTan_32f_A21 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsTan_32f_A24 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsTan_64f_A26 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsTan_64f_A50 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsTan_64f_A53 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsTan_32fc_A11 (const Ipp32fc* pSrc, Ipp32fc* pDst, Ipp32s len);
IppStatus ippsTan_32fc_A21 (const Ipp32fc* pSrc, Ipp32fc* pDst, Ipp32s len);
IppStatus ippsTan_32fc_A24 (const Ipp32fc* pSrc, Ipp32fc* pDst, Ipp32s len);
IppStatus ippsTan_64fc_A26 (const Ipp64fc* pSrc, Ipp64fc* pDst, Ipp32s len);
IppStatus ippsTan_64fc_A50 (const Ipp64fc* pSrc, Ipp64fc* pDst, Ipp32s len);
IppStatus ippsTan_64fc_A53 (const Ipp64fc* pSrc, Ipp64fc* pDst, Ipp32s len);
```

Include Files

`ippvm.h`

Domain Dependencies

Headers: `ippcore.h`

Libraries: `ippcore.lib`

Parameters

`pSrc`	Pointer to the source vector.
`pDst`	Pointer to the destination vector.
`len`	Number of elements in the vectors.

Description

This function computes the tangent of each element of `pSrc`, and stores the result in the corresponding element of `pDst`.

For single precision data:

function flavors `ippsTan_32f_A11` and `ippsTan_32cf_A11` guarantee 11 correctly rounded bits of significand, or at least 3 exact decimal digits;

function flavors `ippsTan_32f_A21` and `ippsTan_32fc_A21` guarantee 21 correctly rounded bits of significand, or 4 ulps, or about 6 exact decimal digits;

function flavors `ippsTan_32f_A24` and `ippsTan_32fc_A24` guarantee 24 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

For double precision data:

function flavors `ippsTan_64f_A26` and `ippsTan_64fc_A26` guarantee 26 correctly rounded bits of significand, or 6.7E+7 ulps, or approximately 8 exact decimal digits;

function flavors ippsTan_64f_A50 and ippsTan_64fc_A50 guarantee 50 correctly rounded bits of significand, or 4 ulps, or approximately 15 exact decimal digits;

function flavors ippsTan_64f_A53 and ippsTan_64fc_A53 guarantee 53 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

The computation is performed as follows:

pDst[n] = tan(pSrc[n]), 0 ≤ n <len.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pSrc or pDst pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.
IppStsDomain	In real functions, indicates a warning that the argument is out of the function domain, that is, at least one of the elements of pSrc is equal to ± INF.

Example

The example below shows how to use the function ippsTan.

```
IppStatus ippsTan_32f_A21_sample(void) {
    const Ipp32f x[4] = {7519.456, 4533.524, 9118.015, 8514.359};
    Ipp32f y[4];
    IppStatus st = ippsTan_32f_A21( x, y, 4 );


    printf(" ippsTan_32f_A21:\n");
    printf(" x = %.3f %.3f %.3f %.3f \n", x[0], x[1], x[2], x[3]);
    printf(" y = %.3f %.3f %.3f %.3f \n", y[0], y[1], y[2], y[3]);
    return st;
}
```

Output:

```
ippsTan_32f_A21:
x = 7519.456 4533.524 9118.015 8514.359
y = -18.656 0.209 2.028 0.750
```

Acos

Computes inverse cosine of each vector element.

Syntax

```
IppStatus ippsAcos_32f_A11 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsAcos_32f_A21 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsAcos_32f_A24 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsAcos_64f_A26 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
```

```
IppStatus ippsAcos_64f_A50 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsAcos_64f_A53 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsAcos_32fc_A11 (const Ipp32fc* pSrc, Ipp32fc* pDst, Ipp32s len);
IppStatus ippsAcos_32fc_A21 (const Ipp32fc* pSrc, Ipp32fc* pDst, Ipp32s len);
IppStatus ippsAcos_32fc_A24 (const Ipp32fc* pSrc, Ipp32fc* pDst, Ipp32s len);
IppStatus ippsAcos_64fc_A26 (const Ipp64fc* pSrc, Ipp64fc* pDst, Ipp32s len);
IppStatus ippsAcos_64fc_A50 (const Ipp64fc* pSrc, Ipp64fc* pDst, Ipp32s len);
IppStatus ippsAcos_64fc_A53 (const Ipp64fc* pSrc, Ipp64fc* pDst, Ipp32s len);
```

Include Files

`ippvm.h`

Domain Dependencies

Headers: `ippcore.h`

Libraries: `ippcore.lib`

Parameters

`pSrc`	Pointer to the source vector.
`pDst`	Pointer to the destination vector.
`len`	Number of elements in the vectors.

Description

This function computes the inverse cosine of each element of `pSrc`, and stores the result in the corresponding element of `pDst`.

For single precision data:

function flavors `ippsAcos_32f_A11` and `ippsAcos_32cf_A11` guarantee 11 correctly rounded bits of significand, or at least 3 exact decimal digits;

function flavors `ippsAcos_32f_A21` and `ippsAcos_32fc_A21` guarantee 21 correctly rounded bits of significand, or 4 ulps, or about 6 exact decimal digits;

function flavors `ippsAcos_32f_A24` and `ippsAcos_32fc_A24` guarantee 24 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

For double precision data:

function flavors `ippsAcos_64f_A26` and `ippsAcos_64fc_A26` guarantee 26 correctly rounded bits of significand, or 6.7E+7 ulps, or approximately 8 exact decimal digits;

function flavors `ippsAcos_64f_A50` and `ippsAcos_64fc_A50` guarantee 50 correctly rounded bits of significand, or 4 ulps, or approximately 15 exact decimal digits;

function flavors `ippsAcos_64f_A53` and `ippsAcos_64fc_A53` guarantee 53 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

The computation is performed as follows:

`pDst[n] = acos(pSrc[n]), 0 ≤ n <len.`

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pSrc or pDst pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.
IppStsDomain	In real functions, indicates a warning that the argument is out of the function domain, that is, at least one of the elements of pSrc has an absolute value greater than 1.

Example

The example below shows how to use the function ippsAcos.

```
IppStatus ippsAcos_32f_A21_sample(void) {
    const Ipp32f x[4] = {0.079, -0.715, -0.076, -0.529};
    Ipp32f y[4];
    IppStatus st = ippsAcos_32f_A21( x, y, 4 );


    printf(" ippsAcos_32f_A21:\n");
    printf(" x = %.3f %.3f %.3f %.3f \n", x[0], x[1], x[2], x[3]);
    printf(" y = %.3f %.3f %.3f %.3f \n", y[0], y[1], y[2], y[3]);
    return st;
}
```

Output:

```
ippsAcos_32f_A21:
x = 0.079 -0.715 -0.076 -0.529
y = 1.492  2.368  1.647  2.129
```

Asin

Computes inverse sine of each vector element.

Syntax

```
IppStatus ippsAsin_32f_A11 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsAsin_32f_A21 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsAsin_32f_A24 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsAsin_64f_A26 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsAsin_64f_A50 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsAsin_64f_A53 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsAsin_32fc_A11 (const Ipp32fc* pSrc, Ipp32fc* pDst, Ipp32s len);
IppStatus ippsAsin_32fc_A21 (const Ipp32fc* pSrc, Ipp32fc* pDst, Ipp32s len);
IppStatus ippsAsin_32fc_A24 (const Ipp32fc* pSrc, Ipp32fc* pDst, Ipp32s len);
IppStatus ippsAsin_64fc_A26 (const Ipp64fc* pSrc, Ipp64fc* pDst, Ipp32s len);
```

```
IppStatus ippsAsin_64fc_A50 (const Ipp64fc* pSrc, Ipp64fc* pDst, Ipp32s len);
IppStatus ippsAsin_64fc_A53 (const Ipp64fc* pSrc, Ipp64fc* pDst, Ipp32s len);
```

Include Files

`ippvm.h`

Domain Dependencies

Headers: `ippcore.h`

Libraries: `ippcore.lib`

Parameters

`pSrc`	Pointer to the source vector.
`pDst`	Pointer to the destination vector.
`len`	Number of elements in the vectors.

Description

This function computes the inverse sine of each element of `pSrc`, and stores the result in the corresponding element of `pDst`.

For single precision data:

function flavors `ippsAsin_32f_A11` and `ippsAsin_32cf_A11` guarantee 11 correctly rounded bits of significand, or at least 3 exact decimal digits;

function flavors `ippsAsin_32f_A21` and `ippsAsin_32fc_A21` guarantee 21 correctly rounded bits of significand, or 4 ulps, or about 6 exact decimal digits;

function flavors `ippsAsin_32f_A24` and `ippsAsin_32fc_A24` guarantee 24 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

For double precision data:

function flavors `ippsAsin_64f_A26` and `ippsAsin_64fc_A26` guarantee 26 correctly rounded bits of significand, or 6.7E+7 ulps, or approximately 8 exact decimal digits;

function flavors `ippsAsin_64f_A50` and `ippsAsin_64fc_A50` guarantee 50 correctly rounded bits of significand, or 4 ulps, or approximately 15 exact decimal digits;

function flavors `ippsAsin_64f_A53` and `ippsAsin_64fc_A53` guarantee 53 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

The computation is performed as follows:

`pDst[n] = asin(pSrc[n]), 0 ≤ n <len.`

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when `pSrc` or `pDst` pointer is `NULL`.

ippStsSizeErr	Indicates an error when len is less than or equal to 0.
IppStsDomain	In real functions, indicates a warning that the argument is out of the function domain, that is, at least one of the elements of pSrc has an absolute value greater than 1.

Example

The example below shows how to use the function ippsAsin.

```
IppStatus ippsAsin_32f_A21_sample(void) {
    const Ipp32f x[4] = {0.724, -0.581, 0.559, 0.687};
    Ipp32f y[4];
    IppStatus st = ippsAsin_32f_A21( x, y, 4 );


    printf(" ippsAsin_32f_A21:\n");
    printf(" x = %.3f %.3f %.3f %.3f \n", x[0], x[1], x[2], x[3]);
    printf(" y = %.3f %.3f %.3f %.3f \n", y[0], y[1], y[2], y[3]);
    return st;
}
```

Output:

```
ippsAsin_32f_A21:
x = 0.724 -0.581 0.559 0.687
y = 0.810 -0.620 0.594 0.758
```

Atan

Computes inverse tangent of each vector element.

Syntax

```
IppStatus ippsAtan_32f_A11 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsAtan_32f_A21 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsAtan_32f_A24 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsAtan_64f_A26 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsAtan_64f_A50 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsAtan_64f_A53 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsAtan_32fc_A11 (const Ipp32fc* pSrc, Ipp32fc* pDst, Ipp32s len);
IppStatus ippsAtan_32fc_A21 (const Ipp32fc* pSrc, Ipp32fc* pDst, Ipp32s len);
IppStatus ippsAtan_32fc_A24 (const Ipp32fc* pSrc, Ipp32fc* pDst, Ipp32s len);
IppStatus ippsAtan_64fc_A26 (const Ipp64fc* pSrc, Ipp64fc* pDst, Ipp32s len);
IppStatus ippsAtan_64fc_A50 (const Ipp64fc* pSrc, Ipp64fc* pDst, Ipp32s len);
IppStatus ippsAtan_64fc_A53 (const Ipp64fc* pSrc, Ipp64fc* pDst, Ipp32s len);
```

Include Files

ippvm.h

Domain Dependencies

Headers: ippcore.h

Libraries: ippcore.lib

Parameters

pSrc	Pointer to the source vector.
pDst	Pointer to the destination vector.
len	Number of elements in the vectors.

Description

This function computes the inverse tangent of each element of pSrc, and stores the result in the corresponding element of pDst.

For single precision data:

function flavors ippsAtan_32f_A11 and ippsAtan_32cf_A11 guarantee 11 correctly rounded bits of significand, or at least 3 exact decimal digits;

function flavors ippsAtan_32f_A21 and ippsAtan_32fc_A21 guarantee 21 correctly rounded bits of significand, or 4 ulps, or about 6 exact decimal digits;

function flavors ippsAtan_32f_A24 and ippsAtan_32fc_A24 guarantee 24 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

For double precision data:

function flavors ippsAtan_64f_A26 and ippsAtan_64fc_A26 guarantee 26 correctly rounded bits of significand, or 6.7E+7 ulps, or approximately 8 exact decimal digits;

function flavors ippsAtan_64f_A50 and ippsAtan_64fc_A50 guarantee 50 correctly rounded bits of significand, or 4 ulps, or approximately 15 exact decimal digits;

function flavors ippsAtan_64f_A53 and ippsAtan_64fc_A53 guarantee 53 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

The computation is performed as follows:

pDst[n] = atan(pSrc[n]), 0 ≤ n <len.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pSrc or pDst pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.

Example

The example below shows how to use the function `ippsAtan`.

```
IppStatus ippsAtan_32f_A21_sample(void) {
    const Ipp32f x[4] = {0.994, 0.999, 0.223, -0.215};
    Ipp32f y[4];
    IppStatus st = ippsAtan_32f_A21( x, y, 4 );


    printf(" ippsAtan_32f_A21:\n");
    printf(" x = %.3f %.3f %.3f %.3f \n", x[0], x[1], x[2], x[3]);
    printf(" y = %.3f %.3f %.3f %.3f \n", y[0], y[1], y[2], y[3]);
    return st;
}
```

Output:

```
ippsAtan_32f_A21:
x = 0.994 0.999 0.223 -0.215
y = 0.782 0.785 0.219 -0.212
```

Atan2

Computes four-quadrant inverse tangent of elements of two vectors.

Syntax

```
IppStatus ippsAtan2_32f_A11 (const Ipp32f* pSrc1, const Ipp32f* pSrc2, Ipp32f* pDst,
Ipp32s len);
```

```
IppStatus ippsAtan2_32f_A21 (const Ipp32f* pSrc1, const Ipp32f* pSrc2, Ipp32f* pDst,
Ipp32s len);
```

```
IppStatus ippsAtan2_32f_A24 (const Ipp32f* pSrc1, const Ipp32f* pSrc2, Ipp32f* pDst,
Ipp32s len);
```

```
IppStatus ippsAtan2_64f_A26 (const Ipp64f* pSrc1, const Ipp64f* pSrc2, Ipp64f* pDst,
Ipp32s len);
```

```
IppStatus ippsAtan2_64f_A50 (const Ipp64f* pSrc1, const Ipp64f* pSrc2, Ipp64f* pDst,
Ipp32s len);
```

```
IppStatus ippsAtan2_64f_A53 (const Ipp64f* pSrc1, const Ipp64f* pSrc2, Ipp64f* pDst,
Ipp32s len);
```

Include Files

`ippvm.h`

Domain Dependencies

Headers: `ippcore.h`

Libraries: `ippcore.lib`

Parameters

`pSrc1`	Pointer to the first source vector.

pSrc2	Pointer to the second source vector.
pDst	Pointer to the destination vector.
len	Number of elements in the vectors.

Description

This function computes the angle between the X axis and the line from the origin to the point (X,Y), for each element of pSrc1 as a Y (the ordinate) and corresponding element of pSrc2 as an X (the abscissa), and stores the result in the corresponding element of pDst. The result angle varies from - π to + π.

For single precision data:

function flavor ippsAtan2_32f_A11 guarantees 11 correctly rounded bits of significand, or at least 3 exact decimal digits;

function flavor ippsAtan2_32f_A21 guarantees 21 correctly rounded bits of significand, or 4 ulps, or about 6 exact decimal digits;

function flavor ippsAtan2_32f_A24 guarantees 24 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

For double precision data:

function flavor ippsAtan2_64f_A26 guarantees 26 correctly rounded bits of significand, or 6.7E+7 ulps, or approximately 8 exact decimal digits;

function flavor ippsAtan2_64f_A50 guarantees 50 correctly rounded bits of significand, or 4 ulps, or approximately 15 exact decimal digits;

function flavor ippsAtan2_64f_A53 guarantees 53 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

The computation is performed as follows:

pDst[n] = atan2(pSrc1[n],pSrc2[n]),0 ≤ n <len.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pSrc1, pSrc2 or pDst pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.

Example

The example below shows how to use the function ippsAtan2.

```
IppStatus ippsAtan2_32f_A21_sample(void) {
    const Ipp32f x1[4] = {1.492, 1.700, 1.147, 1.142};
    const Ipp32f x2[4] = {1.064, 1.505, 1.950, 1.905};
    Ipp32f y[4];
    IppStatus st = ippsAtan2_32f_A21( x1, x2, y, 4 );


    printf(" ippsAtan2_32f_A21:\n");
    printf(" x1 = %.3f %.3f %.3f %.3f \n", x1[0], x1[1], x1[2], x1[3]);
```

```
    printf(" x2 = %.3f %.3f %.3f %.3f \n", x2[0], x2[1], x2[2], x2[3]);
    printf(" y  = %.3f %.3f %.3f %.3f \n", y[0],  y[1],  y[2],  y[3]);
    return st;
}
```

Output:

```
ippsAtan2_32f_A21:
x1 = 1.492 1.700 1.147 1.142
x2 = 1.064 1.505 1.950 1.905
y  = 0.951 0.846 0.532 0.540
```

Hyperbolic Functions

- Cosh
- Sinh
- Tanh
- Acosh
- Asinh
- Atanh

Cosh

Computes hyperbolic cosine of each vector element.

Syntax

```
IppStatus ippsCosh_32f_A11 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsCosh_32f_A21 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsCosh_32f_A24 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsCosh_64f_A26 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsCosh_64f_A50 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsCosh_64f_A53 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsCosh_32fc_A11 (const Ipp32fc* pSrc, Ipp32fc* pDst, Ipp32s len);
IppStatus ippsCosh_32fc_A21 (const Ipp32fc* pSrc, Ipp32fc* pDst, Ipp32s len);
IppStatus ippsCosh_32fc_A24 (const Ipp32fc* pSrc, Ipp32fc* pDst, Ipp32s len);
IppStatus ippsCosh_64fc_A26 (const Ipp64fc* pSrc, Ipp64fc* pDst, Ipp32s len);
IppStatus ippsCosh_64fc_A50 (const Ipp64fc* pSrc, Ipp64fc* pDst, Ipp32s len);
IppStatus ippsCosh_64fc_A53 (const Ipp64fc* pSrc, Ipp64fc* pDst, Ipp32s len);
```

Include Files

`ippvm.h`

Domain Dependencies

Headers: `ippcore.h`

Libraries: `ippcore.lib`

Parameters

pSrc	Pointer to the source vector.
pDst	Pointer to the destination vector.
len	Number of elements in the vectors.

Description

This function computes the hyperbolic cosine of each element of pSrc, and stores the result in the corresponding element of pDst.

For single precision data:

function flavors ippsCosh_32f_A11 and ippsCosh_32cf_A11 guarantee 11 correctly rounded bits of significand, or at least 3 exact decimal digits;

function flavors ippsCosh_32f_A21 and ippsCosh_32fc_A21 guarantee 21 correctly rounded bits of significand, or 4 ulps, or about 6 exact decimal digits;

function flavors ippsCosh_32f_A24 and ippsCosh_32fc_A24 guarantee 24 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

For double precision data:

function flavors ippsCosh_64f_A26 and ippsCosh_64fc_A26 guarantee 26 correctly rounded bits of significand, or 6.7E+7 ulps, or approximately 8 exact decimal digits;

function flavors ippsCosh_64f_A50 and ippsCosh_64fc_A50 guarantee 50 correctly rounded bits of significand, or 4 ulps, or approximately 15 exact decimal digits;

function flavors ippsCosh_64f_A53 and ippsCosh_64fc_A53 guarantee 53 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

The computation is performed as follows:

pDst[n] = cosh(pSrc[n]), 0 ≤ n <len.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pSrc or pDst pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.
IppStsOverflow	In real functions, indicates a warning that the function overflows, that is, at least one of elements of pSrc has the absolute value greater than Ln(FPMAX) + Ln(2), where FPMAX is the maximum representable floating-point number.

Example

The example below shows how to use the function ippsCosh.

```
IppStatus ippsCosh_32f_A21_sample(void) {
    const Ipp32f x[4] = {-4.676, -4.054, 6.803, -9.525};
    Ipp32f y[4];
```

```
    IppStatus st = ippsCosh_32f_A21( x, y, 4 );


    printf(" ippsCosh_32f_A21:\n");
    printf(" x = %.3f %.3f %.3f %.3f \n", x[0], x[1], x[2], x[3]);
    printf(" y = %.3f %.3f %.3f %.3f \n", y[0], y[1], y[2], y[3]);
    return st;
}
```

Output:

```
ippsCosh_32f_A21:
x = -4.676 -4.054 6.803 -9.525
y = 53.661 28.833 450.219 6849.870
```

Sinh

Computes hyperbolic sine of each vector element.

Syntax

```
IppStatus ippsSinh_32f_A11 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsSinh_32f_A21 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsSinh_32f_A24 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsSinh_64f_A26 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsSinh_64f_A50 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsSinh_64f_A53 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsSinh_32fc_A11 (const Ipp32fc* pSrc, Ipp32fc* pDst, Ipp32s len);
IppStatus ippsSinh_32fc_A21 (const Ipp32fc* pSrc, Ipp32fc* pDst, Ipp32s len);
IppStatus ippsSinh_32fc_A24 (const Ipp32fc* pSrc, Ipp32fc* pDst, Ipp32s len);
IppStatus ippsSinh_64fc_A26 (const Ipp64fc* pSrc, Ipp64fc* pDst, Ipp32s len);
IppStatus ippsSinh_64fc_A50 (const Ipp64fc* pSrc, Ipp64fc* pDst, Ipp32s len);
IppStatus ippsSinh_64fc_A53 (const Ipp64fc* pSrc, Ipp64fc* pDst, Ipp32s len);
```

Include Files

`ippvm.h`

Domain Dependencies

Headers: `ippcore.h`

Libraries: `ippcore.lib`

Parameters

`pSrc`	Pointer to the source vector.
`pDst`	Pointer to the destination vector.

len	Number of elements in the vectors.

Description

This function computes the hyperbolic sine of each element of pSrc, and stores the result in the corresponding element of pDst.

For single precision data:

function flavors ippsSinh_32f_A11 and ippsSinh_32cf_A11 guarantee 11 correctly rounded bits of significand, or at least 3 exact decimal digits;

function flavors ippsSinh_32f_A21 and ippsSinh_32fc_A21 guarantee 21 correctly rounded bits of significand, or 4 ulps, or about 6 exact decimal digits;

function flavors ippsSinh_32f_A24 and ippsSinh_32fc_A24 guarantee 24 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

For double precision data:

function flavors ippsSinh_64f_A26 and ippsSinh_64fc_A26 guarantee 26 correctly rounded bits of significand, or 6.7E+7 ulps, or approximately 8 exact decimal digits;

function flavors ippsSinh_64f_A50 and ippsSinh_64fc_A50 guarantee 50 correctly rounded bits of significand, or 4 ulps, or approximately 15 exact decimal digits;

function flavors ippsSinh_64f_A53 and ippsSinh_64fc_A53 guarantee 53 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

The computation is performed as follows:

pDst[n] = sinh(pSrc[n]), 0 ≤ n <len.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pSrc or pDst pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.
IppStsOverflow	In real functions, indicates a warning that the function overflows, that is, at least one of elements of pSrc has the absolute value greater than Ln(FPMAX) + Ln(2), where FPMAX is the maximum representable floating-point number.

Example

The example below shows how to use the function ippsSinh.

```
IppStatus ippsSinh_32f_A21_sample(void) {
    const Ipp32f x[4] = {-2.483, -8.148, 3.544, -8.876};
    Ipp32f y[4];
    IppStatus st = ippsSinh_32f_A21( x, y, 4 );


    printf(" ippsSinh_32f_A21:\n");
    printf(" x = %.3f %.3f %.3f %.3f \n", x[0], x[1], x[2], x[3]);
```

```
    printf(" y = %.3f %.3f %.3f %.3f \n", y[0], y[1], y[2], y[3]);
    return st;
}
```

Output:

```
ippsSinh_32f_A21:
x = -2.483 -8.148 3.544 -8.876
y = -5.945 -1727.412 17.290 -3577.970
```

Tanh

Computes hyperbolic tangent of each vector element.

Syntax

```
IppStatus ippsTanh_32f_A11 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsTanh_32f_A21 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsTanh_32f_A24 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsTanh_64f_A26 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsTanh_64f_A50 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsTanh_64f_A53 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsTanh_32fc_A11 (const Ipp32fc* pSrc, Ipp32fc* pDst, Ipp32s len);
IppStatus ippsTanh_32fc_A21 (const Ipp32fc* pSrc, Ipp32fc* pDst, Ipp32s len);
IppStatus ippsTanh_32fc_A24 (const Ipp32fc* pSrc, Ipp32fc* pDst, Ipp32s len);
IppStatus ippsTanh_64fc_A26 (const Ipp64fc* pSrc, Ipp64fc* pDst, Ipp32s len);
IppStatus ippsTanh_64fc_A50 (const Ipp64fc* pSrc, Ipp64fc* pDst, Ipp32s len);
IppStatus ippsTanh_64fc_A53 (const Ipp64fc* pSrc, Ipp64fc* pDst, Ipp32s len);
```

Include Files

`ippvm.h`

Domain Dependencies

Headers: `ippcore.h`

Libraries: `ippcore.lib`

Parameters

`pSrc`	Pointer to the source vector.
`pDst`	Pointer to the destination vector.
`len`	Number of elements in the vectors.

Description

This function computes the hyperbolic tangent of each element of `pSrc` and stores the result in the corresponding element of `pDst`.

For single precision data:

function flavors `ippsTanh_32f_A11` and `ippsTanh_32cf_A11` guarantee 11 correctly rounded bits of significand, or at least 3 exact decimal digits;

function flavors `ippsTanh_32f_A21` and `ippsTanh_32fc_A21` guarantee 21 correctly rounded bits of significand, or 4 ulps, or about 6 exact decimal digits;

function flavors `ippsTanh_32f_A24` and `ippsTanh_32fc_A24` guarantee 24 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

For double precision data:

function flavors `ippsTanh_64f_A26` and `ippsTanh_64fc_A26` guarantee 26 correctly rounded bits of significand, or 6.7E+7 ulps, or approximately 8 exact decimal digits;

function flavors `ippsTanh_64f_A50` and `ippsTanh_64fc_A50` guarantee 50 correctly rounded bits of significand, or 4 ulps, or approximately 15 exact decimal digits;

function flavors `ippsTanh_64f_A53` and `ippsTanh_64fc_A53` guarantee 53 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

The computation is performed as follows:

`pDst[n] = tanh(pSrc[n]), 0 ≤ n <len.`

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when `pSrc` or `pDst` pointer is `NULL`.
`ippStsSizeErr`	Indicates an error when `len` is less than or equal to 0.

Example

The example below shows how to use the function `ippsTanh`.

```
IppStatus ippsTanh_32f_A21_sample(void) {
    const Ipp32f x[4] = {-0.982, 0.838, -0.448, -0.454};
    Ipp32f y[4];
    IppStatus st = ippsTanh_32f_A21( x, y, 4 );


    printf(" ippsTanh_32f_A21:\n");
    printf(" x = %.3f %.3f %.3f %.3f \n", x[0], x[1], x[2], x[3]);
    printf(" y = %.3f %.3f %.3f %.3f \n", y[0], y[1], y[2], y[3]);
    return st;
}
```

Output:

```
ippsTanh_32f_A21:
x = -0.982 0.838 -0.448 -0.454
y = -0.754 0.685 -0.420 -0.425
```

Acosh

Computes inverse hyperbolic cosine of each vector element.

Syntax

```
IppStatus ippsAcosh_32f_A11 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsAcosh_32f_A21 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsAcosh_32f_A24 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsAcosh_64f_A26 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsAcosh_64f_A50 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsAcosh_64f_A53 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsAcosh_32fc_A11 (const Ipp32fc* pSrc, Ipp32fc* pDst, Ipp32s len);
IppStatus ippsAcosh_32fc_A21 (const Ipp32fc* pSrc, Ipp32fc* pDst, Ipp32s len);
IppStatus ippsAcosh_32fc_A24 (const Ipp32fc* pSrc, Ipp32fc* pDst, Ipp32s len);
IppStatus ippsAcosh_64fc_A26 (const Ipp64fc* pSrc, Ipp64fc* pDst, Ipp32s len);
IppStatus ippsAcosh_64fc_A50 (const Ipp64fc* pSrc, Ipp64fc* pDst, Ipp32s len);
IppStatus ippsAcosh_64fc_A53 (const Ipp64fc* pSrc, Ipp64fc* pDst, Ipp32s len);
```

Include Files

`ippvm.h`

Domain Dependencies

Headers: `ippcore.h`

Libraries: `ippcore.lib`

Parameters

`pSrc`	Pointer to the source vector.
`pDst`	Pointer to the destination vector.
`len`	Number of elements in the vectors.

Description

This function computes the inverse (nonnegative) hyperbolic cosine of each element of `pSrc`, and stores the result in the corresponding element of `pDst`.

For single precision data:

function flavors `ippsAcosh_32f_A11` and `ippsAcosh_32cf_A11` guarantee 11 correctly rounded bits of significand, or at least 3 exact decimal digits;

function flavors `ippsAcosh_32f_A21` and `ippsAcosh_32fc_A21` guarantee 21 correctly rounded bits of significand, or 4 ulps, or about 6 exact decimal digits;

function flavors `ippsAcosh_32f_A24` and `ippsAcosh_32fc_A24` guarantee 24 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

For double precision data:

function flavors `ippsAcosh_64f_A26` and `ippsAcosh_64fc_A26` guarantee 26 correctly rounded bits of significand, or 6.7E+7 ulps, or approximately 8 exact decimal digits;

function flavors ippsAcosh_64f_A50 and ippsAcosh_64fc_A50 guarantee 50 correctly rounded bits of significand, or 4 ulps, or approximately 15 exact decimal digits;

function flavors ippsAcosh_64f_A53 and ippsAcosh_64fc_A53 guarantee 53 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

The computation is performed as follows:

pDst[n] = acosh(pSrc[n]), 0 ≤ n <len.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pSrc or pDst pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.
IppStsDomain	In real functions, indicates a warning that the argument is out of the function domain, that is, at least one of the elements of pSrc is less than 1.

Example

The example below shows how to use the function ippsAcosh.

```
IppStatus ippsAcosh_32f_A21_sample(void) {
    const Ipp32f x[4] = {588.321, 691.492, 837.773, 726.767};
    Ipp32f y[4];
    IppStatus st = ippsAcosh_32f_A21( x, y, 4 );


    printf(" ippsAcosh_32f_A21:\n");
    printf(" x = %.3f %.3f %.3f %.3f \n", x[0], x[1], x[2], x[3]);
    printf(" y = %.3f %.3f %.3f %.3f \n", y[0], y[1], y[2], y[3]);
    return st;
}
```

Output:

```
ippsAcosh_32f_A21:
x = 588.321 691.492 837.773 726.767
y = 7.070 7.232 7.424 7.282
```

Asinh

Computes inverse hyperbolic sine of each vector element.

Syntax

```
IppStatus ippsAsinh_32f_A11(const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsAsinh_32f_A21(const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsAsinh_32f_A24(const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsAsinh_64f_A26(const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsAsinh_64f_A50(const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
```

```
IppStatus ippsAsinh_64f_A53(const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsAsinh_32fc_A11(const Ipp32fc* pSrc, Ipp32fc* pDst, Ipp32s len);
IppStatus ippsAsinh_32fc_A21(const Ipp32fc* pSrc, Ipp32fc* pDst, Ipp32s len);
IppStatus ippsAsinh_32fc_A24(const Ipp32fc* pSrc, Ipp32fc* pDst, Ipp32s len);
IppStatus ippsAsinh_64fc_A26(const Ipp64fc* pSrc, Ipp64fc* pDst, Ipp32s len);
IppStatus ippsAsinh_64fc_A50(const Ipp64fc* pSrc, Ipp64fc* pDst, Ipp32s len);
IppStatus ippsAsinh_64fc_A53(const Ipp64fc* pSrc, Ipp64fc* pDst, Ipp32s len);
```

Include Files

`ippvm.h`

Domain Dependencies

Headers: `ippcore.h`

Libraries: `ippcore.lib`

Parameters

`pSrc`	Pointer to the source vector.
`pDst`	Pointer to the destination vector.
`len`	Number of elements in the vectors.

Description

This function computes the inverse hyperbolic sine of each element of `pSrc`, and stores the result in the corresponding element of `pDst`.

For single precision data:

function flavors `ippsAsinh_32f_A11` and `ippsAsinh_32cf_A11` guarantee 11 correctly rounded bits of significand, or at least 3 exact decimal digits;

function flavors `ippsAsinh_32f_A21` and `ippsAsinh_32fc_A21` guarantee 21 correctly rounded bits of significand, or 4 ulps, or about 6 exact decimal digits;

function flavors `ippsAsinh_32f_A24` and `ippsAsinh_32fc_A24` guarantee 24 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

For double precision data:

function flavors `ippsAsinh_64f_A26` and `ippsAsinh_64fc_A26` guarantee 26 correctly rounded bits of significand, or 6.7E+7 ulps, or approximately 8 exact decimal digits;

function flavors `ippsAsinh_64f_A50` and `ippsAsinh_64fc_A50` guarantee 50 correctly rounded bits of significand, or 4 ulps, or approximately 15 exact decimal digits;

function flavors `ippsAsinh_64f_A53` and `ippsAsinh_64fc_A53` guarantee 53 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

The computation is performed as follows:

`pDst[n] = asinh(pSrc[n]), 0 ≤ n <len.`

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pSrc or pDst pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.

Example

The example below shows how to use the function ippsAsinh.

```
IppStatus ippsAsinh_32f_A21_sample(void) {
    const Ipp32f x[4] = {-30.122, -589.282, 487.472, -63.082};
    Ipp32f y[4];
    IppStatus st = ippsAsinh_32f_A21( x, y, 4 );


    printf(" ippsAsinh_32f_A21:\n");
    printf(" x = %.3f %.3f %.3f %.3f \n", x[0], x[1], x[2], x[3]);
    printf(" y = %.3f %.3f %.3f %.3f \n", y[0], y[1], y[2], y[3]);
    return st;
}
```

Output:

```
ippsAsinh_32f_A21:
x = -30.122 -589.282 487.472 -63.082
y = -4.099 -7.072 6.882 -4.838
```

Atanh

Computes inverse hyperbolic tangent of each vector element.

Syntax

```
IppStatus ippsAtanh_32f_A11 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsAtanh_32f_A21 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsAtanh_32f_A24 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsAtanh_64f_A26 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsAtanh_64f_A50 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsAtanh_64f_A53 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsAtanh_32fc_A11 (const Ipp32fc* pSrc, Ipp32fc* pDst, Ipp32s len);
IppStatus ippsAtanh_32fc_A21 (const Ipp32fc* pSrc, Ipp32fc* pDst, Ipp32s len);
IppStatus ippsAtanh_32fc_A24 (const Ipp32fc* pSrc, Ipp32fc* pDst, Ipp32s len);
IppStatus ippsAtanh_64fc_A26 (const Ipp64fc* pSrc, Ipp64fc* pDst, Ipp32s len);
IppStatus ippsAtanh_64fc_A50 (const Ipp64fc* pSrc, Ipp64fc* pDst, Ipp32s len);
IppStatus ippsAtanh_64fc_A53 (const Ipp64fc* pSrc, Ipp64fc* pDst, Ipp32s len);
```

Include Files

ippvm.h

Domain Dependencies

Headers: ippcore.h

Libraries: ippcore.lib

Parameters

pSrc	Pointer to the source vector.
pDst	Pointer to the destination vector.
len	Number of elements in the vectors.

Description

This function computes the inverse hyperbolic tangent of each element of pSrc, and stores the result in the corresponding element of pDst.

For single precision data:

function flavors ippsAtanh_32f_A11 and ippsAtanh_32fc_A11 guarantee 11 correctly rounded bits of significand, or at least 3 exact decimal digits;

function flavors ippsAtanh_32f_A21 and ippsAtanh_32fc_A21 guarantee 21 correctly rounded bits of significand, or 4 ulps, or about 6 exact decimal digits;

function flavors ippsAtanh_32f_A24 and ippsAtanh_32fc_A24 guarantee 24 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

For double precision data:

function flavors ippsAtanh_64f_A26 and ippsAtanh_64fc_A26 guarantee 26 correctly rounded bits of significand, or 6.7E+7 ulps, or approximately 8 exact decimal digits;

function flavors ippsAtanh_64f_A50 and ippsAtanh_64fc_A50 guarantee 50 correctly rounded bits of significand, or 4 ulps, or approximately 15 exact decimal digits;

function flavors ippsAtanh_64f_A53 and ippsAtanh_64fc_A53 guarantee 53 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

The computation is performed as follows:

pDst[n] = atanh(pSrc[n]), 0 ≤ n <len.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pSrc or pDst pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.

ippStsDomain	In real functions, indicates a warning that the argument is out of the function domain, that is, at least one of the elements of pSrc has absolute value greater than 1.
ippStsSingularity	In real functions, indicates a warning that the argument is the singularity point, that is, at least one of the elements of pSrc has absolute value equal to 1.

Example

The example below shows how to use the function ippsAtanh.

```
IppStatus ippsAtanh_32f_A21_sample(void) {
    const Ipp32f x[4] = {-0.076, 0.808, 0.440, -0.705};
    Ipp32f y[4];
    IppStatus st = ippsAtanh_32f_A21( x, y, 4 );


    printf(" ippsAtanh_32f_A21:\n");
    printf(" x = %.3f %.3f %.3f %.3f \n", x[0], x[1], x[2], x[3]);
    printf(" y = %.3f %.3f %.3f %.3f \n", y[0], y[1], y[2], y[3]);
    return st;
}
```

Output:

```
ippsAtanh_32f_A21:
x = -0.076 0.808 0.440 -0.705
y = -0.076 1.123 0.472 -0.877
```

Special Functions

- Erf
- Erfc
- CdfNorm
- ErfInv
- ErfcInv
- CdfNormInv

Erf

Computes the error function value.

Syntax

```
IppStatus ippsErf_32f_A11 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsErf_32f_A21 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsErf_32f_A24 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsErf_64f_A26 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsErf_64f_A50 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsErf_64f_A53 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
```

Include Files

ippvm.h

Domain Dependencies

Headers: ippcore.h

Libraries: ippcore.lib

Parameters

pSrc	Pointer to the source vector.
pDst	Pointer to the destination vector.
len	Number of elements in the vectors.

Description

This function computes the error function value for each element of pSrc and stores the result in the corresponding element of pDst.

For single precision data:

function flavor ippsErf_32f_A11 guarantees 11 correctly rounded bits of significand, or at least 3 exact decimal digits;

function flavor ippsErf_32f_A21 guarantees 21 correctly rounded bits of significand, or 4 ulps, or about 6 exact decimal digits;

function flavor ippsErf_32f_A24 guarantees 24 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

For double precision data:

function flavor ippsErf_64f_A26 guarantees 26 correctly rounded bits of significand, or 6.7E+7 ulps, or approximately 8 exact decimal digits;

function flavor ippsErf_64f_A50 guarantees 50 correctly rounded bits of significand, or 4 ulps, or approximately 15 exact decimal digits;

function flavor ippsErf_64f_A53 guarantees 53 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

The computation is performed as follows:

pDst[n] = erf(pSrc[n]), 0 ≤ n <len,
where

$$\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt .$$

Return Values

ippStsNoErr	Indicates no error.

ippStsNullPtrErr	Indicates an error when pSrc or pDst pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.

Example

The example below shows how to use the function ippsErf.

```
IppStatus ippsErf_32f_A21_sample(void) {
    const Ipp32f x[4] = {-0.982, 0.838, -0.448, -0.454};
    Ipp32f y[4];
    IppStatus st = ippsErf_32f_A21( x, y, 4 );


    printf(" ippsErf_32f_A21:\n");
    printf(" x = %.3f %.3f %.3f %.3f \n", x[0], x[1], x[2], x[3]);
    printf(" y = %.3f %.3f %.3f %.3f \n", y[0], y[1], y[2], y[3]);
    return st;
}
```

Output:

```
ippsErf_32f_A21:
x = -0.982 0.838 -0.448 -0.454
y = -0.835 0.764 -0.474 -0.479
```

Erfc

Computes the complementary error function value.

Syntax

```
IppStatus ippsErfc_32f_A11 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsErfc_32f_A21 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsErfc_32f_A24 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsErfc_64f_A26 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsErfc_64f_A50 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsErfc_64f_A53 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
```

Include Files

ippvm.h

Domain Dependencies

Headers: ippcore.h

Libraries: ippcore.lib

Parameters

pSrc	Pointer to the source vector.

pDst	Pointer to the destination vector.
len	Number of elements in the vectors.

Description

This function computes the complementary error function value for each element of pSrc and stores the result in the corresponding element of pDst.

For single precision data:

function flavor ippsErfc_32f_A11 guarantees 11 correctly rounded bits of significand, or at least 3 exact decimal digits;

function flavor ippsErfc_32f_A21 guarantees 21 correctly rounded bits of significand, or 4 ulps, or about 6 exact decimal digits;

function flavor ippsErfc_32f_A24 guarantees 24 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

For double precision data:

function flavor ippsErfc_64f_A26 guarantees 26 correctly rounded bits of significand, or 6.7E+7 ulps, or approximately 8 exact decimal digits;

function flavor ippsErfc_64f_A50 guarantees 50 correctly rounded bits of significand, or 4 ulps, or approximately 15 exact decimal digits;

function flavor ippsErfc_64f_A53 guarantees 53 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

The computation is performed as follows:

pDst[n] = erfc(pSrc[n]), 0 ≤ n <len, where

$$\mathrm{erfc}(x) = 1 - \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt \ .$$

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pSrc or pDst pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.
IppStsUnderflow	Indicates a warning that the function underflows, that is, at least one element of pSrc is less than some threshold value, where the function result is less than the minimum positive floating-point value in target precision.

Example

The example below shows how to use the function `ippsErfc`.

```
IppStatus ippsErfc_32f_A21_sample(void) {
    const Ipp32f x[4] = {-0.982, 0.838, -0.448, -0.454};
    Ipp32f y[4];
    IppStatus st = ippsErfc_32f_A21( x, y, 4 );
    printf(" ippsErfc_32f_A21:\n");
    printf(" x = %.3f %.3f %.3f %.3f \n", x[0], x[1], x[2], x[3]);
    printf(" y = %.3f %.3f %.3f %.3f \n", y[0], y[1], y[2], y[3]);
    return st;
}
```

Output:

```
ippsErfc_32f_A21:
x = -0.982 0.838 -0.448 -0.454
y = -0.754 0.685 -0.420 -0.425
```

CdfNorm

Computes the cumulative normal distribution function values of vector element.

Syntax

```
IppStatus ippsCdfNorm_32f_A11 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsCdfNorm_32f_A21 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsCdfNorm_32f_A24 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsCdfNorm_64f_A26 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsCdfNorm_64f_A50 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsCdfNorm_64f_A53 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
```

Include Files

`ippvm.h`

Domain Dependencies

Headers: `ippcore.h`

Libraries: `ippcore.lib`

Parameters

`pSrc`	Pointer to the source vector.
`pDst`	Pointer to the destination vector.
`len`	Number of elements in the vectors.

Description

This function computes the cumulative normal distribution function values of `pSrc` element and stores the result in the corresponding element of `pDst`.

For single precision data:

function flavor ippsCdfNorm_32f_A11 guarantees 11 correctly rounded bits of significand, or at least 3 exact decimal digits;

function flavor ippsCdfNorm_32f_A21 guarantees 21 correctly rounded bits of significand, or 4 ulps, or about 6 exact decimal digits;

function flavor ippsCdfNorm_32f_A24 guarantees 24 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

For double precision data:

function flavor ippsCdfNorm_64f_A26 guarantees 26 correctly rounded bits of significand, or 6.7E+7 ulps, or approximately 8 exact decimal digits;

function flavor ippsCdfNorm_64f_A50 guarantees 50 correctly rounded bits of significand, or 4 ulps, or approximately 15 exact decimal digits;

function flavor ippsCdfNorm_64f_A53 guarantees 53 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

The computation is performed as follows:

pDst[n] = CdfNorm(pSrc[n]), 0 ≤ n <len, where

$$\mathrm{CdfNorm}(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{t^2}{2}} dt \ .$$

The example below shows how to use the function ippsCdfNorm.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pSrc or pDst pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.
IppStsUnderflow	Indicates a warning that the function underflows, that is, at least one element of pSrc is less than some threshold value, where the function result is less than the minimum positive floating-point value in the target precision.

Using ippsCdfNorm Function

```
IppStatus ippsCdfNorm_32f_A24_sample(void)
{
const Ipp32f x[4] = {+4.885, -0.543, -3.809, -4.953};
Ipp32f          y[4];

IppStatus st = ippsCdfNorm_32f_A24( x, y, 4 );

printf(" ippsCdfNorm_32f_A24:\n");
printf(" x = %+.3f %+.3f %+.3f %+.3f \n", x[0], x[1], x[2], x[3]);
printf(" y = %+.3f %+.3f %+.3f %+.3f \n", y[0], y[1], y[2], y[3]);
return st;
}

Output results:

ippsCdfNorm_32f_A24:
x = +4.885 -0.543 -3.809 -4.953
y = +1.000 +0.294 +0.000 +0.000
```

ErfInv

Computes the inverse error function value.

Syntax

```
IppStatus ippsErfInv_32f_A11 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsErfInv_32f_A21 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsErfInv_32f_A24 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsErfInv_64f_A26 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsErfInv_64f_A50 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsErfInv_64f_A53 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
```

Include Files

`ippvm.h`

Domain Dependencies

Headers: `ippcore.h`

Libraries: `ippcore.lib`

Parameters

pSrc	Pointer to the source vector.
pDst	Pointer to the destination vector.
len	Number of elements in the vectors.

Description

This function computes the inverse error function value for each element of pSrc and stores the result in the corresponding element of pDst.

For single precision data:

function flavor ippsErfInv_32f_A11 guarantees 11 correctly rounded bits of significand, or at least 3 exact decimal digits;

function flavor ippsErfInv_32f_A21 guarantees 21 correctly rounded bits of significand, or 4 ulps, or about 6 exact decimal digits;

function flavor ippsErfInv_32f_A24 guarantees 24 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

For double precision data:

function flavor ippsErfInv_64f_A26 guarantees 26 correctly rounded bits of significand, or 6.7E+7 ulps, or approximately 8 exact decimal digits;

function flavor ippsErfInv_64f_A50 guarantees 50 correctly rounded bits of significand, or 4 ulps, or approximately 15 exact decimal digits;

function flavor ippsErfInv_64f_A53 guarantees 53 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

The computation is performed as follows:

pDst[n] = erfinv(pSrc[n]), $0 \le n <$ len, where erfinv(x) = erf^{-1}(x), and erf(x) denotes the error function defined as given by:

$$\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pSrc or pDst pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.
ippStsDomain	Indicates a warning that the argument is out of the function domain, that is, at least one of pSrc elements has the absolute value greater than 1.

ippStsSingularity	Indicates a warning that the argument is a singularity point, that is, at least one of the elements of pSrc has the absolute value equal to 1.

Example

The example below shows how to use the function ippsErfcInv.

```
IppStatus ippsErfInv_32f_A21_sample(void) {
    const Ipp32f x[4] = {-0.842, 0.638, -0.345, -0.774};
    Ipp32f y[4];
    IppStatus st = ippsErfInv_32f_A21( x, y, 4 );


    printf(" ippsErfInv_32f_A21:\n");
    printf(" x = %.3f %.3f %.3f %.3f \n", x[0], x[1], x[2], x[3]);
    printf(" y = %.3f %.3f %.3f %.3f \n", y[0], y[1], y[2], y[3]);
    return st;
}
```

Output:

```
ippsErfInv_32f_A21:
x = -0.842 0.638 -0.345 -0.774
y = -0.998 0.645 -0.316 -0.856
```

ErfcInv

Computes the inverse complementary error function value of vector element.

Syntax

```
IppStatus ippsErfcInv_32f_A11 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsErfcInv_32f_A21 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsErfcInv_32f_A24 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsErfcInv_64f_A26 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsErfcInv_64f_A50 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsErfcInv_64f_A53 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
```

Include Files

ippvm.h

Domain Dependencies

Headers: ippcore.h

Libraries: ippcore.lib

Parameters

pSrc	Pointer to the source vector.
pDst	Pointer to the destination vector.

len	Number of elements in the vectors.

Description

This function computes the inverse complementary error function value of each pSrc vector element and stores the result in the corresponding element of pDst.

For single precision data:

function flavor ippsErfcInv_32f_A11 guarantees 11 correctly rounded bits of significand, or at least 3 exact decimal digits;

function flavor ippsErfcInv_32f_A21 guarantees 21 correctly rounded bits of significand, or 4 ulps, or about 6 exact decimal digits;

function flavor ippsErfcInv_32f_A24 guarantees 24 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

For double precision data:

function flavor ippsErfcInv_64f_A26 guarantees 26 correctly rounded bits of significand, or 6.7E+7 ulps, or approximately 8 exact decimal digits;

function flavor ippsErfcInv_64f_A50 guarantees 50 correctly rounded bits of significand, or 4 ulps, or approximately 15 exact decimal digits;

function flavor ippsErfcInv_64f_A53 guarantees 53 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

The computation is performed as follows:

pDst[n] = erfcinv(pSrc[n]), 0 ≤ n <len, where erfcinv(x) = erfinv(1 –x, and erfinv(x) denotes the error function defined as given by:

$$\mathrm{erfinv}(x) = \mathrm{erf}^{-1}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt,$$

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pSrc or pDst pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.
ippStsDomain	Indicates a warning that the argument is out of the function domain, that is, at least one of pSrc elements is outside the function domain [0; 2].
ippStsSingularity	Indicates a warning that the argument is a singularity point, that is, at least one of the elements of pSrc is equal to 0 or 2.

Example

The example below shows how to use the function `ippsErfcInv`.

```
IppStatus ippsErfcInv_32f_A24_sample(void) {
    const Ipp32f x[4] = {+0.885, +0.543, +1.809, +0.953};
    Ipp32f y[4];
    IppStatus st = ippsErfcInv_32f_A24( x, y, 4 );


    printf(" ippsErfcInv_32f_A24:\n");
    printf(" x = %+.3f %+.3f %+.3f %+.3f \n", x[0], x[1], x[2], x[3]);
    printf(" y = %+.3f %+.3f %+.3f %+.3f \n", y[0], y[1], y[2], y[3]);
    return st;
}
```

Output:

```
ippsErfcInv_32f_A24:
x = +0.885 +0.543 +1.809 +0.953
y = +0.102 +0.430 -0.925 +0.042
```

CdfNormInv

Computes the inverse cumulative normal distribution function values of vector elements.

Syntax

```
IppStatus ippsCdfNormInv_32f_A11 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsCdfNormInv_32f_A21 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsCdfNormInv_32f_A24 (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsCdfNormInv_64f_A26 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsCdfNormInv_64f_A50 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
IppStatus ippsCdfNormInv_64f_A53 (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
```

Include Files

`ippvm.h`

Domain Dependencies

Headers: `ippcore.h`

Libraries: `ippcore.lib`

Parameters

`pSrc`	Pointer to the source vector.
`pDst`	Pointer to the destination vector.
`len`	Number of elements in the vectors.

Description

This function computes the inverse cumulative normal distribution function values of `pSrc` vector elements and stores the result in the corresponding element of `pDst`.

For single precision data:

function flavor `ippsCdfNormInv_32f_A11` guarantees 11 correctly rounded bits of significand, or at least 3 exact decimal digits;

function flavor `ippsCdfNormInv_32f_A21` guarantees 21 correctly rounded bits of significand, or 4 ulps, or about 6 exact decimal digits;

function flavor `ippsCdfNormInv_32f_A24` guarantees 24 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

For double precision data:

function flavor `ippsCdfNormInv_64f_A26` guarantees 26 correctly rounded bits of significand, or 6.7E+7 ulps, or approximately 8 exact decimal digits;

function flavor `ippsCdfNormInv_64f_A50` guarantees 50 correctly rounded bits of significand, or 4 ulps, or approximately 15 exact decimal digits;

function flavor `ippsCdfNormInv_64f_A53` guarantees 53 correctly rounded bits of significand, including the implied bit, with the maximum guaranteed error within 1 ulp.

The computation is performed as follows:

`pDst[n] = CdfNormInv(pSrc[n]),0 ≤ n <len`, where `CdfNormInv(x) = CdfNorm`$^{-1}$`(x)`, and `CdfNorm(x)` denotes the cumulative normal distribution function:

$$\mathrm{CdfNorm}(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{t^2}{2}} dt \ .$$

The example below shows how to use the function `ippsCdfNormInv`.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when `pSrc` or `pDst` pointer is `NULL`.
`ippStsSizeErr`	Indicates an error when `len` is less than or equal to 0.
`ippStsDomain`	Indicates a warning that the argument is out of the function domain, that is, at least one of `pSrc` elements is outside the function domain [0; 1].
`ippStsSingularity`	Indicates a warning that the argument is a singularity point, that is, at least one of the elements of `pSrc` is equal to 0 or 1.

Using ippsCdfNormInv Function

```
IppStatus ippsCdfNormInv_32f_A24_sample(void)
{
const Ipp32f x[4] = {+0.085, +0.543, +1.809, +0.953};
Ipp32f       y[4];

IppStatus st = ippsCdfNormInv_32f_A24( x, y, 4 );

printf(" ippsCdfNormInv_32f_A24:\n");
printf(" x = %+.3f %+.3f %+.3f %+.3f \n", x[0], x[1], x[2], x[3]);
printf(" y = %+.3f %+.3f %+.3f %+.3f \n", y[0], y[1], y[2], y[3]);
return st;
}

Output results:

ippsCdfNormInv_32f_A24:
x = +0.085 +0.543 +1.809 +0.953
y = -1.372 +0.108 +0.874 +1.675
```

Rounding Functions

- Floor
- Frac
- Ceil
- Trunc
- Round
- NearbyInt
- Rint
- Modf

Floor

Computes integer value rounded toward minus infinity for each vector element.

Syntax

```
IppStatus ippsFloor_32f (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsFloor_64f (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
```

Include Files

`ippvm.h`

Domain Dependencies

Headers: `ippcore.h`

Libraries: ippcore.lib

Parameters

pSrc	Pointer to the source vector.
pDst	Pointer to the destination vector.
len	Number of elements in the vectors.

Description

This function computes an integer value rounded towards minus infinity for each element of the vector pSrc, and stores the result in the corresponding element of the vector pDst.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pSrc or pDst pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than or equal to 0.

Example

The example below shows how to use the function ippsFloor.

```
IppStatus ippsFloor_32f_ sample(void) {
    const Ipp32f x[4] = {-0.883, -0.265, 0.176, 0.752};
    Ipp32f y[4];
    IppStatus st = ippsFloor_32f ( x, y, 4 );


    printf(" ippsFloor_32f:\n");
    printf(" x = %.3f %.3f %.3f %.3f \n", x[0], x[1], x[2], x[3]);
    printf(" y = %.3f %.3f %.3f %.3f \n", y[0], y[1], y[2], y[3]);
    return st;
}
```

Output:

```
ippsFloor_32f:
x = -0.883 -0.265 0.176 0.752
y = -1.000 -1.000 0.000 0.000
```

Frac

Computes a signed fractional part for each element of a vector.

Syntax

```
IppStatus ippsFrac_32f (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsFrac_64f (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
```

Include Files

ippvm.h

Domain Dependencies

Headers: ippcore.h

Libraries: ippcore.lib

Parameters

pSrc	Pointer to the source vector.
pDst	Pointer to the destination vector.
len	Number of elements in the vectors.

Description

This function computes a fractional part of each element of the pSrc vector. The result is stored in the corresponding element of the pDst vector.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pSrc or pDst pointer is NULL.
ippStsSizeErr	Indicates an error when len is less than, or equal to zero.

Example

The example below shows how to use the ippsFrac function.

```
IppStatus ippsFrac_32f_sample(void)
{
const Ipp32f x[4] = {-1.883, -0.265, 0.176, 1.752};
Ipp32f y[4];
IppStatus st = ippsFrac_32f ( x, y, 4 );
printf(" ippsFrac_32f:\n");
printf(" x = %.3f %.3f %.3f %.3f \n", x[0], x[1], x[2], x[3]);
printf(" y = %.3f %.3f %.3f %.3f \n", y[0], y[1], y[2], y[3]);
return st;
}
```

Result:

```
ippsFrac_32f:
x = -1.883 -0.265 0.176 1.752
y = -0.883 -0.265 0.176 0.752
```

Ceil

Computes integer value rounded toward plus infinity for each vector element.

Syntax

```
IppStatus ippsCeil_32f (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsCeil_64f (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
```

Include Files

`ippvm.h`

Domain Dependencies

Headers: `ippcore.h`

Libraries: `ippcore.lib`

Parameters

`pSrc`	Pointer to the source vector.
`pDst`	Pointer to the destination vector.
`len`	Number of elements in the vectors.

Description

This function computes an integer value rounded towards plus infinity for each element of the vector `pSrc`, and stores the result in the corresponding element of the vector `pDst`.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when `pSrc` or `pDst` pointer is `NULL`.
`ippStsSizeErr`	Indicates an error when `len` is less than or equal to 0.

Example

The example below shows how to use the function `ippsCeil`.

```
IppStatus ippsCeil_32f_sample(void) {
    const Ipp32f x[4] = {-0.883, -0.265, 0.176, 0.752};
    Ipp32f y[4];
    IppStatus st = ippsCeil_32f ( x, y, 4 );


    printf(" ippsCeil_32f:\n");
    printf(" x = %.3f %.3f %.3f %.3f \n", x[0], x[1], x[2], x[3]);
    printf(" y = %.3f %.3f %.3f %.3f \n", y[0], y[1], y[2], y[3]);
    return st;
}
```

Output:

```
ippsCeil_32f:
x = -0.883 -0.265 0.176 0.752
y = 0.000 0.000 1.000 1.000
```

Trunc

Computes integer value rounded toward zero for each vector element.

Syntax

```
IppStatus ippsTrunc_32f (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsTrunc_64f (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
```

Include Files

`ippvm.h`

Domain Dependencies

Headers: `ippcore.h`

Libraries: `ippcore.lib`

Parameters

`pSrc`	Pointer to the source vector.
`pDst`	Pointer to the destination vector.
`len`	Number of elements in the vectors.

Description

This function computes an integer value rounded towards zero for each element of the vector `pSrc`, and stores the result in the corresponding element of the vector `pDst`.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when `pSrc` or `pDst` pointer is `NULL`.
`ippStsSizeErr`	Indicates an error when `len` is less than or equal to 0.

Example

The example below shows how to use the function `ippsTrunc`.

```
IppStatus ippsTrunc_32f_ sample(void) {
    const Ipp32f x[4] = {-1.883, -0.265, 0.176, 1.752};
    Ipp32f y[4];
    IppStatus st = ippsTrunc_32f ( x, y, 4 );


    printf(" ippsTrunc_32f:\n");
    printf(" x = %.3f %.3f %.3f %.3f \n", x[0], x[1], x[2], x[3]);
    printf(" y = %.3f %.3f %.3f %.3f \n", y[0], y[1], y[2], y[3]);
    return st;
}
```

Output:

```
ippsTrunc_32f:
x = -1.883 -0.265 0.176 1.752
y = -1.000 0.000 0.000 1.000
```

Round

Computes integer value rounded to nearest for each vector element.

Syntax

```
IppStatus ippsRound_32f (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsRound_64f (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
```

Include Files

`ippvm.h`

Domain Dependencies

Headers: `ippcore.h`

Libraries: `ippcore.lib`

Parameters

`pSrc`	Pointer to the source vector.
`pDst`	Pointer to the destination vector.
`len`	Number of elements in the vectors.

Description

This function computes a rounded to the nearest integer value for each element of the vector `pSrc`, and stores the result in the corresponding element of the vector `pDst`. Halfway values, that is, 0.5, -1.5, and the like, are rounded off away from zero, that is, 0.5 -> 1, -1.5 -> -2, and so on.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when `pSrc` or `pDst` pointer is `NULL`.
`ippStsSizeErr`	Indicates an error when `len` is less than or equal to 0.

Example

The example below shows how to use the function `ippsRound`.

```
IppStatus ippsRound_32f_ sample(void) {
    const Ipp32f x[4] = {-1.883, -0.265, 0.176, 1.752};
    Ipp32f y[4];
    IppStatus st = ippsRound_32f ( x, y, 4 );
```

```
    printf(" ippsRound_32f:\n");
    printf(" x = %.3f %.3f %.3f %.3f \n", x[0], x[1], x[2], x[3]);
    printf(" y = %.3f %.3f %.3f %.3f \n", y[0], y[1], y[2], y[3]);
    return st;
}
```

Output:

```
ippsRound_32f:
x = -1.883 -0.265 0.176 1.752
y = -2.000  0.000 0.000 2.000
```

NearbyInt

Computes rounded integer value in current rounding mode for each vector element.

Syntax

```
IppStatus ippsNearbyInt_32f (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);
IppStatus ippsNearbyInt_64f (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
```

Include Files

`ippvm.h`

Domain Dependencies

Headers: `ippcore.h`

Libraries: `ippcore.lib`

Parameters

`pSrc`	Pointer to the source vector.
`pDst`	Pointer to the destination vector.
`len`	Number of elements in the vectors.

Description

This function computes a rounded integer value in a current rounding mode for each element of the vector `pSrc`, and stores the result in the corresponding element of the vector `pDst`.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when `pSrc` or `pDst` pointer is `NULL`.
`ippStsSizeErr`	Indicates an error when `len` is less than or equal to 0.

Example

The example below shows how to use the `ippsNearbyInt` function.

```
#include <fenv.h>


void ippsNearbyInt_32f_ sample(void) {
    const Ipp32f x[4] = {-1.883, -0.265, 0.176, 1.752};
    Ipp32f y1[4], y2[4];
    fesetround(FE_TONEAREST);
    ippsNearbyInt_32f ( x, y1, 4 );
    fesetround(FE_TOWARDZERO);
    ippsNearbyInt_32f ( x, y2, 4 );


    printf(" ippsNearbyInt_32f:\n");
    printf(" x = %.3f %.3f %.3f %.3f \n", x[0], x[1], x[2], x[3]);
    printf(" y1 = %.3f %.3f %.3f %.3f \n", y1[0], y1[1], y1[2], y1[3]);
    printf(" y2 = %.3f %.3f %.3f %.3f \n", y2[0], y2[1], y2[2], y2[3]);
}
```

Output:

```
ippsNearInt_32f:
x = -1.883 -0.265 0.176 1.752
y1 = -2.000  0.000 0.000 2.000
y2 = -1.000  0.000 0.000 1.000
```

Rint

Computes rounded integer value in current rounding mode for each vector element with inexact result exception raised for each changed value.

Syntax

```
IppStatus ippsRint_32f (const Ipp32f* pSrc, Ipp32f* pDst, Ipp32s len);

IppStatus ippsRint_64f (const Ipp64f* pSrc, Ipp64f* pDst, Ipp32s len);
```

Include Files

`ippvm.h`

Domain Dependencies

Headers: `ippcore.h`

Libraries: `ippcore.lib`

Parameters

`pSrc`	Pointer to the source vector.
`pDst`	Pointer to the destination vector.
`len`	Number of elements in the vectors.

Description

This function computes a rounded integer value in a current rounding mode for each element of the vector `pSrc`, and stores the result in the corresponding element of the vector `pDst` raising inexact result exception if the value has changed.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when `pSrc` or `pDst` pointer is `NULL`.
`ippStsSizeErr`	Indicates an error when `len` is less than or equal to 0.

Example

The example below shows how to use the `ippsRint` function.

```
#include <fenv.h>


void ippsRint_32f_ sample(void) {
    const Ipp32f x[4] = {-1.883, -0.265, 0.176, 1.752};
    Ipp32f y1[4], y2[4];
    fesetround(FE_TONEAREST);
    ippsRint_32f ( x, y1, 4 );
    fesetround(FE_TOWARDZERO);
    ippsRint_32f ( x, y2, 4 );


    printf(" ippsRint_32f:\n");
    printf(" x = %.3f %.3f %.3f %.3f \n", x[0], x[1], x[2], x[3]);
    printf(" y1 = %.3f %.3f %.3f %.3f \n", y1[0], y1[1], y1[2], y1[3]);
    printf(" y2 = %.3f %.3f %.3f %.3f \n", y2[0], y2[1], y2[2], y2[3]);
}
```

Output:

```
ippsRint_32f:
x = -1.883 -0.265 0.176 1.752
y1 = -2.000  0.000 0.000 2.000
y2 = -1.000  0.000 0.000 1.000
```

Modf

Computes truncated integer value and remaining fraction part for each vector element.

Syntax

```
IppStatus ippsModf_32f (const Ipp32f* pSrc, Ipp32f* pDst1, Ipp32f* pDst2, Ipp32s len);
IppStatus ippsModf_64f (const Ipp64f* pSrc, Ipp64f* pDst1, Ipp64f* pDst2, Ipp32s len);
```

Include Files

`ippvm.h`

Domain Dependencies

Headers: `ippcore.h`

Libraries: `ippcore.lib`

Parameters

`pSrc`	Pointer to the source vector.
`pDst1`	Pointer to the first destination vector.
`pDst2`	Pointer to the second destination vector.
`len`	Number of elements in the vectors.

Description

This function computes a truncated value and a remainder of each element of the vector `pSrc`. The truncated integer value is stored in the corresponding element of the `pDst1` vector and the remainder is stored in the corresponding element of the `pDst2` vector.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when `pSrc` or `pDst1` or `pDst2` pointer is `NULL`.
`ippStsSizeErr`	Indicates an error when `len` is less than or equal to 0.

Example

The example below shows how to use the function `ippsModf`.

```
IppStatus ippsModf_32f_sample(void) {
    const Ipp32f x[4] = {-1.883, -0.265, 0.176, 1.752};
    Ipp32f y1[4], y2[4];
    IppStatus st = ippsModf_32f ( x, y1, y2, 4 );


    printf(" ippsModf_32f:\n");
    printf(" x = %.3f %.3f %.3f %.3f \n", x[0], x[1], x[2], x[3]);
    printf(" y1 = %.3f %.3f %.3f %.3f \n", y1[0], y1[1], y1[2], y1[3]);
    printf(" y2 = %.3f %.3f %.3f %.3f \n", y2[0], y2[1], y2[2], y2[3]);
    return st;
}
```

Output results:

```
ippsModf_32f:
x = -1.883 -0.265 0.176 1.752
y1 = -1.000 0.000 0.000 1.000
y2 = -0.883 -0.265 0.176 0.752
```

Long Term Evolution (LTE) Wireless Support Functions

NOTE This functionality is available only within the Intel® System Studio suite.

This section describes functions that implement the Long Term Evolution (LTE) multiple input multiple output (MIMO) algorithm to estimate the minimum mean square error (MMSE).

The LTE MIMO uplink provides the following:

- *Spatial multiplexing* to enable high data rates within a limited bandwidth
- Additional *diversity* against fading on the radio channel
- *Beam-forming* to shape the overall antenna beam in a certain way to maximize the overall antenna gain in the direction of the target receiver

- CRC_8u
- ippsGenCRCOptPoly_8u
- CRC16
- CRC24a, CRC24b, CRC24c

CRC_8u

Computes checksum for a given data vector.

Syntax

```
IppStatus ippsCRC_8u(Ipp8u* pSrc, int len, Ipp64u poly, Ipp8u optPoly[128], Ipp32u init, Ipp32u* pCRC16);
```

Include Files

`ippe.h`

Domain Dependencies

`ippcore.h`

Libraries

`ippe.lib`

Parameters

`pSrc`	The pointer to the source vector.
`len`	The length of the vector, number of items.
`poly`	CRC polynomial with explicit leading 1. Indicates CRC length: 8/16/24/32 bits.
`optPoly`	The initialized data table (NULL, by default).
`init`	The initial value of a register.
`pCRC`	Pointer to the CRC value.

Description

This function computes the CRC value with the polynomial `poly` and the initial value `init` for the input vector `pSrc` with the length `len` bytes. The default `optPoly` value is NULL.

This function supports only 8, 16, 24, and 32 bytes-length polynomials. The bytes number of the CRC algorithm (8, 16, 24, or 32) is defined by the position of the most significant 1 bit in `poly`. The polynomial must be specified in full. For example, for CRC16 with the 0x1021 polynomial, the poly value must be 0x11021.

Low-level `ippsCRC_8u` optimization requires special tables for every `poly` value. This function calls the optimized code only for the fixed set of polynomials by default and returns `ippStsNoErr` status. If such table is not available for `poly`, it calculates CRC using non-optimized code and returns the `ippStsNonOptimalPathSelected` warning.

To compute CRC for an arbitrary polynomial with low-level optimization, you need to initialize the `optPoly` table first with the ippsGenCRCOptPoly_8u function and transfer `optPoly` into `ippsCRC_8u`.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

//   Computing CRC16 for the 0x1021 polynomial
int main()
{
    Ipp8u *src = "123456789";
    IppStatus status;
    Ipp32u CRC;
    Ipp64u poly = 0x11021; // Default polynomial
    Ipp32u init = 0xFFFF;

    status = ippsCRC_8u(src, 9, poly, NULL, init, &CRC);
    printf("status = '%s'\n", ippGetStatusString(status));
    printf("CRC=0x%x\n", CRC);
    return 0;
}
```

The result:

```
status = 'ippStsNoErr: No errors'
CRC=0x29b1
```

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if the pointer to the source vector is `NULL`.

ippStsSizeErr	Indicates an error if the length of the source vector is less than or equal to 0.
ippStsAlgTypeErr	Indicates an error if the most significant 1 bit is in the wrong position.
ippStsBadArgErr	Indicates an error if the optPoly value does not match poly value.
ippStsNonOptimalPathSelected	Indicates an error if the function returns the positive warning. Call ippsGenCRCOptPoly_8u to compute the table.

ippsGenCRCOptPoly_8u

Computes optimization table for ippsCRC_8u.

Syntax

```
IppStatus ippsGenCRCOptPoly_8u(Ipp64u* poly, Ipp8u optPoly[128]);
```

Include Files

ippe.h

Domain Dependencies

ippcore.h

Libraries

ippe.lib

Parameters

poly	CRC polynomial with explicit leading 1. Indicates CRC length: 8/16/24/32 bits.
optPoly	The initialized data table (NULL, by default).

Description

The ippsGenCRCOptPoly_8u function is auxiliary for ippsCRC_8u and computes a table for low-level optimization in ippsCRC_8u. You can use this function in the initialization procedure of the application.

Low-level ippsCRC_8u optimization requires special tables for every poly value. This function calls the optimized code only for the fixed set of polynomials by default and returns the ippStsNoErr status. If such table is not available for poly, it calculates CRC using non-optimized code and returns the ippStsNonOptimalPathSelected warning.

To compute CRC for an arbitrary polynomial with low-level optimization, you need to initialize the optPoly table first with the ippsGenCRCOptPoly_8u function and transfer optPoly into ippsCRC_8u.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

//   Computing the table for the function that does not support CRC16 with the 0x8005 polynomial.
int main()
{
    Ipp8u *src = "123456789";
    IppStatus status;
    Ipp32u CRC;
    Ipp64u poly = 0x18005;
    Ipp32u init = 0;
    Ipp8u optPoly[128];

    // function returns ippStsNonOptimalPathSelected
    status = ippsCRC_8u(src, 9, poly, NULL, init, &CRC);
    printf("status = '%s'\n", ippGetStatusString(status));
    printf("CRC=0x%x\n", CRC);

    // function returns ippStsNoErr and
    // calls optimized code
    status = ippsGenCRCOptPoly_8u(poly, optPoly);
    status = ippsCRC_8u(src, 9, poly, optPoly, init, &CRC);
    printf("status = '%s'\n", ippGetStatusString(status));
    printf("CRC=0x%x\n", CRC);

    return 0;
}
```

The result:

```
status = 'The function is inefficient due to the combination of input parameters'
CRC=0xfee8
status = 'ippStsNoErr: No errors'
CRC=0xfee8
```

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
```

```
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *****************************************************************************/

//   Computing CRC6
int main()
{
    Ipp8u *src = "123456789";
    IppStatus status;
    Ipp32u CRC;
    Ipp64u poly = 0x61; // CRC6 polynomial;
    Ipp32u init = 0x0;
    Ipp8u optPoly[128];

    // Function calculates crc8/crc16/crc24/crc32
    // Shift 2 bits left, CRC6 -> CRC8
    poly <<= 2;
    status = ippsGenCRCOptPoly_8u(poly, optPoly);
    status = ippsCRC_8u(src, 9, poly, optPoly, init, &CRC);
    printf("status = %d, '%s'\n", status, ippGetStatusString(status));
    // Shift 2 bits right, CRC8 -> CRC6
    printf("crc6=%x\n", CRC >> 2);
    return 0;
}
```

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if the pointer to the source vector is NULL.
ippStsAlgTypeErr	Indicates an error if the most significant 1 bit is in the wrong position.

CRC16

Computes the CRC16 checksum for the source data buffer.

Syntax

```
IppStatus ippsCRC16_8u(Ipp8u* pSrc, int len, Ipp32u* pCRC16);
```

```
IppStatus ippsCRC16_1u(Ipp8u* pSrc, int srcBitOffset, Ipp8u* pDst, int dstBitOffset,
int bitLen);
```

Include Files

ippe.h

Domain Dependencies

ippcore.h

Libraries

ippcore.lib

Parameters

pSrc	Pointer to the source data buffer.
srcBitOffset	Offset in bits from the source data buffer.
pDst	Pointer to the destination data buffer.
dstBitOffset	Offset in bits from the destination data buffer.
len	Number of elements in the source data buffer.
pCRC16	Pointer to the checksum value.
bitLen	Length of the input source vector, in bits.

Description

This function computes the checksum for srcLen elements of the source data buffer pSrc and stores it in the pCRC16 respectively. The following polynomial representation is used:

$x^{16} + X^{12} + X^5 + X + 1$

``ippsCRC16_1u``. This function flavor computes the CRC16 checksum of a vector that has a 8u data type. It means that each byte consists of eight consecutive elements of the vector (1 bit per element). You need to specify the offsets from the source and destination data buffers in the srcBitOffset and dstBitOffset parameters, respectively.

This function can be used to compute the accumulated value of the checksum for multiple buffers in the data stream by specifying as an input parameter the checksum value obtained in the preceding function call.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if the pSrc pointer is NULL.
ippStsSizeErr	Indicates an error if the length of the source vector is less than or equal to 0.

CRC24a, CRC24b, CRC24c

Computes the CRC24 checksum for the source data buffer.

Syntax

```
IppStatus ippsCRC24a_8u(Ipp8u* pSrc, int len, Ipp32u* pCRC24);
IppStatus ippsCRC24b_8u(Ipp8u* pSrc, int len, Ipp32u* pCRC24);
IppStatus ippsCRC24c_8u(Ipp8u* pSrc, int len, Ipp32u* pCRC24);
IppStatus ippsCRC24a_1u(Ipp8u* pSrc, int srcBitOffset, Ipp8u* pDst, int dstBitOffset,
int bitLen);
```

```
IppStatus ippsCRC24b_1u(Ipp8u* pSrc, int srcBitOffset, Ipp8u* pDst, int dstBitOffset,
int bitLen);

IppStatus ippsCRC24c_1u(Ipp8u* pSrc, int srcBitOffset, Ipp8u* pDst, int dstBitOffset,
int bitLen);
```

Include Files

ippe.h

Domain Dependencies

ippcore.h

Libraries

ippe.lib

Parameters

pSrc	Pointer to the source data buffer.
srcBitOffset	Offset in bits from the source data buffer.
pDst	Pointer to the destination data buffer.
dstBitOffset	Offset in bits from the destination data buffer.
len	Number of elements in the source data buffer.
pCRC24	Pointer to the checksum value.
bitLen	Length of the input source vector, in bits.

Description

These functions compute the checksum for srcLen elements of the source data buffer pSrc using different polynomials and store it in the pCRC24 respectively. The following polynomial representations are used:

ippsCRC24a	$X^{24} + X^{23} + X^{18} + X^{17} + X^{14} + X^{11} + X^{10} + X^7 + X^6 + X^5 + X^4 + X^3 + X + 1$
ippsCRC24b	$X^{24} + X^{23} + X^6 + X^5 + X + 1$
ippsCRC24c	$X^{24} + X^{23} + X^{21} + X^{20} + X^{17} + X^{15} + X^{13} + X^{12} + X^8 + X^4 + X^2 + X + 1$

These functions can be used to compute the accumulated value of the checksum for multiple buffers in the data stream by specifying as an input parameter the checksum value obtained in the preceding function call.

``ippsCRC24{a|b|c}_1u``. These function flavors compute the checksum of vectors that have a 8u data type. It means that each byte consists of eight consecutive elements of the vector (1 bit per element). You need to specify the offsets from the source and destination vectors in the srcBitOffset and dstBitOffset parameters, respectively.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if the pSrc pointer is NULL.

ippStsSizeErr	Indicates an error if the length of the source vector is less than or equal to 0.

Data Compression Functions

This chapter describes the Intel® IPP functions for data compression that support a number of different compression methods: Huffman and variable-length coding, dictionary-based coding methods (including support of ZLIB compression), and methods based on Burrows-Wheeler Transform.

- Application Notes
- Dictionary-Based Compression Functions
- BWT-Based Compression Functions
- ZFP Compression Functions

Application Notes

- The functions in this domain can be divided into two types: the functions that actually compress data, and transformation functions. The latters do not compress data but only modify them and prepare for further compression. The examples of such transformation are the *Burrows-Weller Transform*, or *MoveToFront* algorithm. To do data compression efficient, you should develop the proper consequence of functions of different type that will transform data and then compress them.
- Compression ratio depends on the statistics of input data. For some types of input data no compression could be achieved at all.
- The size of memory required for the output of data compression functions typically is not obvious. As a rule encoding functions uses less memory for output than the size of the input buffer, on the contrary decoding functions use more memory for output than the size of the input buffer. You should account these issues and allocate the proper quantity of output memory using the techniques provided by functions in this domain. For example, you can use a double-pointer technique for automatic shifting the user submitted pointer. For some other functions it is possible to compute the upper limit of the size of the required output buffer.

Dictionary-Based Compression Functions

This section describes the Intel IPP functions that use different dictionary-based compression methods.

- LZSS Compression Functions
- ZLIB Coding Functions
- LZO Compression Functions
- LZ4 Compression Functions
- LZ4 Compression Functions for High Compression (HC) Mode

LZSS Compression Functions

These functions implement the LZSS (Lempel-Ziv-Storer-Szymanski) compression algorithm [Storer82]. The functions perform LZSS coding with a vocabulary size of 32KB and 256-byte maximum match string length.

- EncodeLZSSInit
- LZSSGetSize
- EncodeLZSS
- EncodeLZSSFlush
- DecodeLZSSInit
- DecodeLZSS

EncodeLZSSInit

Initializes the LZSS encoder state structure.

Syntax

```
IppStatus ippsEncodeLZSSInit_8u (IppLZSSState_8u* pLZSSState);
```

Include Files

`ippdc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pLZSSState`	Pointer to the LZSS encoder state structure.

Description

This function initializes the LZSS state structure `pLZSSState` in the external buffer. Its size must be computed previously by calling the function ippsLZSSGetSize.

The LZSS encoder state structure is required for the encoder functions ippsEncodeLZSS and ippsEncodeLZSSFlush.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if the pointer `pLZSSState` is `NULL`.

LZSSGetSize

Computes the size of the LZSS state structure.

Syntax

```
IppStatus ippsLZSSGetSize_8u (int* pLZSSStateSize);
```

Include Files

`ippdc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

pLZSSStateSize	Pointer to the size of the LZSS state structure.

Description

This function computes the size in bytes of the LZSS state structure for encoding and decoding and stores it to an integer pointed to by pLZSSStateSize. The function must be called prior to the function ippsEncodeLZSSInit or ippsDecodeLZSSInit.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if the pointer pLZSSStateSize is NULL.

EncodeLZSS

Performs LZSS encoding.

Syntax

```
IppStatus ippsEncodeLZSS_8u (Ipp8u** ppSrc, int* pSrcLen, Ipp8u** ppDst, int* pDstLen,
IppLZSSState_8u* pLZSSState);
```

Include Files

ippdc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

ppSrc	Double pointer to the source buffer.
pSrcLen	Pointer to the number of elements in the source buffer; it is updated after encoding.
ppDst	Double pointer to the destination buffer.
pDstLen	Pointer to the length of the destination buffer; it is updated and returns the length of the destination buffer after encoding.
pLZSSState	Pointer to the LZSS encoding state structure.

Description

This function performs LZSS encoding of data in the source buffer ppSrc of length pSrcLen and stores the result in the destination buffer pDst of length pDstLen. The LZSS encoder state structure pLZSSState must be initialized by ippsEncodeLZSSInit beforehand.

After encoding the function returns the pointers to source and destination buffers shifted by the number of successfully read and encoded bytes, respectively. The function updates `pSrcLen` and `pDstLen` so they return the actual number of elements in the source and destination buffers respectively.

Code example shows how to use the function `ippsEncodeLZSS_8u` and supporting functions.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error if `srcLen` is less than or equal to 0.
`ippStsDstSizeLessExpected`	Indicates a warning that the size of the destination buffer is insufficient for completing the operation.

EncodeLZSSFlush

Encodes the last few bits in the bitstream and aligns the output data on the byte boundary.

Syntax

```
IppStatus ippsEncodeLZSSFlush_8u (Ipp8u** ppDst, int* pDstLen, IppLZSSState_8u*
pLZSSState);
```

Include Files

`ippdc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`ppDst`	Double pointer to the destination buffer.
`pDstLen`	Pointer to the length of destination buffer.
`pLZSSState`	Pointer to the LZSS encoder state structure.

Description

This function encodes the last few bits (remainder) in the bitstream, writes them to `ppDst`, and aligns the output data on a byte boundary.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if one of the pointers is `NULL`.

ippStsSizeErr	Indicates an error if pDstLen is less than or equal to 0.
ippStsDstSizeLessExpected	Indicates a warning that the size of the destination buffer is insufficient for completing the operation.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

// The example below shows how to use the function ippsEncodeLZSS_8u and supporting functions:
//     ippsLZSSGetSize_8u, ippsEncodeLZSSInit_8u
//     ippsEncodeLZSS_8u
//     ippsEncodeLZSSFlush_8u

#include <stdio.h>
#include "ipp.h"

#define INBLOCKSIZE  65536
#define OUTBLOCKSIZE 16384

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define
check_sts(st)
                                  \
    if ((st) !=
ippStsNoErr)
                          \
        goto exitLine; /* Go to Exit if Intel(R) Integrated Performance Primitives (Intel(R)
IPP) function returned status different from \
                           ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main(void)
{
    FILE *inFile = NULL, *outFile = NULL;
    Ipp8u *pSrc = NULL, *pDst = NULL, *startpSrc = NULL, *startpDst = NULL;
    int srcLen, dstLen = 0;
    int commonDstLen = 0;
    IppLZSSState_8u *pLZSSState = NULL;
```

```
int szState = 0;
IppStatus status = ippStsNoErr;

inFile = fopen("EncodeLZSSFlush.c", "rb");
if (!inFile) {
    printf("\nCan't open input data file "
           "EncodeLZSSFlush.c"
           ". program terminated.\n");
    goto exitLine;
}
outFile = fopen("EncodeLZSSFlush.bin", "wb");
if (!outFile) {
    printf("\nCan't open output data file "
           "EncodeLZSSFlush.bin"
           ". program terminated.\n");
    goto exitLine;
}

/*************************************************************/
/* Memory allocation for input data and output code buffers, */
/* and for the internal encoding state structure:            */
/*************************************************************/
startpSrc = ippsMalloc_8u((INBLOCKSIZE) * sizeof(char));
startpDst = ippsMalloc_8u((OUTBLOCKSIZE) * sizeof(char));

check_sts(status = ippsLZSSGetSize_8u(&szState));

pLZSSState = (IppLZSSState_8u *)ippsMalloc_8u(szState);

check_sts(status = ippsEncodeLZSSInit_8u(pLZSSState));

/*************************************************************/
/* Initializing the arguments, reading srcLen bytes of       */
/* input data from input data file to startpSrc:             */
/*************************************************************/
commonDstLen = (OUTBLOCKSIZE);
pDst = startpDst;
dstLen = commonDstLen;
srcLen = fread(startpSrc, sizeof(Ipp8u), INBLOCKSIZE, inFile);

if (srcLen <= 0) {
    printf("\nInput data file "
           "EncodeLZSSFlush.c"
           " is corrupted. program terminated.\n");
    goto exitLine;
}
pSrc = startpSrc;

/*************************************************************/
/* The main loop. In every iteration program calls the       */
/* ippsEncodeLZSS_8u. ippsEncodeLZSS_8u changes the values    */
/* of pSrc, srcLen, pDst and dstLen. ippsEncodeLZSS_8u       */
/* returns ippStsDstSizeLessExpected if there is no          */
/* room in the destination buffer pDst to continue encoding.  */
/* In this case dstLen == 0. Such case is handled by         */
/* flushing the pDst to the output file. If ippsEncodeLZSS_8u */
/* returns ippStsNoErr, then the program flushes the current  */
/* pDst of length of (commonDstLen - dstLen) bytes to        */
```

```
    /* the output file and reads next srcLen bytes of input data  */
    /* to startpSrc.                                              */
    /**************************************************************/
    for (;;) {
        status = ippsEncodeLZSS_8u(&pSrc, &srcLen, &pDst, &dstLen, pLZSSState);
        if (status == ippStsDstSizeLessExpected) {
            fwrite(startpDst, sizeof(Ipp8u), (commonDstLen - dstLen), outFile);
            dstLen = commonDstLen;
            pDst = startpDst;

        } else if (status == ippStsNoErr) {
            fwrite(startpDst, sizeof(Ipp8u), (commonDstLen - dstLen), outFile);
            pDst = startpDst;
            dstLen = commonDstLen;
            srcLen = fread(startpSrc, sizeof(Ipp8u), INBLOCKSIZE, inFile);
            if (srcLen <= 0)
                break;
            pSrc = startpSrc;
        } else {
            check_sts(status);
        }
    }

    /**************************************************************/
    /*  The call of ippsEncodeLZSSFlush_8u flushes the last few   */
    /*  bits of code to the destination buffer:                   */
    /**************************************************************/
    check_sts(ippsEncodeLZSSFlush_8u(&pDst, &dstLen, pLZSSState));
    fwrite(startpDst, sizeof(Ipp8u), (commonDstLen - dstLen), outFile);
    printf("\n EncodeLZSSFlush.c successfully encoded to EncodeLZSSFlush.bin \n");

    EXIT_MAIN
    ippsFree(pLZSSState);
    ippsFree(startpSrc);
    ippsFree(startpDst);
    if (inFile)
        fclose(inFile);
    if (outFile)
        fclose(outFile);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

DecodeLZSSInit

Initializes the LZSS decoder state structure.

Syntax

```
IppStatus ippsDecodeLZSSInit_8u (IppLZSSState_8u* pLZSSState);
```

Include Files

`ippdc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pLZSSState	Pointer to the LZSS decoder state structure.

Description

This function initializes the LZSS decoder state structure in the external buffer, the size of which must be computed previously by calling the function ippsLZSSGetSize.

The LZSS decoder state structure is required for the function ippsDecodeLZSS.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if the pointer pLZSSState is NULL.

DecodeLZSS

Performs LZSS decoding.

Syntax

```
IppStatus ippsDecodeLZSS_8u (Ipp8u** ppSrc, int* pSrcLen, Ipp8u** ppDst, int* pDstLen,
IppLZSSState_8u* pLZSSState);
```

Include Files

ippdc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

ppSrc	Double pointer to the source buffer.
pSrcLen	Pointer to the length of the source buffer.
ppDst	Double pointer to the destination buffer.
pDstLen	Pointer to the length of the destination buffer.
pLZSSState	Pointer to the LZSS decoding state structure.

Description

This function performs LZSS decoding of the pSrcLen elements of the ppSrc source buffer and stores the result in the pDst destination vector. The length of the destination vector is stored in pDstLen. The LZSS decoder state structure pLZSSState must be initialized by ippsDecodeLZSSInit beforehand.

After decoding the function returns the pointers to source and destination buffers shifted by the number of successfully read and decoded bytes respectively. The function updates `pSrcLen` so it is equal to the actual number of elements in the source buffer.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error if `pSrcLen` or `pDstLen` is negative.
`ippStsDstSizeLessExpected`	Indicates a warning that the size of the destination buffer is insufficient for completing the operation.

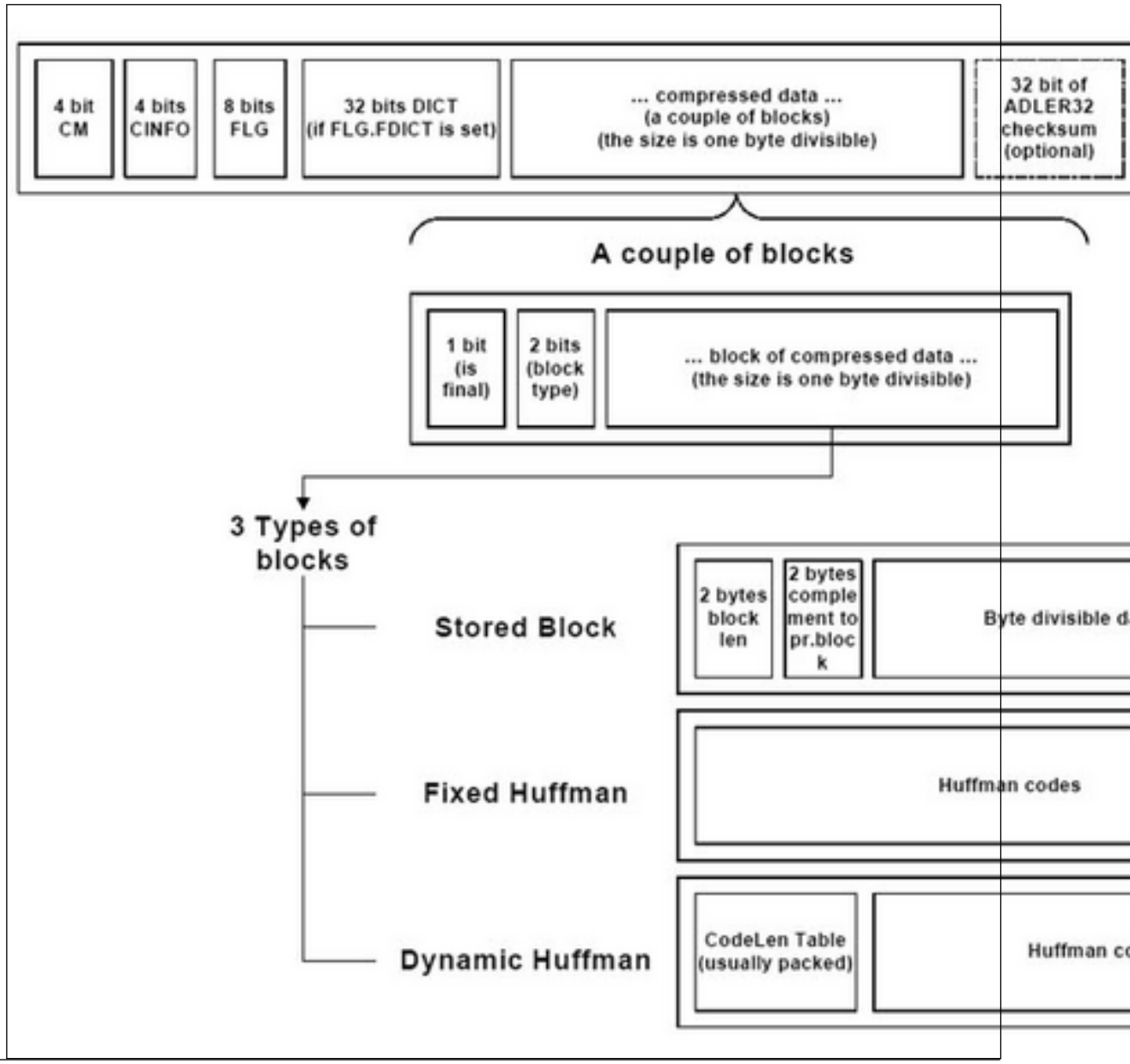
ZLIB Coding Functions

This section describes Intel IPP data compression functions that implement compression methods and data formats defined by the following specifications: [RFC1950], [RFC1951], and [RFC1952]. These formats are also known as ZLIB, DEFLATE, and GZIP, respectively.

A basic algorithm for these data compression methods is based on the Lempel-Ziv (LZ77) [Ziv77] dictionary-based compression.

The structure of ZLIB data is schematically shown in Figure ZLIB Data Structure.

ZLIB Data Structure



The full version of the `zlib` library is included with the product at `<ipp directory>/interfaces/data-compression/ipp_zlib`.

- Special Parameters
- Adler32
- CRC32, CRC32C
- DeflateLZ77
- DeflateLZ77Fast
- DeflateLZ77Fastest
- DeflateLZ77FastestGenHeader
- DeflateLZ77FastestGenHuffTable
- DeflateLZ77FastestGetStat
- DeflateLZ77FastestPrecompHeader

- DeflateLZ77Slow
- DeflateDictionarySet
- DeflateUpdate Hash
- DeflateHuff
- InflateBuildHuffTable
- Inflate

Special Parameters

The ZLIB coding functions have several special parameters.

The `comprLevel` parameter specifies the level of compression rate and compression ratio. The table below lists the possible values of the `comprLevel` parameter and their meanings.

Value	Descriptions
`IppLZ77FastCompr`	Fast compression, maximum compression rate, and below average compression ratio
`IppLZ77AverageCompr`	Average compression rate, average compression ratio
`IppLZ77BestCompr`	Slow compression, maximum compression ratio

The `checksum` parameter specifies what algorithm is used to compute checksum for input data. The table below lists the possible values of the `checksum` parameter and their meanings.

Value	Description
`IppLZ77NoChcksm`	Checksum is not calculated.
`IppLZ77Adler32`	Checksum is calculated using Adler32 algorithm.
`IppLZ77CRC32`	Checksum is calculated using the CRC32 algorithm.

The `flush` parameter specifies the encoding mode for data block encoding. The table below lists the possible values of the `flush` parameter and their meanings.

Value	Descriptions
`IppLZ77NoFlush`	The end of the block is aligned to a byte boundary.
`IppLZ77SyncFlush`	The end of the block is aligned to a byte boundary, and 4-byte marker is written to pDst.
`IppLZ77FullFlush`	The end of the block is aligned to a byte boundary, 4-byte marker is written to pDst, sliding dictionary is zeroed.
`IppLZ77FinishFlush`	The end of the block is aligned to a byte boundary and the function returns the `ippStsStreamEnd` status.

The `deflateStatus` parameter specifies the encoding status to ensure the compatibility with the RFC1951 specification. This parameter is used by Intel IPP ZLIB encoding functions. The table below lists the possible values of the `deflateStatus` parameter and their meanings.

Value	Descriptions
`IppLZ77StatusInit`	Specified the deflate implementation of encoding functions, must be used before stream encoding.

IppLZ77StatusLZ77Process	Call the deflate implementation of encoding function.
IppLZ77StatusHuffProcess	Call the deflate implementation of the encoding function with the fixed Huffman codes.
IppLZ77StatusFinal	Specified the last block in the stream.

The inflateStatus parameter specifies the decoding status to ensure the compatibility with the RFC1951 specification. This parameter is used by Intel IPP ZLIB decoding functions. The table below lists the possible values of the inflateStatus parameter and their meanings.

Value	Descriptions
IppLZ77inflateStatusInit	Specified the deflate implementation of encoding functions, must be used before stream encoding.
IppLZ77InflateStatusHuffProcess	Call the deflate implementation of the encoding function with the fixed Huffman codes.
IppLZ77InflateStatusLZ77Process	Call the deflate implementation of encoding function.
IppLZ77InflateStatusFinal	Specified the last block in the stream.

Adler32

Computes the Adler32 checksum for the source data buffer.

Syntax

```
IppStatus ippsAdler32_8u (const Ipp8u* pSrc, int srcLen, Ipp32u* pAdler32);
```

Include Files

ippdc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc	Pointer to the source data buffer.
srcLen	Number of elements in the source data buffer.
pAdler32	Pointer to the checksum value.

Description

This function computes the checksum for srcLen elements of the source data buffer pSrc and stores it in the pAdler32. The checksum is computed using the Adler32 algorithm that is a modified version of the Fletcher algorithm [Flet82], [ITU224], [RFC1950].

You need to call the Adler32 function twice: once with a NULL/zero length buffer to prime the checksum to 1, then call it again to compute the checksum on the buffer.

You can use this function to compute the accumulated value of the checksum for multiple buffers in the data stream by specifying as an input parameter the checksum value obtained in the preceding function call.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if the pSrc pointer is NULL.
ippStsSizeErr	Indicates an error if srcLen is less than or equal to 0.

CRC32, CRC32C

Computes the CRC32 checksum for the source data buffer.

Syntax

```
IppStatus ippsCRC32_8u (const Ipp8u* pSrc, int srcLen, Ipp32u* pCRC32);
IppStatus ippsCRC32C_8u(const Ipp8u* pSrc, Ipp32u srcLen, Ipp32u* pCRC32C);
```

Include Files

ippdc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc	Pointer to the source data buffer.
srcLen	Number of elements in the source data buffer.
pCRC32, pCRC32C	Pointer to the checksum value.

Description

These functions compute the checksum for srcLen elements of the source data buffer pSrc using different algorithms and stores it in the pCRC32 or pCRC32C respectively.

The function ippsCRC32 uses algorithm described in [Griff87], [Nel92], the function ippsCRC32C uses algorithm described in [Cast93].

These functions can be used to compute the accumulated value of the checksum for multiple buffers in the data stream by specifying as an input parameter the checksum value obtained in the preceding function call.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if the pSrc pointer is NULL.

ippStsSizeErr	Indicates an error if the length of the source vector is less than or equal to 0.

Example

The example below shows how to use the function ippsCRC32C_8u.

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

#include <iostream>
#include <iomanip>
#include "ipp.h"

using namespace std;

void crc32c_core(Ipp8u *src, Ipp32u src_len)
{
    Ipp32u crc32c = ~(Ipp32u)0;
    ippsCRC32C_8u(src, src_len, &crc32c);
    ippsSwapBytes_32u_I(&crc32c, 1);
    cout << "0x" << setbase(16) << ~crc32c << endl;
}

int main()
{
    Ipp8u buff[48] = {0x01, 0xc0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00,
                      0x14, 0x00, 0x00, 0x00, 0x00, 0x00, 0x04, 0x00, 0x00, 0x00, 0x00, 0x14,
0x00, 0x00, 0x00, 0x18,
                      0x28, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x02, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00};
    cout << "An iSCSI - SCSI Read (10) Command PDU: ";
    crc32c_core(buff, 48);

    cout << "32 bytes of zeroes: ";
    for (int i = 0; i < 32; i++)
        buff[i] = 0;
    crc32c_core(buff, 32);

    cout << "32 bytes of ones: ";
    for (int i = 0; i < 32; i++)
        buff[i] = 0xff;
    crc32c_core(buff, 32);
```

```
    cout << "32 bytes of incrementing 00..1f: ";
    for (int i = 0; i < 32; i++)
        buff[i] = i;
    crc32c_core(buff, 32);

    cout << "32 bytes of decrementing 1f..00: ";
    for (int i = 0; i < 32; i++)
        buff[i] = 31 - i;
    crc32c_core(buff, 32);

    return 0;
}
```

```
Output: An iSCSI - SCSI Read (10) Command PDU: 0x563a96d9 32 bytes of zeroes: 0xaa36918a 32
bytes of ones: 0x43aba862 32 bytes of incrementing 00..1f: 0x4e79dd46 32 bytes of decrementing
1f..00: 0x5cdb3f11
```

DeflateLZ77

Performs LZ77 encoding according to the specified compression level.

Syntax

```
IppStatus ippsDeflateLZ77_8u(const Ipp8u** ppSrc, Ipp32u* pSrcLen, Ipp32u* pSrcIdx,
const Ipp8u* pWindow, Ipp32u winSize, Ipp32s* pHashHead, Ipp32s* pHashPrev, Ipp32u
hashSize, IppDeflateFreqTable pLitFreqTable[286], IppDeflateFreqTable
pDistFreqTable[30], Ipp8u* pLitDst, Ipp16u* pDistDst, Ipp32u* pDstLen, int comprLevel,
IppLZ77Flush flush);
```

Include Files

`ippdc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`ppSrc`	Double pointer to the source vector.
`pSrcLen`	Pointer to the length of the source vector.
`pSrcIdx`	Pointer to the index of the current position in the source vector.
`pWindow`	Pointer to the sliding window (the dictionary for the LZ77 algorithm).
`winSize`	Size of the sliding window and the `pHashPrev` table.
`pHashHead`	Pointer to the table containing heads of the hash chains.

pHashPrev	Pointer to the table contating indexes to the previous strings with the same hash key.
hashSize	Size of the pHashHead table.
pLitFreqTable	Pointer to the literals/lengths frequency table.
pDistFreqTable	Pointer to the distances frequency table.
pLitDst	Pointer to the destination vector containing literals/ lengths.
pDistDst	Pointer to the destination vector containing distances.
pDstLen	Pointer to the length of the destination vectors.
comprLevel	Compression level in range [0..9] in accordance with ZLIB.
flush	Specifies the encoding mode for data blocks (see flush parameter).

Description

This function performs LZ77 encoding of source data ppSrc according to the compression level comprLevel, which is similar to the ZLIB compression level.

To correctly process the first bytes of the source vector, initialize the pHashHead table with the winSize value.

The pSrcIdx parameter returns the index of the current position in the source vector, and is used to establish a correlation between the current position in the source vector and indexes in hash tables. After processing each 2GB of source data, the index and hash tables must be normalized (instead of 64K of source data in ZLIB).

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one of the specified pointers is NULL.

See Also

Special Parameters

DeflateLZ77Fast

Performs LZ77 encoding according to the fast algorithm and parameters of a match.

Syntax

```
IppStatus ippsDeflateLZ77Fast_8u(const Ipp8u** ppSrc, Ipp32u* pSrcLen, Ipp32u* pSrcIdx,
const Ipp8u* pWindow, Ipp32u winSize, Ipp32s* pHashHead, Ipp32s* pHashPrev, Ipp32u
hashSize, IppDeflateFreqTable pLitFreqTable[286], IppDeflateFreqTable
pDistFreqTable[30], Ipp8u* pLitDst, Ipp16u* pDistDst, Ipp32u* pDstLen, int* pVecMatch,
IppLZ77Flush flush);
```

Include Files

ippdc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

ppSrc	Double pointer to the source vector.
pSrcLen	Pointer to the length of the source vector.
pSrcIdx	Pointer to the index of the current position in the source vector.
pWindow	Pointer to the sliding window (the dictionary for the LZ77 algorithm).
winSize	Size of the sliding window and the pHashPrev table.
pHashHead	Pointer to the table containing heads of the hash chains.
pHashPrev	Pointer to the table containing indexes to the previous strings with the same hash key.
hashSize	Size of the pHashHead table.
pLitFreqTable	Pointer to the literals/lengths frequency table.
pDistFreqTable	Pointer to the distances frequency table.
pLitDst	Pointer to the destination vector containing literals/lengths.
pDistDst	Pointer to the destination vector containing distances.
pDstLen	Pointer to the length of the destination vectors.
pVecMatch	Pointer to the vector containing the following parameters of a match: max_chain_length, good_match, nice_match, max_lazy_match (for more information, see [ZLIB]).
flush	Specifies the encoding mode for data blocks (see flush parameter).

Description

This function performs LZ77 encoding of the ppSrc data according to the fast algorithm and parameters of a match.

To correctly process the first bytes of the source vector, initialize the pHashHead table with the winSize value.

The pSrcIdx parameter returns the index of the current position in the source vector, and is used to establish a correlation between the current position in the source vector and indexes in hash tables. After processing each 2GB of source data, the index and hash tables must be normalized (instead of 64K of source data in ZLIB).

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when winSize is less than 256, or more than 32768; or hashSize is less than 256, or more than 65536.
ippStsBadArgErr	Indicates an error when good_match, nice_match, or max_lazy_match is less than 4.

See Also

Special Parameters

DeflateLZ77Fastest

Performs LZ77 encoding according to the fastest algorithm.

Syntax

```
IppStatus ippsDeflateLZ77Fastest_8u(const Ipp8u** ppSrc, Ipp32u* pSrcLen, Ipp32u*
pSrcIdx, const Ipp8u* pWindow, Ipp32u winSize, Ipp32s* pHashHead, Ipp32u hashSize,
Ipp16u* pCode, Ipp32u* pCodeLenBits, Ipp8u* pDst, Ipp32u dstLen, Ipp32u* pDstIdx,
IppDeflateHuffCode pLitHuffCodes[286], IppDeflateHuffCode pDistHuffCodes[30],
IppLZ77Flush flush);
```

Include Files

ippdc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

ppSrc	Double pointer to the source vector.
pSrcLen	Pointer to the length of the source vector.
pSrcIdx	Pointer to the index of the current position in the source vector.
pWindow	Pointer to the sliding window (the dictionary for the LZ77 algorithm).

winSize	Size of the sliding window and the pHashPrev table.
pHashHead	Pointer to the table containing heads of the hash chains.
hashSize	Size of the pHashHead table.
pCode	Pointer to the bit buffer.
pCodeLenBits	Pointer to the number of valid bits in the bit buffer.
pDst	Pointer to the destination vector.
dstLen	Lenght of the destination vector.
pDstIdx	Pointer to the index in the destination vector.
pLitHuffCodes	Pointer to the literals/lengths Huffman codes.
pDistHuffCodes	Pointer to the distances Huffman codes.
flush	Specifies the encoding mode for data blocks (see flush parameter).

Description

This function performs LZ77 encoding of the ppSrc data according to the fastest algorithm.

To correctly process the first bytes of the source vector, initialize the pHashHead table with the winSize value.

The pSrcIdx parameter returns the index of the current position in the source vector, and is used to establish a correlation between the current position in the source vector and indexes in hash tables. After processing each 2GB of source data, the index and hash tables must be normalized (instead of 64K of source data in ZLIB).

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error if winSize is less than 256, or more than 32768; or hashSize is less than 256, or more than 65536.

See Also

Special Parameters

DeflateLZ77FastestGenHeader

Computes a header of the provided Huffman tables description.

Syntax

```
IppStatus ippsDeflateLZ77FastestGenHeader_8u(const IppDeflateHuffCode
pLitCodeTable[286], const IppDeflateHuffCode pDistCodeTable[30], Ipp8u* pDstHeader,
int* pDstLen, int* pDstBits);
```

Include Files

`ippdc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pLitCodeTable`	Pointer to literals/lengths in a Huffman code.
`pDistCodeTable`	Pointer to distances in a Huffman code.
`pDstHeader`	Pointer to the header.
`pDstLen`	Pointer to the header length.
`pDstBits`	Pointer to the header length, in bits.

Description

This function gets a header of the provided Huffman tables description.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when one of the specified pointers is `NULL`.

DeflateLZ77FastestGenHuffTable

Builds Huffman tables according to statistical data collections.

Syntax

```
IppStatus ippsDeflateLZ77FastestGenHuffTable_8u(const int pLitStat[286], const int
pDistStat[30], IppDeflateHuffCode pLitCodeTable[286], IppDeflateHuffCode
pDistCodeTable[30]);
```

Include Files

`ippdc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

pLitStat	Pointer to data collection for literals and match lengths.
pDistStat	Pointer to data collection for distances.
pLitCodeTable	Pointer to the literals/lengths Huffman codes.
pDistCodeTable	Pointer to the distances Huffman codes.

Description

This function builds Huffman tables for literals/lengths according to the provided statistical data collection.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.

DeflateLZ77FastestGetStat

Performs statistical data collection required to build user's Huffman table.

Syntax

```
IppStatus ippsDeflateLZ77FastestGetStat_8u(const Ipp8u** ppSrc, Ipp32u* pSrcLen,
Ipp32u* pSrcIdx, const Ipp8u* pWindow, Ipp32u winSize, Ipp32s* pHashHead, Ipp32u
hashSize, int pLitStat[286], int pDistStat[30], IppLZ77Flush flush);
```

Include Files

ippdc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

ppSrc	Double pointer to the source vector.
pSrcLen	Pointer to the length of the source vector.
pSrcIdx	Pointer to the index of the current position in the source vector.
pWindow	Pointer to the sliding window (the dictionary for the LZ77 algorithm).
winSize	Size of the sliding window and the pHashPrev table.

pHashHead	Pointer to the table containing heads of the hash chains.
hashSize	Size of the pHashHead table.
pLitStat	Pointer to data collection for literals and match lengths.
pDistStat	Pointer to data collection for distances.
flush	Specifies the encoding mode for data blocks.

Description

This function performs collection of statistical data. This data is needed for functions building user's Huffman table ippsDeflateLZ77FastestGenHuffTable and functions computing a header of Huffman tables description ippsDeflateLZ77FastestGenHeader.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when winSize is less than 256 or more than 32768, or if hashSize is less than 256 or more than 65536.

DeflateLZ77FastestPrecompHeader

Performs LZ77 encoding using the fastest algorithm with prebuilding of customer Huffman tables and prediction header.

Syntax

```
IppStatus ippsDeflateLZ77FastestPrecompHeader_8u(const Ipp8u** ppSrc, Ipp32u* pSrcLen,
Ipp32u* pSrcIdx, const Ipp8u* pWindow, Ipp32u winSize, Ipp32s* pHashHead, Ipp32u
hashSize, Ipp16u* pCode, Ipp32u* pCodeLenBits, Ipp8u* pDst, Ipp32u dstLen, Ipp32u*
pDstIdx, IppDeflateHuffCode pLitHuffCodes[288], IppDeflateHuffCode pDistHuffCodes[30],
const Ipp8u* pHeaderCodeLens, int numBitsHeader, IppLZ77Flush flush);
```

Include Files

ippdc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

ppSrc	Double pointer to the source vector.

pSrcLen	Pointer to the length of the source vector.
pSrcIdx	Pointer to the index of the current position in the source vector.
pWindow	Pointer to the sliding window (the dictionary for the LZ77 algorithm).
winSize	Size of the sliding window and the pHashPrev table.
pHashHead	Pointer to the table containing heads of the hash chains.
hashSize	Size of the pHashHead table.
pCode	Pointer to the bit buffer.
pCodeLenBits	Pointer to the number of valid bits in the bit buffer.
pDst	Pointer to the destination vector.
dstLen	The length of the destination vector.
pDstIdx	Pointer to the index in the destination vector.
pLitHuffCodes	Pointer to the literals/lengths Huffman codes.
pDistHuffCodes	Pointer to the distances Huffman codes.
pHeaderCodeLens	Pointer to the prediction header with description of Huffman tables.
numBitsHeader	Length of the prediction header, in bits.
flush	Specifies the encoding mode for data blocks.

Description

This function performs LZ77 encoding of source data ppSrc using the fastest algorithm.

To correctly process the first bytes of the source vector, initialize the pHashHead table with-winSize value.

The pSrcIdx parameter returns the index of the current position in the source vector and is used to correlate the current position in the source vector and indexes in the hash tables. After processing each 2GB of source data, this index and hash tables must be normalized (instead of 64K of source data in ZLIB).

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when winSize is less than 256 or more than 32768, or if hashSize is less than 256 or more than 65536, or if * pDstIdx is more than or equal to dstLen.

DeflateLZ77Slow

Performs LZ77 encoding according to the slow algorithm and parameters of a match.

Syntax

```
IppStatus ippsDeflateLZ77Slow_8u(const Ipp8u** ppSrc, Ipp32u* pSrcLen, Ipp32u* pSrcIdx,
const Ipp8u* pWindow, Ipp32u winSize, Ipp32s* pHashHead, Ipp32s* pHashPrev, Ipp32u
hashSize, IppDeflateFreqTable pLitFreqTable[286], IppDeflateFreqTable
pDistFreqTable[30], Ipp8u* pLitDst, Ipp16u* pDistDst, Ipp32u* pDstLen, int* pVecMatch,
IppLZ77Flush flush);
```

Include Files

`ippdc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`ppSrc`	Double pointer to the source vector.
`pSrcLen`	Pointer to the length of the source vector.
`pSrcIdx`	Pointer to the index of the current position in the source vector.
`pWindow`	Pointer to the sliding window (the dictionary for the LZ77 algorithm).
`winSize`	Size of the sliding window and the `pHashPrev` table.
`pHashHead`	Pointer to the table containing heads of the hash chains.
`pHashPrev`	Pointer to the table contating indexes to the previous strings with the same hash key.
`hashSize`	Size of the `pHashHead` table.
`pLitFreqTable`	Pointer to the literals/lengths frequency table.
`pDistFreqTable`	Pointer to the distances frequency table.
`pLitDst`	Pointer to the destination vector containing literals/ lengths.
`pDistDst`	Pointer to the destination vector containing distances.
`pDstLen`	Pointer to the length of the destination vectors.

pVecMatch	Pointer to the vector containing the following parameters of a match: max_chain_length, good_match, nice_match, max_lazy_match (for more information, see [ZLIB]).
flush	Specifies the encoding mode for data blocks (see flush parameter).

Description

This function performs LZ77 encoding of source data ppSrc according to the slow algorithm and match parameters.

To correctly process the first bytes of the source vector, initialize the pHashHead table with the winSize value.

The pSrcIdx parameter returns the index of the current position in the source vector, and is used to establish a correlation between the current position in the source vector and indexes in hash tables. After processing each 2GB of source data, the index and hash tables must be normalized (instead of 64K of source data in ZLIB).

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error if winSize is less than 256 or more than 32768 or hashSize is less than 256 or more than 65536.
ippStsBadArgErr	Indicates an error when good_match, nice_match, or max_lazy_match is less than 4, or max_chain_length is less than 1.

See Also

Special Parameters

DeflateDictionarySet

Presets the user's dictionary for LZ77 encoding.

Syntax

```
IppStatus ippsDeflateDictionarySet_8u(const Ipp8u* pDictSrc, Ipp32u dictLen, Ipp32s*
pHashHeadDst, Ipp32u hashSize, Ipp32s* pHashPrevDst, Ipp8u* pWindowDst, Ipp32u winSize,
int comprLevel);
```

Include Files

ippdc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pDictSrc	Pointer to the user's dictionary.
dictLen	Length of the user's dictionary.
pHashHeadDst	Pointer to the table containing heads of the hash chains.
pHashPrevDst	Pointer to the table containing indexes to the previous strings with the same hash key.
hashSize	Size of the pHashHeadDst table.
pWindowDst	Pointer to the sliding window that is used as the dictionary for LZ77 encoding.
winSize	Size of the sliding window and the elements of the pHashPrevDst table.
comprLevel	Compression level in range [0..9] in accordance with ZLIB.

Description

This function presets the user's dictionary for LZ77 encoding.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if one of the specified pointers is NULL.

DeflateUpdate Hash

Performs LZ77 encoding according to the specified compression level.

Syntax

```
IppStatus ippsDeflateUpdateHash_8u(const Ipp8u* pSrc, Ipp32u srcIdx, Ipp32u srcLen,
Ipp32s* pHashHeadDst, Ipp32u hashSize, Ipp32s* pHashPrevDst, Ipp32u winSize, int
comprLevel);
```

Include Files

ippdc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc	Pointer to the source vector.
srcIdx	Index of the current position in the source vector.
srcLen	Length of the source vector.
pHashHeadDst	Pointer to the table containing heads of the hash chains.
hashSize	Size of the pHashHeadDst table.
pHashPrevDst	Pointer to the table contating indexes to the previous strings with the same hash key.
winSize	Size of the sliding window and the pHashPrevDst table.
comprLevel	Compression level in range [0..9] in accordance with ZLIB.

Description

This function updates hash tables according to the source context.

The function parameter srcIdx - index of the current position in the source vector - is used to correlate the current position in the source vector and indexes in the hash tables. After processing each 2GB of source data, this index and hash tables must be normalized (instead of 64K of source data in ZLIB).

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error if winSize is less than or equal to 256, or greater than 32768; or if hashSize is less than or equal to 256, or greater than 65536.

DeflateHuff

Performs Huffman encoding .

Syntax

```
IppStatus ippsDeflateHuff_8u(const Ipp8u* pLitSrc, const Ipp16u* pDistSrc, Ipp32u
srcLen, Ipp16u* pCode, Ipp32u* pCodeLenBits, IppDeflateHuffCode pLitHuffCodes[286],
IppDeflateHuffCode pDistHuffCodes[30], Ipp8u* pDst, Ipp32u* pDstIdx);
```

Include Files

ippdc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pLitSrc	Pointer to the literals/lengths source vector.
pDistSrc	Pointer to the distances source vector.
srcLen	Length of the source vectors.
pCode	Pointer to the bit buffer.
pCodeLenBits	Pointer to the number of valid bits in the bit buffer.
pLitHuffCodes	Pointer to the literals/lengths Huffman codes.
pDistHuffCodes	Pointer to the distances Huffman codes.
pDst	Pointer to the destination vector.
pDstIdx	Pointer to the index in the destination vector.

Description

This function performs Huffman encoding of source data.

The function parameter pDstIdx returns the index of the current position in the destination vector: zlib uses the intermediate buffer for the Huffman encoding and we need to know the indexes of the first (input parameter) and the last (output parameter) symbols, which are written by the function.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if one of the specified pointers is NULL.

InflateBuildHuffTable

Builds the Huffman code table for compressed block in the “deflate” format.

Syntax

```
IppStatus ippsInflateBuildHuffTable(const Ipp16u* pCodeLens, unsigned int nLitCodeLens,
unsigned int nDistCodeLens, IppInflateState* pIppInflateState);
```

Include Files

ippdc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pCodeLens	Pointer to the common array with lengths of the Huffman codes for literals/lengths and distances.
nLitCodeLens	Number of lengths of the Huffman codes for literals/lengths.
nDistCodeLens	Number of lengths of the Huffman codes for distances.
pIppInflateState	Pointer to the structure with the parameters of decoding.

Description

This function builds tables of Hufman codes for literals/lengths and distances to decode a block compressed with use of the dynamic Huffman codes in accordance with the "deflate" format [RFC1951].

The structure IppInflateState contains the following fields:

pWindow	Pointer to the sliding window (the dictionary for the LZ77 algorithm).
winSize	Size of the sliding window in the range [256, 32768].
tableType	Type of the Huffman code tables. For dynamic Huffman code it is greater than 0, for fixed Huffman codes is equal to 0.
tableBufferSize	Size of the buffer containing the tables. Its value is 8192 - sizeof(IppInflateState). (8192 =ENOUGH*sizeof(code); ENOUGH is defined in ZLIB and is equal to 2048, sizeof(code)=4.)

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if one of the pointers is NULL.
ippStsSizeErr	Indicates an error if nLitCodeLens is greater than 286 or nDistCodeLens is greater than 30.
ippStsSrcDataErr	Indicates an error if a not valid literal/length and distance set occurs in the common lengths array.

Inflate

Decodes data in the "deflate" format.

Syntax

```
IppStatus ippsInflate_8u(Ipp8u** ppSrc, unsigned int* pSrcLen, Ipp32u* pCode, unsigned int* pCodeLenBits, unsigned int winIdx, Ipp8u** ppDst, unsigned int* pDstLen, unsigned int dstIdx, IppInflateMode* pMode, IppInflateState* pIppInflateState);
```

Include Files

ippdc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

ppSrc	Double pointer to the source vector.
pSrcLen	Pointer to the length of source vector.
pCode	Pointer to the bit buffer.
pCodeLenBits	Number of valid bits in the bit buffer.
winIdx	Index of the start position of the sliding window.
ppDst	Double pointer to the destination vector.
pDstLen	Pointer to the length of destination vector.
dstIdx	Index of the current position in the destination vector.
pMode	Pointer to the current decode mode. Possible values are: • ippTYPE - block decoding is completed; • ippLEN - decoding from the beginning of the sequence; • ippLENEXT - extra bits are required to decode the sequence.
pIppInflateState	Pointer to the structure that contains parameters of decoding.

Description

This function decodes the data encoded in the "deflate" format [RFC1951] in accordance with the parameters set in the structure pIppInflateState. If the data is compressed using dynamic Huffman codes, the Huffman code tables must be built by the function ippsInflateBuildHuffTable beforehand. If the data is compressed using the fixed Huffman codes, the fileld tableType in the pIppInflateState must be set to 0, and code tables are not required to be built at all.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if one of the pointers is NULL.
ippStsSizeErr	Indicates an error if pCodeLenBits is greater than 32, or if winIdx is greater than pIppInflateState->winSize, or if dstIdx is greater than pDstLen.

ippStsSrcDataErr	Indicates an error if a not valid literal/length and distance set occurs during decoding.

LZO Compression Functions

This section describes Intel IPP data compression functions, that implement the LZO (Lempel-Ziv-Oberhumer) compressed data format. This format and algorithm use 64KB compression dictionary and do not require additional memory for decompression. (See original code of the LZO library at http://www.oberhumer.com.)

- Special Parameters
- EncodeLZOGetSize
- EncodeLZOInit
- EncodeLZO
- DecodeLZO
- DecodeLZOSafe

Special Parameters

The LZO coding initialization functions have a special parameter `method`. This parameter specifies level of parallelization and generic LZO compatibility to be used in the LZO encoding. The table below lists possible values of the `method` parameter and their meanings.

Value	Descriptions
IppLZO1XST	The compression and decompression are performed sequentially in a single-thread mode with full binary compatibility with generic LZO libraries and applications.
IppLZO1XMT	The compression and decompression are performed in parallel (multi-threaded mode), it is more fast, but not compatible with the generic LZO.
IppLZO1X1ST	The compression and decompression are performed sequentially in a single-threaded mode with full binary compatibility with generic LZO libraries and applications. The compression ratio of this method corresponds to the LZO `lzo1x_1_compress` function (default function for the `lzop` compressor). The `IppLZO1X1ST` method provides lower compression ratio than `IppLZO1XST` but better compression performance.

EncodeLZOGetSize

Calculates the size of LZO encoding structure.

Syntax

```
IppStatus ippsEncodeLZOGetSize(IppLZOMethod method, Ipp32u maxInputLen, Ipp32u* pSize);
```

Include Files

ippdc.h

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`method`	Specifies required LZO compression method, possible values are listed in Table "method Parameter").
`maxInputLen`	Specifies the maximum length of the input data buffer during compression operations. Not required for the `IppLZO1XST` and `IppLZO1X1ST` compression methods.
`pSize`	Pointer to the variable, receiving the size of LZO encoding structure.

Description

This function calculates the size of the memory buffer that must be allocated for the LZO encoding structure.

For the single-thread compression (`method = IppLZO1XST`) the size of the structure is fixed, and the value of the `maxInputLen` parameter is ignored, for example, it can be set to 0.

For the multi-threaded compression (`method = IppLZO1XMT`) `maxInputLen` parameter is important and affects the size of the structure. If it is set to 0, then each compression operation starts with memory allocation for internal buffers and ends with memory freeing. This significantly decreases the performance of compression/decompression.

Code example shows how the Intel IPP functions for the LZO compression can be used.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if the pointer `pSize` is `NULL`.
`ippStsBadArgErr`	Indicates an error if the parameter `method` has an illegal value.

Example

```
/*******************************************************************************
 * Copyright 2017 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
```

```
 *****************************************************************************/

/*
 The example below shows how to use the functions:
        ippsEncodeLZOGetSize
        ippsEncodeLZO_8u
        ippsEncodeLZOInit_8u
        ippsDecodeLZO_8u
        ippsDecodeLZOSafe_8u
*/

#include <stdio.h>
#include <string.h>

#include <ipp/ippdc.h> /* Data compression functions       */
#include <ipp/ipps.h>  /* Malloc/vector initialization function   */

/* Next two defines are created to simplify code reading and understanding          */
#define EXIT_MAIN \
    exitLine: /* Label for Exit    */
#define
check_sts(st)
                                  \
    if ((st) !=
ippStsNoErr)
                           \
        goto exitLine; /* Go to Exit if Intel(R) Integrated Performance Primitives (Intel(R)
IPP) function returned status different from \
                            ippStsNoErr */

#define TEST_SIZE   (1024 * 4)
#define DECOMPR_MAX (TEST_SIZE + 64)

int main(void)
{
    Ipp8u *srcBuf = NULL, *comprBuf = NULL, *decomprBuf = NULL;
    IppStatus st;
    int i;
    Ipp32u stateSize;
    IppLZOState_8u *lzoState = NULL;
    Ipp32u comprLen, decomprLen;

    /* Allocate buffers */
    srcBuf = ippsMalloc_8u(TEST_SIZE);
    decomprBuf = ippsMalloc_8u(DECOMPR_MAX); /* Extra 64 bytes may be required
                                                for non-safe decompression       */
    comprBuf = ippsMalloc_8u(TEST_SIZE + TEST_SIZE / 10);
    /* Initialize source buffer */
    check_sts(st = ippsVectorJaehne_8u(srcBuf, TEST_SIZE, IPP_MAX_8U));
    for (i = 0; i < TEST_SIZE; i++)
        srcBuf[i] >>= 6; /* Decrease source data entropy */
    /* Allocate and initialize LZO structure */
    check_sts(st = ippsEncodeLZOGetSize(IppLZO1XST, TEST_SIZE, &stateSize));
    lzoState = (IppLZOState_8u *)ippsMalloc_8u(stateSize);
    check_sts(st = ippsEncodeLZOInit_8u(IppLZO1XST, TEST_SIZE, lzoState));
    /* LZO compress, decompress and compare */
    comprLen = (TEST_SIZE + TEST_SIZE / 10);
    check_sts(st = ippsEncodeLZO_8u(srcBuf, TEST_SIZE, comprBuf, &comprLen, lzoState));
```

```
    printf("LZO compress: %d bytes compressed into %d bytes\n", TEST_SIZE, (int)comprLen);
    decomprLen = DECOMPR_MAX;
    check_sts(st = ippsDecodeLZO_8u(comprBuf, comprLen, decomprBuf, &decomprLen));
    if (decomprLen != TEST_SIZE || memcmp(decomprBuf, srcBuf, TEST_SIZE) != 0) {
        printf("Non-safe decompress failed!\n");
        st = ippStsErr;
        goto exitLine;
    } else
        printf("Non-safe decompress verified.\n");
    /* Decompress using safe function */
    ippsZero_8u(decomprBuf, DECOMPR_MAX);
    decomprLen = TEST_SIZE; /* Can specify exact size */
    check_sts(st = ippsDecodeLZOSafe_8u(comprBuf, comprLen, decomprBuf, &decomprLen));
    if (decomprLen != TEST_SIZE || memcmp(decomprBuf, srcBuf, TEST_SIZE) != 0) {
        printf("Safe decompress failed!\n");
        st = ippStsErr;
        goto exitLine;
    } else
        printf("Safe decompress verified.\n");

    EXIT_MAIN
    ippsFree(srcBuf);
    ippsFree(decomprBuf);
    ippsFree(comprBuf);
    ippsFree(lzoState);
    return (int)st;
}
```

EncodeLZOInit

Initializes LZO encoding structure.

Syntax

```
IppStatus ippsEncodeLZOInit_8u(IppLZOMethod method, Ipp32u maxInputLen, IppLZOState_8u*
pLZOState);
```

Include Files

`ippdc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`method`	Specifies required LZO compression method, possible values are listed in Table "method Parameter").

maxInputLen	Specifies the maximum length of the input data buffer during compression operations. Not required for the IppLZO1XST and IppLZO1X1ST compression methods.
pLZOState	Pointer to the LZO encoding structure.

Description

This function initializes the LZO encoding structure in the external buffer. Its size must be calculated by calling the function ippsEncodeLZOGetSize beforehand.

The parameter method must be the same for both functions.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if the pointer pLZOState is NULL.
ippStsBadArgErr I	ndicates an error if the parameter method has an illegal value.

Example

```
/*******************************************************************************
 * Copyright 2017 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

/*
 The example below shows how to use the functions:
        ippsEncodeLZOGetSize
        ippsEncodeLZO_8u
        ippsEncodeLZOInit_8u
        ippsDecodeLZO_8u
        ippsDecodeLZOSafe_8u
*/

#include <stdio.h>
#include <string.h>

#include <ipp/ippdc.h> /* Data compression functions       */
#include <ipp/ipps.h>  /* Malloc/vector initialization function   */

/* Next two defines are created to simplify code reading and understanding          */
#define EXIT_MAIN \
```

```
    exitLine: /* Label for Exit    */
#define
check_sts(st)
                                  \
    if ((st) !=
ippStsNoErr)
                          \
        goto exitLine; /* Go to Exit if Intel(R) Integrated Performance Primitives (Intel(R)
IPP) function returned status different from \
                           ippStsNoErr */

#define TEST_SIZE   (1024 * 4)
#define DECOMPR_MAX (TEST_SIZE + 64)

int main(void)
{
    Ipp8u *srcBuf = NULL, *comprBuf = NULL, *decomprBuf = NULL;
    IppStatus st;
    int i;
    Ipp32u stateSize;
    IppLZOState_8u *lzoState = NULL;
    Ipp32u comprLen, decomprLen;

    /* Allocate buffers */
    srcBuf = ippsMalloc_8u(TEST_SIZE);
    decomprBuf = ippsMalloc_8u(DECOMPR_MAX); /* Extra 64 bytes may be required
                                                for non-safe decompression       */
    comprBuf = ippsMalloc_8u(TEST_SIZE + TEST_SIZE / 10);
    /* Initialize source buffer */
    check_sts(st = ippsVectorJaehne_8u(srcBuf, TEST_SIZE, IPP_MAX_8U));
    for (i = 0; i < TEST_SIZE; i++)
        srcBuf[i] >>= 6; /* Decrease source data entropy */
    /* Allocate and initialize LZO structure */
    check_sts(st = ippsEncodeLZOGetSize(IppLZO1XST, TEST_SIZE, &stateSize));
    lzoState = (IppLZOState_8u *)ippsMalloc_8u(stateSize);
    check_sts(st = ippsEncodeLZOInit_8u(IppLZO1XST, TEST_SIZE, lzoState));
    /* LZO compress, decompress and compare */
    comprLen = (TEST_SIZE + TEST_SIZE / 10);
    check_sts(st = ippsEncodeLZO_8u(srcBuf, TEST_SIZE, comprBuf, &comprLen, lzoState));
    printf("LZO compress: %d bytes compressed into %d bytes\n", TEST_SIZE, (int)comprLen);
    decomprLen = DECOMPR_MAX;
    check_sts(st = ippsDecodeLZO_8u(comprBuf, comprLen, decomprBuf, &decomprLen));
    if (decomprLen != TEST_SIZE || memcmp(decomprBuf, srcBuf, TEST_SIZE) != 0) {
        printf("Non-safe decompress failed!\n");
        st = ippStsErr;
        goto exitLine;
    } else
        printf("Non-safe decompress verified.\n");
    /* Decompress using safe function */
    ippsZero_8u(decomprBuf, DECOMPR_MAX);
    decomprLen = TEST_SIZE; /* Can specify exact size */
    check_sts(st = ippsDecodeLZOSafe_8u(comprBuf, comprLen, decomprBuf, &decomprLen));
    if (decomprLen != TEST_SIZE || memcmp(decomprBuf, srcBuf, TEST_SIZE) != 0) {
        printf("Safe decompress failed!\n");
        st = ippStsErr;
        goto exitLine;
    } else
        printf("Safe decompress verified.\n");
```

```
    EXIT_MAIN
    ippsFree(srcBuf);
    ippsFree(decomprBuf);
    ippsFree(comprBuf);
    ippsFree(lzoState);
    return (int)st;
}
```

EncodeLZO

Compresses input data, returns the length of the compressed data.

Syntax

```
IppStatus ippsEncodeLZO_8u (const Ipp8u* pSrc, Ipp32u srcLen, Ipp8u* pDst, Ipp32u*
pDstLen, IppLZOState_8u* pLZOState);
```

Include Files

`ippdc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source buffer.
`srcLen`	Length of the source buffer.
`pDst`	Pointer to the destination buffer.
`pDstLen`	Pointer to the length of the destination buffer.
`pLZOState`	Pointer to the LZO state structure.

Description

This function performs compression of the source data `pSrc` according to the method specified in the LZO state structure `pLZOState`. It must be previously initialized by the function ippsEncodeLZOInit.

Compressed data are stored in the `pDst`, the pointer `pDstLen` points to the number of elements in this buffer.

Code example shows how the Intel IPP functions for the LZO compression can be used.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if one of the specified pointers is `NULL`.

Example

```
/*******************************************************************************
 * Copyright 2017 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

/*
 The example below shows how to use the functions:
        ippsEncodeLZOGetSize
        ippsEncodeLZO_8u
        ippsEncodeLZOInit_8u
        ippsDecodeLZO_8u
        ippsDecodeLZOSafe_8u
*/

#include <stdio.h>
#include <string.h>

#include <ipp/ippdc.h> /* Data compression functions       */
#include <ipp/ipps.h>  /* Malloc/vector initialization function   */

/* Next two defines are created to simplify code reading and understanding          */
#define EXIT_MAIN \
    exitLine: /* Label for Exit    */
#define
check_sts(st)
                                  \
    if ((st) !=
ippStsNoErr)
                          \
        goto exitLine; /* Go to Exit if Intel(R) Integrated Performance Primitives (Intel(R)
IPP) function returned status different from \
                           ippStsNoErr */

#define TEST_SIZE   (1024 * 4)
#define DECOMPR_MAX (TEST_SIZE + 64)

int main(void)
{
    Ipp8u *srcBuf = NULL, *comprBuf = NULL, *decomprBuf = NULL;
    IppStatus st;
    int i;
    Ipp32u stateSize;
    IppLZOState_8u *lzoState = NULL;
    Ipp32u comprLen, decomprLen;

    /* Allocate buffers */
```

```
    srcBuf = ippsMalloc_8u(TEST_SIZE);
    decomprBuf = ippsMalloc_8u(DECOMPR_MAX); /* Extra 64 bytes may be required
                                                for non-safe decompression       */
    comprBuf = ippsMalloc_8u(TEST_SIZE + TEST_SIZE / 10);
    /* Initialize source buffer */
    check_sts(st = ippsVectorJaehne_8u(srcBuf, TEST_SIZE, IPP_MAX_8U));
    for (i = 0; i < TEST_SIZE; i++)
        srcBuf[i] >>= 6; /* Decrease source data entropy */
    /* Allocate and initialize LZO structure */
    check_sts(st = ippsEncodeLZOGetSize(IppLZO1XST, TEST_SIZE, &stateSize));
    lzoState = (IppLZOState_8u *)ippsMalloc_8u(stateSize);
    check_sts(st = ippsEncodeLZOInit_8u(IppLZO1XST, TEST_SIZE, lzoState));
    /* LZO compress, decompress and compare */
    comprLen = (TEST_SIZE + TEST_SIZE / 10);
    check_sts(st = ippsEncodeLZO_8u(srcBuf, TEST_SIZE, comprBuf, &comprLen, lzoState));
    printf("LZO compress: %d bytes compressed into %d bytes\n", TEST_SIZE, (int)comprLen);
    decomprLen = DECOMPR_MAX;
    check_sts(st = ippsDecodeLZO_8u(comprBuf, comprLen, decomprBuf, &decomprLen));
    if (decomprLen != TEST_SIZE || memcmp(decomprBuf, srcBuf, TEST_SIZE) != 0) {
        printf("Non-safe decompress failed!\n");
        st = ippStsErr;
        goto exitLine;
    } else
        printf("Non-safe decompress verified.\n");
    /* Decompress using safe function */
    ippsZero_8u(decomprBuf, DECOMPR_MAX);
    decomprLen = TEST_SIZE; /* Can specify exact size */
    check_sts(st = ippsDecodeLZOSafe_8u(comprBuf, comprLen, decomprBuf, &decomprLen));
    if (decomprLen != TEST_SIZE || memcmp(decomprBuf, srcBuf, TEST_SIZE) != 0) {
        printf("Safe decompress failed!\n");
        st = ippStsErr;
        goto exitLine;
    } else
        printf("Safe decompress verified.\n");

    EXIT_MAIN
    ippsFree(srcBuf);
    ippsFree(decomprBuf);
    ippsFree(comprBuf);
    ippsFree(lzoState);
    return (int)st;
}
```

DecodeLZO

Decompresses input data, returns the length of the decompressed data.

Syntax

```
IppStatus ippsDecodeLZO_8u (const Ipp8u* pSrc, Ipp32u srcLen, Ipp8u* pDst, Ipp32u*
pDstLen);
```

Include Files

```
ippdc.h
```

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source data buffer.
`srcLen`	Number of elements in the source data buffer.
`pDst`	Pointer to the destination data buffer.
`pDstLen`	Pointer to the variable with the number of elements in the destination data buffer.

Description

The function decompresses the source (compressed) data according to the compressed data format. This function can decompress both single-thread and multi-threaded data. Note that the maximum performance can be obtained only in multi-threaded decompression of data compressed in the multi-threaded mode.

NOTE Destination data buffer must have enough free space to hold uncompressed data. No output buffer check is performed and no error code is returned. In the case of doubts use safe version of this function DecodeLZOSafe .

Code example shows how the Intel IPP functions for the LZO compression can be used.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if one of the specified pointers is `NULL`.

Example

```
/*******************************************************************************
 * Copyright 2017 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

/*
 The example below shows how to use the functions:
        ippsEncodeLZOGetSize
```

```
        ippsEncodeLZO_8u
        ippsEncodeLZOInit_8u
        ippsDecodeLZO_8u
        ippsDecodeLZOSafe_8u
*/

#include <stdio.h>
#include <string.h>

#include <ipp/ippdc.h> /* Data compression functions       */
#include <ipp/ipps.h>  /* Malloc/vector initialization function   */

/* Next two defines are created to simplify code reading and understanding          */
#define EXIT_MAIN \
    exitLine: /* Label for Exit    */
#define
check_sts(st)
                                  \
    if ((st) !=
ippStsNoErr)
                           \
        goto exitLine; /* Go to Exit if Intel(R) Integrated Performance Primitives (Intel(R)
IPP) function returned status different from \
                            ippStsNoErr */

#define TEST_SIZE   (1024 * 4)
#define DECOMPR_MAX (TEST_SIZE + 64)

int main(void)
{
    Ipp8u *srcBuf = NULL, *comprBuf = NULL, *decomprBuf = NULL;
    IppStatus st;
    int i;
    Ipp32u stateSize;
    IppLZOState_8u *lzoState = NULL;
    Ipp32u comprLen, decomprLen;

    /* Allocate buffers */
    srcBuf = ippsMalloc_8u(TEST_SIZE);
    decomprBuf = ippsMalloc_8u(DECOMPR_MAX); /* Extra 64 bytes may be required
                                                  for non-safe decompression       */
    comprBuf = ippsMalloc_8u(TEST_SIZE + TEST_SIZE / 10);
    /* Initialize source buffer */
    check_sts(st = ippsVectorJaehne_8u(srcBuf, TEST_SIZE, IPP_MAX_8U));
    for (i = 0; i < TEST_SIZE; i++)
        srcBuf[i] >>= 6; /* Decrease source data entropy */
    /* Allocate and initialize LZO structure */
    check_sts(st = ippsEncodeLZOGetSize(IppLZO1XST, TEST_SIZE, &stateSize));
    lzoState = (IppLZOState_8u *)ippsMalloc_8u(stateSize);
    check_sts(st = ippsEncodeLZOInit_8u(IppLZO1XST, TEST_SIZE, lzoState));
    /* LZO compress, decompress and compare */
    comprLen = (TEST_SIZE + TEST_SIZE / 10);
    check_sts(st = ippsEncodeLZO_8u(srcBuf, TEST_SIZE, comprBuf, &comprLen, lzoState));
    printf("LZO compress: %d bytes compressed into %d bytes\n", TEST_SIZE, (int)comprLen);
    decomprLen = DECOMPR_MAX;
    check_sts(st = ippsDecodeLZO_8u(comprBuf, comprLen, decomprBuf, &decomprLen));
    if (decomprLen != TEST_SIZE || memcmp(decomprBuf, srcBuf, TEST_SIZE) != 0) {
        printf("Non-safe decompress failed!\n");
```

```
        st = ippStsErr;
        goto exitLine;
    } else
        printf("Non-safe decompress verified.\n");
    /* Decompress using safe function */
    ippsZero_8u(decomprBuf, DECOMPR_MAX);
    decomprLen = TEST_SIZE; /* Can specify exact size */
    check_sts(st = ippsDecodeLZOSafe_8u(comprBuf, comprLen, decomprBuf, &decomprLen));
    if (decomprLen != TEST_SIZE || memcmp(decomprBuf, srcBuf, TEST_SIZE) != 0) {
        printf("Safe decompress failed!\n");
        st = ippStsErr;
        goto exitLine;
    } else
        printf("Safe decompress verified.\n");

    EXIT_MAIN
    ippsFree(srcBuf);
    ippsFree(decomprBuf);
    ippsFree(comprBuf);
    ippsFree(lzoState);
    return (int)st;
}
```

DecodeLZOSafe

Decompresses input data with constantly checking integrity of output.

Syntax

```
IppStatus ippsDecodeLZOSafe_8u(const Ipp8u* pSrc, Ipp32u srcLen, Ipp8u* pDst, Ipp32u* pDstLen);
```

Include Files

`ippdc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source data buffer.
`srcLen`	Number of elements in the source data buffer.
`pDst`	Pointer to the destination data buffer.
`pDstLen`	Pointer to the variable with the number of elements in the destination data buffer.

Description

This function is a version of the function `ippsDecodeLZO`. It decompresses the source (compressed) data according to the compressed data format. The function can decompress both single-thread and multi-threaded data. The maximum performance can be obtained only in multi-threaded decompression of data

compressed in the multi-threaded mode. Additionally this function checks the integrity of the destination data buffer, that is checks the buffer boundary limits. This function works slower, it can be used in doubtful cases when the compressed data integrity is not guaranteed, for example, decoding data received via non-reliable communication lines.

Destination data buffer must have enough free space to hold uncompressed data. Prior to the function call the destination buffer size variable pointed to by `pDstLen` must be initialized with actual number of free bytes in the destination buffer.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if one of the specified pointers is `NULL`.
`ippStsLzoBrokenStreamErr`	Indicates an error if compressed data is not valid - not an LZO compressed data.
`ippStsDstSizeLessExpected`	Destination buffer is too small to store decompressed data.

Example

```
/*******************************************************************************
 * Copyright 2017 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

/*
 The example below shows how to use the functions:
        ippsEncodeLZOGetSize
        ippsEncodeLZO_8u
        ippsEncodeLZOInit_8u
        ippsDecodeLZO_8u
        ippsDecodeLZOSafe_8u
*/

#include <stdio.h>
#include <string.h>

#include <ipp/ippdc.h> /* Data compression functions       */
#include <ipp/ipps.h>  /* Malloc/vector initialization function   */

/* Next two defines are created to simplify code reading and understanding          */
#define EXIT_MAIN \
    exitLine: /* Label for Exit    */
```

```
#define
check_sts(st)
                                  \
    if ((st) !=
ippStsNoErr)
                           \
        goto exitLine; /* Go to Exit if Intel(R) Integrated Performance Primitives (Intel(R)
IPP) function returned status different from \
                           ippStsNoErr */

#define TEST_SIZE   (1024 * 4)
#define DECOMPR_MAX (TEST_SIZE + 64)

int main(void)
{
    Ipp8u *srcBuf = NULL, *comprBuf = NULL, *decomprBuf = NULL;
    IppStatus st;
    int i;
    Ipp32u stateSize;
    IppLZOState_8u *lzoState = NULL;
    Ipp32u comprLen, decomprLen;

    /* Allocate buffers */
    srcBuf = ippsMalloc_8u(TEST_SIZE);
    decomprBuf = ippsMalloc_8u(DECOMPR_MAX); /* Extra 64 bytes may be required
                                                for non-safe decompression       */
    comprBuf = ippsMalloc_8u(TEST_SIZE + TEST_SIZE / 10);
    /* Initialize source buffer */
    check_sts(st = ippsVectorJaehne_8u(srcBuf, TEST_SIZE, IPP_MAX_8U));
    for (i = 0; i < TEST_SIZE; i++)
        srcBuf[i] >>= 6; /* Decrease source data entropy */
    /* Allocate and initialize LZO structure */
    check_sts(st = ippsEncodeLZOGetSize(IppLZO1XST, TEST_SIZE, &stateSize));
    lzoState = (IppLZOState_8u *)ippsMalloc_8u(stateSize);
    check_sts(st = ippsEncodeLZOInit_8u(IppLZO1XST, TEST_SIZE, lzoState));
    /* LZO compress, decompress and compare */
    comprLen = (TEST_SIZE + TEST_SIZE / 10);
    check_sts(st = ippsEncodeLZO_8u(srcBuf, TEST_SIZE, comprBuf, &comprLen, lzoState));
    printf("LZO compress: %d bytes compressed into %d bytes\n", TEST_SIZE, (int)comprLen);
    decomprLen = DECOMPR_MAX;
    check_sts(st = ippsDecodeLZO_8u(comprBuf, comprLen, decomprBuf, &decomprLen));
    if (decomprLen != TEST_SIZE || memcmp(decomprBuf, srcBuf, TEST_SIZE) != 0) {
        printf("Non-safe decompress failed!\n");
        st = ippStsErr;
        goto exitLine;
    } else
        printf("Non-safe decompress verified.\n");
    /* Decompress using safe function */
    ippsZero_8u(decomprBuf, DECOMPR_MAX);
    decomprLen = TEST_SIZE; /* Can specify exact size */
    check_sts(st = ippsDecodeLZOSafe_8u(comprBuf, comprLen, decomprBuf, &decomprLen));
    if (decomprLen != TEST_SIZE || memcmp(decomprBuf, srcBuf, TEST_SIZE) != 0) {
        printf("Safe decompress failed!\n");
        st = ippStsErr;
        goto exitLine;
    } else
        printf("Safe decompress verified.\n");
```

```
    EXIT_MAIN
    ippsFree(srcBuf);
    ippsFree(decomprBuf);
    ippsFree(comprBuf);
    ippsFree(lzoState);
    return (int)st;
}
```

LZ4 Compression Functions

This section describes Intel IPP data compression functions that implement the LZ4 compressed data format. This format and algorithm use 64Kb compression dictionary. The original code of the library is available at http://www.lz4.org.

- EncodeLZ4HashTableGetSize
- EncodeLZ4HashTableInit, EncodeLZ4DictHashTableInit
- EncodeLZ4LoadDict
- EncodeLZ4
- EncodeLZ4Safe
- DecodeLZ4

EncodeLZ4HashTableGetSize

Calculates the size of the LZ4 hash table.

Syntax

```
IppStatus ippsEncodeLZ4HashTableGetSize(int* pHashTableSize);
```

Include Files

`ippdc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pHashTableSize`	Pointer to the variable containing the size of the LZ4 hash table.

Description

This function calculates the size of the memory buffer that must be allocated for the LZ4 hash table.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if the `pHashTableSize` pointer is `NULL`.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

/*
 The example below shows how to use the functions:
     ippsEncodeLZ4HashTableGetSize_8u
     ippsEncodeLZ4HashTableInit_8u
     ippsEncodeLZ4_8u
     ippsDecodeLZ4_8u
*/

#include <stdio.h>
#include <string.h>

#include <ipp/ippdc.h>
#include <ipp/ipps.h>

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define
check_sts(st)
                                  \
    if ((st) !=
ippStsNoErr)
                          \
        goto exitLine; /* Go to Exit if Intel(R) Integrated Performance Primitives (Intel(R)
IPP) function returned status different from \
                           ippStsNoErr */

#define TEST_SIZE (1024)

int main(void)
{
    Ipp8u *srcBuf = NULL, *comprBuf = NULL, *decomprBuf = NULL, *hashTable = NULL;
    IppStatus st = ippStsNoErr;
    int hashSize = 0, comprLen = TEST_SIZE + TEST_SIZE / 255 + 16, /* Extra bytes for
                                                                      uncompressible data */
        decomprLen = TEST_SIZE + 33;
    int i;

    srcBuf = ippsMalloc_8u(TEST_SIZE);
    decomprBuf = ippsMalloc_8u(decomprLen); /* Spare bytes for "wild"
                                       (non-safe) decompression */
```

```
    comprBuf = ippsMalloc_8u(comprLen);
    /* Initialize source buffer */
    check_sts(st = ippsVectorJaehne_8u(srcBuf, TEST_SIZE, IPP_MAX_8U));
    for (i = 0; i < TEST_SIZE; i++)
        srcBuf[i] >>= 6; /* Decrease source data entropy */
    /* Init and allocate hash table */
    check_sts(st = ippsEncodeLZ4HashTableGetSize_8u(&hashSize));
    hashTable = ippsMalloc_8u(hashSize);
    check_sts(st = ippsEncodeLZ4HashTableInit_8u(hashTable, TEST_SIZE));
    /* Compress source data */
    check_sts(st = ippsEncodeLZ4_8u((const Ipp8u *)srcBuf, TEST_SIZE, comprBuf, &comprLen,
hashTable));
    /* Print compression result */
    printf("Compression: %d bytes compressed into %d bytes\n", TEST_SIZE, comprLen);
    /* Decompression */
    decomprLen = TEST_SIZE + 33;
    check_sts(st = ippsDecodeLZ4_8u((const Ipp8u *)comprBuf, comprLen, decomprBuf, &decomprLen));
    /* Check */
    if (decomprLen == TEST_SIZE) /* Decompressed size must be equal to source data size */
    {
        if (memcmp(srcBuf, decomprBuf, TEST_SIZE) != 0) {
            printf("Wrong decompression!\n");
            st = ippStsErr;
        } else
            printf("Decompressed by ippsDecodeLZ4_8u OK\n");
    } else
        printf("Invalid decompressed length %d\n", decomprLen);

    EXIT_MAIN
    ippsFree(srcBuf);
    ippsFree(comprBuf);
    ippsFree(decomprBuf);
    ippsFree(hashTable);
    return (int)st;
}
```

EncodeLZ4HashTableInit, EncodeLZ4DictHashTableInit

Initializes the LZ4 hash table.

Syntax

```
IppStatus ippsEncodeLZ4HashTableInit_8u(Ipp8u* pHashTable, int srcLen);
```

```
IppStatus ippsEncodeLZ4DictHashTableInit_8u(Ipp8u* pHashTable, int srcLen);
```

Include Files

`ippdc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

pHashTable	Pointer to the LZ4 hash table.
srcLen	Length of the source data for compression.

Description

This function initializes the LZ4 hash table. Before using this function, compute the size of the LZ4 hash table using the EncodeLZ4HashTableGetSize function.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if the pHashTable pointer is NULL.
ippStsSizeErr	Indicates an error if the srcLen value is less than, or equal to zero.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

/*
 The example below shows how to use the functions:
     ippsEncodeLZ4HashTableGetSize_8u
     ippsEncodeLZ4HashTableInit_8u
     ippsEncodeLZ4_8u
     ippsDecodeLZ4_8u
*/

#include <stdio.h>
#include <string.h>

#include <ipp/ippdc.h>
#include <ipp/ipps.h>

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define
check_sts(st)
```

```
                                    \
    if ((st) !=
ippStsNoErr)
                            \
        goto exitLine; /* Go to Exit if Intel(R) Integrated Performance Primitives (Intel(R)
IPP) function returned status different from \
                            ippStsNoErr */

#define TEST_SIZE (1024)

int main(void)
{
    Ipp8u *srcBuf = NULL, *comprBuf = NULL, *decomprBuf = NULL, *hashTable = NULL;
    IppStatus st = ippStsNoErr;
    int hashSize = 0, comprLen = TEST_SIZE + TEST_SIZE / 255 + 16, /* Extra bytes for
                                                                      uncompressible data */
        decomprLen = TEST_SIZE + 33;
    int i;

    srcBuf = ippsMalloc_8u(TEST_SIZE);
    decomprBuf = ippsMalloc_8u(decomprLen); /* Spare bytes for "wild"
                                       (non-safe) decompression */
    comprBuf = ippsMalloc_8u(comprLen);
    /* Initialize source buffer */
    check_sts(st = ippsVectorJaehne_8u(srcBuf, TEST_SIZE, IPP_MAX_8U));
    for (i = 0; i < TEST_SIZE; i++)
        srcBuf[i] >>= 6; /* Decrease source data entropy */
    /* Init and allocate hash table */
    check_sts(st = ippsEncodeLZ4HashTableGetSize_8u(&hashSize));
    hashTable = ippsMalloc_8u(hashSize);
    check_sts(st = ippsEncodeLZ4HashTableInit_8u(hashTable, TEST_SIZE));
    /* Compress source data */
    check_sts(st = ippsEncodeLZ4_8u((const Ipp8u *)srcBuf, TEST_SIZE, comprBuf, &comprLen,
hashTable));
    /* Print compression result */
    printf("Compression: %d bytes compressed into %d bytes\n", TEST_SIZE, comprLen);
    /* Decompression */
    decomprLen = TEST_SIZE + 33;
    check_sts(st = ippsDecodeLZ4_8u((const Ipp8u *)comprBuf, comprLen, decomprBuf, &decomprLen));
    /* Check */
    if (decomprLen == TEST_SIZE) /* Decompressed size must be equal to source data size */
    {
        if (memcmp(srcBuf, decomprBuf, TEST_SIZE) != 0) {
            printf("Wrong decompression!\n");
            st = ippStsErr;
        } else
            printf("Decompressed by ippsDecodeLZ4_8u OK\n");
    } else
        printf("Invalid decompressed length %d\n", decomprLen);

    EXIT_MAIN
    ippsFree(srcBuf);
    ippsFree(comprBuf);
    ippsFree(decomprBuf);
    ippsFree(hashTable);
    return (int)st;
}
```

EncodeLZ4LoadDict

Initializes the LZ4 hash table that uses dictionary.

Syntax

```
IppStatus ippsEncodeLZ4LoadDict_8u(Ipp8u* pHashTable, const Ipp8u* pDict, int dictLen);
```

Include Files

`ippdc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pHashTable`	Pointer to the LZ4 hash table.
`pDict`	Pointer to the dictionary.
`dictLen`	Length of the dictionary.

Description

This function initializes the LZ4 hash table with values from the dictionary.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if the `pHashTable` pointer is `NULL`.
`ippStsSizeErr`	Indicates an error if the `dictLen` value is less than, or equal to zero.

EncodeLZ4

Performs LZ4 encoding.

Syntax

```
IppStatus ippsEncodeLZ4_8u(const Ipp8u* pSrc, int srcLen, Ipp8u* pDst, int* pDstLen,
Ipp8u* pHashTable);
```

```
IppStatus ippsEncodeLZ4Fast_8u(const Ipp8u* pSrc, int srcLen, Ipp8u* pDst, int*
pDstLen, Ipp8u* pHashTable, int* acceleration);
```

```
IppStatus ippsEncodeLZ4Safe_8u(const Ipp8u* pSrc, int* pSrcLen, Ipp8u* pDst, int*
pDstLen, Ipp8u* pHashTable);
```

```
IppStatus ippsEncodeLZ4Dict_8u(const Ipp8u* pSrc, int srcIdx, int srcLen, Ipp8u* pDst,
int* pDstLen, Ipp8u* pHashTable, const Ipp8u* pDict, int dictLen);
```

```
IppStatus ippsEncodeLZ4DictSafe_8u(const Ipp8u* pSrc, int srcIdx, int* pSrcLen, Ipp8u*
pDst, int* pDstLen, Ipp8u* pHashTable, const Ipp8u* pDict, int dictLen);
```

Include Files

`ippdc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source data.
`srcLen`	Length of the source data for compression.
`srcIdx`	Index of the starting byte in the source vector.
`pSrcLen`	Pointer to the length of the source data for compression.
`pDst`	Pointer to the compressed data.
`pDstLen`	Pointer to the length of the compressed data.
`pHashTable`	Pointer to the LZ4 hash table.
`pDict`	Pointer to the dictionary.
`dictLen`	Length of the dictionary.
`acceleraion`	Acceleration value.

Description

These functions perform encoding of the source data `pSrc` using the LZ4 algorithm. The destination buffer must have sufficient length for the operation. The length of the compressed data is set to `pDstLen`.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if at least one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error if the `srcLen` value is less than, or equal to zero.
`ippStsBadArgErr`	Indicates an error if the index of the starting byte is less than zero.
`ippStsDstSizeLessExpected`	Indicates an error if the length of the destination buffer is not sufficient.

Example

LZ4:

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

/*
 The example below shows how to use the functions:
     ippsEncodeLZ4HashTableGetSize_8u
     ippsEncodeLZ4HashTableInit_8u
     ippsEncodeLZ4_8u
     ippsDecodeLZ4_8u
*/

#include <stdio.h>
#include <string.h>

#include <ipp/ippdc.h>
#include <ipp/ipps.h>

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define
check_sts(st)
                                  \
    if ((st) !=
ippStsNoErr)
                          \
        goto exitLine; /* Go to Exit if Intel(R) Integrated Performance Primitives (Intel(R)
IPP) function returned status different from \
                           ippStsNoErr */

#define TEST_SIZE (1024)

int main(void)
{
    Ipp8u *srcBuf = NULL, *comprBuf = NULL, *decomprBuf = NULL, *hashTable = NULL;
    IppStatus st = ippStsNoErr;
    int hashSize = 0, comprLen = TEST_SIZE + TEST_SIZE / 255 + 16, /* Extra bytes for
                                                                      uncompressible data */
        decomprLen = TEST_SIZE + 33;
    int i;

    srcBuf = ippsMalloc_8u(TEST_SIZE);
```

```
    decomprBuf = ippsMalloc_8u(decomprLen); /* Spare bytes for "wild"
                                       (non-safe) decompression */
    comprBuf = ippsMalloc_8u(comprLen);
    /* Initialize source buffer */
    check_sts(st = ippsVectorJaehne_8u(srcBuf, TEST_SIZE, IPP_MAX_8U));
    for (i = 0; i < TEST_SIZE; i++)
        srcBuf[i] >>= 6; /* Decrease source data entropy */
    /* Init and allocate hash table */
    check_sts(st = ippsEncodeLZ4HashTableGetSize_8u(&hashSize));
    hashTable = ippsMalloc_8u(hashSize);
    check_sts(st = ippsEncodeLZ4HashTableInit_8u(hashTable, TEST_SIZE));
    /* Compress source data */
    check_sts(st = ippsEncodeLZ4_8u((const Ipp8u *)srcBuf, TEST_SIZE, comprBuf, &comprLen,
hashTable));
    /* Print compression result */
    printf("Compression: %d bytes compressed into %d bytes\n", TEST_SIZE, comprLen);
    /* Decompression */
    decomprLen = TEST_SIZE + 33;
    check_sts(st = ippsDecodeLZ4_8u((const Ipp8u *)comprBuf, comprLen, decomprBuf, &decomprLen));
    /* Check */
    if (decomprLen == TEST_SIZE) /* Decompressed size must be equal to source data size */
    {
        if (memcmp(srcBuf, decomprBuf, TEST_SIZE) != 0) {
            printf("Wrong decompression!\n");
            st = ippStsErr;
        } else
            printf("Decompressed by ippsDecodeLZ4_8u OK\n");
    } else
        printf("Invalid decompressed length %d\n", decomprLen);

    EXIT_MAIN
    ippsFree(srcBuf);
    ippsFree(comprBuf);
    ippsFree(decomprBuf);
    ippsFree(hashTable);
    return (int)st;
}
```

LZ4Dict:

```
/*******************************************************************************
 * Copyright 2017 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

/*
 The example below shows how to use the functions:
     ippsEncodeLZ4HashTableGetSize_8u
```

```
     ippsEncodeLZ4HashTableInit_8u
     ippsEncodeLZ4Dict_8u
     ippsDecodeLZ4_8u
*/

#include <stdio.h>
#include <string.h>

#include <ipp.h>

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define
check_sts(st)
                                  \
    if ((st) !=
ippStsNoErr)
                           \
        goto exitLine; /* Go to Exit if Intel(R) Integrated Performance Primitives (Intel(R)
IPP) function returned status different from \
                            ippStsNoErr */

#define TEST_SIZE  (1024 * 4)
#define CHUNK_SIZE (512)
#define RATIO      (TEST_SIZE / CHUNK_SIZE)

static int Verify(const Ipp8u *comprBuf, int comprLen, Ipp8u *decomprBuf, int maxDecomprLen,
const Ipp8u *srcBuf, int srcLen, const Ipp8u *dictBuf);

int main(void)
{
    Ipp8u *srcBuf = NULL, *comprBuf = NULL, *decomprBuf = NULL, *hashTable = NULL;
    Ipp8u *comprPtr, *dict;
    IppStatus st = ippStsNoErr;
    int hashSize = 0, maxChunkCompr = CHUNK_SIZE + CHUNK_SIZE / 255 + 16, comprLen =
(maxChunkCompr)*RATIO, /* Extra bytes for uncompressible

                                       data */
        decomprLen = TEST_SIZE + 33;
    int destLen;
    int i;

    srcBuf = ippsMalloc_8u(TEST_SIZE);
    decomprBuf = ippsMalloc_8u(decomprLen); /* Spare bytes for "wild" (non-safe)
                                                                    decompression */
    comprBuf = ippsMalloc_8u(comprLen);
    /* Initialize source buffer */
    check_sts(st = ippsVectorJaehne_8u(srcBuf, TEST_SIZE, IPP_MAX_8U));
    for (i = 0; i < TEST_SIZE; i++)
        srcBuf[i] >>= 6; /* Decrease source data entropy */
    /* Allocate and initialize hash table */
    check_sts(st = ippsEncodeLZ4HashTableGetSize_8u(&hashSize));
    hashTable = ippsMalloc_8u(hashSize);
    check_sts(st = ippsEncodeLZ4HashTableInit_8u(hashTable, TEST_SIZE));
    /* Compress source data using chunks without dictionary*/
    comprPtr = comprBuf;
    for (i = 0; i < RATIO; i++) {
```

```
        destLen = maxChunkCompr;
        check_sts(st = ippsEncodeLZ4_8u(srcBuf + CHUNK_SIZE * i, CHUNK_SIZE, comprPtr + 2,
&destLen, hashTable));
        *comprPtr++ = destLen & 0xFF;
        *comprPtr++ = (destLen >> 8) & 0xFF;
        comprPtr += destLen;
    }
    /* Print compression result */
    comprLen = (int)(comprPtr - comprBuf);
    printf("Compression w/o dictionary: %d bytes compressed into %d bytes.", TEST_SIZE,
comprLen);
    if (!Verify(comprBuf, comprLen, decomprBuf, decomprLen, srcBuf, TEST_SIZE, NULL)) {
        printf("Verification failed!\n");
        st = ippStsErr;
        goto exitLine;
    } else
        printf("Verified.\n");

    /* Compress source data with dictionary*/
    comprPtr = comprBuf;
    dict = srcBuf;
    for (i = 0; i < RATIO; i++) {
        int srcIdx, dictLen;
        destLen = maxChunkCompr;
        srcIdx = CHUNK_SIZE * i;
        dictLen = CHUNK_SIZE * i; /* All previous chunks as dictionary */

        check_sts(st = ippsEncodeLZ4Dict_8u(srcBuf + CHUNK_SIZE * i, srcIdx, CHUNK_SIZE,
comprPtr + 2, &destLen, hashTable, dict, dictLen));
        *comprPtr++ = destLen & 0xFF;
        *comprPtr++ = (destLen >> 8) & 0xFF;
        comprPtr += destLen;
    }
    /* Print compression result */
    comprLen = (int)(comprPtr - comprBuf);
    printf("Compression with dictionary: %d bytes compressed into %d bytes.", TEST_SIZE,
comprLen);
    if (!Verify(comprBuf, comprLen, decomprBuf, decomprLen, srcBuf, TEST_SIZE, dict)) {
        printf("Verification failed!\n");
        st = ippStsErr;
        goto exitLine;
    } else
        printf("Verified.\n");

    EXIT_MAIN
    if (st != ippStsNoErr)
        printf("Function status: %s\n", ippGetStatusString(st));
    ippsFree(srcBuf);
    ippsFree(comprBuf);
    ippsFree(decomprBuf);
    ippsFree(hashTable);
    return (int)st;
}

static int Verify(const Ipp8u *comprBuf, int comprLen, Ipp8u *decomprBuf, int maxDecomprLen,
const Ipp8u *srcBuf, int srcLen, const Ipp8u *dictBuf)
{
    const Ipp8u *locPtr = comprBuf;
```

```
    Ipp8u *decomprPtr = decomprBuf;
    IppStatus st;
    int decomprLen = 0;
    int dictLen = 0;

    while ((int)(locPtr - comprBuf) < comprLen) {
        int nextInLen;
        int nextOutLen;

        nextInLen = (*locPtr++);
        nextInLen += (*locPtr++) << 8;
        nextOutLen = maxDecomprLen - decomprLen;
        st = ippsDecodeLZ4Dict_8u(locPtr, &nextInLen, decomprPtr, dictLen, &nextOutLen, dictBuf,
dictLen);
        if (st != ippStsNoErr)
            return 0;
        locPtr += nextInLen;
        decomprLen += nextOutLen;
        dictLen += nextOutLen;
    }
    if (decomprLen != srcLen || memcmp(decomprBuf, srcBuf, srcLen) != 0)
        return 0;
    return 1;
}
```

LZ4Safe:

```
/*******************************************************************************
 * Copyright 2017 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

/*
 The example below shows how to use the functions:
     ippsEncodeLZ4HashTableGetSize_8u
     ippsEncodeLZ4HashTableInit_8u
     ippsEncodeLZ4Safe_8u
*/

#include <stdio.h>
#include <string.h>

#include <ipp/ippdc.h>
#include <ipp/ipps.h>

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
```

```
#define
check_sts(st)
                                    \
    if ((st) !=
ippStsNoErr)
                            \
        goto exitLine; /* Go to Exit if Intel(R) Integrated Performance Primitives (Intel(R)
IPP) function returned status different from \
                           ippStsNoErr */

#define TEST_SIZE (1024)
#define COMPR_LEN (TEST_SIZE / 20)

int main(void)
{
    Ipp8u *srcBuf = NULL, *comprBuf = NULL, *hashTable = NULL;
    IppStatus st = ippStsNoErr;
    int i, hashSize, remBytes, srcLen, dstLen;

    srcBuf = ippsMalloc_8u(TEST_SIZE);
    comprBuf = ippsMalloc_8u(COMPR_LEN);
    /* Initialize source buffer */
    check_sts(st = ippsVectorJaehne_8u(srcBuf, TEST_SIZE, IPP_MAX_8U));
    for (i = 0; i < TEST_SIZE; i++)
        srcBuf[i] >>= 6; /* Decrease source data entropy */
    /* Init and allocate hash table */
    check_sts(st = ippsEncodeLZ4HashTableGetSize_8u(&hashSize));
    hashTable = ippsMalloc_8u(hashSize);
    check_sts(st = ippsEncodeLZ4HashTableInit_8u(hashTable, TEST_SIZE));
    /* Compress source data */
    remBytes = TEST_SIZE;
    while (remBytes > 0) {
        srcLen = remBytes;
        dstLen = COMPR_LEN;
        /* Compressing into knowingly small buffer */
        st = ippsEncodeLZ4Safe_8u((const Ipp8u *)srcBuf + TEST_SIZE - remBytes, &srcLen,
comprBuf, &dstLen, hashTable);
        if (st == ippStsDstSizeLessExpected) {
            /* Print compression result */
            printf("Compression: %d bytes compressed into %d bytes\n", srcLen, dstLen);
        } else if (st < ippStsNoErr) {
            printf("Error in compression function\n");
            break;
        }
        remBytes -= srcLen;
    }
    EXIT_MAIN
    ippsFree(srcBuf);
    ippsFree(comprBuf);
    ippsFree(hashTable);
    return (int)st;
}
```

EncodeLZ4Safe

Performs LZ4 encoding.

Syntax

```
IppStatus ippsEncodeLZ4Safe_8u(const Ipp8u* pSrc, int* srcLen, Ipp8u* pDst, int* pDstLen, Ipp8u* pHashTable);
```

Include Files

`ippdc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source data.
`srcLen`	Length of the source data for compression.
`pDst`	Pointer to the compressed data.
`pDstLen`	Pointer to the length of the destination buffer and the length of the compressed data.
`pHashTable`	Pointer to the LZ4 hash table.

Description

This function performs encoding of the source data `pSrc` using the LZ4 algorithm. The length of the compressed data is set to `pDstLen`. The length of the processed source data is set to `pSrcLen`

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if at lease one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error if the `srcLen` or `dstLen` value is less than, or equal to zero.
`ippStsDstSizeLessExpectedErr`	Indicates an error if the destination buffer has insufficient length.

DecodeLZ4

Performs LZ4 decoding.

Syntax

```
IppStatus ippsDecodeLZ4_8u(const Ipp8u* pSrc, int srcLen, Ipp8u* pDst, int* pDstLen);
IppStatus ippsDecodeLZ4Dict_8u(const Ipp8u* pSrc, int* pSrcLen, Ipp8u* pDst, int dstIdx, int* pDstLen, const Ipp8u* pDict, int dictSize);
```

Include Files

ippdc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc	Pointer to the source data.
srcLen	Length of the source data for decompression.
pSrcLen	Pointer to the length of the source data for decompression.
pDst	Pointer to the compressed data.
pDstLen	Pointer to the length of the uncompressed data.
dstIdx	Index of the starting byte in the destination vector.
pDict	Pointer to the dictionary.
dictSize	Length of the dictionary.

Description

This function performs decoding of the source data pSrc using the LZ4 algorithm. The destination buffer must have sufficient length for the operation. The length of the uncompressed data is set to pDstLen.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if at lease one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error if the srcLen value is less than, or equal to zero.
ippStsMemAllocErr	Indicates an error if the size of the allocated memory is not sufficient for decompression.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
```

```
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *****************************************************************************/

/*
 The example below shows how to use the functions:
     ippsEncodeLZ4HashTableGetSize_8u
     ippsEncodeLZ4HashTableInit_8u
     ippsEncodeLZ4_8u
     ippsDecodeLZ4_8u
*/

#include <stdio.h>
#include <string.h>

#include <ipp/ippdc.h>
#include <ipp/ipps.h>

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define
check_sts(st)
                                  \
    if ((st) !=
ippStsNoErr)
                          \
        goto exitLine; /* Go to Exit if Intel(R) Integrated Performance Primitives (Intel(R)
IPP) function returned status different from \
                           ippStsNoErr */

#define TEST_SIZE (1024)

int main(void)
{
    Ipp8u *srcBuf = NULL, *comprBuf = NULL, *decomprBuf = NULL, *hashTable = NULL;
    IppStatus st = ippStsNoErr;
    int hashSize = 0, comprLen = TEST_SIZE + TEST_SIZE / 255 + 16, /* Extra bytes for
                                                                      uncompressible data */
        decomprLen = TEST_SIZE + 33;
    int i;

    srcBuf = ippsMalloc_8u(TEST_SIZE);
    decomprBuf = ippsMalloc_8u(decomprLen); /* Spare bytes for "wild"
                                       (non-safe) decompression */
    comprBuf = ippsMalloc_8u(comprLen);
    /* Initialize source buffer */
    check_sts(st = ippsVectorJaehne_8u(srcBuf, TEST_SIZE, IPP_MAX_8U));
    for (i = 0; i < TEST_SIZE; i++)
        srcBuf[i] >>= 6; /* Decrease source data entropy */
    /* Init and allocate hash table */
    check_sts(st = ippsEncodeLZ4HashTableGetSize_8u(&hashSize));
    hashTable = ippsMalloc_8u(hashSize);
    check_sts(st = ippsEncodeLZ4HashTableInit_8u(hashTable, TEST_SIZE));
    /* Compress source data */
    check_sts(st = ippsEncodeLZ4_8u((const Ipp8u *)srcBuf, TEST_SIZE, comprBuf, &comprLen,
hashTable));
```

```
    /* Print compression result */
    printf("Compression: %d bytes compressed into %d bytes\n", TEST_SIZE, comprLen);
    /* Decompression */
    decomprLen = TEST_SIZE + 33;
    check_sts(st = ippsDecodeLZ4_8u((const Ipp8u *)comprBuf, comprLen, decomprBuf, &decomprLen));
    /* Check */
    if (decomprLen == TEST_SIZE) /* Decompressed size must be equal to source data size */
    {
        if (memcmp(srcBuf, decomprBuf, TEST_SIZE) != 0) {
            printf("Wrong decompression!\n");
            st = ippStsErr;
        } else
            printf("Decompressed by ippsDecodeLZ4_8u OK\n");
    } else
        printf("Invalid decompressed length %d\n", decomprLen);

    EXIT_MAIN
    ippsFree(srcBuf);
    ippsFree(comprBuf);
    ippsFree(decomprBuf);
    ippsFree(hashTable);
    return (int)st;
}
```

LZ4 Compression Functions for High Compression (HC) Mode

This section describes Intel IPP data compression functions that implement the LZ4 compressed data format and can be used in high compression (HC) mode. This format and algorithm use 64Kb compression dictionary. The original code of the library is available at http://www.lz4.org.

- EncodeLZ4HCHashTableGetSize
- EncodeLZ4HCHashTableInit
- EncodeLZ4HC

EncodeLZ4HCHashTableGetSize

Calculates the size of the LZ4 `HashTable` and `PrevTable` for HC mode.

Syntax

```
IppStatus ippsEncodeLZ4HCHashTableGetSize(int* pHashTableSize, int* pPrevTableSize);
```

Include Files

`ippdc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pHashTableSize`	Pointer to the variable containing the size of the LZ4 `HashTable`.

pHashPrevSize	Pointer to the variable containing the size of the LZ4 PrevTable.

Description

This function calculates the size of the memory buffer that must be allocated for the LZ4 HashTable and PrevTable in HC mode.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if pHashTableSize or pPrevTableSize is NULL.

Example

```
/*******************************************************************************
 * Copyright 2017 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

/*
 The example below shows how to use the functions:
     ippsEncodeLZ4HCHashTableGetSize_8u
     ippsEncodeLZ4HCHashTableInit_8u
     ippsEncodeLZ4HC_8u
     ippsDecodeLZ4_8u
*/

#include <stdio.h>
#include <string.h>

#include <ipp/ippdc.h>
#include <ipp/ipps.h>

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define
check_sts(st)
                                  \
    if ((st) !=
ippStsNoErr)
                          \
        goto exitLine; /* Go to Exit if Intel(R) Integrated Performance Primitives (Intel(R)
```

```
IPP) function returned status different from \
                          ippStsNoErr */

#define TEST_SIZE (1024)

static int checkDecompr(Ipp8u *decomprAddr, int decomprLen, Ipp8u *srcAddr, int srcLen);

int main(void)
{
    Ipp8u *srcBuf = NULL, *comprBuf = NULL, *decomprBuf = NULL, *hashTable = NULL, *prevTable =
NULL;
    Ipp8u *tables[2];
    IppStatus st = ippStsNoErr;
    int hashSize = 0, prevSize = 0, srcLen = TEST_SIZE, comprLen = TEST_SIZE + TEST_SIZE / 255 +
16, /* Extra bytes for

        uncompressible data */
        decomprLen = TEST_SIZE +
33;                                                              /* Spare bytes for "wild"

    (non-safe) decompression */
    int i;

    srcBuf = ippsMalloc_8u(TEST_SIZE);
    decomprBuf = ippsMalloc_8u(decomprLen);
    comprBuf = ippsMalloc_8u(comprLen);
    /* Initialize source buffer */
    check_sts(st = ippsVectorJaehne_8u(srcBuf, TEST_SIZE, IPP_MAX_8U));
    for (i = 0; i < TEST_SIZE; i++)
        srcBuf[i] >>= 6; /* Decrease source data entropy */
    /* Init and allocate hash table */
    check_sts(st = ippsEncodeLZ4HCHashTableGetSize_8u(&hashSize, &prevSize));
    hashTable = ippsMalloc_8u(hashSize);
    prevTable = ippsMalloc_8u(prevSize);
    tables[0] = hashTable;
    tables[1] = prevTable;
    check_sts(st = ippsEncodeLZ4HCHashTableInit_8u(tables));
    /* Compress source data at level 1*/
    check_sts(st = ippsEncodeLZ4HC_8u((const Ipp8u *)srcBuf, 0, &srcLen, comprBuf, &comprLen,
tables, NULL, 0, 1));
    /* Print compression result */
    printf("Compression lev 1: %d bytes compressed into %d bytes\n", TEST_SIZE, comprLen);
    /* Decompression */
    decomprLen = TEST_SIZE + 33;
    check_sts(st = ippsDecodeLZ4_8u((const Ipp8u *)comprBuf, comprLen, decomprBuf, &decomprLen));
    check_sts(st = checkDecompr(decomprBuf, decomprLen, srcBuf, srcLen));

    /* Compress source data at level 6*/
    comprLen = TEST_SIZE + TEST_SIZE / 255 + 16;
    check_sts(st = ippsEncodeLZ4HC_8u((const Ipp8u *)srcBuf, 0, &srcLen, comprBuf, &comprLen,
tables, NULL, 0, 6));
    /* Print compression result */
    printf("Compression lev 6: %d bytes compressed into %d bytes\n", TEST_SIZE, comprLen);
    /* Decompression */
    decomprLen = TEST_SIZE + 33;
    check_sts(st = ippsDecodeLZ4_8u((const Ipp8u *)comprBuf, comprLen, decomprBuf, &decomprLen));
    check_sts(st = checkDecompr(decomprBuf, decomprLen, srcBuf, srcLen));
```

```
    /* Compress source data at level 9*/
    comprLen = TEST_SIZE + TEST_SIZE / 255 + 16;
    check_sts(st = ippsEncodeLZ4HC_8u((const Ipp8u *)srcBuf, 0, &srcLen, comprBuf, &comprLen,
tables, NULL, 0, 9));
    /* Print compression result */
    printf("Compression lev 9: %d bytes compressed into %d bytes\n", TEST_SIZE, comprLen);
    /* Decompression */
    decomprLen = TEST_SIZE + 33;
    check_sts(st = ippsDecodeLZ4_8u((const Ipp8u *)comprBuf, comprLen, decomprBuf, &decomprLen));
    st = checkDecompr(decomprBuf, decomprLen, srcBuf, srcLen);

    EXIT_MAIN
    ippsFree(srcBuf);
    ippsFree(comprBuf);
    ippsFree(decomprBuf);
    ippsFree(hashTable);
    ippsFree(prevTable);
    return (int)st;
}
static IppStatus checkDecompr(Ipp8u *decomprAddr, int decomprLen, Ipp8u *srcAddr, int srcLen)
{
    /* Check */
    if (decomprLen != srcLen || memcmp(decomprAddr, srcAddr, srcLen) != 0) {
        printf("Wrong decompression!\n");
        return ippStsErr;
    }
    printf("Decompressed OK into %d bytes\n", decomprLen);
    return ippStsNoErr;
}
```

EncodeLZ4HCHashTableInit

Initializes LZ4 hash tables for HC mode.

Syntax

```
IppStatus ippsEncodeLZ4HCHashTableInit_8u(Ipp8u** ppHashTables);
```

Include Files

`ippdc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`ppHashTables`	Pointer to an array of pointers to LZ4 hash tables for HC mode; `ppHashTables[0] = pHashTable`, `ppHashTables[1] = pPrevTable`.

Description

This function initializes the LZ4 `HashTable` and `PrevTable`. Before using this function, compute the size of the LZ4 hash tables using the EncodeLZ4HCHashTableGetSize function.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if any of the `ppHashTables` pointers is `NULL`.

Example

```
/*******************************************************************************
 * Copyright 2017 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

/*
 The example below shows how to use the functions:
     ippsEncodeLZ4HCHashTableGetSize_8u
     ippsEncodeLZ4HCHashTableInit_8u
     ippsEncodeLZ4HC_8u
     ippsDecodeLZ4_8u
*/

#include <stdio.h>
#include <string.h>

#include <ipp/ippdc.h>
#include <ipp/ipps.h>

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define
check_sts(st)
                                  \
    if ((st) !=
ippStsNoErr)
                          \
        goto exitLine; /* Go to Exit if Intel(R) Integrated Performance Primitives (Intel(R)
IPP) function returned status different from \
                           ippStsNoErr */

#define TEST_SIZE (1024)
```

```
static int checkDecompr(Ipp8u *decomprAddr, int decomprLen, Ipp8u *srcAddr, int srcLen);

int main(void)
{
    Ipp8u *srcBuf = NULL, *comprBuf = NULL, *decomprBuf = NULL, *hashTable = NULL, *prevTable =
NULL;
    Ipp8u *tables[2];
    IppStatus st = ippStsNoErr;
    int hashSize = 0, prevSize = 0, srcLen = TEST_SIZE, comprLen = TEST_SIZE + TEST_SIZE / 255 +
16, /* Extra bytes for

        uncompressible data */
        decomprLen = TEST_SIZE +
33;                                                              /* Spare bytes for "wild"

    (non-safe) decompression */
    int i;

    srcBuf = ippsMalloc_8u(TEST_SIZE);
    decomprBuf = ippsMalloc_8u(decomprLen);
    comprBuf = ippsMalloc_8u(comprLen);
    /* Initialize source buffer */
    check_sts(st = ippsVectorJaehne_8u(srcBuf, TEST_SIZE, IPP_MAX_8U));
    for (i = 0; i < TEST_SIZE; i++)
        srcBuf[i] >>= 6; /* Decrease source data entropy */
    /* Init and allocate hash table */
    check_sts(st = ippsEncodeLZ4HCHashTableGetSize_8u(&hashSize, &prevSize));
    hashTable = ippsMalloc_8u(hashSize);
    prevTable = ippsMalloc_8u(prevSize);
    tables[0] = hashTable;
    tables[1] = prevTable;
    check_sts(st = ippsEncodeLZ4HCHashTableInit_8u(tables));
    /* Compress source data at level 1*/
    check_sts(st = ippsEncodeLZ4HC_8u((const Ipp8u *)srcBuf, 0, &srcLen, comprBuf, &comprLen,
tables, NULL, 0, 1));
    /* Print compression result */
    printf("Compression lev 1: %d bytes compressed into %d bytes\n", TEST_SIZE, comprLen);
    /* Decompression */
    decomprLen = TEST_SIZE + 33;
    check_sts(st = ippsDecodeLZ4_8u((const Ipp8u *)comprBuf, comprLen, decomprBuf, &decomprLen));
    check_sts(st = checkDecompr(decomprBuf, decomprLen, srcBuf, srcLen));

    /* Compress source data at level 6*/
    comprLen = TEST_SIZE + TEST_SIZE / 255 + 16;
    check_sts(st = ippsEncodeLZ4HC_8u((const Ipp8u *)srcBuf, 0, &srcLen, comprBuf, &comprLen,
tables, NULL, 0, 6));
    /* Print compression result */
    printf("Compression lev 6: %d bytes compressed into %d bytes\n", TEST_SIZE, comprLen);
    /* Decompression */
    decomprLen = TEST_SIZE + 33;
    check_sts(st = ippsDecodeLZ4_8u((const Ipp8u *)comprBuf, comprLen, decomprBuf, &decomprLen));
    check_sts(st = checkDecompr(decomprBuf, decomprLen, srcBuf, srcLen));

    /* Compress source data at level 9*/
    comprLen = TEST_SIZE + TEST_SIZE / 255 + 16;
    check_sts(st = ippsEncodeLZ4HC_8u((const Ipp8u *)srcBuf, 0, &srcLen, comprBuf, &comprLen,
tables, NULL, 0, 9));
    /* Print compression result */
```

```
    printf("Compression lev 9: %d bytes compressed into %d bytes\n", TEST_SIZE, comprLen);
    /* Decompression */
    decomprLen = TEST_SIZE + 33;
    check_sts(st = ippsDecodeLZ4_8u((const Ipp8u *)comprBuf, comprLen, decomprBuf, &decomprLen));
    st = checkDecompr(decomprBuf, decomprLen, srcBuf, srcLen);

    EXIT_MAIN
    ippsFree(srcBuf);
    ippsFree(comprBuf);
    ippsFree(decomprBuf);
    ippsFree(hashTable);
    ippsFree(prevTable);
    return (int)st;
}
static IppStatus checkDecompr(Ipp8u *decomprAddr, int decomprLen, Ipp8u *srcAddr, int srcLen)
{
    /* Check */
    if (decomprLen != srcLen || memcmp(decomprAddr, srcAddr, srcLen) != 0) {
        printf("Wrong decompression!\n");
        return ippStsErr;
    }
    printf("Decompressed OK into %d bytes\n", decomprLen);
    return ippStsNoErr;
}
```

EncodeLZ4HC

Performs LZ4 encoding in HC mode.

Syntax

```
IppStatus ippsEncodeLZ4HC_8u(const Ipp8u* pSrc, int srcIdx, int* pSrcLen, Ipp8u* pDst,
int* pDstLen, Ipp8u** ppHashTables, const Ipp8u* pDict, int dictLen, int level);
```

```
IppStatus ippsEncodeLZ4HCDictLimit_8u(const Ipp8u* pSrc, int srcIdx, int* pSrcLen,
Ipp8u* pDst, int* pDstLen, Ipp8u** ppHashTables, const Ipp8u* pDict, int dictLen, int
level, int lowDictIdx);
```

Include Files

`ippdc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source data.
`srcIdx`	Index of the starting byte in the source vector.
`pSrcLen`	Pointer to the length of the source data for compression.

pDst	Pointer to the compressed data.
pDstLen	Pointer to the length of the compressed data.
ppHashTables	Pointer to an array of pointers to the LZ4 hash tables for HC mode; ppHashTables[0] = pHashTable, ppHashTables[1] = pPrevTable.
pDict	Pointer to the dictionary.
dictLen	Length to the dictionary.
level	Compression level.
lowDictIdx	Lowest valid index in dictionary.

Description

These functions perform encoding of the source data pSrc using the LZ4 algorithm in HC (High Compression) mode. The destination buffer must have sufficient length for the operation. The length of the compressed data is set to pDstLen.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if at least one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error if the srcLen value is less than, or equal to zero.
ippStsBadArgErr	Indicates an error if the index of the starting byte is less than zero.
ippStsNotSupportedModeErr	Indicates an error if the function does not support the specified combination of parameters' values.

Example

```
/*******************************************************************************
 * Copyright 2017 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

/*
 The example below shows how to use the functions:
```

```
     ippsEncodeLZ4HCHashTableGetSize_8u
     ippsEncodeLZ4HCHashTableInit_8u
     ippsEncodeLZ4HC_8u
     ippsDecodeLZ4_8u
*/

#include <stdio.h>
#include <string.h>

#include <ipp/ippdc.h>
#include <ipp/ipps.h>

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define
check_sts(st)
                                  \
    if ((st) !=
ippStsNoErr)
                           \
        goto exitLine; /* Go to Exit if Intel(R) Integrated Performance Primitives (Intel(R)
IPP) function returned status different from \
                            ippStsNoErr */

#define TEST_SIZE (1024)

static int checkDecompr(Ipp8u *decomprAddr, int decomprLen, Ipp8u *srcAddr, int srcLen);

int main(void)
{
    Ipp8u *srcBuf = NULL, *comprBuf = NULL, *decomprBuf = NULL, *hashTable = NULL, *prevTable =
NULL;
    Ipp8u *tables[2];
    IppStatus st = ippStsNoErr;
    int hashSize = 0, prevSize = 0, srcLen = TEST_SIZE, comprLen = TEST_SIZE + TEST_SIZE / 255 +
16, /* Extra bytes for

        uncompressible data */
        decomprLen = TEST_SIZE +
33;                                                                  /* Spare bytes for "wild"

    (non-safe) decompression */
    int i;

    srcBuf = ippsMalloc_8u(TEST_SIZE);
    decomprBuf = ippsMalloc_8u(decomprLen);
    comprBuf = ippsMalloc_8u(comprLen);
    /* Initialize source buffer */
    check_sts(st = ippsVectorJaehne_8u(srcBuf, TEST_SIZE, IPP_MAX_8U));
    for (i = 0; i < TEST_SIZE; i++)
        srcBuf[i] >>= 6; /* Decrease source data entropy */
    /* Init and allocate hash table */
    check_sts(st = ippsEncodeLZ4HCHashTableGetSize_8u(&hashSize, &prevSize));
    hashTable = ippsMalloc_8u(hashSize);
    prevTable = ippsMalloc_8u(prevSize);
    tables[0] = hashTable;
    tables[1] = prevTable;
```

```
    check_sts(st = ippsEncodeLZ4HCHashTableInit_8u(tables));
    /* Compress source data at level 1*/
    check_sts(st = ippsEncodeLZ4HC_8u((const Ipp8u *)srcBuf, 0, &srcLen, comprBuf, &comprLen,
tables, NULL, 0, 1));
    /* Print compression result */
    printf("Compression lev 1: %d bytes compressed into %d bytes\n", TEST_SIZE, comprLen);
    /* Decompression */
    decomprLen = TEST_SIZE + 33;
    check_sts(st = ippsDecodeLZ4_8u((const Ipp8u *)comprBuf, comprLen, decomprBuf, &decomprLen));
    check_sts(st = checkDecompr(decomprBuf, decomprLen, srcBuf, srcLen));

    /* Compress source data at level 6*/
    comprLen = TEST_SIZE + TEST_SIZE / 255 + 16;
    check_sts(st = ippsEncodeLZ4HC_8u((const Ipp8u *)srcBuf, 0, &srcLen, comprBuf, &comprLen,
tables, NULL, 0, 6));
    /* Print compression result */
    printf("Compression lev 6: %d bytes compressed into %d bytes\n", TEST_SIZE, comprLen);
    /* Decompression */
    decomprLen = TEST_SIZE + 33;
    check_sts(st = ippsDecodeLZ4_8u((const Ipp8u *)comprBuf, comprLen, decomprBuf, &decomprLen));
    check_sts(st = checkDecompr(decomprBuf, decomprLen, srcBuf, srcLen));

    /* Compress source data at level 9*/
    comprLen = TEST_SIZE + TEST_SIZE / 255 + 16;
    check_sts(st = ippsEncodeLZ4HC_8u((const Ipp8u *)srcBuf, 0, &srcLen, comprBuf, &comprLen,
tables, NULL, 0, 9));
    /* Print compression result */
    printf("Compression lev 9: %d bytes compressed into %d bytes\n", TEST_SIZE, comprLen);
    /* Decompression */
    decomprLen = TEST_SIZE + 33;
    check_sts(st = ippsDecodeLZ4_8u((const Ipp8u *)comprBuf, comprLen, decomprBuf, &decomprLen));
    st = checkDecompr(decomprBuf, decomprLen, srcBuf, srcLen);

    EXIT_MAIN
    ippsFree(srcBuf);
    ippsFree(comprBuf);
    ippsFree(decomprBuf);
    ippsFree(hashTable);
    ippsFree(prevTable);
    return (int)st;
}
static IppStatus checkDecompr(Ipp8u *decomprAddr, int decomprLen, Ipp8u *srcAddr, int srcLen)
{
    /* Check */
    if (decomprLen != srcLen || memcmp(decomprAddr, srcAddr, srcLen) != 0) {
        printf("Wrong decompression!\n");
        return ippStsErr;
    }
    printf("Decompressed OK into %d bytes\n", decomprLen);
    return ippStsNoErr;
}
```

BWT-Based Compression Functions

This section describes the Intel IPP functions that support composed algorithms based on the Burrows-Wheeler transform (BWT).

- Burrows-Wheeler Transform
- Move To Front Functions
- bzip2 Coding Functions

Burrows-Wheeler Transform

Burrows-Wheeler Transform (BWT) does not compress data, but it simplifies the structure of input data and makes more effective further compression. One of the distinctive feature of this method is operation on the block of data (as a rule of size 512kB - 2 mB). The main idea of this method is block sorting which groups symbols with a similar context. Let us consider how BWT works on the input data block ‘`abracadabra`’. The first step is to create a matrix containing all its possible cyclic permutations. The first row is input string, the second is created by shifting it to the left by one symbol and so on:

```
abracadabra bracadabraa racadabraab acadabraabr cadabraabra adabraabrac dabraabraca abraabracad
braabracada raabracadab aabracadabr
```

Then all rows are sorted in accordance with the lexicographic order:

```
0 aabracadabr  1 abraabracad  2 abracadabra  3 acadabraabr  4 adabraabrac  5 braabracada  6
bracadabraa  7 cadabraabra  8 dabraabraca  9 raabracadab 10 racadabraab
```

The last step is to write out the last column and the index of the input string: `rdarcaaaabb`, 2 - this is a result of the forward BWT transform.

Inverse BTW is performed as follows:

elements of the input string are numbered in ascending order

```
0  r  1  d  2  a  3  r  4  c  5  a  6  a  7  a  8  a  9  b 10  b
```

and sorted in accordance with the lexicographic order:

```
2  a  5  a  6  a  7  a  8  a  9  b 10  b  4  c  1  d  0  r  3  r
```

This index array is a vector of the inverse transform (`Inv`), the further reconstruction of the string is performed in the following manner:

```
src[] = "rdarcaaaabb";

Inv[] = {2,5,6,7,8,9,10,4,1,0,3};

index = 2; // index of the initial string is known from the forward BWT

for( i = 0; i < len; i++ ) {

index = Inv[index];

dst[i] = src[index];

}
```

- BWTFwdGetSize
- BWTFwd
- BWTFwdGetBufSize_SelectSort
- BWTFwd_SelectSort
- BWTInvGetSize
- BWTInv

BWTFwdGetSize

Computes the size of the external buffer for the forward BWT transform.

Syntax

```
IppStatus ippsBWTFwdGetSize_8u(int wndSize, int* pBWTFwdBuffSize);
```

Include Files

ippdc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

wndSize	Window size for BWT transform.
pBWTFwdBuffSize	Pointer to the computed size of the additional buffer.

Description

This function computes the size of memory (in bytes) of the external buffer that is required by the function ippsBWTFwd for the forward BWT transform.

Code example shows how to use the function ippsBWTFwdGetSize_8u.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if pBWTFwdBuffSize pointer is NULL.

BWTFwd

Performs the forward BWT transform.

Syntax

```
IppStatus ippsBWTFwd_8u(const Ipp8u* pSrc, Ipp8u* pDst, int len, int* pIndex, Ipp8u* pBWTFwdBuff);
```

Include Files

ippdc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc	Pointer to the source vector.
pDst	Pointer to the destination vector.

len	Number of elements in the source and destination vectors.
pIndex	Pointer to the index of first position for the forward BWT transform.
pBWTFwdBuff	Pointer to the additional buffer.

Description

This function performs the forward BWT transform of len elements starting from pIndex element of the source vector pSrc and stores result in the vector pDst. The function uses the external buffer pBWTFwdBuff. The size of this buffer must be computed by calling the function ippsBWTFwdGetSize beforehand.

Code example shows how to use the function ippsBWTFwd_8u.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error if len is less than or equal to 0.

BWTFwdGetBufSize_SelectSort

Computes the size of the external buffer for the forward BWT transform.

Syntax

```
IppStatus ippsBWTFwdGetBufSize_SelectSort_8u(Ipp32u wndSize, Ipp32u* pBWTFwdBufSize,
IppBWTSortAlgorithmHint sortAlgorithmHint);
```

Include Files

ippdc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

wndSize	Window size for BWT transform.
pBWTFwdBufSize	Pointer to the computed size of the additional buffer.
sortAlgorithmHint	Specifies the sort algorithm used. Possible values are:

Description

This function computes the size of memory (in bytes) of the external buffer that is required by the function BWTFwd_SelectSort for the forward BWT transform.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if pBuffSize pointer is NULL.

BWTFwd_SelectSort

Performs the forward BWT transform with specified sort algorithm.

Syntax

```
IppStatus ippsBWTFwd_SelectSort_8u(const Ipp8u* pSrc, Ipp8u* pDst, Ipp32u len, Ipp32u* index, Ipp8u* pBWTFwdBuf, IppBWTSortAlgorithmHint sortAlgorithmHint);
```

Include Files

ippdc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc	Pointer to the source vector.
pDst	Pointer to the destination vector.
len	Number of elements in the source and destination vectors.
index	Index of the first position for the forward BWT transform.
pBWTFwdBuf	Pointer to the additional buffer.
sortAlgorithmHint	Specifies the sort algorithm used. Possible values are:

Description

This function performs the forward BWT transform of len elements starting from pIndex element of the source vector pSrc and stores result in the vector pDst. The parameter sortAlgorithmHint specifies the desired algorithm of sorting. The function uses the external buffer pBuff. The size of this buffer must be computed by calling the function BWTFwdGetBufSize_SelectSort beforehand.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error if len is less than or equal to 0.

BWTInvGetSize

Computes the size of the external buffer for the inverse BWT transform.

Syntax

```
IppStatus ippsBWTInvGetSize_8u(int wndSize, int* pBWTInvBuffSize);
```

Include Files

ippdc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

wndSize	Window size for BWT transform.
pBWTInvBuffSize	Pointer to the computed size of the additional buffer.

Description

This function computes the size of memory (in bytes) of the external buffer that is required by the function ippsBWTInv for the inverse BWT transform.

Code example shows how to use the function ippsBWTInvGetSize_8u.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if pBWTInvBuffSize pointer is NULL.

BWTInv

Performs the inverse BWT transform.

Syntax

```
IppStatus ippsBWTInv_8u(const Ipp8u* pSrc, Ipp8u* pDst, int len, int index, Ipp8u*
pBWTInvBuff);
```

Include Files

ippdc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc	Pointer to the source vector.
pDst	Pointer to the destination vector.
len	Number of elements in the source and destination vectors.
index	Index of first position for the inverse BWT transform.
pBWTInvBuff	Pointer to the additional buffer.

Description

This function performs the inverse BWT transform of len elements starting from pIndex element of the source vector pSrc and stores result in the vector pDst. The function uses the external buffer pBWTInvBuff. The size of this buffer must be computed by calling the function ippsBWTInvGetSize beforehand.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error if len is less than or equal to 0.

Example

The code example below shows how to use the function ippsBWTInv_8u.

```
void func_BWT() {
    int wndSize = 8;
    int pBWTFwdBuffSize;
    int pBWTInvBuffSize;


    Ipp8u pSrc[] = "baadeffg";
    int len = 8;
    int pIndex;


    Ipp8u* pDst = ippsMalloc_8u(len);
    Ipp8u* pDstInv = ippsMalloc_8u(len);


    ippsBWTFwdGetSize_8u(wndSize, &pBWTFwdBuffSize);
    Ipp8u* pBWTFwdBuff = ippsMalloc_8u(pBWTFwdBuffSize);
```

```
    ippsBWTFwd_8u(pSrc, pDst, len, &pIndex, pBWTFwdBuff);


    ippsBWTInvGetSize_8u( wndSize, &pBWTInvBuffSize);
    Ipp8u* pBWTInvBuff = ippsMalloc_8u(pBWTInvBuffSize);
    ippsBWTInv_8u(pDst, pDstInv, len, pIndex, pBWTInvBuff);
}
```

Result:

```
pDst ->     "bagadeff"
pDstInv ->  "baadeffg"
```

Move To Front Functions

This section describes the functions that performs Move To Front (MTF) data transform method. The basic idea is to represent the symbols of the source sequence as the current indexes of that symbols in the modified alphabet. This alphabet is a list where frequently used symbols are placed in the upper lines. When the given symbols occurs it is replaced by its index in the list, then this symbol is moved in the first position in the list, and all indexes are updated. For example, the sequence " `baabbfffacczzdd` " contains symbols that form the ordered 'alphabet'`{'a', 'b', 'c', 'd', 'f', 'z'}`. The function will operate in the following manner:

source	destination	alphabet
		0, 1, 2, 3, 4, 5, 6
		'a', 'b', 'c', 'd', 'f', 'z'
b	1	'b', 'a', 'c', 'd', 'f', 'z'
a	1	'a', 'b', 'c', 'd', 'f', 'z'
a	0	'a', 'b', 'c', 'd', 'f', 'z'
b	1	'b', 'a', 'c', 'd', 'f', 'z'
f	4	'f', 'b', 'a', 'c', 'd', 'z'
f	0	'f', 'b', 'a', 'c', 'd', 'z'
f	0	'f', 'b', 'a', 'c', 'd', 'z'
a	2	'a', 'f', 'b', 'c', 'd', 'z'
c	3	'c', 'a', 'f', 'b','d', 'z'
c	0	'c', 'a', 'f', 'b', 'd', 'z'
z	5	'z', 'c', 'a', 'f', 'b', 'd'
z	0	'z', 'c', 'a', 'f', 'b', 'd'
d	5	'd', 'z', 'c', 'a', 'f', 'b'
d	0	'd', 'z', 'c', 'a', 'f', 'b'

Finally, the function returns the destination sequence: 11014002305050.

These transformed data can be used for the following effective compression. This method is often used after Burrows-Wheeler transform.

- MTFInit
- MTFGetSize

- MTFFwd
- MTFInv

MTFInit

Initializes the MTF structure.

Syntax

```
IppStatus ippsMTFInit_8u(IppMTFState_8u* pMTFState);
```

Include Files

`ippdc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pMTFState`	Pointer to the MTF structure.

Description

This function initializes the MTF structure that contains parameters for the MTF transform in the external buffer. This structure is used by the functions ippsMTFFwd and ippsMTFInv. The size of this buffer must be computed previously by calling the function ippsMTFGetSize.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if `pMTFState` pointer is `NULL`.

MTFGetSize

Computes the size of the MTF structure.

Syntax

```
IppStatus ippsMTFGetSize_8u(int* pMTFStateSize);
```

Include Files

`ippdc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

pMTFStateSize	Pointer to the computed MTF structure size.

Description

This function computes the size of memory (in bytes) that is required for the MTF structure. This function must be called prior to the function ippsMTFInit.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if pMTFStateSize pointer is NULL.

MTFFwd

Performs the forward MTF transform.

Syntax

```
IppStatus ippsMTFFwd_8u(const Ipp8u* pSrc, Ipp8u* pDst, int len, IppMTFState_8u* pMTFState);
```

Include Files

ippdc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc	Pointer to the source buffer.
pDst	Pointer to the destination buffer.
len	Number of elements in the source and destination buffers.
pMTFState	Pointer to the MTF structure.

Description

This function performs the forward MTF transform of len elements of the data in the source buffer pSrc and stores result in the buffer pDst. The parameters of the MTF transform are specified in the MTF structure pMTFState that must be initialized by ippsMTFInit beforehand.

Return Values

ippStsNoErr	Indicates no error.

ippStsNullPtrErr	Indicates an error if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error if len is less than or equal to 0.

MTFInv

Performs the inverse MTF transform.

Syntax

```
IppStatus ippsMTFInv_8u(const Ipp8u* pSrc, Ipp8u* pDst, int len, IppMTFState_8u*
pMTFState);
```

Include Files

ippdc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc	Pointer to the source buffer.
pDst	Pointer to the destination buffer.
len	Number of elements in the source and destination buffers.
pMTFState	Pointer to the MTF structure.

Description

This function performs the inverse MTF transform of len elements of data in the source buffer pSrc and stores result in the buffer pDst. The parameters of the MTF transform are specified in the MTF structure pMTFState that must be initialized by ippsMTFInit beforehand.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error if len is less than or equal to 0.

bzip2 Coding Functions

This section describes different Intel IPP functions to perform bzip2 encoding and decoding.

- EncodeRLEInit_BZ2

- RLEGetSize_BZ2
- EncodeRLE_BZ2
- EncodeRLEFlush_BZ2
- RLEGetInUseTable
- DecodeRLEStateInit_BZ2
- DecodeRLEState_BZ2
- DecodeRLEStateFlush_BZ2
- EncodeZ1Z2_BZ2
- DecodeZ1Z2_BZ2
- ReduceDictionary
- ExpandDictionary
- CRC32_BZ2
- EncodeHuffGetSize_BZ2
- EncodeHuffInit_BZ2
- PackHuffContext_BZ2
- EncodeHuff_BZ2
- DecodeHuffGetSize_BZ2
- DecodeHuffInit_BZ2
- UnpackHuffContext_BZ2
- DecodeHuff_BZ2
- DecodeBlockGetSize_BZ2
- DecodeBlock_BZ2

EncodeRLEInit_BZ2

Initializes the bzip2-specific RLE structure.

Syntax

```
IppStatus ippsEncodeRLEInit_BZ2_8u(IppRLEState_BZ2* pRLEState);
```

Include Files

`ippdc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pRLEState`	Pointer to the bzip2-specific RLE structure.

Description

This function initializes the bzip2-specific RLE structure that contains parameters for the RLE in the external buffer. This structure is used by the function ippsEncodeRLE_BZ2. The size of this buffer must be computed previously by calling the function ippsRLEGetSize_BZ2.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if `pRLEState` pointer is `NULL`.

RLEGetSize_BZ2

Compute the size of the state structure for the bzip2-specific RLE.

Syntax

```
IppStatus ippsRLEGetSize_BZ2_8u(int* pRLEStateSize);
```

Include Files

`ippdc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pRLEStateSize`	Pointer to the size of the state structure for bzip2-specific RLE.

Description

This function computes the size of memory (in bytes) of the internal state structure for the bzip2-specific RLE.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if `pRLEStateSize` pointer is `NULL`.

EncodeRLE_BZ2

Performs the bzip2-specific RLE.

Syntax

```
IppStatus ippsEncodeRLE_BZ2_8u(Ipp8u** ppSrc, int* pSrcLen, Ipp8u* pDst, int* pDstLen,
IppRLEState_BZ2* pRLEState);
```

Include Files

`ippdc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

ppSrc	Double pointer to the source buffer.
pSrcLen	Pointer to the length of the source buffer.
pDst	Pointer to the destination buffer.
pDstLen	Pointer to the length of the destination buffer.
pRLEState	Pointer to the bzip2-specific RLE state structure.

Description

This function performs RLE encoding with thresholding equal to 4. It processes the input data ppSrc and writes the results to the pDst buffer. The function uses the bzip2-specific RLE state structure pRLEState. This structure must be initialized by ippsEncodeRLEInit_BZ2 beforehand.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if one of the pointers is NULL.
ippStsSizeErr	Indicates an error if length of the source or destination buffer is less than or equal to 0.
ippStsDstSizeLessExpected	Indicates a warning if size of the destination buffer is insufficient to store all output elements.

EncodeRLEFlush_BZ2

Flushes the remaining data after RLE.

Syntax

```
IppStatus ippsEncodeRLEFlush_BZ2_8u(Ipp8u* pDst, int* pDstLen, IppRLEState_BZ2*
pRLEState);
```

Include Files

ippdc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pDst	Pointer to the destination buffer.
pDstLen	Pointer to the length of the destination buffer.
pRLEState	Pointer to the bzip2-specific RLE state structure.

Description

This function flushes the remaining data after RLE encoding with thresholding equal to 4. The function uses the initialized bzip2-specific RLE state structure `pRLEState`.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if one of the pointers is `NULL`.
`ippStsSizeErr`	Indicates an error if length of the destination buffer is less than or equal to 0.

RLEGetInUseTable

Gets the pointer to the `inUse` vector from the RLE state structure.

Syntax

```
IppStatus ippsRLEGetInUseTable_8u(Ipp8u inUse[256], IppRLEState_BZ2* pRLEState);
```

Include Files

`ippdc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`inUse`	Pointer to the `inUse` vector.
`pRLEState`	Pointer to the bzip2-specific RLE state structure.

Description

This function gets the pointer to the `inUse` vector (table) from the initialized bzip2-specific RLE state structure `pRLEState`.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if one of the pointers is `NULL`.

DecodeRLEStateInit_BZ2

Initializes the bzip2-specific RLE structure.

Syntax

```
IppStatus ippsDecodeRLEStateInit_BZ2_8u(IppRLEState_BZ2* pRLEState);
```

Include Files

ippdc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pRLEState	Pointer to the bzip2-specific RLE structure.

Description

This function initializes the bzip2-specific RLE structure that contains parameters for the RLE in the external buffer. This structure is used by the function DecodeRLEState_BZ2.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if pRLEState pointer is NULL.

DecodeRLEState_BZ2

Performs the bzip2-specific RLE decoding.

Syntax

```
IppStatus ippsDecodeRLEState_BZ2_8u(Ipp8u** ppSrc, Ipp32u* pSrcLen, Ipp8u** ppDst,
Ipp32u* pDstLen, IppRLEState_BZ2* pRLEState);
```

Include Files

ippdc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

ppSrc	Double pointer to the source buffer.
pSrcLen	Pointer to the length of the source buffer.
ppDst	Double pointer to the destination buffer.
pDstLen	Pointer to the length of the destination buffer.
pRLEState	Pointer to the bzip2-specific RLE state structure.

Description

This function performs RLE decoding with thresholding equal to 4. It processes the input data `ppSrc` and writes the results to the `ppDst` buffer. The function uses the bzip2-specific RLE state structure `pRLEState`. This structure must be initialized by the functions DecodeRLEStateInit_BZ2 beforehand.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if one of the pointers is `NULL`.
`ippStsSizeErr`	Indicates an error if length of the source or destination buffer is less than or equal to 0.
`ippStsDstSizeLessExpected`	Indicates a warning if size of the destination buffer is insufficient to store all output elements.

DecodeRLEStateFlush_BZ2

Flushes the remaining data after RLE decoding.

Syntax

```
IppStatus ippsDecodeRLEStateFlush_BZ2_8u(IppRLEState_BZ2* pRLEState, Ipp8u** ppDst,
Ipp32u* pDstLen);
```

Include Files

`ippdc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pRLEState`	Pointer to the bzip2-specific RLE state structure.
`ppDst`	Double pointer to the destination buffer.
`pDstLen`	Pointer to the length of the destination buffer.

Description

This function flushes the remaining data after RLE decoding with thresholding equal to 4. The function uses the initialized bzip2-specific RLE state structure `pRLEState`.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if one of the pointers is `NULL`.

ippStsSizeErr	Indicates an error if length of the destination buffer is less than or equal to 0.

EncodeZ1Z2_BZ2

Performs the bzip2-specific Z1Z2 encoding.

Syntax

```
IppStatus ippsEncodeZ1Z2_BZ2_8u16u(Ipp8u** ppSrc, int* pSrcLen, Ipp16u* pDst, int* pDstLen, int freqTable[258]);
```

Include Files

ippdc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

ppSrc	Double pointer to the source buffer.
pSrcLen	Pointer to the length of the source buffer, after decoding - pointer to the size of the remaining data.
pDst	Pointer to the destination buffer.
pDstLen	Pointer to the length of the destination buffer, after decoding - pointer to the resulting size of the destination buffer.
freqTable	Table of frequencies collected for the alphabet symbols.

Description

This function performs the bzip2-specific Z1Z2 encoding.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if one of the pointers is NULL.
ippStsSizeErr	Indicates an error if length of the source or destination buffer is less than or equal to 0.
ippStsDstSizeLessExpected	Indicates a warning if size of the destination buffer is insufficient to store all output elements.

DecodeZ1Z2_BZ2

Performs the bzip2-specific Z1Z2 decoding.

Syntax

```
IppStatus ippsDecodeZ1Z2_BZ2_16u8u(Ipp16u** ppSrc, int* pSrcLen, Ipp8u* pDst, int* pDstLen);
```

Include Files

`ippdc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`ppSrc`	Double pointer to the source buffer.
`pSrcLen`	Pointer to the length of the source buffer, after decoding - pointer to the size of the remaining data.
`pDst`	Pointer to the destination buffer.
`pDstLen`	Pointer to the length of the destination buffer, after decoding - pointer to the resulting size of the destination buffer.

Description

This function performs the bzip2-specific Z1Z2 decoding.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if one of the pointers is `NULL`.
`ippStsSizeErr`	Indicates an error if length of the source or destination buffer is less than or equal to 0.
`ippStsDstSizeLessExpected`	Indicates a warning if size of the destination buffer is insufficient to store all output elements.

ReduceDictionary

Performs the dictionary reducing.

Syntax

```
IppStatus ippsReduceDictionary_8u_I(const Ipp8u inUse[256], Ipp8u* pSrcDst, int srcDstLen, int* pSizeDictionary);
```

Include Files

ippdc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

inUse	Table of 256 values of the Ipp8u type.
pSrcDst	Pointer to the source and destination buffer.
srcDstLen	Length of the source and destination buffer.
pSizeDictionary	Pointer to the size of the dictionary on entry, and to the size of reduced dictionary after operation.

Description

This function performs the dictionary reducing.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if one of the pointers is NULL.
ippStsSizeErr	Indicates an error if length of the source and destination buffer is less than or equal to 0.

ExpandDictionary

Performs the dictionary expanding.

Syntax

```
IppStatus ippsExpandDictionary_8u_I(const Ipp8u inUse[256], Ipp8u* pSrcDst, int
srcDstLen, int sizeDictionary);
```

Include Files

ippdc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

inUse	Table of 256 values of the Ipp8u type.

pSrcDst	Pointer to the source and destination buffer.
srcDstLen	Length of the source and destination buffer.
sizeDictionary	Size of the dictionary on entry, and to the size of expanded dictionary after operation.

Description

This function performs the dictionary expanding.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if one of the pointers is NULL.
ippStsSizeErr	Indicates an error if length of the source and destination buffer is less than or equal to 0.

CRC32_BZ2

Computes the CRC32 checksum for the source data buffer.

Syntax

```
IppStatus ippsCRC32_BZ2_8u(const Ipp8u* pSrc, int srcLen, Ipp32u* pCRC32);
```

Include Files

ippdc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc	Pointer to the source data buffer.
srcLen	Number of elements in the source data buffer.
pCRC32	Pointer to the accumulated checksum value.

Description

This function computes the checksum for srcLen elements of the source data buffer pSrc and stores it in the pCRC32. The checksum is computed using the CRC32 direct algorithm that is specific for the bzip2 coding.

You can use this function to compute the accumulated value of the checksum for multiple buffers by specifying as an input parameter the checksum value obtained in the preceding function call.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if the pSrc pointer is NULL.
ippStsSizeErr	Indicates an error if the length of the source vector is less than or equal to 0.

EncodeHuffGetSize_BZ2

Computes the size of the internal state for bzip2-specific Huffman encoding.

Syntax

```
IppStatus ippsEncodeHuffGetSize_BZ2_16u8u(int wndSize, int* pEncodeHuffStateSize);
```

Include Files

ippdc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

wndSize	Size of the block to be processed.
pEncodeHuffStateSize	Pointer to the size of the internal state for bzip2-specific Huffman coding.

Description

This function computes the size of the internal state structure for bzip2-specific Huffman encoding in dependence of the size of the block to be encoded.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if the pointer pEncodeHuffStateSize is NULL.
ippStsSizeErr	Indicates an error if wndSize is less than or equal to 0.

EncodeHuffInit_BZ2

Initializes the elements of the bzip2-specific internal state for Huffman encoding.

Syntax

```
IppStatus ippsEncodeHuffInit_BZ2_16u8u(int sizeDictionary, const int freqTable[258],
const Ipp16u* pSrc, int srcLen, IppEncodeHuffState_BZ2* pEncodeHuffState);
```

Include Files

ippdc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

sizeDictionary	Size of the dictionary.
freqTable	Table of frequencies of symbols.
pSrc	Pointer to the source vector.
srcLen	Length of the source vector.
pEncodeHuffState	Pointer to internal state structure for bzip2 specific Huffman coding.

Description

This function initializes the elements of the bzip2-specific internal state for Huffman encoding. This structure is used by the function ippsEncodeHuff_BZ2. The size of this buffer must be computed previously by calling the function ippsEncodeHuffGetSize_BZ2.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if one of the pointers is NULL.
ippStsSizeErr	Indicates an error if length of the source buffer is less than or equal to 0.

PackHuffContext_BZ2

Performs the bzip2-specific encoding of Huffman context.

Syntax

```
IppStatus ippsPackHuffContext_BZ2_16u8u(Ipp32u* pCode, int* pCodeLenBits, Ipp8u* pDst,
int* pDstLen, IppEncodeHuffState_BZ2* pEncodeHuffState);
```

Include Files

ippdc.h

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pCode`	Pointer to the bit buffer.
`pCodeLenBits`	Number of valid bits in the bit buffer.
`pDst`	Pointer to the destination vector.
`pDstLen`	Pointer to the size of destination buffer on input, pointer to the resulting length of the destination vector on output.
`pEncodeHuffState`	Pointer to internal state structure for bzip2 specific Huffman encoding.

Description

This function performs the bzip2-specific encoding of the `Huffman context`. The function uses the bzip2-specific Huffman encoding state structure `pEncodeHuffState`. This structure must be initialized by ippsEncodeHuffInit_BZ2 beforehand.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if one of the pointers is `NULL`.
`ippStsSizeErr`	Indicates an error if length of the destination buffer is less than or equal to 0.
`ippStsDstSizeLessExpected`	Indicates a warning if size of the destination buffer is insufficient to store all output elements.

EncodeHuff_BZ2

Performs the bzip2-specific Huffman encoding.

Syntax

```
IppStatus ippsEncodeHuff_BZ2_16u8u(Ipp32u* pCode, int* pCodeLenBits, Ipp16u** ppSrc,
int* pSrcLen, Ipp8u* pDst, int* pDstLen, IppEncodeHuffState_BZ2* pEncodeHuffState);
```

Include Files

`ippdc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

pCode	Pointer to the bit buffer.
pCodeLenBits	Number of valid bits in the bit buffer.
ppSrc	Double pointer to the source vector.
pSrcLen	Pointer to the length of source vector.
pDst	Pointer to the destination vector.
pDstLen	Pointer to the size of destination buffer on input, pointer to the resulting length of the destination vector on output.
pEncodeHuffState	Pointer to internal state structure for bzip2 specific Huffman encoding.

Description

This function performs the bzip2-specific Huffman encoding. The function uses the bzip2-specific Huffman encoding state structure pEncodeHuffState. This structure must be initialized by ippsEncodeHuffInit_BZ2 beforehand.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if one of the pointers is NULL.
ippStsSizeErr	Indicates an error if length of the source or destination buffer is less than or equal to 0.
ippStsDstSizeLessExpected	Indicates a warning if size of the destination buffer is insufficient to store all output elements.

DecodeHuffGetSize_BZ2

Computes the size of the internal state for bzip2-specific Huffman decoding.

Syntax

```
IppStatus ippsDecodeHuffGetSize_BZ2_8u16u(int wndSize, int* pDecodeHuffStateSize);
```

Include Files

ippdc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

wndSize	Size of the block to be processed.
pDecodeHuffStateSize	Pointer to the size of the internal state for bzip2-specific Huffman coding.

Description

This function computes the size of the internal state structure for bzip2-specific Huffman decoding.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if the pointer pDecodeHuffStateSize is NULL.
ippStsSizeErr	Indicates an error if wndSize is less than or equal to 0.

DecodeHuffInit_BZ2

Initializes the elements of the bzip2-specific internal state for Huffman decoding.

Syntax

```
IppStatus ippsDecodeHuffInit_BZ2_8u16u(int sizeDictionary, IppDecodeHuffState_BZ2*
pDecodeHuffState);
```

Include Files

ippdc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

sizeDictionary	Size of the dictionary.
pDecodeHuffState	Pointer to internal state structure for bzip2 specific Huffman coding.

Description

This function initializes the elements of the bzip2-specific internal state for Huffman decoding. This structure is used by the function ippsDecodeHuff_BZ2. The size of this buffer must be computed previously by calling the function ippsDecodeHuffGetSize_BZ2.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if the pDecodeHuffState pointer is NULL.
ippStsSizeErr	Indicates an error if sizeDictionary is less than or equal to 0.

UnpackHuffContext_BZ2

Performs the bzip2-specific decoding of Huffman context.

Syntax

```
IppStatus ippsUnpackHuffContext_BZ2_8u16u(Ipp32u* pCode, int* pCodeLenBits, Ipp8u**
ppSrc, int* pSrcLen, IppDecodeHuffState_BZ2* pDecodeHuffState);
```

Include Files

ippdc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pCode	Pointer to the bit buffer.
pCodeLenBits	Number of valid bits in the bit buffer.
ppSrc	Double pointer to the source vector.
pSrcLen	Pointer to the size of source buffer on input, pointer to the resulting length of the source vector on output.
pDecodeHuffState	Pointer to internal state structure for bzip2 specific Huffman decoding.

Description

This function performs the bzip2-specific decoding of the *Huffman context*. The function uses the bzip2-specific Huffman decoding state structure pDecodeHuffState. This structure must be initialized by ippsDecodeHuffInit_BZ2 beforehand.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if one of the pointers is NULL.

ippStsSizeErr	Indicates an error if length of the destination buffer is less than or equal to 0.
ippStsSrcSizeLessExpected	Indicates a warning if size of the source buffer is insufficient to store all output elements.

DecodeHuff_BZ2

Performs the bzip2-specific Huffman decoding.

Syntax

```
IppStatus ippsDecodeHuff_BZ2_8u16u(Ipp32u* pCode, int* pCodeLenBits, Ipp8u** ppSrc,
int* pSrcLen, Ipp16u* pDst, int* pDstLen, IppDecodeHuffState_BZ2* pDecodeHuffState);
```

Include Files

`ippdc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

pCode	Pointer to the bit buffer.
pCodeLenBits	Number of valid bits in the bit buffer.
ppSrc	Double pointer to the source vector.
pSrcLen	Pointer to the size of source buffer.
pDst	Pointer to the destination vector.
pDstLen	Pointer to the size of destination buffer on input, pointer to the resulting length of the destination vector on output.
pDecodeHuffState	Pointer to internal state structure for bzip2 specific Huffman decoding.

Description

This function performs the bzip2-specific Huffman decoding. The function uses the bzip2-specific Huffman decoding state structure `pDecodeHuffState`. This structure must be initialized by ippsDecodeHuffInit_BZ2 beforehand.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if one of the pointers is `NULL`.

ippStsSizeErr	Indicates an error if length of the destination buffer is less than or equal to 0.
ippStsSrcSizeLessExpected	Indicates a warning if size of the source buffer is insufficient to store all output elements.

DecodeBlockGetSize_BZ2

Computes the size of the additional buffer for bzip2-specific block decoding.

Syntax

```
IppStatus ippsDecodeBlockGetSize_BZ2_8u(int blockSize, int* pBuffSize);
```

Include Files

ippdc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

blockSize	Size of the block to be processed.
pBuffSize	Pointer to the size of the buffer for bzip2-specific decoding.

Description

This function computes the size of the additional buffer for bzip2-specific decoding.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if the pointer pBuffSize is NULL.

DecodeBlock_BZ2

Performs the bzip2-specific block decoding.

Syntax

```
IppStatus ippsDecodeBlock_BZ2_16u8u(const Ipp16u* pSrc, int srcLen, Ipp8u* pDst, int*
pDstLen, int index, int dictSize, const Ipp8u inUse[256], Ipp8u* pBuff);
```

Include Files

ippdc.h

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source vector.
`srcLen`	Pointer to the length of the source vector.
`pDst`	Pointer to the destination vector.
`pDstLen`	Pointer to the size of destination buffer on input, pointer to the resulting length of the destination vector on output.
`index`	Index of first position for the inverse BWT transform
`dictSize`	Size of the reduced dictionary.
`inUse`	Table of 256 values of `Ipp8u` type.
`pBuff`	Pointer to the additional buffer.

Description

This function performs the bzip2-specific block decoding. The function uses the bzip2-specific additional buffer `pBuff`. The size of this buffer must be computed by the function DecodeBlockGetSize_BZ2 beforehand.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if one of the pointers is `NULL`.
`ippStsSizeErr`	Indicates an error if length of the source or destination buffer is less than or equal to 0; or if `index` is greater than or equal to `srcLen`.
`ippStsSrcSizeLessExpected`	Indicates a warning if size of the source buffer is insufficient to store all output elements.

ZFP Compression Functions

This section describes the Intel® Integrated Performance Primitives data compression functions that implement the ZFP compressed data format. You can use the ZFP algorithm to perform lossy compression of 3D floating point data. The ZFP format and algorithm are described in [ZFP].

- EncodeZfpGetStateSize
- EncodeZfpInit, EncodeZfpInitLong
- EncodeZfpSet
- EncodeZfpSetAccuracy
- EncodeZfp444
- EncodeZfpGetCompressedBitSize

- EncodeZfpFlush
- EncodeZfpGetCompressedSize, EncodeZfpGetCompressedSizeLong
- DecodeZfpGetStateSize
- DecodeZfpInit, DecodeZfpInitLong
- DecodeZfpSet
- DecodeZfpSetAccuracy
- Decode Zfp444
- DecodeZfpGetCompressedSize, DecodeZfpGetCompressedSizeLong

EncodeZfpGetStateSize

Calculates the size of the buffer for the ZFP compression structure.

Syntax

```
IppStatus ippsEncodeZfpGetStateSize_32f(int* pStateSize);
```

Include Files

`ippdc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pStateSize`	Pointer to the variable receiving the size of the ZFP compression structure.

Description

This function calculates the size of the memory buffer that your application must allocate to hold the ZFP compression structure.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if the pointer `pStateSize` is `NULL`.

EncodeZfpInit, EncodeZfpInitLong

Initializes the ZFP compression structure with default values.

Syntax

```
IppStatus ippsEncodeZfpInit_32f(Ipp8u* pDst, int dstLen, IppEncodeZfpState_32f*
pState);
```

```
IppStatus ippsEncodeZfpInitLong_32f(Ipp8u* pDst, Ipp64u dstLen, IppEncodeZfpState_32f*
pState);
```

Include Files

ippdc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pDst	Pointer to the destination buffer for compressed data.
dstLen	Length of the destination buffer.
pState	Pointer to the ZFP compression structure.

Description

This function initializes the ZFP compression structure. Its size must be calculated by calling the function EncodeZfpGetStateSize beforehand. Use the EncodeZfpInitLong function to process large 3Dfloating point arrays.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if any of the pointers is NULL.
ippStsSizeErr	Indicates an error if the parameter dstLen is less than or equal to zero (applies only to the EncodeZfpInit function.

EncodeZfpSet

Populates fields of the ZFP compression structure with input values.

Syntax

```
IppStatus ippsEncodeZfpSet_32f(int minBits, int maxBits, int maxPrec, int minExp,
IppEncodeZfpState_32f* pState);
```

Include Files

ippdc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

minBits	Minimum number of bits for a compressed block; the default value is IppZFPMINBITS.
maxBits	Maximum number of bits for a compressed block; the default value is IppZFPMAXBITS.
maxPrec	Maximum level of precision; the default value is IppZFPMAXPREC.
minExp	Minimum level of exponent; the default value is IppZFPMINEXP.
pState	Pointer to the ZFP compression structure.

Description

This function populates fields of the ZFP compression structure with the corresponding input values. Refer to [ZFP] and associated documentation for more information about the compression parameters.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if the pState pointer is NULL.
ippStsContexMatchErr	Indicates an error if the ZFP compression structure data is invalid.

EncodeZfpSetAccuracy

Sets the desired precision value in the ZFP compression structure.

Syntax

```
IppStatus ippsEncodeZfpSetAccuracy_32f(Ipp64f precision, IppEncodeZfpState_32f*
pState);
```

Include Files

ippdc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

precision	Value to assign to the minExp field of the ZFP compression structure.
pState	Pointer to the ZFP compression structure.

Description

This function sets the value of the `minExp` field in the ZFP compression structure to the value of the `precision` input parameter.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if the `pState` pointer is `NULL`.
`ippStsContexMatchErr`	Indicates an error if the ZFP compression structure data is invalid.

EncodeZfp444

Encodes a 4x4x4 block of 3D floating point data.

Syntax

```
IppStatus ippsEncodeZfp444_32f(const Ipp32f* pSrc, int srcStep, int srcPlaneStep,
IppEncodeZfpState_32f* pState);
```

Include Files

`ippdc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the value at the coordinates (0, 0, 0) of the 3D source data block.
`srcStep`	Row step in bytes.
`srcPlaneStep`	Plane step in bytes.
`pState`	Pointer to the ZFP compression structure.

Description

This function encodes a 3D 4x4x4 block of floating point values. The function adds the encoded data to an internal bit stream. The pointer to the bit stream is initialized by calling the EncodeZfpInit function and is updated automatically.

The 3D source data must be arranged in memory in such a way that for any set of coordinates (x, y, z) within the 4x4x4 block,

```
pXYZ = (float *)((char *)pSrc + x + y * srcStep + z * srcPlaneStep)
```

is a pointer to the value at (x, y, z).

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if any of the pointers is NULL.
ippStsContexMatchErr	Indicates an error if ZFP compression structure data is invalid.

EncodeZfpGetCompressedBitSize

Returns the current compressed data size.

Syntax

```
IppStatus ippsEncodeZfpGetCompressedBitSize_32f(IppEncodeZfpState_32f* pState, Ipp64u* pCompressedBitSize);
```

Include Files

ippdc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pState	Pointer to the ZFP compression structure.
pCompressedBitSize	Pointer to the variable receiving the compressed data size, in bits.

Description

This function returns the current compressed data size in bits. You can call this function during compression or after the compression of your array is finished. Call this function before using EncodeZfpFlush to get actual bits size. Otherwise,the function returns size which is rounded (aligned) to an upperbyte.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if any of the pointers is NULL.
ippStsContexMatchErr	Indicates an error if the ZFP compression structure data is invalid.

EncodeZfpFlush

Writes any buffered encoded data from the ZFP compression structure to the destination buffer.

Syntax

```
IppStatus ippsEncodeZfpFlush_32f(IppEncodeZfpState_32f* pState);
```

Include Files

`ippdc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pState`	Pointer to the ZFP compression structure.

Description

This function writes any buffered encoded data from the ZFP compression structure to the destination buffer. Your application must call this function after the last call to EncodeZfp444 to properly finish all compression operations.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if the pointer to `pState` is `NULL`.
`ippStsContexMatchErr`	Indicates an error if the ZFP compression structure data is invalid.

EncodeZfpGetCompressedSize, EncodeZfpGetCompressedSizeLong

Returns the current compressed data size.

Syntax

```
IppStatus ippsEncodeZfpGetCompressedSize_32f(IppEncodeZfpState_32f* pState, int*
pCompressedSize);
```

```
IppStatus ippsEncodeZfpGetCompressedSizeLong_32f(IppEncodeZfpState_32f* pState, Ipp64u*
pCompressedSize);
```

Include Files

`ippdc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

pState	Pointer to the ZFP compression structure.
pCompressedSize	Pointer to the variable receiving the compressed data size.

Description

This function returns the current compressed data size, in bytes. You can call this function during compression or after the compression of your array is finished.

Use EncodeZfpGetCompressedSizeLong for large 3D floating point arrays.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if any of the pointers is NULL.
ippStsContexMatchErr	Indicates an error if the ZFP compression structure data is invalid.

DecodeZfpGetStateSize

Calculates the size of the buffer for the ZFP decompression structure.

Syntax

```
IppStatus ippsDecodeZfpGetStateSize_32f(int* pStateSize);
```

Include Files

ippdc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pStateSize	Pointer to the variable receiving the size of the ZFP decompression structure.

Description

This function calculates the size of the memory buffer that your application must allocate for ZFP decompression operations.

Return Values

ippStsNoErr	Indicates no error.

ippStsNullPtrErr	Indicates an error if the pStateSize pointer is NULL.

DecodeZfpInit, DecodeZfpInitLong

Initializes the ZFP decompression structure with default values.

Syntax

```
IppStatus ippsDecodeZfpInit_32f(const Ipp8u* pSrc, int srcLen, IppDecodeZfpState_32f* pState);
```

```
IppStatus ippsDecodeZfpInitLong_32f(const Ipp8u* pSrc, Ipp64u srcLen, IppDecodeZfpState_32f* pState);
```

Include Files

ippdc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc	Pointer to the input buffer holding the compressed data.
srcLen	Length of the compressed data buffer.
pState	Pointer to the ZFP decompression structure.

Description

This function initializes the ZFP decompression structure. Before using this function, calculate the size of the structure calling the DecodeZfpGetStateSize function.

Use ippsDecodeZfpInitLong to process large 3D floating point arrays.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if any of the pointers is NULL.
ippStsSizeErr	Indicates an error if the parameter srcLen is less than or equal to zero (applies only to ippsDecodeZfpInit).

DecodeZfpSet

Populates fields of the ZFP decompression structure with input values.

Syntax

```
IppStatus ippsDecodeZfpSet_32f(int minBits, int maxBits, int maxPrec, int minExp,
IppDecodeZfpState_32f* pState);
```

Include Files

`ippdc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`minBits`	Minimum number of bits for a compressed block; the default value is `IppZFPMINBITS`.
`maxBits`	Maximum number of bits for a compressed block; the default value is `IppZFPMAXBITS`.
`maxPrec`	Maximum level of precision; the default value is `IppZFPMAXPREC`.
`minExp`	Minimum level of exponent; the default value is `IppZFPMINEXP`.
`pState`	Pointer to the ZFP decompression structure.

Description

This function populates fields of the ZFP decompression structure with the corresponding input values. Refer to [ZFP] and associated documentation for more information about the compression parameters.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if the `pState` pointer is `NULL`.
`ippStsContexMatchErr`	Indicates an error if the ZFP decompression structure is invalid.

DecodeZfpSetAccuracy

Sets the desired precision value in the ZFP decompression structure.

Syntax

```
IppStatus ippsDecodeZfpSetAccuracy_32f(Ipp64f precision, IppDecodeZfpState_32f*
pState);
```

Include Files

`ippdc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`precision`	Value to assign to the `minExp` field of the ZFP decompression structure.
`pState`	Pointer to the ZFP decompression structure.

Description

This function sets the value of the `minExp` field in the ZFP decompression structure to the value of the `precision` input parameter.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if the `pState` pointer is `NULL`.
`ippStsContexMatchErr`	Indicates an error if the ZFP decompression structure is invalid.

DecodeZfp444

Decodes a 4x4x4 block of 3D floating point data.

Syntax

```
IppStatus ippsDecodeZfp444_32f(IppDecodeZfpState_32f* pState, Ipp32f* pDst, int dstStep, int dstPlaneStep);
```

Include Files

`ippdc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pState`	Pointer to the ZFP decompression structure.
`pDst`	Pointer to the value at the coordinates (0, 0, 0) of the 3D destination buffer.
`dstStep`	Row step in bytes.
`dstPlaneStep`	Plane step in bytes.

Description

This function decodes a 4x4x4 block of 3D floating point data. The source data pointer is initialized by calling DecodeZfpInit and is updated automatically.

The function arranges the decompressed 3D data in the destination buffer in such a way that for any set of coordinates (x, y, z) within the 4x4x4 block,

```
pXYZ = (float *)((char *)pDst + x + y * dstStep + z * dstPlaneStep)
```

is a pointer to the value at (x, y, z).

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if any of the pointers is NULL.
ippStsContexMatchErr	Indicates an error if the ZFP decompression structure data is invalid.

DecodeZfpGetCompressedSize, DecodeZfpGetCompressedSizeLong

Returns the current decompressed data size.

Syntax

```
IppStatus ippsDecodeZfpGetCompressedSize_32f(IppDecodeZfpState_32f* pState, int* pDecompressedSize);
```

```
IppStatus ippsDecodeZfpGetCompressedSizeLong_32f(IppDecodeZfpState_32f* pState, Ipp64u* pDecompressedSize);
```

Include Files

ippdc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pState	Pointer to the ZFP compression structure.
pDeompressedSize	Pointer to the variable receiving the decompressed data size.

Description

This function returns the current decompressed data size, in bytes. You can call this function during decompression or after the decompression of your array is finished.

Use DecodeZfpGetCompressedSizeLong for large 3D floating point arrays.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if any of the pointers is NULL.
ippStsContexMatchErr	Indicates an error if the ZFP decompression structure data is invalid.

Appendix A: Handling of Special Cases

Some mathematical functions implemented in Intel IPP are not defined for all possible argument values. This appendix describes how the corresponding Intel IPP functions used in signal processing domains handle situations when their input arguments fall outside the range of function definition or may lead to ambiguously determined output results.

The table *"Special Cases for Intel IPP Signal Processing Functions"* below summarizes these special cases for general vector functions described in Essential Functions and lists result values together with status codes returned by these functions. The status codes ending with Err (except for the ippStsNoErr status) indicate an error. When an error occurs, the function execution is interrupted. All other status codes indicate that the input argument is outside the range, but the function execution is continued with the corresponding result value.

Function Base Name	Data Type	Case Description	Result Value	Status Code
Sqrt	16s32f64s64f	Sqrt (x<0)Sqrt (x<0)Sqrt (x<0)Sqrt (x<0)	0NAN_32F0NAN_64F	ippStsSqrtNegArgippStsSqrtNegArgippStsSqrtNegArgippStsSqrtNegArg
Div, Div_Round	8u16s16sc32f32fc64f64fc	Div (0/0)Div (x/0)Div (0/0)Div (x/0), x>0Div (x/0), x<0Div (0/0)Div (x/0)Div (0/0)Div (x/0), x>0Div (x/0), x<0Div (0/0)Div (x/0)Div (0/0)Div (x/0), x>0Div (x/0), x<0Div (0/0)Div (x/0)	0IPP_MAX_8U0IPP_MAX_16SIPP_MIN_16S00NAN_32FINF_32FINF_NEG_32FNAN_32FNAN_32FNAN_32FINF_32FINF_NEG_32FNAN_32FNAN_32F	ippStsDivByZero ippStsDivByZero ippStsDivByZero ippStsDivByZero ippStsDivByZero ippStsDivByZero ippStsDivByZero ippStsDivByZero ippStsDivByZero ippStsDivByZero ippStsDivByZero ippStsDivByZero ippStsDivByZero ippStsDivByZero ippStsDivByZero ippStsDivByZero ippStsDivByZero ippStsDivByZero
DivC	all	Div(x/0)	–	ippStsDivByZeroErr

Ln	16s32s32f64f	Ln (0)Ln (x<0)Ln (0)Ln (x<0)Ln (x<0)Ln(x<IPP_MINABS_32F)Ln (x<0)Ln(x<IPP_MINABS_64F)	IPP_MIN_16SIPP_MIN_16SIPP_MIN_32SIPP_MIN_32SNAN_32FINF_NEG_32FNAN_64FINF_NEG_64F	ippStsLnZeroArg ippStsLnZeroArg ippStsLnZeroArg ippStsLnNegArgi ppStsLnNegArgip pStsLnZeroArgip pStsLnNegArgipp StsLnZeroArg
Exp	16s32s64s32f64f	overflowoverflowoverflowoverflowoverflowoverflow	IPP_MAX_16SIPP_MAX_32SIPP_MAX_64SINF_32FINF_64F	ippStsNoErrippStsNoErrippStsNoErrippStsNoErrippStsNoErr

Here x denotes an input value. For the definition of the constants used, see Data Ranges in Intel® Integrated Performance Primitives Concepts.

Note that flavors of the same math function operating on different data types may produce different results for equal argument values. However, for a given function and a fixed data type, handling of special cases is the same for all function flavors that have different descriptors in their names. For example, the logarithm function ippiLn operating on 16s data treats zero argument values in the same way for all its flavors ippsLn_16s_Sfs and ippiLn_16s_ISfs.

The table below summarizes special cases for fixed-accuracy arithmetic functions.

Function Base Name	Data Type	Case Description	Result Value	Status Code
Inv	32f64f	Inv (x=+0)Inv (x=-0)Inv (x=+0)Inv (x=-0)	INF_32F-INF_32FINF_64F-INF_64F	ippStsSingularityippStsSingularityippStsSingularityippStsSingularity
Div	32f64f	Div (x>0, y=+0)Div (x>0, y=-0)Div (x<0, y=+0)Div (x<0, y=-0)Div (x=0, y=0)Div (x>0, y=+0)Div (x>0, y=-0)Div (x<0, y=+0)Div (x<0, y=-0)Div (x=0, y=0)	INF_32F-INF_32FINF_32F-INF_32FNAN_32FINF_64F-INF_64FINF_64F-INF_64FNAN_64F	ippStsSingularityippStsSingularityippStsSingularityippStsSingularityippStsSingularityippStsSingularityippStsSingularityippStsSingularityippStsSingularityippStsSingularityippStsSingularityippStsSingularity
Sqrt	32f64f	Sqrt(x<0)Sqrt(x=-INF)Sqrt(x<0)Sqrt(x=-INF)	NAN_32FNAN_32FNAN_64FNAN_64F	ippStsDomainippStsDomainippStsDomainippStsDomain
InvSqrt	32f64f	InvSqrt (x<0)InvSqrt (x=+0)InvSqrt (x=-0)InvSqrt	NAN_32FINF_32F-INF_32FNAN_32FNAN_64FINF_64F-INF_64FNAN_64F	ippStsDomainippStsSingularityippStsSingularityippStsDomainip

		(x=-INF)InvSqrt (x<0)InvSqrt (x=+0)InvSqrt (x=-0)InvSqrt (x=-INF)		pStsDomainippStsSingularityippStsSingularityippStsDomain
InvCbrt	32f64f	InvCbrt (x=+0)InvCbrt (x=-0)InvCbrt (x=+0)InvCbrt (x=-0)	INF_32F-INF_32FINF_64F-INF_64F	ippStsSingularityippStsSingularityippStsSingularityippStsSingularity
Pow3o2	32f64f	Pow3o2 (x<0)Pow3o2 (x=-INF)Pow3o2 (x<0)Pow3o2 (x=-INF)	NAN_32FNAN_32FNAN_64FNAN_64F	ippStsDomainippStsDomainippStsDomainippStsDomain
Pow, Powx	32f64f	Pow (x=+0, y=-ODD_INT)Pow (x=-0, y=-ODD_INT)Pow (x=0, y=-EVEN_INT)Pow (x=0, y=NON_INT_NEG)Pow (x<0, y=NON_INT_POS)Pow (x=0, y=-INF)Pow (x=+0, y=-ODD_INT)Pow (x=-0, y=-ODD_INT)Pow (x=0, y=-EVEN_INT)Pow (x=0, y=NON_INT_NEG)Pow (x<0, y=NON_INT_POS)Pow (x=0, y=-INF)	INF_32F-INF_32FINF_32FINF_32FNAN_32FINF_32FINF_64F-INF_64FINF_64FINF_64FNAN_64FINF_64F	ippStsSingularityippStsSingularityippStsSingularityippStsSingularityippStsDomainippStsSingularityippStsSingularityippStsSingularityippStsSingularityippStsDomainippStsSingularity
Exp	32f64f	Exp (x), x<underflowExp (x), x>overflowExp (x), x<underflowExp (x), x>overflow	0INF_32F0INF_64F	ippStsUnderflowippStsOverflowippStsUnderflowippStsOverflow
Expm1	32f64f	Expm1(x), x>overflowExpm1(x), x>overflow	INF_32FINF_64F	ippStsOverflowippStsOverflow

Ln, Log10	32f64f	Ln(x<0)Ln(x=-INF)Ln(x=0)Ln(x<0)Ln(x=-INF)Ln(x=0)	NAN_32FNAN_32F-INF_32FNAN_64FNAN_64F-INF_64F	ippStsDomainippStsDomainippStsSingularityippStsDomainippStsDomainippStsSingularity
Log1p	32f64f	Ln(x<-1)Ln(x=-INF)Ln(x=-1)Ln(x<-1)Ln(x=-INF)Ln(x=-1)	NAN_32FNAN_32F-INF_32FNAN_64FNAN_64F-INF_64F	ippStsDomainippStsDomainippStsSingularityippStsDomainippStsDomainippStsSingularity
Cos	32f64f	Cos(INF)Cos(INF)	NAN_32FNAN_64F	ippStsDomainippStsDomain
Sin	32f64f	Sin(INF)Sin(INF)	NAN_32FNAN_64F	ippStsDomainippStsDomain
SinCos	32f64f	SinCos(INF)SinCos(INF)	NAN_32F, NAN_32FNAN_64F, NAN_64F	ippStsDomainippStsDomain
CIS	32fc64fc	CIS(INF)CIS(INF)	NAN_32F, NAN_32FNAN_64F, NAN_64F	ippStsDomainippStsDomain
Tan	32f64f	Tan(INF)Tan(INF)	NAN_32FNAN_64F	ippStsDomainippStsDomain
Acos	32f64f	Acos(x), |x|>1Acos(INF)Acos(x), |x|>1Acos(INF)	NAN_32FNAN_32FNAN_64FNAN_64F	ippStsDomainippStsDomainippStsDomainippStsDomain
Asin	32f64f	Asin(x), |x|>1Asin(INF)Asin(x), |x|>1Asin(INF)	NAN_32FNAN_32FNAN_64FNAN_64F	ippStsDomainippStsDomainippStsDomainippStsDomain
Cosh	32f64f	Cosh(x), |x|>overflowCosh(x), |x|>overflow	INF_32FINF_64F	ippStsOverflowippStsOverflow
Sinh	32f64f	Sinh(x), |x|>overflowSinh(x), |x|>overflow	INF_32FINF_64F	ippStsOverflowippStsOverflow

Acosh	32f64f	Acosh(x<1)Acosh(x=-INF)Acosh(x<1)Acosh(x=-INF)	NAN_32FNAN_32FNAN_64FNAN_64F	ippStsDomainippStsDomainippStsDomainippStsDomain
Atanh	32f64f	Atanh(x=1)Atanh(x=-1)Atanh(x), |x|>1Atanh(INF)Atanh(x=1)Atanh(x=-1)Atanh(x), |x|>1Atanh(INF)	INF_32F-INF_32FNAN_32FNAN_32FINF_64F-INF_64FNAN_64FNAN_64F	ippStsSingularityippStsSingularityippStsDomainippStsDomainippStsSingularityippStsSingularityippStsDomainippStsDomain
Erfc	32f64f	Erfc(x), |x|>underflowErfc(x), |x|>underflow	00	ippStsUnderflowippStsUnderflow
ErfInv	32f64f	ErfInv(x=1)ErfInv(x=-1)ErfInv(x), |x|>1ErfInv(INF)ErfInv(x=1)ErfInv(x=-1)ErfInv(x), |x|>1ErfInv(INF)	INF_32F-INF_32FNAN_32FNAN_32FINF_64F-INF_64FNAN_64FNAN_64F	ippStsSingularityippStsSingularityippStsDomainippStsDomainippStsSingularityippStsSingularityippStsDomainippStsDomain
ErfcInv	32f64f	ErfcInv(x=2)ErfcInv(x=0)ErfcInv(x), |x|<0ErfcInv(x), |x|>2ErfcInv(INF)ErfcInv(x=2)ErfcInv(x=0)ErfcInv(x), |x|<0ErfcInv(x), |x|>2ErfcInv(INF)	-INF_32FINF_32FNAN_32FNAN_32FNAN_32F-INF_64FINF_64FNAN_64FNAN_64FNAN_64F	ippStsSingularityippStsSingularityippStsDomainippStsDomainippStsSingularityippStsSingularityippStsDomainippStsDomainippStsDomain
CdfNorm	32f64f	CdfNorm(x), |x|<underflowCdfNorm(x), |x|<underflow	00	ippStsUnderflowippStsUnderflow
CdfNormInv	32f64f	CdfNormInv(x=1)CdfNormInv(x=0)CdfNormInv(x), x<0CdfNormInv(x), x>1CdfNormInv(INF)CdfNormInv(x=1)CdfNormInv(x	INF_32F-INF_32FNAN_32FNAN_32FNAN_32FINF_64F-INF_64FNAN_64FNAN_64F	ippStsSingularityippStsSingularityippStsDomainippStsDomainippStsSingularityippStsSingularityippStsDomainippStsDomainippStsSingularityippStsSingularityippStsDomainippStsDomainippStsSingularityippStsSingularityi

	=0)CdfNormInv(x), x<0CdfNormInv(x), x>1CdfNormInv(I NF)	ppStsDomainippS tsDomainippStsD omain

Appendix B: Removed Functions for Signal Processing

This appendix contains tables that list the functions removed from Intel IPP 9.0. If an application created with the previous versions calls a function listed here, then the source code must be modified. The tables specify the corresponding Intel IPP 9.0 functions or workaround to replace the removed functions:

- ippcore.h
- ipps.h
- ippdi.h
- ippch.h
- ippdc.h
- ippsc.h - the whole domain is removed
- ippac.h - the whole domain is removed
- ippgen.h - the whole domain is removed

NOTE To get information on possible alternatives to the removed functions that do not have substitution or workaround in Intel IPP , file a support request at Online Service Center.

ippcore.h:

Removed from 9.0	Substitution or Workaround
ippEnableCpu	N/A
ippGetCpuType	ippGetCpuFeatures
ippGetMessageStatusI18n	N/A
ippGetNumCoresOnDie	N/A
ippInitCpu	ippSetCpuFeatures
ippMessageCatalogCloseI18n	N/A
ippMessageCatalogOpenI18n	N/A
ippSetAffinity	N/A
ippStaticInit	ippInit
ippStatusToMessageIdI18n	N/A

ipps.h:

Removed from 9.0	Substitution or Workaround

ippsl0Log10_32s_ISfs	Use ippsConvert_32s32f + ippsLog10_32f_A24 (ippVM)
ippsl0Log10_32s_Sfs	Use ippsConvert_32s32f + ippsLog10_32f_A24 (ippVM)
ippsALawToLin_8u16s	Use any open-source g711 implementation
ippsALawToLin_8u32f	Use any open-source g711 implementation
ippsALawToMuLaw_8u	Use any open-source g711 implementation
ippsAutoCorr_16s_Sfs	Use ippsConvert_16s32f and ippsAutoCorrNorm_32f
ippsAutoCorr_32f	ippsAutoCorrNorm
ippsAutoCorr_32fc	ippsAutoCorrNorm
ippsAutoCorr_64f	ippsAutoCorrNorm
ippsAutoCorr_64fc	ippsAutoCorrNorm
ippsAutoCorr_NormA_16s_Sfs	ippsAutoCorrNorm
ippsAutoCorr_NormA_32f	ippsAutoCorrNorm
ippsAutoCorr_NormA_32fc	ippsAutoCorrNorm
ippsAutoCorr_NormA_64f	ippsAutoCorrNorm
ippsAutoCorr_NormA_64fc	ippsAutoCorrNorm
ippsAutoCorr_NormB_16s_Sfs	ippsAutoCorrNorm
ippsAutoCorr_NormB_32f	ippsAutoCorrNorm
ippsAutoCorr_NormB_32fc	ippsAutoCorrNorm
ippsAutoCorr_NormB_64f	ippsAutoCorrNorm
ippsAutoCorr_NormB_64fc	ippsAutoCorrNorm
ippsBuildSymblTableDV4D_16sc	N/A
ippsCalcStatesDV_16sc	N/A
ippsCauchyDD2_32f_I	N/A
ippsCauchyD_32f_I	N/A
ippsCauchy_32f_I	N/A
ippsConjCcs_16sc	ippsConvert_16s32f + ippsConjCcs_32fc
ippsConjCcs_16sc_I	ippsConvert_16s32f + ippsConjCcs_32fc_I
ippsConjPack_16sc	ippsConvert_16s32f + ippsConjPack_32fc
ippsConjPack_16sc_I	ippsConvert_16s32f + ippsConjPack_32fc_I

ippsConjPerm_16sc	ippsConvert_16s32f + ippsConjPerm_32fc
ippsConjPerm_16sc_I	ippsConvert_16s32f + ippsConjPerm_32fc_I
ippsConvCyclic4x4_32f32fc	N/A
ippsConvCyclic8x8_16s_Sfs	N/A
ippsConvCyclic8x8_32f	N/A
ippsConv_16s_Sfs	ippsConvert_16s32f + ippsConvolve_32f
ippsConv_32f	ippsConvolve_32f
ippsConv_64f	ippsConvolve_64f
ippsCopy_1u	ippsCopyLE_1u and ippsCopyBE_1u
ippsCrossCorr_16s64s	Use ippsConvert_16s32f and ippsCrossCorrNorm_32f
ippsCrossCorr_16s_Sfs	Use ippsConvert_16s32f and ippsCrossCorrNorm_32f
ippsCrossCorr_32f	ippsCrossCorrNorm_32f
ippsCrossCorr_32fc	ippsCrossCorrNorm_32fc
ippsCrossCorr_64f	ippsCrossCorrNorm_64f
ippsCrossCorr_64fc	ippsCrossCorrNorm_64fc
ippsDCTFwdFree_16s	Use ippsFree
ippsDCTFwdFree_32f	Use ippsFree
ippsDCTFwdFree_64f	Use ippsFree
ippsDCTFwdGetBufSize_16s	ippsDCTGetSize
ippsDCTFwdGetBufSize_32f	ippsDCTGetSize
ippsDCTFwdGetBufSize_64f	ippsDCTGetSize
ippsDCTFwdGetSize_16s	ippsDCTFwdGetSize_32f
ippsDCTFwdInitAlloc_16s	ippsDCTGetSize + ippsMalloc + ippsDCTInit
ippsDCTFwdInitAlloc_32f	ippsDCTGetSize + ippsMalloc + ippsDCTInit
ippsDCTFwdInitAlloc_64f	ippsDCTGetSize + ippsMalloc + ippsDCTInit
ippsDCTFwdInit_16s	ippsDCTFwdInit_32f
ippsDCTFwd_16s_ISfs	ippsDCTFwd_32f_I
ippsDCTFwd_16s_Sfs	ippsDCTFwd_32f
ippsDCTInvFree_16s	ippsFree

ippsDCTInvFree_32f	ippsFree
ippsDCTInvFree_64f	ippsFree
ippsDCTInvGetBufSize_16s	ippsDCTGetSize
ippsDCTInvGetBufSize_32f	ippsDCTGetSize
ippsDCTInvGetBufSize_64f	ippsDCTGetSize
ippsDCTInvGetSize_16s	ippsDCTInvGetSize_32f
ippsDCTInvInitAlloc_16s	ippsDCTGetSize + ippsMalloc + ippsDCTInit
ippsDCTInvInitAlloc_32f	ippsDCTGetSize + ippsMalloc + ippsDCTInit
ippsDCTInvInitAlloc_64f	ippsDCTGetSize + ippsMalloc + ippsDCTInit
ippsDCTInvInit_16s	ippsDCTInvInit_32f
ippsDCTInv_16s_ISfs	ippsDCTInv_32f_I
ippsDCTInv_16s_Sfs	ippsDCTInv_32f
ippsDFTFree_C_16s	ippsFree
ippsDFTFree_C_16sc	ippsFree
ippsDFTFree_C_32f	ippsFree
ippsDFTFree_C_32fc	ippsFree
ippsDFTFree_C_64f	ippsFree
ippsDFTFree_C_64fc	ippsFree
ippsDFTFree_R_16s	ippsFree
ippsDFTFree_R_32f	ippsFree
ippsDFTFree_R_64f	ippsFree
ippsDFTFwd_CToC_16s_Sfs	ippsDFTFwd_CToC_32f
ippsDFTFwd_CToC_16sc_Sfs	ippsDFTFwd_CToC_32fc
ippsDFTFwd_RToCCS_16s_Sfs	ippsDFTFwd_RToCCS_32f
ippsDFTFwd_RToPack_16s_Sfs	ippsDFTFwd_RToPack_32f
ippsDFTFwd_RToPerm_16s_Sfs	ippsDFTFwd_RToPerm_32f
ippsDFTGetBufSize_C_16s	ippsDFTGetSize
ippsDFTGetBufSize_C_16sc	ippsDFTGetSize
ippsDFTGetBufSize_C_32f	ippsDFTGetSize
ippsDFTGetBufSize_C_32fc	ippsDFTGetSize
ippsDFTGetBufSize_C_64f	ippsDFTGetSize

ippsDFTGetBufSize_C_64fc	ippsDFTGetSize
ippsDFTGetBufSize_R_16s	ippsDFTGetSize
ippsDFTGetBufSize_R_32f	ippsDFTGetSize
ippsDFTGetBufSize_R_64f	ippsDFTGetSize
ippsDFTInitAlloc_C_16s	ippsDFTGetSize + ippsMalloc + ippsDFTInit
ippsDFTInitAlloc_C_16sc	ippsDFTGetSize + ippsMalloc + ippsDFTInit
ippsDFTInitAlloc_C_32f	ippsDFTGetSize + ippsMalloc + ippsDFTInit
ippsDFTInitAlloc_C_32fc	ippsDFTGetSize + ippsMalloc + ippsDFTInit
ippsDFTInitAlloc_C_64f	ippsDFTGetSize + ippsMalloc + ippsDFTInit
ippsDFTInitAlloc_C_64fc	ippsDFTGetSize + ippsMalloc + ippsDFTInit
ippsDFTInitAlloc_R_16s	ippsDFTGetSize + ippsMalloc + ippsDFTInit
ippsDFTInitAlloc_R_32f	ippsDFTGetSize + ippsMalloc + ippsDFTInit
ippsDFTInitAlloc_R_64f	ippsDFTGetSize + ippsMalloc + ippsDFTInit
ippsDFTInv_CCSToR_16s_Sfs	ippsDFTInv_CCSToR_32f
ippsDFTInv_CToC_16s_Sfs	ippsDFTInv_CToC_32f
ippsDFTInv_CToC_16sc_Sfs	ippsDFTInv_CToC_32fc
ippsDFTInv_PackToR_16s_Sfs	ippsDFTInv_PackToR_32f
ippsDFTInv_PermToR_16s_Sfs	ippsDFTInv_PermToR_32f
ippsDFTOutOrdFree_C_32fc	N/A
ippsDFTOutOrdFree_C_64fc	N/A
ippsDFTOutOrdFwd_CToC_32fc	N/A
ippsDFTOutOrdFwd_CToC_64fc	N/A
ippsDFTOutOrdGetBufSize_C_32fc	N/A
ippsDFTOutOrdGetBufSize_C_64fc	N/A
ippsDFTOutOrdInitAlloc_C_32fc	N/A
ippsDFTOutOrdInitAlloc_C_64fc	N/A
ippsDFTOutOrdInv_CToC_32fc	N/A
ippsDFTOutOrdInv_CToC_64fc	N/A
ippsDemodulateFM_CToR_16s	N/A
ippsDotProd_16s16sc32fc	Use simple C-loop and Intel compiler, or convert to supported data type and use another flavor of DotProd function

ippsDotProd_16s16sc32sc_Sfs	Use simple C-loop and Intel compiler, or convert to supported data type and use another flavor of DotProd function
ippsDotProd_16s16sc_Sfs	Use ippsDotProd_16s16sc64sc
ippsDotProd_16s_Sfs	Use ippsDotProd_16s64s
ippsDotProd_16sc32fc	Use simple C-loop and Intel compiler, or convert to supported data type and use another flavor of DotProd function
ippsDotProd_16sc32sc_Sfs	Use simple C-loop and Intel compiler, or convert to supported data type and use another flavor of DotProd function
ippsDotProd_16sc_Sfs	Use ippsDotProd_16sc64sc
ippsDotProd_32s32sc_Sfs	Use simple C-loop and Intel compiler, or convert to supported data type and use another flavor of DotProd function
ippsDotProd_32sc_Sfs	Use simple C-loop and Intel compiler, or convert to supported data type and use another flavor of DotProd function
ippsExp_32f64f	Use ippsConvert_32f64f and ippsExp_64f
ippsExp_64s_ISfs	Use ippsConvert_64s64f and ippsExp_64f_I
ippsExp_64s_Sfs	Use ippsConvert_64s64f and ippsExp_64f
ippsFFTFree_C_16s	ippsFree
ippsFFTFree_C_16sc	ippsFree
ippsFFTFree_C_32f	ippsFree
ippsFFTFree_C_32fc	ippsFree
ippsFFTFree_C_32s	ippsFree
ippsFFTFree_C_32sc	ippsFree
ippsFFTFree_C_64f	ippsFree
ippsFFTFree_C_64fc	ippsFree
ippsFFTFree_R_16s	ippsFree
ippsFFTFree_R_16s32s	ippsFree
ippsFFTFree_R_32f	ippsFree
ippsFFTFree_R_32s	ippsFree
ippsFFTFree_R_64f	ippsFree
ippsFFTFwd_CToC_16s_ISfs	ippsFFTFwd_CToC_32f_I

ippsFFTFwd_CToC_16s_Sfs	ippsFFTFwd_CToC_32f
ippsFFTFwd_CToC_16sc_ISfs	ippsFFTFwd_CToC_32fc_I
ippsFFTFwd_CToC_16sc_Sfs	ippsFFTFwd_CToC_32fc
ippsFFTFwd_CToC_32s_ISfs	ippsFFTFwd_CToC_64f_I
ippsFFTFwd_CToC_32s_Sfs	ippsFFTFwd_CToC_64f
ippsFFTFwd_CToC_32sc_ISfs	ippsFFTFwd_CToC_64fc_I
ippsFFTFwd_CToC_32sc_Sfs	ippsFFTFwd_CToC_64fc
ippsFFTFwd_RToCCS_16s32s_Sfs	ippsFFTFwd_RToCCS_32f
ippsFFTFwd_RToCCS_16s_ISfs	ippsFFTFwd_RToCCS_32f_I
ippsFFTFwd_RToCCS_16s_Sfs	ippsFFTFwd_RToCCS_32f
ippsFFTFwd_RToCCS_32s_ISfs	ippsFFTFwd_RToCCS_64f_I
ippsFFTFwd_RToCCS_32s_Sfs	ippsFFTFwd_RToCCS_64f
ippsFFTFwd_RToPack_16s_ISfs	ippsFFTFwd_RToPack_32f_I
ippsFFTFwd_RToPack_16s_Sfs	ippsFFTFwd_RToPack_32f
ippsFFTFwd_RToPack_32s_ISfs	ippsFFTFwd_RToPack_64f_I
ippsFFTFwd_RToPack_32s_Sfs	ippsFFTFwd_RToPack_64f
ippsFFTFwd_RToPerm_16s_ISfs	ippsFFTFwd_RToPerm_32f_I
ippsFFTFwd_RToPerm_16s_Sfs	ippsFFTFwd_RToPerm_32f
ippsFFTFwd_RToPerm_32s_ISfs	ippsFFTFwd_RToPerm_64f_I
ippsFFTFwd_RToPerm_32s_Sfs	ippsFFTFwd_RToPerm_64f
ippsFFTGetBufSize_C_16s	ippsFFTGetSize
ippsFFTGetBufSize_C_16sc	ippsFFTGetSize
ippsFFTGetBufSize_C_32f	ippsFFTGetSize
ippsFFTGetBufSize_C_32fc	ippsFFTGetSize
ippsFFTGetBufSize_C_32s	ippsFFTGetSize
ippsFFTGetBufSize_C_32sc	ippsFFTGetSize
ippsFFTGetBufSize_C_64f	ippsFFTGetSize
ippsFFTGetBufSize_C_64fc	ippsFFTGetSize
ippsFFTGetBufSize_R_16s	ippsFFTGetSize
ippsFFTGetBufSize_R_16s32s	ippsFFTGetSize
ippsFFTGetBufSize_R_32f	ippsFFTGetSize

ippsFFTGetBufSize_R_32s	ippsFFTGetSize
ippsFFTGetBufSize_R_64f	ippsFFTGetSize
ippsFFTGetSize_C_16s	Use 16s->32f conversion and 32f flavor of FFT - ippsFFTGetSize_C_32f
ippsFFTGetSize_C_16sc	Use 16s->32f conversion and 32f flavor of FFT - ippsFFTGetSize_C_32fc
ippsFFTGetSize_C_32s	Use 32s->32f (or to 64f) conversion and 32f (64f) flavor of FFT-ippsFFTGetSize_C_32f (64f)
ippsFFTGetSize_C_32sc	Use 32s->32f (or to 64f) conversion and 32f (64f) flavor of FFT-ippsFFTGetSize_C_32fc (64fc)
ippsFFTGetSize_R_16s	Use 16s->32f conversion and 32f flavor of FFT ippsFFTGetSize_R_32f
ippsFFTGetSize_R_16s32s	Use 16s->32f conversion and 32f flavor of FFT - ippsFFTGetSize_R_32f
ippsFFTGetSize_R_32s	Use 32s->32f (or to 64f) conversion and 32f (64f) flavor of FFT-ippsFFTGetSize_R_32f (64f)
ippsFFTInitAlloc_C_16s	ippsFFTGetSize + ippsMalloc + ippsFFTInit
ippsFFTInitAlloc_C_16sc	ippsFFTGetSize + ippsMalloc + ippsFFTInit
ippsFFTInitAlloc_C_32f	ippsFFTGetSize + ippsMalloc + ippsFFTInit
ippsFFTInitAlloc_C_32fc	ippsFFTGetSize + ippsMalloc + ippsFFTInit
ippsFFTInitAlloc_C_32s	ippsFFTGetSize + ippsMalloc + ippsFFTInit
ippsFFTInitAlloc_C_32sc	ippsFFTGetSize + ippsMalloc + ippsFFTInit
ippsFFTInitAlloc_C_64f	ippsFFTGetSize + ippsMalloc + ippsFFTInit
ippsFFTInitAlloc_C_64fc	ippsFFTGetSize + ippsMalloc + ippsFFTInit
ippsFFTInitAlloc_R_16s	ippsFFTGetSize + ippsMalloc + ippsFFTInit
ippsFFTInitAlloc_R_16s32s	ippsFFTGetSize + ippsMalloc + ippsFFTInit
ippsFFTInitAlloc_R_32f	ippsFFTGetSize + ippsMalloc + ippsFFTInit
ippsFFTInitAlloc_R_32s	ippsFFTGetSize + ippsMalloc + ippsFFTInit
ippsFFTInitAlloc_R_64f	ippsFFTGetSize + ippsMalloc + ippsFFTInit
ippsFFTInit_C_16s	Use 16s->32f conversion and 32f flavor of FFT - ippsFFTInit_C_32f
ippsFFTInit_C_16sc	Use 16s->32f conversion and 32f flavor of FFT - ippsFFTInit_C_32fc
ippsFFTInit_C_32s	Use 32s->32f (or to 64f) conversion and 32f (64f) flavor of FFT-ippsFFTInit_C_32f (64f)

ippsFFTInit_C_32sc	Use 32s->32f (or to 64f) conversion and 32f (64f) flavor of FFT-ippsFFTInit_C_32fc (64fc)
ippsFFTInit_R_16s	Use 16s->32f conversion and 32f flavor of FFT ippsFFTInit_R_32f
ippsFFTInit_R_16s32s	Use 16s->32f conversion and 32f flavor of FFT - ippsFFTInit_R_32f
ippsFFTInit_R_32s	Use 32s->32f (or to 64f) conversion and 32f (64f) flavor of FFT-ippsFFTInit_R_32f (64f)
ippsFFTInv_CCSToR_16s_ISfs	ippsFFTInv_CCSToR_32f_I
ippsFFTInv_CCSToR_16s_Sfs	ippsFFTInv_CCSToR_32f
ippsFFTInv_CCSToR_32s16s_Sfs	ippsFFTInv_CCSToR_32f
ippsFFTInv_CCSToR_32s_ISfs	ippsFFTInv_CCSToR_64f_I
ippsFFTInv_CCSToR_32s_Sfs	ippsFFTInv_CCSToR_64f
ippsFFTInv_CToC_16s_ISfs	ippsFFTInv_CToC_32f_I
ippsFFTInv_CToC_16s_Sfs	ippsFFTInv_CToC_32f
ippsFFTInv_CToC_16sc_ISfs	ippsFFTInv_CToC_32fc_I
ippsFFTInv_CToC_16sc_Sfs	ippsFFTInv_CToC_32fc
ippsFFTInv_CToC_32s_ISfs	ippsFFTInv_CToC_64f_I
ippsFFTInv_CToC_32s_Sfs	ippsFFTInv_CToC_64f
ippsFFTInv_CToC_32sc_ISfs	ippsFFTInv_CToC_64fc_I
ippsFFTInv_CToC_32sc_Sfs	ippsFFTInv_CToC_64fc
ippsFFTInv_PackToR_16s_ISfs	ippsFFTInv_PackToR_32f_I
ippsFFTInv_PackToR_16s_Sfs	ippsFFTInv_PackToR_32f
ippsFFTInv_PackToR_32s_ISfs	ippsFFTInv_PackToR_64f_I
ippsFFTInv_PackToR_32s_Sfs	ippsFFTInv_PackToR_64f
ippsFFTInv_PermToR_16s_ISfs	ippsFFTInv_PermToR_32f_I
ippsFFTInv_PermToR_16s_Sfs	ippsFFTInv_PermToR_32f
ippsFFTInv_PermToR_32s_ISfs	ippsFFTInv_PermToR_64f_I
ippsFFTInv_PermToR_32s_Sfs	ippsFFTInv_PermToR_64f
ippsFIR32f_16s_ISfs	Use new FIRSR and FIRMR APIs
ippsFIR32f_16s_Sfs	Use new FIRSR and FIRMR APIs
ippsFIR32f_Direct_16s_ISfs	Use new FIRSR and FIRMR APIs

ippsFIR32f_Direct_16s_Sfs	Use new FIRSR and FIRMR APIs
ippsFIR32fc_16sc_ISfs	Use new FIRSR and FIRMR APIs
ippsFIR32fc_16sc_Sfs	Use new FIRSR and FIRMR APIs
ippsFIR32fc_Direct_16sc_ISfs	Use new FIRSR and FIRMR APIs
ippsFIR32fc_Direct_16sc_Sfs	Use new FIRSR and FIRMR APIs
ippsFIR32s_16s_ISfs	Use new FIRSR and FIRMR APIs
ippsFIR32s_16s_Sfs	Use new FIRSR and FIRMR APIs
ippsFIR32s_Direct_16s_ISfs	Use new FIRSR and FIRMR APIs
ippsFIR32s_Direct_16s_Sfs	Use new FIRSR and FIRMR APIs
ippsFIR32sc_16sc_ISfs	Use new FIRSR and FIRMR APIs
ippsFIR32sc_16sc_Sfs	Use new FIRSR and FIRMR APIs
ippsFIR32sc_Direct_16sc_ISfs	Use new FIRSR and FIRMR APIs
ippsFIR32sc_Direct_16sc_Sfs	Use new FIRSR and FIRMR APIs
ippsFIR64f_16s_ISfs	Use new FIRSR and FIRMR APIs
ippsFIR64f_16s_Sfs	Use new FIRSR and FIRMR APIs
ippsFIR64f_32f	Use new FIRSR and FIRMR APIs
ippsFIR64f_32f_I	Use new FIRSR and FIRMR APIs
ippsFIR64f_32s_ISfs	Use new FIRSR and FIRMR APIs
ippsFIR64f_32s_Sfs	Use new FIRSR and FIRMR APIs
ippsFIR64f_Direct_16s_ISfs	Use new FIRSR and FIRMR APIs
ippsFIR64f_Direct_16s_Sfs	Use new FIRSR and FIRMR APIs
ippsFIR64f_Direct_32f	Use new FIRSR and FIRMR APIs
ippsFIR64f_Direct_32f_I	Use new FIRSR and FIRMR APIs
ippsFIR64f_Direct_32s_ISfs	Use new FIRSR and FIRMR APIs
ippsFIR64f_Direct_32s_Sfs	Use new FIRSR and FIRMR APIs
ippsFIR64fc_16sc_ISfs	Use new FIRSR and FIRMR APIs
ippsFIR64fc_16sc_Sfs	Use new FIRSR and FIRMR APIs
ippsFIR64fc_32fc	Use new FIRSR and FIRMR APIs
ippsFIR64fc_32fc_I	Use new FIRSR and FIRMR APIs
ippsFIR64fc_32sc_ISfs	Use new FIRSR and FIRMR APIs
ippsFIR64fc_32sc_Sfs	Use new FIRSR and FIRMR APIs

ippsFIR64fc_Direct_16sc_ISfs	Use new FIRSR and FIRMR APIs
ippsFIR64fc_Direct_16sc_Sfs	Use new FIRSR and FIRMR APIs
ippsFIR64fc_Direct_32fc	Use new FIRSR and FIRMR APIs
ippsFIR64fc_Direct_32fc_I	Use new FIRSR and FIRMR APIs
ippsFIR64fc_Direct_32sc_ISfs	Use new FIRSR and FIRMR APIs
ippsFIR64fc_Direct_32sc_Sfs	Use new FIRSR and FIRMR APIs
ippsFIRFree32f_16s	Use new FIRSR and FIRMR APIs
ippsFIRFree32fc_16sc	Use new FIRSR and FIRMR APIs
ippsFIRFree32s_16s	Use new FIRSR and FIRMR APIs
ippsFIRFree32sc_16sc	Use new FIRSR and FIRMR APIs
ippsFIRFree64f_16s	Use new FIRSR and FIRMR APIs
ippsFIRFree64f_32f	Use new FIRSR and FIRMR APIs
ippsFIRFree64f_32s	Use new FIRSR and FIRMR APIs
ippsFIRFree64fc_16sc	Use new FIRSR and FIRMR APIs
ippsFIRFree64fc_32fc	Use new FIRSR and FIRMR APIs
ippsFIRFree64fc_32sc	Use new FIRSR and FIRMR APIs
ippsFIRFree_16s	Use new FIRSR and FIRMR APIs
ippsFIRFree_32f	Use new FIRSR and FIRMR APIs
ippsFIRFree_32fc	Use new FIRSR and FIRMR APIs
ippsFIRFree_32s	Use new FIRSR and FIRMR APIs
ippsFIRFree_64f	Use new FIRSR and FIRMR APIs
ippsFIRFree_64fc	Use new FIRSR and FIRMR APIs
ippsFIRGetDlyLine32f_16s	Use new FIRSR and FIRMR APIs
ippsFIRGetDlyLine32fc_16sc	Use new FIRSR and FIRMR APIs
ippsFIRGetDlyLine32s_16s	Use new FIRSR and FIRMR APIs
ippsFIRGetDlyLine32sc_16sc	Use new FIRSR and FIRMR APIs
ippsFIRGetDlyLine64f_16s	Use new FIRSR and FIRMR APIs
ippsFIRGetDlyLine64f_32f	Use new FIRSR and FIRMR APIs
ippsFIRGetDlyLine64f_32s	Use new FIRSR and FIRMR APIs
ippsFIRGetDlyLine64fc_16sc	Use new FIRSR and FIRMR APIs
ippsFIRGetDlyLine64fc_32fc	Use new FIRSR and FIRMR APIs

ippsFIRGetDlyLine64fc_32sc	Use new FIRSR and FIRMR APIs
ippsFIRGetDlyLine_16s	Use new FIRSR and FIRMR APIs
ippsFIRGetDlyLine_32f	Use new FIRSR and FIRMR APIs
ippsFIRGetDlyLine_32fc	Use new FIRSR and FIRMR APIs
ippsFIRGetDlyLine_64f	Use new FIRSR and FIRMR APIs
ippsFIRGetDlyLine_64fc	Use new FIRSR and FIRMR APIs
ippsFIRGetStateSize32f_16s	Use new FIRSR and FIRMR APIs
ippsFIRGetStateSize32fc_16sc	Use new FIRSR and FIRMR APIs
ippsFIRGetStateSize32s_16s	Use new FIRSR and FIRMR APIs
ippsFIRGetStateSize32s_16s32f	Use new FIRSR and FIRMR APIs
ippsFIRGetStateSize32sc_16sc	Use new FIRSR and FIRMR APIs
ippsFIRGetStateSize32sc_16sc32fc	Use new FIRSR and FIRMR APIs
ippsFIRGetStateSize64f_16s	Use new FIRSR and FIRMR APIs
ippsFIRGetStateSize64f_32f	Use new FIRSR and FIRMR APIs
ippsFIRGetStateSize64f_32s	Use new FIRSR and FIRMR APIs
ippsFIRGetStateSize64fc_16sc	Use new FIRSR and FIRMR APIs
ippsFIRGetStateSize64fc_32fc	Use new FIRSR and FIRMR APIs
ippsFIRGetStateSize64fc_32sc	Use new FIRSR and FIRMR APIs
ippsFIRGetStateSize_16s	Use new FIRSR and FIRMR APIs
ippsFIRGetStateSize_32f	Use new FIRSR and FIRMR APIs
ippsFIRGetStateSize_32fc	Use new FIRSR and FIRMR APIs
ippsFIRGetStateSize_32s	Use new FIRSR and FIRMR APIs
ippsFIRGetStateSize_64f	Use new FIRSR and FIRMR APIs
ippsFIRGetStateSize_64fc	Use new FIRSR and FIRMR APIs
ippsFIRGetTaps32f_16s	Use new FIRSR and FIRMR APIs
ippsFIRGetTaps32fc_16sc	Use new FIRSR and FIRMR APIs
ippsFIRGetTaps32s_16s	Use new FIRSR and FIRMR APIs
ippsFIRGetTaps32s_16s32f	Use new FIRSR and FIRMR APIs
ippsFIRGetTaps32sc_16sc	Use new FIRSR and FIRMR APIs
ippsFIRGetTaps32sc_16sc32fc	Use new FIRSR and FIRMR APIs
ippsFIRGetTaps64f_16s	Use new FIRSR and FIRMR APIs

ippsFIRGetTaps64f_32f	Use new FIRSR and FIRMR APIs
ippsFIRGetTaps64f_32s	Use new FIRSR and FIRMR APIs
ippsFIRGetTaps64fc_16sc	Use new FIRSR and FIRMR APIs
ippsFIRGetTaps64fc_32fc	Use new FIRSR and FIRMR APIs
ippsFIRGetTaps64fc_32sc	Use new FIRSR and FIRMR APIs
ippsFIRGetTaps_16s	Use new FIRSR and FIRMR APIs
ippsFIRGetTaps_32f	Use new FIRSR and FIRMR APIs
ippsFIRGetTaps_32fc	Use new FIRSR and FIRMR APIs
ippsFIRGetTaps_32s	Use new FIRSR and FIRMR APIs
ippsFIRGetTaps_64f	Use new FIRSR and FIRMR APIs
ippsFIRGetTaps_64fc	Use new FIRSR and FIRMR APIs
ippsFIRInit32f_16s	Use new FIRSR and FIRMR APIs
ippsFIRInit32fc_16sc	Use new FIRSR and FIRMR APIs
ippsFIRInit32s_16s	Use new FIRSR and FIRMR APIs
ippsFIRInit32s_16s32f	Use new FIRSR and FIRMR APIs
ippsFIRInit32sc_16sc	Use new FIRSR and FIRMR APIs
ippsFIRInit32sc_16sc32fc	Use new FIRSR and FIRMR APIs
ippsFIRInit64f_16s	Use new FIRSR and FIRMR APIs
ippsFIRInit64f_32f	Use new FIRSR and FIRMR APIs
ippsFIRInit64f_32s	Use new FIRSR and FIRMR APIs
ippsFIRInit64fc_16sc	Use new FIRSR and FIRMR APIs
ippsFIRInit64fc_32fc	Use new FIRSR and FIRMR APIs
ippsFIRInit64fc_32sc	Use new FIRSR and FIRMR APIs
ippsFIRInitAlloc32f_16s	Use new FIRSR and FIRMR APIs
ippsFIRInitAlloc32fc_16sc	Use new FIRSR and FIRMR APIs
ippsFIRInitAlloc32s_16s	Use new FIRSR and FIRMR APIs
ippsFIRInitAlloc32s_16s32f	Use new FIRSR and FIRMR APIs
ippsFIRInitAlloc32sc_16sc	Use new FIRSR and FIRMR APIs
ippsFIRInitAlloc32sc_16sc32fc	Use new FIRSR and FIRMR APIs
ippsFIRInitAlloc64f_16s	Use new FIRSR and FIRMR APIs
ippsFIRInitAlloc64f_32f	Use new FIRSR and FIRMR APIs

ippsFIRInitAlloc64f_32s	Use new FIRSR and FIRMR APIs
ippsFIRInitAlloc64fc_16sc	Use new FIRSR and FIRMR APIs
ippsFIRInitAlloc64fc_32fc	Use new FIRSR and FIRMR APIs
ippsFIRInitAlloc64fc_32sc	Use new FIRSR and FIRMR APIs
ippsFIRInitAlloc_16s	Use new FIRSR and FIRMR APIs
ippsFIRInitAlloc_32f	Use new FIRSR and FIRMR APIs
ippsFIRInitAlloc_32fc	Use new FIRSR and FIRMR APIs
ippsFIRInitAlloc_32s	Use new FIRSR and FIRMR APIs
ippsFIRInitAlloc_64f	Use new FIRSR and FIRMR APIs
ippsFIRInitAlloc_64fc	Use new FIRSR and FIRMR APIs
ippsFIRInit_16s	Use new FIRSR and FIRMR APIs
ippsFIRInit_32f	Use new FIRSR and FIRMR APIs
ippsFIRInit_32fc	Use new FIRSR and FIRMR APIs
ippsFIRInit_32s	Use new FIRSR and FIRMR APIs
ippsFIRInit_64f	Use new FIRSR and FIRMR APIs
ippsFIRInit_64fc	Use new FIRSR and FIRMR APIs
ippsFIRLMSFree32f_16s	ippsFree
ippsFIRLMSFree_32f	ippsFree
ippsFIRLMSInitAlloc32f_16s	FIRLMSGetSize+ippsMalloc + FIRLMSInit
ippsFIRLMSInitAlloc_32f	FIRLMSGetSize+ippsMalloc + FIRLMSInit
ippsFIRLMSMRFree32s_16s	N/A
ippsFIRLMSMRFree32sc_16sc	N/A
ippsFIRLMSMRGetDlyLine32s_16s	N/A
ippsFIRLMSMRGetDlyLine32sc_16sc	N/A
ippsFIRLMSMRGetDlyVal32s_16s	N/A
ippsFIRLMSMRGetDlyVal32sc_16sc	N/A
ippsFIRLMSMRGetTaps32s_16s	N/A
ippsFIRLMSMRGetTaps32sc_16sc	N/A
ippsFIRLMSMRGetTapsPointer32s_16s	N/A
ippsFIRLMSMRGetTapsPointer32sc_16sc	N/A
ippsFIRLMSMRInitAlloc32s_16s	N/A

ippsFIRLMSMRInitAlloc32sc_16sc	N/A
ippsFIRLMSMROne32s_16s	N/A
ippsFIRLMSMROne32sc_16sc	N/A
ippsFIRLMSMROneVal32s_16s	N/A
ippsFIRLMSMROneVal32sc_16sc	N/A
ippsFIRLMSMRPutVal32s_16s	N/A
ippsFIRLMSMRPutVal32sc_16sc	N/A
ippsFIRLMSMRSetDlyLine32s_16s	N/A
ippsFIRLMSMRSetDlyLine32sc_16sc	N/A
ippsFIRLMSMRSetMu32s_16s	N/A
ippsFIRLMSMRSetMu32sc_16sc	N/A
ippsFIRLMSMRSetTaps32s_16s	N/A
ippsFIRLMSMRSetTaps32sc_16sc	N/A
ippsFIRLMSMRUpdateTaps32s_16s	N/A
ippsFIRLMSMRUpdateTaps32sc_16sc	N/A
ippsFIRLMSOne_Direct32f_16s	Use new FIRSR and FIRMR APIs
ippsFIRLMSOne_DirectQ15_16s	Use new FIRSR and FIRMR APIs
ippsFIRLMSOne_Direct_32f	Use new FIRSR and FIRMR APIs
ippsFIRMR32f_Direct_16s_ISfs	Use new FIRSR and FIRMR APIs
ippsFIRMR32f_Direct_16s_Sfs	Use new FIRSR and FIRMR APIs
ippsFIRMR32fc_Direct_16sc_ISfs	Use new FIRSR and FIRMR APIs
ippsFIRMR32fc_Direct_16sc_Sfs	Use new FIRSR and FIRMR APIs
ippsFIRMR32s_Direct_16s_ISfs	Use new FIRSR and FIRMR APIs
ippsFIRMR32s_Direct_16s_Sfs	Use new FIRSR and FIRMR APIs
ippsFIRMR32sc_Direct_16sc_ISfs	Use new FIRSR and FIRMR APIs
ippsFIRMR32sc_Direct_16sc_Sfs	Use new FIRSR and FIRMR APIs
ippsFIRMR64f_Direct_16s_ISfs	Use new FIRSR and FIRMR APIs
ippsFIRMR64f_Direct_16s_Sfs	Use new FIRSR and FIRMR APIs
ippsFIRMR64f_Direct_32f	Use new FIRSR and FIRMR APIs
ippsFIRMR64f_Direct_32f_I	Use new FIRSR and FIRMR APIs
ippsFIRMR64f_Direct_32s_ISfs	Use new FIRSR and FIRMR APIs

ippsFIRMR64f_Direct_32s_Sfs	Use new FIRSR and FIRMR APIs
ippsFIRMR64fc_Direct_16sc_ISfs	Use new FIRSR and FIRMR APIs
ippsFIRMR64fc_Direct_16sc_Sfs	Use new FIRSR and FIRMR APIs
ippsFIRMR64fc_Direct_32fc	Use new FIRSR and FIRMR APIs
ippsFIRMR64fc_Direct_32fc_I	Use new FIRSR and FIRMR APIs
ippsFIRMR64fc_Direct_32sc_ISfs	Use new FIRSR and FIRMR APIs
ippsFIRMR64fc_Direct_32sc_Sfs	Use new FIRSR and FIRMR APIs
ippsFIRMRGetStateSize32f_16s	Use new FIRSR and FIRMR APIs
ippsFIRMRGetStateSize32fc_16sc	Use new FIRSR and FIRMR APIs
ippsFIRMRGetStateSize32s_16s	Use new FIRSR and FIRMR APIs
ippsFIRMRGetStateSize32s_16s32f	Use new FIRSR and FIRMR APIs
ippsFIRMRGetStateSize32sc_16sc	Use new FIRSR and FIRMR APIs
ippsFIRMRGetStateSize32sc_16sc32fc	Use new FIRSR and FIRMR APIs
ippsFIRMRGetStateSize64f_16s	Use new FIRSR and FIRMR APIs
ippsFIRMRGetStateSize64f_32f	Use new FIRSR and FIRMR APIs
ippsFIRMRGetStateSize64f_32s	Use new FIRSR and FIRMR APIs
ippsFIRMRGetStateSize64fc_16sc	Use new FIRSR and FIRMR APIs
ippsFIRMRGetStateSize64fc_32fc	Use new FIRSR and FIRMR APIs
ippsFIRMRGetStateSize64fc_32sc	Use new FIRSR and FIRMR APIs
ippsFIRMRGetStateSize_16s	Use new FIRSR and FIRMR APIs
ippsFIRMRGetStateSize_32f	Use new FIRSR and FIRMR APIs
ippsFIRMRGetStateSize_32fc	Use new FIRSR and FIRMR APIs
ippsFIRMRGetStateSize_64f	Use new FIRSR and FIRMR APIs
ippsFIRMRGetStateSize_64fc	Use new FIRSR and FIRMR APIs
ippsFIRMRInit32f_16s	Use new FIRSR and FIRMR APIs
ippsFIRMRInit32fc_16sc	Use new FIRSR and FIRMR APIs
ippsFIRMRInit32s_16s	Use new FIRSR and FIRMR APIs
ippsFIRMRInit32s_16s32f	Use new FIRSR and FIRMR APIs
ippsFIRMRInit32sc_16sc	Use new FIRSR and FIRMR APIs
ippsFIRMRInit32sc_16sc32fc	Use new FIRSR and FIRMR APIs
ippsFIRMRInit64f_16s	Use new FIRSR and FIRMR APIs

ippsFIRMRInit64f_32f	Use new FIRSR and FIRMR APIs
ippsFIRMRInit64f_32s	Use new FIRSR and FIRMR APIs
ippsFIRMRInit64fc_16sc	Use new FIRSR and FIRMR APIs
ippsFIRMRInit64fc_32fc	Use new FIRSR and FIRMR APIs
ippsFIRMRInit64fc_32sc	Use new FIRSR and FIRMR APIs
ippsFIRMRInitAlloc32f_16s	Use new FIRSR and FIRMR APIs
ippsFIRMRInitAlloc32fc_16sc	Use new FIRSR and FIRMR APIs
ippsFIRMRInitAlloc32s_16s	Use new FIRSR and FIRMR APIs
ippsFIRMRInitAlloc32s_16s32f	Use new FIRSR and FIRMR APIs
ippsFIRMRInitAlloc32sc_16sc	Use new FIRSR and FIRMR APIs
ippsFIRMRInitAlloc32sc_16sc32fc	Use new FIRSR and FIRMR APIs
ippsFIRMRInitAlloc64f_16s	Use new FIRSR and FIRMR APIs
ippsFIRMRInitAlloc64f_32f	Use new FIRSR and FIRMR APIs
ippsFIRMRInitAlloc64f_32s	Use new FIRSR and FIRMR APIs
ippsFIRMRInitAlloc64fc_16sc	Use new FIRSR and FIRMR APIs
ippsFIRMRInitAlloc64fc_32fc	Use new FIRSR and FIRMR APIs
ippsFIRMRInitAlloc64fc_32sc	Use new FIRSR and FIRMR APIs
ippsFIRMRInitAlloc_16s	Use new FIRSR and FIRMR APIs
ippsFIRMRInitAlloc_32f	Use new FIRSR and FIRMR APIs
ippsFIRMRInitAlloc_32fc	Use new FIRSR and FIRMR APIs
ippsFIRMRInitAlloc_64f	Use new FIRSR and FIRMR APIs
ippsFIRMRInitAlloc_64fc	Use new FIRSR and FIRMR APIs
ippsFIRMRInit_16s	Use new FIRSR and FIRMR APIs
ippsFIRMRInit_32f	Use new FIRSR and FIRMR APIs
ippsFIRMRInit_32fc	Use new FIRSR and FIRMR APIs
ippsFIRMRInit_64f	Use new FIRSR and FIRMR APIs
ippsFIRMRInit_64fc	Use new FIRSR and FIRMR APIs
ippsFIRMRStreamGetStateSize_16s	Use new FIRSR and FIRMR APIs
ippsFIRMRStreamGetStateSize_32f	Use new FIRSR and FIRMR APIs
ippsFIRMRStreamInitAlloc_16s	Use new FIRSR and FIRMR APIs
ippsFIRMRStreamInitAlloc_32f	Use new FIRSR and FIRMR APIs

ippsFIRMRStreamInit_16s	Use new FIRSR and FIRMR APIs
ippsFIRMRStreamInit_32f	Use new FIRSR and FIRMR APIs
ippsFIRMR_Direct_32f	Use new FIRSR and FIRMR APIs
ippsFIRMR_Direct_32f_I	Use new FIRSR and FIRMR APIs
ippsFIRMR_Direct_32fc	Use new FIRSR and FIRMR APIs
ippsFIRMR_Direct_32fc_I	Use new FIRSR and FIRMR APIs
ippsFIRMR_Direct_64f	Use new FIRSR and FIRMR APIs
ippsFIRMR_Direct_64f_I	Use new FIRSR and FIRMR APIs
ippsFIRMR_Direct_64fc	Use new FIRSR and FIRMR APIs
ippsFIRMR_Direct_64fc_I	Use new FIRSR and FIRMR APIs
ippsFIROne32f_16s_Sfs	Use new FIRSR and FIRMR APIs
ippsFIROne32f_Direct_16s_ISfs	Use new FIRSR and FIRMR APIs
ippsFIROne32f_Direct_16s_Sfs	Use new FIRSR and FIRMR APIs
ippsFIROne32fc_16sc_Sfs	Use new FIRSR and FIRMR APIs
ippsFIROne32fc_Direct_16sc_ISfs	Use new FIRSR and FIRMR APIs
ippsFIROne32fc_Direct_16sc_Sfs	Use new FIRSR and FIRMR APIs
ippsFIROne32s_16s_Sfs	Use new FIRSR and FIRMR APIs
ippsFIROne32s_Direct_16s_ISfs	Use new FIRSR and FIRMR APIs
ippsFIROne32s_Direct_16s_Sfs	Use new FIRSR and FIRMR APIs
ippsFIROne32sc_16sc_Sfs	Use new FIRSR and FIRMR APIs
ippsFIROne32sc_Direct_16sc_ISfs	Use new FIRSR and FIRMR APIs
ippsFIROne32sc_Direct_16sc_Sfs	Use new FIRSR and FIRMR APIs
ippsFIROne64f_16s_Sfs	Use new FIRSR and FIRMR APIs
ippsFIROne64f_32f	Use new FIRSR and FIRMR APIs
ippsFIROne64f_32s_Sfs	Use new FIRSR and FIRMR APIs
ippsFIROne64f_Direct_16s_ISfs	Use new FIRSR and FIRMR APIs
ippsFIROne64f_Direct_16s_Sfs	Use new FIRSR and FIRMR APIs
ippsFIROne64f_Direct_32f	Use new FIRSR and FIRMR APIs
ippsFIROne64f_Direct_32f_I	Use new FIRSR and FIRMR APIs
ippsFIROne64f_Direct_32s_ISfs	Use new FIRSR and FIRMR APIs
ippsFIROne64f_Direct_32s_Sfs	Use new FIRSR and FIRMR APIs

ippsFIROne64fc_16sc_Sfs	Use new FIRSR and FIRMR APIs
ippsFIROne64fc_32fc	Use new FIRSR and FIRMR APIs
ippsFIROne64fc_32sc_Sfs	Use new FIRSR and FIRMR APIs
ippsFIROne64fc_Direct_16sc_ISfs	Use new FIRSR and FIRMR APIs
ippsFIROne64fc_Direct_16sc_Sfs	Use new FIRSR and FIRMR APIs
ippsFIROne64fc_Direct_32fc	Use new FIRSR and FIRMR APIs
ippsFIROne64fc_Direct_32fc_I	Use new FIRSR and FIRMR APIs
ippsFIROne64fc_Direct_32sc_ISfs	Use new FIRSR and FIRMR APIs
ippsFIROne64fc_Direct_32sc_Sfs	Use new FIRSR and FIRMR APIs
ippsFIROne_16s_Sfs	Use new FIRSR and FIRMR APIs
ippsFIROne_32f	Use new FIRSR and FIRMR APIs
ippsFIROne_32fc	Use new FIRSR and FIRMR APIs
ippsFIROne_32s_Sfs	Use new FIRSR and FIRMR APIs
ippsFIROne_64f	Use new FIRSR and FIRMR APIs
ippsFIROne_64fc	Use new FIRSR and FIRMR APIs
ippsFIROne_Direct_16s_ISfs	Use new FIRSR and FIRMR APIs
ippsFIROne_Direct_16s_Sfs	Use new FIRSR and FIRMR APIs
ippsFIROne_Direct_32f	Use new FIRSR and FIRMR APIs
ippsFIROne_Direct_32f_I	Use new FIRSR and FIRMR APIs
ippsFIROne_Direct_32fc	Use new FIRSR and FIRMR APIs
ippsFIROne_Direct_32fc_I	Use new FIRSR and FIRMR APIs
ippsFIROne_Direct_64f	Use new FIRSR and FIRMR APIs
ippsFIROne_Direct_64f_I	Use new FIRSR and FIRMR APIs
ippsFIROne_Direct_64fc	Use new FIRSR and FIRMR APIs
ippsFIROne_Direct_64fc_I	Use new FIRSR and FIRMR APIs
ippsFIRSetDlyLine32f_16s	Use new FIRSR and FIRMR APIs
ippsFIRSetDlyLine32fc_16sc	Use new FIRSR and FIRMR APIs
ippsFIRSetDlyLine32s_16s	Use new FIRSR and FIRMR APIs
ippsFIRSetDlyLine32sc_16sc	Use new FIRSR and FIRMR APIs
ippsFIRSetDlyLine64f_16s	Use new FIRSR and FIRMR APIs
ippsFIRSetDlyLine64f_32f	Use new FIRSR and FIRMR APIs

ippsFIRSetDlyLine64f_32s	Use new FIRSR and FIRMR APIs
ippsFIRSetDlyLine64fc_16sc	Use new FIRSR and FIRMR APIs
ippsFIRSetDlyLine64fc_32fc	Use new FIRSR and FIRMR APIs
ippsFIRSetDlyLine64fc_32sc	Use new FIRSR and FIRMR APIs
ippsFIRSetDlyLine_16s	Use new FIRSR and FIRMR APIs
ippsFIRSetDlyLine_32f	Use new FIRSR and FIRMR APIs
ippsFIRSetDlyLine_32fc	Use new FIRSR and FIRMR APIs
ippsFIRSetDlyLine_64f	Use new FIRSR and FIRMR APIs
ippsFIRSetDlyLine_64fc	Use new FIRSR and FIRMR APIs
ippsFIRSetTaps32f_16s	Use new FIRSR and FIRMR APIs
ippsFIRSetTaps32fc_16sc	Use new FIRSR and FIRMR APIs
ippsFIRSetTaps32s_16s	Use new FIRSR and FIRMR APIs
ippsFIRSetTaps32s_16s32f	Use new FIRSR and FIRMR APIs
ippsFIRSetTaps32sc_16sc	Use new FIRSR and FIRMR APIs
ippsFIRSetTaps32sc_16sc32fc	Use new FIRSR and FIRMR APIs
ippsFIRSetTaps64f_16s	Use new FIRSR and FIRMR APIs
ippsFIRSetTaps64f_32f	Use new FIRSR and FIRMR APIs
ippsFIRSetTaps64f_32s	Use new FIRSR and FIRMR APIs
ippsFIRSetTaps64fc_16sc	Use new FIRSR and FIRMR APIs
ippsFIRSetTaps64fc_32fc	Use new FIRSR and FIRMR APIs
ippsFIRSetTaps64fc_32sc	Use new FIRSR and FIRMR APIs
ippsFIRSetTaps_16s	Use new FIRSR and FIRMR APIs
ippsFIRSetTaps_32f	Use new FIRSR and FIRMR APIs
ippsFIRSetTaps_32fc	Use new FIRSR and FIRMR APIs
ippsFIRSetTaps_32s	Use new FIRSR and FIRMR APIs
ippsFIRSetTaps_64f	Use new FIRSR and FIRMR APIs
ippsFIRSetTaps_64fc	Use new FIRSR and FIRMR APIs
ippsFIRStreamGetStateSize_16s	Use new FIRSR and FIRMR APIs
ippsFIRStreamGetStateSize_32f	Use new FIRSR and FIRMR APIs
ippsFIRStreamInitAlloc_16s	Use new FIRSR and FIRMR APIs
ippsFIRStreamInitAlloc_32f	Use new FIRSR and FIRMR APIs

ippsFIRStreamInit_16s	Use new FIRSR and FIRMR APIs
ippsFIRStreamInit_32f	Use new FIRSR and FIRMR APIs
ippsFIR_16s_ISfs	Use new FIRSR and FIRMR APIs
ippsFIR_16s_Sfs	Use new FIRSR and FIRMR APIs
ippsFIR_32f	Use new FIRSR and FIRMR APIs
ippsFIR_32f_I	Use new FIRSR and FIRMR APIs
ippsFIR_32fc	Use new FIRSR and FIRMR APIs
ippsFIR_32fc_I	Use new FIRSR and FIRMR APIs
ippsFIR_32s_ISfs	Use new FIRSR and FIRMR APIs
ippsFIR_32s_Sfs	Use new FIRSR and FIRMR APIs
ippsFIR_64f	Use new FIRSR and FIRMR APIs
ippsFIR_64f_I	Use new FIRSR and FIRMR APIs
ippsFIR_64fc	Use new FIRSR and FIRMR APIs
ippsFIR_64fc_I	Use new FIRSR and FIRMR APIs
ippsFIR_Direct_16s_ISfs	Use new FIRSR and FIRMR APIs
ippsFIR_Direct_16s_Sfs	Use new FIRSR and FIRMR APIs
ippsFIR_Direct_32f	Use new FIRSR and FIRMR APIs
ippsFIR_Direct_32f_I	Use new FIRSR and FIRMR APIs
ippsFIR_Direct_32fc	Use new FIRSR and FIRMR APIs
ippsFIR_Direct_32fc_I	Use new FIRSR and FIRMR APIs
ippsFIR_Direct_64f	Use new FIRSR and FIRMR APIs
ippsFIR_Direct_64f_I	Use new FIRSR and FIRMR APIs
ippsFIR_Direct_64fc	Use new FIRSR and FIRMR APIs
ippsFIR_Direct_64fc_I	Use new FIRSR and FIRMR APIs
ippsGetVarPointDV_16sc	N/A
ippsGoertzQ15_16sc_Sfs	Q15 is not supported anymore - use 32f or 16s data types
ippsGoertzTwoQ15_16sc_Sfs	Q15 is not supported anymore - use 32f or 16s data types
ippsGoertzTwo_16sc_Sfs	2x ippsGoertz_16sc_Sfs
ippsGoertzTwo_32f	2x ippsGoertz_32f
ippsGoertzTwo_32fc	2x ippsGoertz_32fc

ippsGoertzTwo_64f	2x ippsGoertz_64f
ippsGoertzTwo_64fc	2x ippsGoertz_64fc
ippsHilbertFree_16s16sc	ippsFree
ippsHilbertFree_16s32fc	ippsFree
ippsHilbertFree_32f32fc	ippsFree
ippsHilbertInitAlloc_16s16sc	Use ippsHilbertGetSize + ippsMalloc + ippsHilbertInit for 32f data type
ippsHilbertInitAlloc_16s32fc	Use ippsHilbertGetSize + ippsMalloc + ippsHilbertInit for 32f data type
ippsHilbertInitAlloc_32f32fc	Use ippsHilbertGetSize + ippsMalloc + ippsHilbertInit
ippsHilbert_16s16sc_Sfs	Use ippsConvert_16s32f + ippsHilbert_32f32fc
ippsHilbert_16s32fc	Use ippsConvert_16s32f + ippsHilbert_32f32fc
ippsIIR32s_16s_ISfs	Use IIR with 32f taps
ippsIIR32s_16s_Sfs	Use IIR with 32f taps
ippsIIR32sc_16sc_ISfs	Use IIR with 32f taps
ippsIIR32sc_16sc_Sfs	Use IIR with 32f taps
ippsIIRFree32f_16s	ippsFree
ippsIIRFree32fc_16sc	ippsFree
ippsIIRFree32s_16s	ippsFree
ippsIIRFree32sc_16sc	ippsFree
ippsIIRFree64f_16s	ippsFree
ippsIIRFree64f_32f	ippsFree
ippsIIRFree64f_32s	ippsFree
ippsIIRFree64fc_16sc	ippsFree
ippsIIRFree64fc_32fc	ippsFree
ippsIIRFree64fc_32sc	ippsFree
ippsIIRFree_32f	ippsFree
ippsIIRFree_32fc	ippsFree
ippsIIRFree_64f	ippsFree
ippsIIRFree_64fc	ippsFree

ippsIIRGetDlyLine32s_16s	Use IIR with 32f taps
ippsIIRGetDlyLine32sc_16sc	Use IIR with 32f taps
ippsIIRGetStateSize32s_16s	Use IIR with 32f taps
ippsIIRGetStateSize32s_16s32f	Use IIR with 32f taps
ippsIIRGetStateSize32s_BiQuad_16s	Use IIR with 32f taps
ippsIIRGetStateSize32s_BiQuad_16s32f	Use IIR with 32f taps
ippsIIRGetStateSize32sc_16sc	Use IIR with 32f taps
ippsIIRGetStateSize32sc_16sc32fc	Use IIR with 32f taps
ippsIIRGetStateSize32sc_BiQuad_16sc	Use IIR with 32f taps
ippsIIRGetStateSize32sc_BiQuad_16sc32fc	Use IIR with 32f taps
ippsIIRInit32s_16s	Use IIR with 32f taps
ippsIIRInit32s_16s32f	Use IIR with 32f taps
ippsIIRInit32s_BiQuad_16s	Use IIR with 32f taps
ippsIIRInit32s_BiQuad_16s32f	Use IIR with 32f taps
ippsIIRInit32sc_16sc	Use IIR with 32f taps
ippsIIRInit32sc_16sc32fc	Use IIR with 32f taps
ippsIIRInit32sc_BiQuad_16sc	Use IIR with 32f taps
ippsIIRInit32sc_BiQuad_16sc32fc	Use IIR with 32f taps
ippsIIRInitAlloc32f_16s	ippsIIRGetSize + ippsMalloc + ippsIIRInit
ippsIIRInitAlloc32f_BiQuad_16s	ippsIIRGetSize + ippsMalloc + ippsIIRInit
ippsIIRInitAlloc32fc_16sc	ippsIIRGetSize + ippsMalloc + ippsIIRInit
ippsIIRInitAlloc32fc_BiQuad_16sc	ippsIIRGetSize + ippsMalloc + ippsIIRInit
ippsIIRInitAlloc32s_16s	Use ippsIIRGetSize + ippsMalloc + ippsIIRInit for 32f taps
ippsIIRInitAlloc32s_16s32f	Use ippsIIRGetSize + ippsMalloc + ippsIIRInit for 32f taps
ippsIIRInitAlloc32s_BiQuad_16s	Use ippsIIRGetSize + ippsMalloc + ippsIIRInit for 32f taps
ippsIIRInitAlloc32s_BiQuad_16s32f	Use ippsIIRGetSize + ippsMalloc + ippsIIRInit for 32f taps
ippsIIRInitAlloc32sc_16sc	Use ippsIIRGetSize + ippsMalloc + ippsIIRInit for 32f taps

ippsIIRInitAlloc32sc_16sc32fc	Use ippsIIRGetSize + ippsMalloc + ippsIIRInit for 32f taps
ippsIIRInitAlloc32sc_BiQuad_16sc	Use ippsIIRGetSize + ippsMalloc + ippsIIRInit for 32f taps
ippsIIRInitAlloc32sc_BiQuad_16sc32fc	Use ippsIIRGetSize + ippsMalloc + ippsIIRInit for 32f taps
ippsIIRInitAlloc64f_16s	ippsIIRGetSize + ippsMalloc + ippsIIRInit
ippsIIRInitAlloc64f_32f	ippsIIRGetSize + ippsMalloc + ippsIIRInit
ippsIIRInitAlloc64f_32s	ippsIIRGetSize + ippsMalloc + ippsIIRInit
ippsIIRInitAlloc64f_BiQuad_16s	ippsIIRGetSize + ippsMalloc + ippsIIRInit
ippsIIRInitAlloc64f_BiQuad_32f	ippsIIRGetSize + ippsMalloc + ippsIIRInit
ippsIIRInitAlloc64f_BiQuad_32s	ippsIIRGetSize + ippsMalloc + ippsIIRInit
ippsIIRInitAlloc64f_BiQuad_DF1_32s	ippsIIRGetSize + ippsMalloc + ippsIIRInit
ippsIIRInitAlloc64fc_16sc	ippsIIRGetSize + ippsMalloc + ippsIIRInit
ippsIIRInitAlloc64fc_32fc	ippsIIRGetSize + ippsMalloc + ippsIIRInit
ippsIIRInitAlloc64fc_32sc	ippsIIRGetSize + ippsMalloc + ippsIIRInit
ippsIIRInitAlloc64fc_BiQuad_16sc	ippsIIRGetSize + ippsMalloc + ippsIIRInit
ippsIIRInitAlloc64fc_BiQuad_32fc	ippsIIRGetSize + ippsMalloc + ippsIIRInit
ippsIIRInitAlloc64fc_BiQuad_32sc	ippsIIRGetSize + ippsMalloc + ippsIIRInit
ippsIIRInitAlloc_32f	ippsIIRGetSize + ippsMalloc + ippsIIRInit
ippsIIRInitAlloc_32fc	ippsIIRGetSize + ippsMalloc + ippsIIRInit
ippsIIRInitAlloc_64f	ippsIIRGetSize + ippsMalloc + ippsIIRInit
ippsIIRInitAlloc_64fc	ippsIIRGetSize + ippsMalloc + ippsIIRInit
ippsIIRInitAlloc_BiQuad_32f	ippsIIRGetSize + ippsMalloc + ippsIIRInit
ippsIIRInitAlloc_BiQuad_32fc	ippsIIRGetSize + ippsMalloc + ippsIIRInit
ippsIIRInitAlloc_BiQuad_64f	ippsIIRGetSize + ippsMalloc + ippsIIRInit
ippsIIRInitAlloc_BiQuad_64fc	ippsIIRGetSize + ippsMalloc + ippsIIRInit
ippsIIRInitAlloc_BiQuad_DF1_32f	ippsIIRGetSize + ippsMalloc + ippsIIRInit
ippsIIROne32f_16s_Sfs	ippsIIR32f_16s_Sfs
ippsIIROne32fc_16sc_Sfs	ippsIIR32fc_16sc_Sfs
ippsIIROne32s_16s_Sfs	Use IIR with 32f taps
ippsIIROne32sc_16sc_Sfs	Use IIR with 32f taps

ippsIIROne64f_16s_Sfs	ippsIIR64f_16s_Sfs
ippsIIROne64f_32f	ippsIIR64f_32f
ippsIIROne64f_32s_Sfs	ippsIIR64f_32s_Sfs
ippsIIROne64fc_16sc_Sfs	ippsIIR64fc_16sc_Sfs
ippsIIROne64fc_32fc	ippsIIR64fc_32fc
ippsIIROne64fc_32sc_Sfs	ippsIIR64fc_32sc_Sfs
ippsIIROne_32f	ippsIIR_32f
ippsIIROne_32fc	ippsIIR_32fc
ippsIIROne_64f	ippsIIR_64f
ippsIIROne_64fc	ippsIIR_64fc
ippsIIROne_BiQuadDirect_16s	Use IIR with 32f taps
ippsIIROne_BiQuadDirect_16s_I	Use IIR with 32f taps
ippsIIROne_Direct_16s	Use IIR with 32f taps
ippsIIROne_Direct_16s_I	Use IIR with 32f taps
ippsIIRSetDlyLine32s_16s	Use IIR with 32f taps
ippsIIRSetDlyLine32sc_16sc	Use IIR with 32f taps
ippsIIRSetTaps32f_16s	Set new taps with ippsIIRInit function
ippsIIRSetTaps32fc_16sc	Set new taps with ippsIIRInit function
ippsIIRSetTaps32s_16s	Set new taps with ippsIIRInit function
ippsIIRSetTaps32s_16s32f	Set new taps with ippsIIRInit function
ippsIIRSetTaps32sc_16sc	Set new taps with ippsIIRInit function
ippsIIRSetTaps32sc_16sc32fc	Set new taps with ippsIIRInit function
ippsIIRSetTaps64f_16s	Set new taps with ippsIIRInit function
ippsIIRSetTaps64f_32f	Set new taps with ippsIIRInit function
ippsIIRSetTaps64f_32s	Set new taps with ippsIIRInit function
ippsIIRSetTaps64fc_16sc	Set new taps with ippsIIRInit function
ippsIIRSetTaps64fc_32fc	Set new taps with ippsIIRInit function
ippsIIRSetTaps64fc_32sc	Set new taps with ippsIIRInit function
ippsIIRSetTaps_32f	Set new taps with ippsIIRInit function
ippsIIRSetTaps_32fc	Set new taps with ippsIIRInit function
ippsIIRSetTaps_64f	Set new taps with ippsIIRInit function

ippsIIRSetTaps_64fc	Set new taps with ippsIIRInit function
ippsIIR_BiQuadDirect_16s	Use IIR with 32f taps
ippsIIR_BiQuadDirect_16s_I	Use IIR with 32f taps
ippsIIR_Direct_16s	Use IIR with 32f taps
ippsIIR_Direct_16s_I	Use IIR with 32f taps
ippsJoinScaled_32f16s_D2L	Use simple C-loop and Intel compiler
ippsJoinScaled_32f24s_D2L	Use simple C-loop and Intel compiler
ippsLinToALaw_16s8u	Use any open-source g711 implementation
ippsLinToALaw_32f8u	Use any open-source g711 implementation
ippsLinToMuLaw_16s8u	Use any open-source g711 implementation
ippsLinToMuLaw_32f8u	Use any open-source g711 implementation
ippsLn_32s16s_Sfs	Use ippsLn32s_Sfs and ippsConvert_32s16s
ippsLn_64f32f	Use ippsLn_64f and ippsConvert_64f32f
ippsMagSquared_32fc64f	Use ippsConvert_32f64f + ippsPowerSpectr_64fc
ippsMagSquared_32sc32s_Sfs	Use ippsConvert_32s32f + ippsPowerSpectr_32fc
ippsMuLawToALaw_8u	Use any open-source g711 implementation
ippsMuLawToLin_8u16s	Use any open-source g711 implementation
ippsMuLawToLin_8u32f	Use any open-source g711 implementation
ippsMulPack_16s_ISfs	ippsMulPack_32f_I
ippsMulPack_16s_Sfs	ippsMulPack_32f
ippsMulPerm_16s_ISfs	ippsMulPerm_32f_I
ippsMulPerm_16s_Sfs	ippsMulPerm_32f
ippsMul_32s32sc_ISfs	Use simple C-loop and Intel compiler, or convert to supported data type and use another flavor of Mul function
ippsMul_32s32sc_Sfs	Use simple C-loop and Intel compiler, or convert to supported data type and use another flavor of Mul function
ippsMul_Low_32s_Sfs	Use simple C-loop and Intel compiler, or convert to supported data type and use another flavor of Mul function
ippsPackBits_32u8u	N/A

ippsPhase_32sc_Sfs	Use ippsConvert_32s32f + ippsPhase_32fc
ippsPolarToCart_16sc	ippsConvert_16s32f + ippsPolarToCart_32fc
ippsPolarToCart_32sc	ippsConvert_32s32f + ippsPolarToCart_32fc
ippsPreemphasize_16s	Use simple C-loop and Intel compiler
ippsPreemphasize_32f	Use simple C-loop and Intel compiler
ippsRandGaussFree_16s	ippsFree
ippsRandGaussFree_32f	ippsFree
ippsRandGaussFree_8u	ippsFree
ippsRandGaussInitAlloc_16s	ippsRandGaussGetSize_16s + ippsMalloc_8u + ippsRandGaussInit_16s
ippsRandGaussInitAlloc_32f	ippsRandGaussGetSize_32f + ippsMalloc_8u + ippsRandGaussInit_32f
ippsRandGaussInitAlloc_8u	ippsRandGaussGetSize_8u + ippsMalloc_8u + ippsRandGaussInit_8u
ippsRandGauss_Direct_16s	ippsRandGauss_16s
ippsRandGauss_Direct_32f	ippsRandGauss_32f
ippsRandGauss_Direct_64f	ippsRandGauss_32f + ippsConvert_32f64f
ippsRandUniformFree_16s	ippsFree
ippsRandUniformFree_32f	ippsFree
ippsRandUniformFree_8u	ippsFree
ippsRandUniformInitAlloc_16s	ippsRandUniformGetSize_16s + ippsMalloc_8u + ippsRandUniformInit_16s
ippsRandUniformInitAlloc_32f	ippsRandUniformGetSize_32f + ippsMalloc_8u + ippsRandUniformInit_32f
ippsRandUniformInitAlloc_8u	ippsRandUniformGetSize_8u + ippsMalloc_8u + ippsRandUniformInit_8u
ippsRandUniform_Direct_16s	ippsRandUniform_16s
ippsRandUniform_Direct_32f	ippsRandUniform_32f
ippsRandUniform_Direct_64f	ippsRandUniform_32f + ippsConvert_32f64f
ippsSplitScaled_16s32f_D2L	Use simple C-loop and Intel compiler
ippsSplitScaled_24s32f_D2L	Use simple C-loop and Intel compiler
ippsSqrt_64s16s_Sfs	Convert to supported data type
ippsSqrt_64s_ISfs	Convert to 64f and use ippsSqrt_64f_I

ippsSqrt_64s_Sfs	Convert to 64f and use ippsSqrt_64f
ippsToneFree	N/A
ippsToneGetStateSizeQ15_16s	N/A
ippsToneInitAllocQ15_16s	N/A
ippsToneInitQ15_16s	N/A
ippsToneQ15_16s	Q15 is not supported anymore; use ippsTone_16s + manual conversion to Q15 format
ippsToneQ15_Direct_16s	Q15 is not supported anymore; use ippsTone_16s + manual conversion to Q15 format
ippsTone_Direct_16s	ippsTone_16s
ippsTone_Direct_16sc	ippsTone_16sc
ippsTone_Direct_32f	ippsTone_32f
ippsTone_Direct_32fc	ippsTone_32fc
ippsTone_Direct_64f	ippsTone_64f
ippsTone_Direct_64fc	ippsTone_64fc
ippsTriangleFree	N/A
ippsTriangleGetStateSizeQ15_16s	N/A
ippsTriangleInitAllocQ15_16s	N/A
ippsTriangleInitQ15_16s	N/A
ippsTriangleQ15_16s	Q15 is not supported anymore; use ippsTriangle_16s + manual conversion to Q15 format
ippsTriangleQ15_Direct_16s	Q15 is not supported anymore; use ippsTriangle_16s + manual conversion to Q15 format
ippsTriangle_Direct_16s	ippsTriangle_16s
ippsTriangle_Direct_16sc	ippsTriangle_16sc
ippsTriangle_Direct_32f	ippsTriangle_32f
ippsTriangle_Direct_32fc	ippsTriangle_32fc
ippsTriangle_Direct_64f	ippsTriangle_64f
ippsTriangle_Direct_64fc	ippsTriangle_64fc
ippsUpdateLinear_16s32s_I	Use simple C-loop and Intel compiler
ippsUpdatePathMetricsDV_16u	N/A

ippsUpdatePower_16s32s_I	Use simple C-loop and Intel compiler
ippsVectorJaehne_32u	32u is not supported anymore; if still required - can be emulated with ippsVectorJaehne_32s + level shift: (Ipp32u)(x[i]+(-IPP_MIN_32S))
ippsVectorJaehne_8s	8s is not supported anymore; if still required - can be emulated with ippsVectorJaehne_16s + ippsConvert_16s8s_Sfs
ippsVectorRamp_16s	ippsVectorSlope_16s
ippsVectorRamp_16u	ippsVectorSlope_16u
ippsVectorRamp_32f	ippsVectorSlope_32f
ippsVectorRamp_32s	ippsVectorSlope_32s
ippsVectorRamp_32u	ippsVectorSlope_32u
ippsVectorRamp_64f	ippsVectorSlope_64f
ippsVectorRamp_8s	8s is not supported anymore; if still required - can be emulated with ippsVectorSlope_16s + ippsConvert_16s8s_Sfs
ippsVectorRamp_8u	ippsVectorSlope_8u
ippsVectorSlope_8s	Convert 8s to 8u or 16s data type
ippsWTFwdFree_16s32f	ippsFree
ippsWTFwdFree_16u32f	ippsFree
ippsWTFwdFree_32f	ippsFree
ippsWTFwdFree_8s32f	ippsFree
ippsWTFwdFree_8u32f	ippsFree
ippsWTFwdGetDlyLine_8s32f	Convert 8s to 8u or 16s data type
ippsWTFwdInitAlloc_16s32f	Use ippsWTFGetSize + ippsMalloc + ippsWTFInit
ippsWTFwdInitAlloc_16u32f	Use ippsWTFGetSize + ippsMalloc + ippsWTFInit
ippsWTFwdInitAlloc_32f	Use ippsWTFGetSize + ippsMalloc + ippsWTFInit
ippsWTFwdInitAlloc_8s32f	Use ippsWTFGetSize + ippsMalloc + ippsWTFInit
ippsWTFwdInitAlloc_8u32f	Use ippsWTFGetSize + ippsMalloc + ippsWTFInit
ippsWTFwdSetDlyLine_8s32f	Convert 8s to 8u or 16s data type

ippsWTFwd_8s32f	Convert 8s to 8u or 16s data type
ippsWTHaarFwd_16s	ippsWTHaarFwd_16s_Sfs
ippsWTHaarFwd_32s	ippsConvert_32s32f + ippsWTHaarFwd_32f
ippsWTHaarFwd_32s_Sfs	ippsConvert_32s32f + ippsWTHaarFwd_32f
ippsWTHaarFwd_64s	ippsConvert_64s64f + ippsWTHaarFwd_64f
ippsWTHaarFwd_64s_Sfs	ippsConvert_64s64f + ippsWTHaarFwd_64f
ippsWTHaarFwd_8s	ippsConvert_8s16s + ippsWTHaarFwd_16s_Sfs
ippsWTHaarFwd_8s_Sfs	ippsConvert_8s16s + ippsWTHaarFwd_16s_Sfs
ippsWTHaarInv_16s	ippsWTHaarInv_16s_Sfs
ippsWTHaarInv_32s	ippsConvert_32s32f + ippsWTHaarInv_32f
ippsWTHaarInv_32s_Sfs	ippsConvert_32s32f + ippsWTHaarInv_32f
ippsWTHaarInv_64s	ippsConvert_64s64f + ippsWTHaarInv_64f
ippsWTHaarInv_64s_Sfs	ippsConvert_64s64f + ippsWTHaarInv_64f
ippsWTHaarInv_8s	ippsConvert_8s16s + ippsWTHaarInv_16s_Sfs
ippsWTHaarInv_8s_Sfs	ippsConvert_8s16s + ippsWTHaarInv_16s_Sfs
ippsWTInvFree_32f	ippsFree
ippsWTInvFree_32f16s	ippsFree
ippsWTInvFree_32f16u	ippsFree
ippsWTInvFree_32f8s	ippsFree
ippsWTInvFree_32f8u	ippsFree
ippsWTInvGetDlyLine_32f8s	Convert 8s to 8u or 16s data type
ippsWTInvInitAlloc_32f	ippsWTFGetSize + ippsMalloc + ippsWTFInit
ippsWTInvInitAlloc_32f16s	ippsWTFGetSize + ippsMalloc + ippsWTFInit
ippsWTInvInitAlloc_32f16u	ippsWTFGetSize + ippsMalloc + ippsWTFInit
ippsWTInvInitAlloc_32f8s	ippsWTFGetSize + ippsMalloc + ippsWTFInit
ippsWTInvInitAlloc_32f8u	ippsWTFGetSize + ippsMalloc + ippsWTFInit
ippsWTInvSetDlyLine_32f8s	Convert 8s to 8u or 16s data type
ippsWTInv_32f8s	Convert 8s to 8u or 16s data type
ippsWinBlackmanQ15_16s	Q15 is not supported anymore, use 32f or 16s data type
ippsWinBlackmanQ15_16s_I	Q15 is not supported anymore, use 32f or 16s data type

ippsWinBlackmanQ15_16s_ISfs	Q15 is not supported anymore, use 32f or 16s data type
ippsWinBlackmanQ15_16sc	Q15 is not supported anymore, use 32f or 16s data type
ippsWinBlackmanQ15_16sc_I	Q15 is not supported anymore, use 32f or 16s data type
ippsWinKaiserQ15_16s	Q15 is not supported anymore, use 32f or 16s data type
ippsWinKaiserQ15_16s_I	Q15 is not supported anymore, use 32f or 16s data type
ippsWinKaiserQ15_16sc	Q15 is not supported anymore, use 32f or 16s data type
ippsWinKaiserQ15_16sc_I	Q15 is not supported anymore, use 32f or 16s data type

ippdi.h:

Removed from 9.0	Substitution or Workaround
ippdiGetLibVersion	N/A
ippsGFAdd_8u	N/A
ippsGFDiv_8u	N/A
ippsGFExpAlpha_8u	N/A
ippsGFGetSize_8u	N/A
ippsGFInit_8u	N/A
ippsGFInv_8u	N/A
ippsGFLogAlpha_8u	N/A
ippsGFMul_8u	N/A
ippsGFNeg_8u	N/A
ippsGFPow_8u	N/A
ippsGFSub_8u	N/A
ippsPolyGFAdd_8u	N/A
ippsPolyGFCopy_8u	N/A
ippsPolyGFDerive_8u	N/A
ippsPolyGFDiv_8u	N/A
ippsPolyGFGCD_8u	N/A

ippsPolyGFGetRef_8u	N/A
ippsPolyGFGetSize_8u	N/A
ippsPolyGFInit_8u	N/A
ippsPolyGFIrreducible_8u	N/A
ippsPolyGFMod_8u	N/A
ippsPolyGFMul_8u	N/A
ippsPolyGFPrimitive_8u	N/A
ippsPolyGFRoots_8u	N/A
ippsPolyGFSetCoeffs_8u	N/A
ippsPolyGFSetDegree_8u	N/A
ippsPolyGFShlC_8u	N/A
ippsPolyGFShrC_8u	N/A
ippsPolyGFSub_8u	N/A
ippsPolyGFValue_8u	N/A
ippsRSDecodeBMGetBufferSize_8u	N/A
ippsRSDecodeBM_8u	N/A
ippsRSDecodeEEGetBufferSize_8u	N/A
ippsRSDecodeEE_8u	N/A
ippsRSDecodeGetSize_8u	N/A
ippsRSDecodeInit_8u	N/A
ippsRSEncodeGetBufferSize_8u	N/A
ippsRSEncodeGetSize_8u	N/A
ippsRSEncodeInit_8u	N/A
ippsRSEncode_8u	N/A

ippch.h:

Removed from 9.0	Substitution or Workaround
ippsCompareIgnoreCaseLatin_16u	N/A
ippsCompareIgnoreCase_16u	N/A
ippsCompare_16u	N/A
ippsConcatC_16u_D2L	N/A

ippsConcat_16u	N/A
ippsConcat_16u_D2L	N/A
ippsEqual_16u	N/A
ippsFindCAny_16u	N/A
ippsFindC_16u	N/A
ippsFindC_Z_16u	N/A
ippsFindRevCAny_16u	N/A
ippsFindRevC_16u	N/A
ippsFindRev_16u	N/A
ippsFind_16u	N/A
ippsFind_Z_16u	N/A
ippsInsert_16u	N/A
ippsInsert_16u_I	N/A
ippsLowercaseLatin_16u	N/A
ippsLowercaseLatin_16u_I	N/A
ippsLowercase_16u	N/A
ippsLowercase_16u_I	N/A
ippsRegExpFree	ippFree
ippsRegExpInitAlloc	ippsRegExpGetSize + ippMalloc + ippsRegExpInit
ippsRegExpMultiAdd	N/A
ippsRegExpMultiDelete	N/A
ippsRegExpMultiFind_8u	N/A
ippsRegExpMultiFree	N/A
ippsRegExpMultiGetSize	N/A
ippsRegExpMultiInit	N/A
ippsRegExpMultiInitAlloc	N/A
ippsRegExpMultiModify	N/A
ippsRemove_16u	N/A
ippsRemove_16u_I	N/A
ippsReplaceC_16u	N/A

ippsSplitC_16u_D2L	N/A
ippsTrimCAny_16u	N/A
ippsTrimC_16u	N/A
ippsTrimC_16u_I	N/A
ippsTrimEndCAny_16u	N/A
ippsTrimStartCAny_16u	N/A
ippsUppercaseLatin_16u	N/A
ippsUppercaseLatin_16u_I	N/A
ippsUppercase_16u	N/A
ippsUppercase_16u_I	N/A

ippdc.h:

Removed from 9.0	Substitution or Workaround
ippsBWTFwd_SmallBlock_8u	N/A
ippsBWTGetSize_SmallBlock_8u	N/A
ippsBWTInv_SmallBlock_8u	N/A
ippsDecodeGITGetSize_8u	N/A
ippsDecodeGITInitAlloc_8u	N/A
ippsDecodeGITInit_8u	N/A
ippsDecodeGIT_8u	N/A
ippsDecodeHuffFree_BZ2_8u16u	ippFree
ippsDecodeHuffInitAlloc_8u	N/A
ippsDecodeHuffInitAlloc_BZ2_8u16u	ippsDecodeHuffGetSize_BZ2_8u16u + ippMalloc + ippsDecodeHuffInit_BZ2_8u16u
ippsDecodeHuffInit_8u	N/A
ippsDecodeHuffOne_8u	N/A
ippsDecodeHuff_8u	N/A
ippsDecodeLZ77CopyState_8u	N/A
ippsDecodeLZ77DynamicHuffFull_8u	N/A
ippsDecodeLZ77DynamicHuff_8u	N/A
ippsDecodeLZ77FixedHuffFull_8u	N/A

ippsDecodeLZ77FixedHuff_8u	N/A
ippsDecodeLZ77GetBlockType_8u	N/A
ippsDecodeLZ77GetPairs_8u	N/A
ippsDecodeLZ77GetSize_8u	N/A
ippsDecodeLZ77GetStatus_8u	N/A
ippsDecodeLZ77InitAlloc_8u	N/A
ippsDecodeLZ77Init_8u	N/A
ippsDecodeLZ77Reset_8u	N/A
ippsDecodeLZ77SetDictionary_8u	N/A
ippsDecodeLZ77SetPairs_8u	N/A
ippsDecodeLZ77SetStatus_8u	N/A
ippsDecodeLZ77StoredBlock_8u	N/A
ippsDecodeLZ77StoredHuff_8u	N/A
ippsDecodeLZ77_8u	N/A
ippsDecodeLZSSInitAlloc_8u	ippsLZSSGetSize_8u + ippMalloc + ippsDecodeLZSSInit_8u
ippsDecodeRLE_8u	N/A
ippsDecodeRLE_BZ2_8u	N/A
ippsEncodeGITGetSize_8u	N/A
ippsEncodeGITInitAlloc_8u	N/A
ippsEncodeGITInit_8u	N/A
ippsEncodeGIT_8u	N/A
ippsEncodeHuffFinal_8u	N/A
ippsEncodeHuffFree_BZ2_16u8u	ippFree
ippsEncodeHuffInitAlloc_8u	N/A
ippsEncodeHuffInitAlloc_BZ2_16u8u	ippsEncodeHuffGetSize_BZ2_16u8u + ippMalloc + ippsEncodeHuffInit_BZ2_16u8u
ippsEncodeHuffInit_8u	N/A
ippsEncodeHuffOne_8u	N/A
ippsEncodeHuff_8u	N/A
ippsEncodeLZ77DynamicHuff_8u	N/A

ippsEncodeLZ77FixedHuff_8u	N/A
ippsEncodeLZ77Flush_8u	N/A
ippsEncodeLZ77GetPairs_8u	N/A
ippsEncodeLZ77GetSize_8u	N/A
ippsEncodeLZ77GetStatus_8u	N/A
ippsEncodeLZ77InitAlloc_8u	N/A
ippsEncodeLZ77Init_8u	N/A
ippsEncodeLZ77Reset_8u	N/A
ippsEncodeLZ77SelectHuffMode_8u	N/A
ippsEncodeLZ77SetDictionary_8u	N/A
ippsEncodeLZ77SetPairs_8u	N/A
ippsEncodeLZ77SetStatus_8u	N/A
ippsEncodeLZ77StoredBlock_8u	N/A
ippsEncodeLZ77_8u	N/A
ippsEncodeLZSSInitAlloc_8u	ippsLZSSGetSize_8u + ippMalloc + ippsEncodeLZSSInit_8u
ippsEncodeRLEInitAlloc_BZ2_8u	ippsRLEGetSize_BZ2_8u + ippMalloc + ippsEncodeRLEInit_BZ2_8u
ippsEncodeRLE_8u	N/A
ippsGITFree_8u	N/A
ippsHuffFree_8u	N/A
ippsHuffGetDstBuffSize_8u	N/A
ippsHuffGetLenCodeTable_8u	N/A
ippsHuffGetSize_8u	N/A
ippsHuffLenCodeTablePack_8u	N/A
ippsHuffLenCodeTableUnpack_8u	N/A
ippsLZ77Free_8u	N/A
ippsLZSSFree_8u	ippFree
ippsMTFFree_8u	ippFree
ippsMTFInitAlloc_8u	ippsMTFGetSize_8u + ippMalloc + ippsMTFInit_8u
ippsRLEFree_BZ2_8u	ippFree

ippsc.h:

Removed from 9.0	Substitution or Workaround
ippsACELPFixedCodebookSearch_G723_16s	N/A
ippsACELPFixedCodebookSearch_G723_32s16s	N/A
ippsALCGetStateSize_G169_16s	N/A
ippsALCInit_G169_16s	N/A
ippsALCSetGain_G169_16s	N/A
ippsALCSetLevel_G169_16s	N/A
ippsALC_G169_16s	N/A
ippsAdaptiveCodebookContribution_G729_16s	N/A
ippsAdaptiveCodebookContribution_G729_32f	N/A
ippsAdaptiveCodebookDecodeGetSize_AMRWB_16s	N/A
ippsAdaptiveCodebookDecodeInit_AMRWB_16s	N/A
ippsAdaptiveCodebookDecodeUpdate_AMRWB_16s	N/A
ippsAdaptiveCodebookDecode_AMRWBE_16s	N/A
ippsAdaptiveCodebookDecode_AMRWB_16s	N/A
ippsAdaptiveCodebookDecode_GSMAMR_16s	N/A
ippsAdaptiveCodebookGainCoeff_AMRWB_16s	N/A
ippsAdaptiveCodebookGainCoeffs_GSMAMR_16s	N/A
ippsAdaptiveCodebookGain_G7291_16s	N/A
ippsAdaptiveCodebookGain_G729A_16s	N/A
ippsAdaptiveCodebookGain_G729_16s	N/A
ippsAdaptiveCodebookGain_GSMAMR_16s	N/A
ippsAdaptiveCodebookSearch_AMRWBE_16s	N/A
ippsAdaptiveCodebookSearch_AMRWB_16s	N/A
ippsAdaptiveCodebookSearch_G723	N/A
ippsAdaptiveCodebookSearch_G7291_16s	N/A
ippsAdaptiveCodebookSearch_G729A_16s	N/A
ippsAdaptiveCodebookSearch_G729D_16s	N/A
ippsAdaptiveCodebookSearch_G729_16s	N/A

ippsAdaptiveCodebookSearch_GSMAMR_16s	N/A
ippsAdaptiveCodebookSearch_RTA_32f	N/A
ippsAlgebraicCodebookDecode_AMRWB_16s	N/A
ippsAlgebraicCodebookSearchEX_AMRWB_16s	N/A
ippsAlgebraicCodebookSearchEX_GSMAMR_16s	N/A
ippsAlgebraicCodebookSearchGetBufferSize_AMRWB_16s	N/A
ippsAlgebraicCodebookSearchGetBufferSize_GSMAMR_16s	N/A
ippsAlgebraicCodebookSearchL1_G7291_16s	N/A
ippsAlgebraicCodebookSearchL2_G7291_16s	N/A
ippsAlgebraicCodebookSearch_AMRWB_16s	N/A
ippsAlgebraicCodebookSearch_GSMAMR_16s	N/A
ippsAutoCorrLagMax_32f	N/A
ippsAutoCorrLagMax_Fwd_16s	N/A
ippsAutoCorrLagMax_Inv_16s	N/A
ippsAutoCorr_16s32s	N/A
ippsAutoCorr_G723_16s	N/A
ippsAutoCorr_G729B	N/A
ippsAutoCorr_GSMAMR_16s32s	N/A
ippsAutoCorr_NormE_16s32s	N/A
ippsAutoCorr_NormE_G723_16s	N/A
ippsAutoCorr_NormE_NR_16s	N/A
ippsAutoScale_16s	N/A
ippsAutoScale_16s_I	N/A
ippsBandJoinUpsample_AMRWBE_16s	N/A
ippsBandJoin_AMRWBE_16s	N/A
ippsBandPassFilter_RTA_32f_I	N/A
ippsBandSplitDownsample_AMRWBE_16s	N/A
ippsBandSplit_AMRWBE_16s	N/A
ippsClassifyFrame_G722_16s_I	N/A

ippsCodebookSearchTCQ_G728_16s8u	N/A
ippsCodebookSearch_G728_16s	N/A
ippsCombinedFilterGetStateSize_G728_16s	N/A
ippsCombinedFilterInit_G728_16s	N/A
ippsCombinedFilterZeroInput_G728_16s	N/A
ippsCombinedFilterZeroState_G728_16s	N/A
ippsCompressEnvelopTime_G7291_16s	N/A
ippsConvPartial_16s32s	N/A
ippsConvPartial_16s_Sfs	N/A
ippsConvPartial_NR_16s	N/A
ippsConvPartial_NR_Low_16s	N/A
ippsCrossCorrLagMax_16s	N/A
ippsCrossCorrLagMax_32f64f	N/A
ippsCrossCorr_16s32s_Sfs	N/A
ippsCrossCorr_NR_16s	N/A
ippsCrossCorr_NR_16s32s	N/A
ippsCrossCorr_NR_16s_Sfs	N/A
ippsCrossCorr_NormM_16s	N/A
ippsDCTFwd_G7221_16s	N/A
ippsDCTFwd_G722_16s	N/A
ippsDCTInv_G7221_16s	N/A
ippsDCTInv_G722_16s	N/A
ippsDecDTXBuffer_AMRWB_16s	N/A
ippsDecDTXBuffer_GSMAMR_16s	N/A
ippsDecodeAdaptiveVector_G723_16s	N/A
ippsDecodeAdaptiveVector_G729_16s	N/A
ippsDecodeAdaptiveVector_G729_16s_I	N/A
ippsDecodeAdaptiveVector_G729_32f_I	N/A
ippsDecodeDemux_AMRWBE_16s	N/A
ippsDecodeGain_AMRWB_16s	N/A
ippsDecodeGain_G729I_16s	N/A

ippsDecodeGain_G729_16s	N/A
ippsDecodeGetStateSize_G726_8u16s	N/A
ippsDecodeInit_G726_8u16s	N/A
ippsDecode_G726_8u16s	N/A
ippsDecomposeDCTToMLT_G7221_16s	N/A
ippsDecomposeDCTToMLT_G722_16s	N/A
ippsDecomposeMLTToDCT_G7221_16s	N/A
ippsDecomposeMLTToDCT_G722_16s	N/A
ippsDeemphasize_AMRWBE_NR_16s_I	N/A
ippsDeemphasize_AMRWB_32s16s	N/A
ippsDeemphasize_AMRWB_NR_16s_I	N/A
ippsDeemphasize_GSMFR_16s_I	N/A
ippsDotProdAutoScale_16s32s_Sfs	N/A
ippsDotProd_G729A_16s32s	N/A
ippsDotProd_G729A_32f	N/A
ippsDownsampleFilter_G722_16s	N/A
ippsDownsample_AMRWBE_16s	N/A
ippsEncDTXBuffer_AMRWB_16s	N/A
ippsEncDTXBuffer_GSMAMR_16s	N/A
ippsEncDTXHandler_GSMAMR_16s	N/A
ippsEncDTXSID_GSMAMR_16s	N/A
ippsEncodeGetStateSize_G726_16s8u	N/A
ippsEncodeInit_G726_16s8u	N/A
ippsEncodeMux_AMRWBE_16s	N/A
ippsEncode_G726_16s8u	N/A
ippsEnvelopFrequency_G7291_16s	N/A
ippsEnvelopTime_G7291_16s	N/A
ippsFFTFwd_RToPerm_AMRWBE_16s	N/A
ippsFFTFwd_RToPerm_GSMAMR_16s_I	N/A
ippsFFTInv_PermToR_AMRWBE_16s	N/A
ippsFIRGenMidBand_AMRWBE_16s	N/A

ippsFIRSubbandAPCoeffUpdate_EC_32fc_I	N/A
ippsFIRSubbandCoeffUpdate_EC_32fc_I	N/A
ippsFIRSubbandCoeffUpdate_EC_32sc_I	N/A
ippsFIRSubbandLowCoeffUpdate_EC_32sc_I	N/A
ippsFIRSubbandLow_EC_32sc_Sfs	N/A
ippsFIRSubband_EC_32fc	N/A
ippsFIRSubband_EC_32sc_Sfs	N/A
ippsFIR_EC_16s	N/A
ippsFIR_EC_32f	N/A
ippsFilterHighband_G722_16s_I	N/A
ippsFilterHighpassGetStateSize_G7291_16s	N/A
ippsFilterHighpassInit_G7291_16s	N/A
ippsFilterHighpass_G7291_16s_ISfs	N/A
ippsFilterLowpass_G7291_16s_I	N/A
ippsFilterNoiseDetectModerate_EC_32f64f	N/A
ippsFilterNoiseDetect_EC_32f64f	N/A
ippsFilterNoiseGetStateSize_EC_32f	N/A
ippsFilterNoiseGetStateSize_RTA_32f	N/A
ippsFilterNoiseInit_EC_32f	N/A
ippsFilterNoiseInit_RTA_32f	N/A
ippsFilterNoiseLevel_EC_32f	N/A
ippsFilterNoiseLevel_RTA_32f	N/A
ippsFilterNoiseSetMode_EC_32f	N/A
ippsFilterNoise_EC_32f	N/A
ippsFilterNoise_EC_32f_I	N/A
ippsFilterNoise_RTA_32f	N/A
ippsFilterNoise_RTA_32f_I	N/A
ippsFilteredExcitation_G729_32f	N/A
ippsFixedCodebookDecode_GSMAMR_16s	N/A
ippsFixedCodebookSearchBuffer_RTA_32f	N/A
ippsFixedCodebookSearchRandom_RTA_32f	N/A

ippsFixedCodebookSearch_G729A_16s	N/A
ippsFixedCodebookSearch_G729A_32f	N/A
ippsFixedCodebookSearch_G729A_32s16s	N/A
ippsFixedCodebookSearch_G729D_16s	N/A
ippsFixedCodebookSearch_G729D_32f	N/A
ippsFixedCodebookSearch_G729E_16s	N/A
ippsFixedCodebookSearch_G729E_32f	N/A
ippsFixedCodebookSearch_G729_16s	N/A
ippsFixedCodebookSearch_G729_32f	N/A
ippsFixedCodebookSearch_G729_32s16s	N/A
ippsFixedCodebookSearch_RTA_GetBufferSize_32f	N/A
ippsFullbandControllerGetSize_EC_16s	N/A
ippsFullbandControllerGetSize_EC_32f	N/A
ippsFullbandControllerInit_EC_16s	N/A
ippsFullbandControllerInit_EC_32f	N/A
ippsFullbandControllerReset_EC_16s	N/A
ippsFullbandControllerReset_EC_32f	N/A
ippsFullbandControllerUpdate_EC_16s	N/A
ippsFullbandControllerUpdate_EC_32f	N/A
ippsFullbandController_EC_16s	N/A
ippsFullbandController_EC_32f	N/A
ippsGainCodebookSearch_G729D_32f	N/A
ippsGainCodebookSearch_G729_32f	N/A
ippsGainControl_G723_16s_I	N/A
ippsGainControl_G7291_16s_I	N/A
ippsGainControl_G729A_16s_I	N/A
ippsGainControl_G729_16s_I	N/A
ippsGainControl_G729_32f_I	N/A
ippsGainDecodeTCX_AMRWBE_16s	N/A
ippsGainQuantTCX_AMRWBE_16s	N/A

ippsGainQuant_AMRWBE_16s	N/A
ippsGainQuant_AMRWB_16s	N/A
ippsGainQuant_G723_16s	N/A
ippsGainQuant_G7291_16s	N/A
ippsGainQuant_G729D_16s	N/A
ippsGainQuant_G729_16s	N/A
ippsGenerateExcitationGetStateSize_G7291_16s	N/A
ippsGenerateExcitationInit_G7291_16s	N/A
ippsGenerateExcitation_G7291_16s	N/A
ippsHarmonicFilter_16s_I	N/A
ippsHarmonicFilter_32f_I	N/A
ippsHarmonicFilter_NR_16s	N/A
ippsHarmonicNoiseSubtract_G723_16s_I	N/A
ippsHarmonicSearch_G723_16s	N/A
ippsHighPassFilterGetDlyLine_AMRWB_16s	N/A
ippsHighPassFilterGetSize_AMRWB_16s	N/A
ippsHighPassFilterInit_AMRWB_16s	N/A
ippsHighPassFilterInit_G729	N/A
ippsHighPassFilterSetDlyLine_AMRWB_16s	N/A
ippsHighPassFilterSize_G729	N/A
ippsHighPassFilter_AMRWB_16s_ISfs	N/A
ippsHighPassFilter_AMRWB_16s_Sfs	N/A
ippsHighPassFilter_Direct_AMRWB_16s	N/A
ippsHighPassFilter_G723_16s	N/A
ippsHighPassFilter_G729_16s_ISfs	N/A
ippsHighPassFilter_GSMFR_16s	N/A
ippsHighPassFilter_RTA_32f	N/A
ippsHuffmanEncode_G722_16s32u	N/A
ippsIIR16sGetStateSize_G728_16s	N/A
ippsIIR16sInit_G728_16s	N/A

ippsIIR16sLow_G729_16s	N/A
ippsIIR16s_G723_16s32s	N/A
ippsIIR16s_G723_16s_I	N/A
ippsIIR16s_G723_32s16s_Sfs	N/A
ippsIIR16s_G728_16s	N/A
ippsIIR16s_G729_16s	N/A
ippsISFQuantDTX_AMRWB_16s	N/A
ippsISFQuantDecodeDTX_AMRWB_16s	N/A
ippsISFQuantDecodeHighBand_AMRWBE_16s	N/A
ippsISFQuantDecode_AMRWBE_16s	N/A
ippsISFQuantDecode_AMRWB_16s	N/A
ippsISFQuantHighBand_AMRWBE_16s	N/A
ippsISFQuant_AMRWB_16s	N/A
ippsISFToISP_AMRWB_16s	N/A
ippsISPToISF_Norm_AMRWB_16s	N/A
ippsISPToLPC_AMRWB_16s	N/A
ippsImpulseResponseEnergy_G728_16s	N/A
ippsImpulseResponseTarget_GSMAMR_16s	N/A
ippsInterpolateC_G729_16s_Sfs	N/A
ippsInterpolateC_G729_32f	N/A
ippsInterpolateC_NR_16s	N/A
ippsInterpolateC_NR_G729_16s_Sfs	N/A
ippsInterpolate_G729_16s	N/A
ippsInterpolate_GSMAMR_16s	N/A
ippsInvSqrt_32s_I	N/A
ippsLPCInverseFilter_G728_16s	N/A
ippsLPCToISP_AMRWBE_16s	N/A
ippsLPCToISP_AMRWB_16s	N/A
ippsLPCToLSF_G723_16s	N/A
ippsLPCToLSP_G729A_16s	N/A
ippsLPCToLSP_G729A_32f	N/A

ippsLPCToLSP_G729_16s	N/A
ippsLPCToLSP_G729_32f	N/A
ippsLPCToLSP_GSMAMR_16s	N/A
ippsLPCToLSP_RTA_32f	N/A
ippsLSFDecodeErased_G729_16s	N/A
ippsLSFDecodeErased_G729_32f	N/A
ippsLSFDecode_G723_16s	N/A
ippsLSFDecode_G7291_16s	N/A
ippsLSFDecode_G729B_16s	N/A
ippsLSFDecode_G729B_32f	N/A
ippsLSFDecode_G729_16s	N/A
ippsLSFDecode_G729_32f	N/A
ippsLSFQuant_G723_16s32s	N/A
ippsLSFQuant_G729B_16s	N/A
ippsLSFQuant_G729B_32f	N/A
ippsLSFQuant_G729_16s	N/A
ippsLSFToLPC_G723_16s	N/A
ippsLSFToLPC_G723_16s_I	N/A
ippsLSFToLSP_G729_16s	N/A
ippsLSFToLSP_GSMAMR_16s	N/A
ippsLSPQuant_G729E_16s	N/A
ippsLSPQuant_G729E_32f	N/A
ippsLSPQuant_G729_16s	N/A
ippsLSPQuant_GSMAMR_16s	N/A
ippsLSPQuant_RTA_32f	N/A
ippsLSPToLPC_G729_16s	N/A
ippsLSPToLPC_G729_32f	N/A
ippsLSPToLPC_GSMAMR_16s	N/A
ippsLSPToLPC_RTA_32f	N/A
ippsLSPToLSF_G729_16s	N/A
ippsLSPToLSF_Norm_G729_16s	N/A

ippsLagWindow_G729_32s_I	N/A
ippsLevinsonDurbin_G723_16s	N/A
ippsLevinsonDurbin_G729B	N/A
ippsLevinsonDurbin_G729_32f	N/A
ippsLevinsonDurbin_G729_32s16s	N/A
ippsLevinsonDurbin_GSMAMR_32s16s	N/A
ippsLevinsonDurbin_RTA_32f	N/A
ippsLongTermPostFilter_G729A_16s	N/A
ippsLongTermPostFilter_G729B_16s	N/A
ippsLongTermPostFilter_G729_16s	N/A
ippsMDCTFwd_G7291_16s	N/A
ippsMDCTInv_G7291_16s	N/A
ippsMDCTPostProcess_G7291_16s	N/A
ippsMDCTQuantFwd_G7291_16s32u	N/A
ippsMDCTQuantInv_G7291_32u16s	N/A
ippsMPMLQFixedCodebookSearch_G723	N/A
ippsMulC_NR_16s_ISfs	N/A
ippsMulC_NR_16s_Sfs	N/A
ippsMulPowerC_NR_16s_Sfs	N/A
ippsMul_NR_16s_ISfs	N/A
ippsMul_NR_16s_Sfs	N/A
ippsNLMS_EC_16s	N/A
ippsNLMS_EC_32f	N/A
ippsOpenLoopPitchSearchDTXVAD1_GSMAMR_16s	N/A
ippsOpenLoopPitchSearchDTXVAD2_GSMAMR_16s32s	N/A
ippsOpenLoopPitchSearchNonDTX_GSMAMR_16s	N/A
ippsOpenLoopPitchSearch_AMRWBE_16s	N/A
ippsOpenLoopPitchSearch_AMRWB_16s	N/A
ippsOpenLoopPitchSearch_G723_16s	N/A
ippsOpenLoopPitchSearch_G729A_16s	N/A

ippsOpenLoopPitchSearch_G729A_32f	N/A
ippsOpenLoopPitchSearch_G729_16s	N/A
ippsPhaseDispersionGetStateSize_G729D_16s	N/A
ippsPhaseDispersionInit_G729D_16s	N/A
ippsPhaseDispersionUpdate_G729D_16s	N/A
ippsPhaseDispersion_G729D_16s	N/A
ippsPitchPeriodExtraction_G728_16s	N/A
ippsPitchPostFilter_G723_16s	N/A
ippsPostFilterAdapterGetStateSize_G728	N/A
ippsPostFilterAdapterStateInit_G728	N/A
ippsPostFilterGetStateSize_G728_16s	N/A
ippsPostFilterGetStateSize_RTA_32f	N/A
ippsPostFilterInit_G728_16s	N/A
ippsPostFilterInit_RTA_32f	N/A
ippsPostFilterLowBand_AMRWBE_16s	N/A
ippsPostFilter_G728_16s	N/A
ippsPostFilter_GSMAMR_16s	N/A
ippsPostFilter_RTA_32f_I	N/A
ippsPreemphasize_32f_I	N/A
ippsPreemphasize_AMRWB_16s_ISfs	N/A
ippsPreemphasize_G729A_16s	N/A
ippsPreemphasize_G729A_16s_I	N/A
ippsPreemphasize_GSMAMR_16s	N/A
ippsPreemphasize_GSMFR_16s	N/A
ippsQMFDecode_G722_16s	N/A
ippsQMFDecode_G7291_16s	N/A
ippsQMFDecode_RTA_32f	N/A
ippsQMFEncode_G722_16s	N/A
ippsQMFEncode_G7291_16s	N/A
ippsQMFEncode_RTA_32f	N/A
ippsQMFGetStateSize_G7291_16s	N/A

ippsQMFGetStateSize_RTA_32f	N/A
ippsQMFInit_G7291_16s	N/A
ippsQMFInit_RTA_32f	N/A
ippsQuantLSPDecode_GSMAMR_16s	N/A
ippsQuantParam_G7291_16s	N/A
ippsQuantTCX_AMRWBE_16s	N/A
ippsRPEQuantDecode_GSMFR_16s	N/A
ippsRandomNoiseExcitation_G729B_16s	N/A
ippsRandomNoiseExcitation_G729B_16s32f	N/A
ippsResamplePolyphase_AMRWBE_16s	N/A
ippsResidualFilter_AMRWB_16s_Sfs	N/A
ippsResidualFilter_G729E_16s	N/A
ippsResidualFilter_G729_16s	N/A
ippsResidualFilter_Low_16s_Sfs	N/A
ippsSBADPCMDecodeInit_G722_16s	N/A
ippsSBADPCMDecodeStateSize_G722_16s	N/A
ippsSBADPCMDecodeStateUpdate_G722_16s	N/A
ippsSBADPCMDecode_G722_16s	N/A
ippsSBADPCMEncodeInit_G722_16s	N/A
ippsSBADPCMEncodeStateSize_G722_16s	N/A
ippsSBADPCMEncode_G722_16s	N/A
ippsSNR_AMRWBE_16s	N/A
ippsSchur_GSMFR_32s16s	N/A
ippsShapeEnvelopFrequency_G7291_16s	N/A
ippsShapeEnvelopTime_G7291_16s	N/A
ippsShortTermAnalysisFilter_GSMFR_16s_I	N/A
ippsShortTermPostFilter_G729_16s	N/A
ippsShortTermPostFilter_G729A_16s	N/A
ippsShortTermSynthesisFilter_GSMFR_16s	N/A
ippsSubbandAPControllerUpdate_EC_32f	N/A
ippsSubbandAnalysis_16s32sc_Sfs	N/A

ippsSubbandAnalysis_32f32fc	N/A
ippsSubbandControllerDTGetSize_EC_16s	N/A
ippsSubbandControllerDTInit_EC_16s	N/A
ippsSubbandControllerDTReset_EC_16s	N/A
ippsSubbandControllerDTUpdate_EC_16s	N/A
ippsSubbandControllerDT_EC_16s	N/A
ippsSubbandControllerGetSize_EC_16s	N/A
ippsSubbandControllerGetSize_EC_32f	N/A
ippsSubbandControllerInit_EC_16s	N/A
ippsSubbandControllerInit_EC_32f	N/A
ippsSubbandControllerReset_EC_16s	N/A
ippsSubbandControllerReset_EC_32f	N/A
ippsSubbandControllerUpdate_EC_16s	N/A
ippsSubbandControllerUpdate_EC_32f	N/A
ippsSubbandController_EC_16s	N/A
ippsSubbandController_EC_32f	N/A
ippsSubbandProcessGetSize_16s	N/A
ippsSubbandProcessGetSize_32f	N/A
ippsSubbandProcessInit_16s	N/A
ippsSubbandProcessInit_32f	N/A
ippsSubbandSynthesis_32fc32f	N/A
ippsSubbandSynthesis_32sc16s_Sfs	N/A
ippsSyntesisFilterZeroInput_G728_16s	N/A
ippsSynthesisFilterGetStateSize_G728_16s	N/A
ippsSynthesisFilterInit_G728_16s	N/A
ippsSynthesisFilterLow_NR_16s_ISfs	N/A
ippsSynthesisFilterZeroStateResponse_NR_16s	N/A
ippsSynthesisFilter_AMRWBE_16s32s_I	N/A
ippsSynthesisFilter_AMRWB_16s32s_I	N/A
ippsSynthesisFilter_G723_16s	N/A

ippsSynthesisFilter_G723_16s32s	N/A
ippsSynthesisFilter_G729E_16s	N/A
ippsSynthesisFilter_G729E_16s_I	N/A
ippsSynthesisFilter_G729_32f	N/A
ippsSynthesisFilter_NR_16s_ISfs	N/A
ippsSynthesisFilter_NR_16s_Sfs	N/A
ippsTiltCompensation_G723_32s16s	N/A
ippsTiltCompensation_G7291_16s	N/A
ippsTiltCompensation_G729A_16s	N/A
ippsTiltCompensation_G729E_16s	N/A
ippsTiltCompensation_G729_16s	N/A
ippsToeplizMatrix_G723_16s	N/A
ippsToeplizMatrix_G723_16s32s	N/A
ippsToeplizMatrix_G729D_32f	N/A
ippsToeplizMatrix_G729_16s	N/A
ippsToeplizMatrix_G729_16s32s	N/A
ippsToeplizMatrix_G729_32f	N/A
ippsToneDetectGetStateSize_EC_16s	N/A
ippsToneDetectGetStateSize_EC_32f	N/A
ippsToneDetectInit_EC_16s	N/A
ippsToneDetectInit_EC_32f	N/A
ippsToneDetect_EC_16s	N/A
ippsToneDetect_EC_32f	N/A
ippsUpsampleEX_AMRWBE_16s	N/A
ippsUpsampleGetBufferSize_AMRWBE_16s	N/A
ippsUpsample_AMRWBE_16s	N/A
ippsVAD1_GSMAMR_16s	N/A
ippsVAD2_GSMAMR_16s	N/A
ippsVADGetEnergyLevel_AMRWB_16s	N/A
ippsVADGetSize_AMRWB_16s	N/A
ippsVADInit_AMRWB_16s	N/A

ippsVAD_AMRWB_16s	N/A
ippsWeightingFilter_GSMFR_16s	N/A
ippsWinHybridBlock_G728_16s	N/A
ippsWinHybridGetStateSize_G728_16s	N/A
ippsWinHybridGetStateSize_G729E_16s	N/A
ippsWinHybridGetStateSize_G729E_32f	N/A
ippsWinHybridInit_G728_16s	N/A
ippsWinHybridInit_G729E_16s	N/A
ippsWinHybridInit_G729E_32f	N/A
ippsWinHybrid_G728_16s	N/A
ippsWinHybrid_G729E_16s32s	N/A
ippsWinHybrid_G729E_32f	N/A
ippscGetLibVersion	N/A

ippac.h:

Removed from 9.0	Substitution or Workaround
ippacGetLibVersion	N/A
ippsAnalysisFilterEncFree_SBR_32f	N/A
ippsAnalysisFilterEncGetSize_SBR_32f	N/A
ippsAnalysisFilterEncInitAlloc_SBR_32f	N/A
ippsAnalysisFilterEncInit_SBR_32f	N/A
ippsAnalysisFilterEnc_SBR_32f32fc	N/A
ippsAnalysisFilterFree_PQMF_MP3_32f	N/A
ippsAnalysisFilterFree_SBRHQ_32s32sc	N/A
ippsAnalysisFilterFree_SBRLP_32s	N/A
ippsAnalysisFilterGetSize_PQMF_MP3_32f	N/A
ippsAnalysisFilterGetSize_SBRHQ_32s32sc	N/A
ippsAnalysisFilterGetSize_SBRLP_32s	N/A
ippsAnalysisFilterGetSize_SBR_RToC_32f	N/A
ippsAnalysisFilterGetSize_SBR_RToC_32f32fc	N/A

ippsAnalysisFilterGetSize_SBR_RToR_32f	N/A
ippsAnalysisFilterInitAlloc_PQMF_MP3_32f	N/A
ippsAnalysisFilterInitAlloc_SBRHQ_32s32sc	N/A
ippsAnalysisFilterInitAlloc_SBRLP_32s	N/A
ippsAnalysisFilterInit_PQMF_MP3_32f	N/A
ippsAnalysisFilterInit_SBRHQ_32s32sc	N/A
ippsAnalysisFilterInit_SBRLP_32s	N/A
ippsAnalysisFilterInit_SBR_RToC_32f	N/A
ippsAnalysisFilterInit_SBR_RToC_32f32fc	N/A
ippsAnalysisFilterInit_SBR_RToR_32f	N/A
ippsAnalysisFilter_PQMF_MP3_32f	N/A
ippsAnalysisFilter_PS_32fc_D2	N/A
ippsAnalysisFilter_SBRHQ_32s32sc	N/A
ippsAnalysisFilter_SBRLP_32s	N/A
ippsAnalysisFilter_SBR_RToC_32f32fc_D2L	N/A
ippsAnalysisFilter_SBR_RToC_32f_D2L	N/A
ippsAnalysisFilter_SBR_RToR_32f_D2L	N/A
ippsAnalysisPQMF_MP3_16s32s	N/A
ippsBitReservoirInit_MP3	N/A
ippsCalcSF_16s32f	N/A
ippsDecodeChanPairElt_AAC	N/A
ippsDecodeChanPairElt_MP4_AAC	N/A
ippsDecodeDatStrElt_AAC	N/A
ippsDecodeExtensionHeader_AAC	N/A
ippsDecodeFillElt_AAC	N/A
ippsDecodeIsStereo_AAC_32s	N/A
ippsDecodeMainHeader_AAC	N/A
ippsDecodeMsPNS_AAC_32s	N/A
ippsDecodeMsStereo_AAC_32s_I	N/A
ippsDecodePNS_AAC_32s	N/A
ippsDecodePrgCfgElt_AAC	N/A

ippsDecodeTNS_AAC_32s_I	N/A
ippsDeinterleaveSpectrum_AAC_32s	N/A
ippsDeinterleave_16s	N/A
ippsDeinterleave_32f	N/A
ippsDetectTransient_SBR_32f	N/A
ippsEncodeTNS_AAC_32s_I	N/A
ippsEstimateTNR_SBR_32f	N/A
ippsFDPFree_32f	N/A
ippsFDPFwd_32f	N/A
ippsFDPGetSize_32f	N/A
ippsFDPInitAlloc_32f	N/A
ippsFDPInit_32f	N/A
ippsFDPInv_32f_I	N/A
ippsFDPResetGroup_32f	N/A
ippsFDPResetSfb_32f	N/A
ippsFDPReset_32f	N/A
ippsFIRBlockFree_32f	N/A
ippsFIRBlockInitAlloc_32f	N/A
ippsFIRBlockOne_32f	N/A
ippsHuffmanDecodeSfbMbp_MP3_1u32s	N/A
ippsHuffmanDecodeSfb_MP3_1u32s	N/A
ippsHuffmanDecode_MP3_1u32s	N/A
ippsHuffmanEncode_MP3_32s1u	N/A
ippsInterleave_16s	N/A
ippsInterleave_32f	N/A
ippsJoin_32f16s_D2L	N/A
ippsJointStereoEncode_MP3_32s_I	N/A
ippsLongTermPredict_AAC_32s	N/A
ippsLongTermReconstruct_AAC_32s	N/A
ippsLtpUpdate_AAC_32s	N/A
ippsMDCTFwdFree_16s	N/A

ippsMDCTFwdFree_32f	N/A
ippsMDCTFwdGetBufSize_16s	N/A
ippsMDCTFwdGetBufSize_32f	N/A
ippsMDCTFwdGetSize_16s	N/A
ippsMDCTFwdGetSize_32f	N/A
ippsMDCTFwdInitAlloc_16s	N/A
ippsMDCTFwdInitAlloc_32f	N/A
ippsMDCTFwdInit_16s	N/A
ippsMDCTFwdInit_32f	N/A
ippsMDCTFwd_16s_Sfs	N/A
ippsMDCTFwd_32f	N/A
ippsMDCTFwd_32f_I	N/A
ippsMDCTFwd_AAC_32s	N/A
ippsMDCTFwd_AAC_32s_I	N/A
ippsMDCTFwd_MP3_32s	N/A
ippsMDCTInvFree_32f	N/A
ippsMDCTInvGetBufSize_32f	N/A
ippsMDCTInvGetSize_32f	N/A
ippsMDCTInvInitAlloc_32f	N/A
ippsMDCTInvInit_32f	N/A
ippsMDCTInvWindow_MP3_32s	N/A
ippsMDCTInv_32f	N/A
ippsMDCTInv_32f_I	N/A
ippsMDCTInv_AAC_32s16s	N/A
ippsMDCTInv_AAC_32s_I	N/A
ippsMDCTInv_MP3_32s	N/A
ippsMakeFloat_16s32f	N/A
ippsNoiselessDecode_AAC	N/A
ippsNoiselessDecoder_LC_AAC	N/A
ippsPackFrameHeader_MP3	N/A
ippsPackScaleFactors_MP3_8s1u	N/A

ippsPackSideInfo_MP3	N/A
ippsPow34_16s_Sfs	N/A
ippsPow34_32f	N/A
ippsPow34_32f16s	N/A
ippsPow43Scale_16s32s_Sf	N/A
ippsPow43_16s32f	N/A
ippsPredictCoef_SBR_C_32f_D2L	N/A
ippsPredictCoef_SBR_C_32fc_D2L	N/A
ippsPredictCoef_SBR_R_32f_D2L	N/A
ippsPredictOneCoef_SBRHQ_32sc_D2L	N/A
ippsPredictOneCoef_SBRLP_32s_D2L	N/A
ippsPsychoacousticModelTwo_MP3_16s	N/A
ippsQuantInv_AAC_32s_I	N/A
ippsQuantize_MP3_32s_I	N/A
ippsReQuantizeSfb_MP3_32s_I	N/A
ippsReQuantize_MP3_32s_I	N/A
ippsScale_32f_I	N/A
ippsSpread_16s_Sfs	N/A
ippsSynthPQMF_MP3_32s16s	N/A
ippsSynthesisDownFilterFree_SBRHQ_32sc32s	N/A
ippsSynthesisDownFilterFree_SBRLP_32s	N/A
ippsSynthesisDownFilterGetSize_SBRHQ_32sc32s	N/A
ippsSynthesisDownFilterGetSize_SBRLP_32s	N/A
ippsSynthesisDownFilterGetSize_SBR_CToR_32f	N/A
ippsSynthesisDownFilterGetSize_SBR_CToR_32fc32f	N/A
ippsSynthesisDownFilterGetSize_SBR_RToR_32f	N/A
ippsSynthesisDownFilterInitAlloc_SBRHQ_32sc32s	N/A

ippsSynthesisDownFilterInitAlloc_SBRLP_32s	N/A
ippsSynthesisDownFilterInit_SBRHQ_32sc32s	N/A
ippsSynthesisDownFilterInit_SBRLP_32s	N/A
ippsSynthesisDownFilterInit_SBR_CToR_32f	N/A
ippsSynthesisDownFilterInit_SBR_CToR_32fc32f	N/A
ippsSynthesisDownFilterInit_SBR_RToR_32f	N/A
ippsSynthesisDownFilter_SBRHQ_32sc32s	N/A
ippsSynthesisDownFilter_SBRLP_32s	N/A
ippsSynthesisDownFilter_SBR_CToR_32f_D2L	N/A
ippsSynthesisDownFilter_SBR_CToR_32fc32f_D2L	N/A
ippsSynthesisDownFilter_SBR_RToR_32f_D2L	N/A
ippsSynthesisFilterFree_DTS_32f	N/A
ippsSynthesisFilterFree_PQMF_MP3_32f	N/A
ippsSynthesisFilterFree_SBRHQ_32sc32s	N/A
ippsSynthesisFilterFree_SBRLP_32s	N/A
ippsSynthesisFilterGetSize_DTS_32f	N/A
ippsSynthesisFilterGetSize_PQMF_MP3_32f	N/A
ippsSynthesisFilterGetSize_SBRHQ_32sc32s	N/A
ippsSynthesisFilterGetSize_SBRLP_32s	N/A
ippsSynthesisFilterGetSize_SBR_CToR_32f	N/A
ippsSynthesisFilterGetSize_SBR_CToR_32fc32f	N/A
ippsSynthesisFilterGetSize_SBR_RToR_32f	N/A
ippsSynthesisFilterInitAlloc_DTS_32f	N/A
ippsSynthesisFilterInitAlloc_PQMF_MP3_32f	N/A
ippsSynthesisFilterInitAlloc_SBRHQ_32sc32s	N/A
ippsSynthesisFilterInitAlloc_SBRLP_32s	N/A
ippsSynthesisFilterInit_DTS_32f	N/A
ippsSynthesisFilterInit_PQMF_MP3_32f	N/A

ippsSynthesisFilterInit_SBRHQ_32sc32s	N/A
ippsSynthesisFilterInit_SBRLP_32s	N/A
ippsSynthesisFilterInit_SBR_CToR_32f	N/A
ippsSynthesisFilterInit_SBR_CToR_32fc32f	N/A
ippsSynthesisFilterInit_SBR_RToR_32f	N/A
ippsSynthesisFilter_DTS_32f	N/A
ippsSynthesisFilter_PQMF_MP3_32f	N/A
ippsSynthesisFilter_SBRHQ_32sc32s	N/A
ippsSynthesisFilter_SBRLP_32s	N/A
ippsSynthesisFilter_SBR_CToR_32f_D2L	N/A
ippsSynthesisFilter_SBR_CToR_32fc32f_D2L	N/A
ippsSynthesisFilter_SBR_RToR_32f_D2L	N/A
ippsUnpackADIFHeader_AAC	N/A
ippsUnpackADTSFrameHeader_AAC	N/A
ippsUnpackFrameHeader_MP3	N/A
ippsUnpackScaleFactors_MP3_1u8s	N/A
ippsUnpackSideInfo_MP3	N/A
ippsVLCCountBits_16s32s	N/A
ippsVLCCountEscBits_AAC_16s32s	N/A
ippsVLCCountEscBits_MP3_16s32s	N/A
ippsVLCDecodeBlock_1u16s	N/A
ippsVLCDecodeEscBlock_AAC_1u16s	N/A
ippsVLCDecodeEscBlock_MP3_1u16s	N/A
ippsVLCDecodeFree_32s	N/A
ippsVLCDecodeGetSize_32s	N/A
ippsVLCDecodeInitAlloc_32s	N/A
ippsVLCDecodeInit_32s	N/A
ippsVLCDecodeOne_1u16s	N/A
ippsVLCDecodeUTupleBlock_1u16s	N/A
ippsVLCDecodeUTupleEscBlock_AAC_1u16s	N/A
ippsVLCDecodeUTupleEscBlock_MP3_1u16s	N/A

ippsVLCDecodeUTupleFree_32s	N/A
ippsVLCDecodeUTupleGetSize_32s	N/A
ippsVLCDecodeUTupleInitAlloc_32s	N/A
ippsVLCDecodeUTupleInit_32s	N/A
ippsVLCDecodeUTupleOne_1u16s	N/A
ippsVLCEncodeBlock_16s1u	N/A
ippsVLCEncodeEscBlock_AAC_16s1u	N/A
ippsVLCEncodeEscBlock_MP3_16s1u	N/A
ippsVLCEncodeFree_32s	N/A
ippsVLCEncodeGetSize_32s	N/A
ippsVLCEncodeInitAlloc_32s	N/A
ippsVLCEncodeInit_32s	N/A
ippsVLCEncodeOne_16s1u	N/A
ippsVQCodeBookFree_32f	N/A
ippsVQCodeBookGetSize_32f	N/A
ippsVQCodeBookInitAlloc_32f	N/A
ippsVQCodeBookInit_32f	N/A
ippsVQIndexSelect_32f	N/A
ippsVQMainSelect_32f	N/A
ippsVQPreliminarySelect_32f	N/A
ippsVQReconstruction_32f	N/A

ippgen.h:

Removed from 9.0	Substitution or Workaround
ippgDCT4Free_32f	N/A
ippgDCT4Free_64f	N/A
ippgDCT4GetSize_32f	N/A
ippgDCT4GetSize_64f	N/A
ippgDCT4InitAlloc_32f	N/A
ippgDCT4InitAlloc_64f	N/A
ippgDCT4Init_32f	N/A

ippgDCT4Init_64f	N/A
ippgDCT4_32f	N/A
ippgDCT4_64f	N/A
ippgDFTFwd_CToC_10_32fc	N/A
ippgDFTFwd_CToC_10_64fc	N/A
ippgDFTFwd_CToC_11_32fc	N/A
ippgDFTFwd_CToC_11_64fc	N/A
ippgDFTFwd_CToC_12_32fc	N/A
ippgDFTFwd_CToC_12_64fc	N/A
ippgDFTFwd_CToC_13_32fc	N/A
ippgDFTFwd_CToC_13_64fc	N/A
ippgDFTFwd_CToC_14_32fc	N/A
ippgDFTFwd_CToC_14_64fc	N/A
ippgDFTFwd_CToC_15_32fc	N/A
ippgDFTFwd_CToC_15_64fc	N/A
ippgDFTFwd_CToC_16_32fc	N/A
ippgDFTFwd_CToC_16_64fc	N/A
ippgDFTFwd_CToC_17_32fc	N/A
ippgDFTFwd_CToC_17_64fc	N/A
ippgDFTFwd_CToC_18_32fc	N/A
ippgDFTFwd_CToC_18_64fc	N/A
ippgDFTFwd_CToC_19_32fc	N/A
ippgDFTFwd_CToC_19_64fc	N/A
ippgDFTFwd_CToC_20_32fc	N/A
ippgDFTFwd_CToC_20_64fc	N/A
ippgDFTFwd_CToC_21_32fc	N/A
ippgDFTFwd_CToC_21_64fc	N/A
ippgDFTFwd_CToC_22_32fc	N/A
ippgDFTFwd_CToC_22_64fc	N/A
ippgDFTFwd_CToC_23_32fc	N/A
ippgDFTFwd_CToC_23_64fc	N/A

ippgDFTFwd_CToC_24_32fc	N/A
ippgDFTFwd_CToC_24_64fc	N/A
ippgDFTFwd_CToC_25_32fc	N/A
ippgDFTFwd_CToC_25_64fc	N/A
ippgDFTFwd_CToC_26_32fc	N/A
ippgDFTFwd_CToC_26_64fc	N/A
ippgDFTFwd_CToC_27_32fc	N/A
ippgDFTFwd_CToC_27_64fc	N/A
ippgDFTFwd_CToC_28_32fc	N/A
ippgDFTFwd_CToC_28_64fc	N/A
ippgDFTFwd_CToC_29_32fc	N/A
ippgDFTFwd_CToC_29_64fc	N/A
ippgDFTFwd_CToC_2_32fc	N/A
ippgDFTFwd_CToC_2_64fc	N/A
ippgDFTFwd_CToC_30_32fc	N/A
ippgDFTFwd_CToC_30_64fc	N/A
ippgDFTFwd_CToC_31_32fc	N/A
ippgDFTFwd_CToC_31_64fc	N/A
ippgDFTFwd_CToC_32_32fc	N/A
ippgDFTFwd_CToC_32_64fc	N/A
ippgDFTFwd_CToC_32fc	N/A
ippgDFTFwd_CToC_33_32fc	N/A
ippgDFTFwd_CToC_33_64fc	N/A
ippgDFTFwd_CToC_34_32fc	N/A
ippgDFTFwd_CToC_34_64fc	N/A
ippgDFTFwd_CToC_35_32fc	N/A
ippgDFTFwd_CToC_35_64fc	N/A
ippgDFTFwd_CToC_36_32fc	N/A
ippgDFTFwd_CToC_36_64fc	N/A
ippgDFTFwd_CToC_37_32fc	N/A
ippgDFTFwd_CToC_37_64fc	N/A

ippgDFTFwd_CToC_38_32fc	N/A
ippgDFTFwd_CToC_38_64fc	N/A
ippgDFTFwd_CToC_39_32fc	N/A
ippgDFTFwd_CToC_39_64fc	N/A
ippgDFTFwd_CToC_3_32fc	N/A
ippgDFTFwd_CToC_3_64fc	N/A
ippgDFTFwd_CToC_40_32fc	N/A
ippgDFTFwd_CToC_40_64fc	N/A
ippgDFTFwd_CToC_41_32fc	N/A
ippgDFTFwd_CToC_41_64fc	N/A
ippgDFTFwd_CToC_42_32fc	N/A
ippgDFTFwd_CToC_42_64fc	N/A
ippgDFTFwd_CToC_43_32fc	N/A
ippgDFTFwd_CToC_43_64fc	N/A
ippgDFTFwd_CToC_44_32fc	N/A
ippgDFTFwd_CToC_44_64fc	N/A
ippgDFTFwd_CToC_45_32fc	N/A
ippgDFTFwd_CToC_45_64fc	N/A
ippgDFTFwd_CToC_46_32fc	N/A
ippgDFTFwd_CToC_46_64fc	N/A
ippgDFTFwd_CToC_47_32fc	N/A
ippgDFTFwd_CToC_47_64fc	N/A
ippgDFTFwd_CToC_48_32fc	N/A
ippgDFTFwd_CToC_48_64fc	N/A
ippgDFTFwd_CToC_49_32fc	N/A
ippgDFTFwd_CToC_49_64fc	N/A
ippgDFTFwd_CToC_4_32fc	N/A
ippgDFTFwd_CToC_4_64fc	N/A
ippgDFTFwd_CToC_50_32fc	N/A
ippgDFTFwd_CToC_50_64fc	N/A
ippgDFTFwd_CToC_51_32fc	N/A

ippgDFTFwd_CToC_51_64fc	N/A
ippgDFTFwd_CToC_52_32fc	N/A
ippgDFTFwd_CToC_52_64fc	N/A
ippgDFTFwd_CToC_53_32fc	N/A
ippgDFTFwd_CToC_53_64fc	N/A
ippgDFTFwd_CToC_54_32fc	N/A
ippgDFTFwd_CToC_54_64fc	N/A
ippgDFTFwd_CToC_55_32fc	N/A
ippgDFTFwd_CToC_55_64fc	N/A
ippgDFTFwd_CToC_56_32fc	N/A
ippgDFTFwd_CToC_56_64fc	N/A
ippgDFTFwd_CToC_57_32fc	N/A
ippgDFTFwd_CToC_57_64fc	N/A
ippgDFTFwd_CToC_58_32fc	N/A
ippgDFTFwd_CToC_58_64fc	N/A
ippgDFTFwd_CToC_59_32fc	N/A
ippgDFTFwd_CToC_59_64fc	N/A
ippgDFTFwd_CToC_5_32fc	N/A
ippgDFTFwd_CToC_5_64fc	N/A
ippgDFTFwd_CToC_60_32fc	N/A
ippgDFTFwd_CToC_60_64fc	N/A
ippgDFTFwd_CToC_61_32fc	N/A
ippgDFTFwd_CToC_61_64fc	N/A
ippgDFTFwd_CToC_62_32fc	N/A
ippgDFTFwd_CToC_62_64fc	N/A
ippgDFTFwd_CToC_63_32fc	N/A
ippgDFTFwd_CToC_63_64fc	N/A
ippgDFTFwd_CToC_64_32fc	N/A
ippgDFTFwd_CToC_64_64fc	N/A
ippgDFTFwd_CToC_64fc	N/A
ippgDFTFwd_CToC_6_32fc	N/A

ippgDFTFwd_CToC_6_64fc	N/A
ippgDFTFwd_CToC_7_32fc	N/A
ippgDFTFwd_CToC_7_64fc	N/A
ippgDFTFwd_CToC_8_32fc	N/A
ippgDFTFwd_CToC_8_64fc	N/A
ippgDFTFwd_CToC_9_32fc	N/A
ippgDFTFwd_CToC_9_64fc	N/A
ippgDFTFwd_RToCCS_10_32f	N/A
ippgDFTFwd_RToCCS_10_64f	N/A
ippgDFTFwd_RToCCS_11_32f	N/A
ippgDFTFwd_RToCCS_11_64f	N/A
ippgDFTFwd_RToCCS_12_32f	N/A
ippgDFTFwd_RToCCS_12_64f	N/A
ippgDFTFwd_RToCCS_13_32f	N/A
ippgDFTFwd_RToCCS_13_64f	N/A
ippgDFTFwd_RToCCS_14_32f	N/A
ippgDFTFwd_RToCCS_14_64f	N/A
ippgDFTFwd_RToCCS_15_32f	N/A
ippgDFTFwd_RToCCS_15_64f	N/A
ippgDFTFwd_RToCCS_16_32f	N/A
ippgDFTFwd_RToCCS_16_64f	N/A
ippgDFTFwd_RToCCS_17_32f	N/A
ippgDFTFwd_RToCCS_17_64f	N/A
ippgDFTFwd_RToCCS_18_32f	N/A
ippgDFTFwd_RToCCS_18_64f	N/A
ippgDFTFwd_RToCCS_19_32f	N/A
ippgDFTFwd_RToCCS_19_64f	N/A
ippgDFTFwd_RToCCS_20_32f	N/A
ippgDFTFwd_RToCCS_20_64f	N/A
ippgDFTFwd_RToCCS_21_32f	N/A
ippgDFTFwd_RToCCS_21_64f	N/A

ippgDFTFwd_RToCCS_22_32f	N/A
ippgDFTFwd_RToCCS_22_64f	N/A
ippgDFTFwd_RToCCS_23_32f	N/A
ippgDFTFwd_RToCCS_23_64f	N/A
ippgDFTFwd_RToCCS_24_32f	N/A
ippgDFTFwd_RToCCS_24_64f	N/A
ippgDFTFwd_RToCCS_25_32f	N/A
ippgDFTFwd_RToCCS_25_64f	N/A
ippgDFTFwd_RToCCS_26_32f	N/A
ippgDFTFwd_RToCCS_26_64f	N/A
ippgDFTFwd_RToCCS_27_32f	N/A
ippgDFTFwd_RToCCS_27_64f	N/A
ippgDFTFwd_RToCCS_28_32f	N/A
ippgDFTFwd_RToCCS_28_64f	N/A
ippgDFTFwd_RToCCS_29_32f	N/A
ippgDFTFwd_RToCCS_29_64f	N/A
ippgDFTFwd_RToCCS_2_32f	N/A
ippgDFTFwd_RToCCS_2_64f	N/A
ippgDFTFwd_RToCCS_30_32f	N/A
ippgDFTFwd_RToCCS_30_64f	N/A
ippgDFTFwd_RToCCS_31_32f	N/A
ippgDFTFwd_RToCCS_31_64f	N/A
ippgDFTFwd_RToCCS_32_32f	N/A
ippgDFTFwd_RToCCS_32_64f	N/A
ippgDFTFwd_RToCCS_32f	N/A
ippgDFTFwd_RToCCS_33_32f	N/A
ippgDFTFwd_RToCCS_33_64f	N/A
ippgDFTFwd_RToCCS_34_32f	N/A
ippgDFTFwd_RToCCS_34_64f	N/A
ippgDFTFwd_RToCCS_35_32f	N/A
ippgDFTFwd_RToCCS_35_64f	N/A

ippgDFTFwd_RToCCS_36_32f	N/A
ippgDFTFwd_RToCCS_36_64f	N/A
ippgDFTFwd_RToCCS_37_32f	N/A
ippgDFTFwd_RToCCS_37_64f	N/A
ippgDFTFwd_RToCCS_38_32f	N/A
ippgDFTFwd_RToCCS_38_64f	N/A
ippgDFTFwd_RToCCS_39_32f	N/A
ippgDFTFwd_RToCCS_39_64f	N/A
ippgDFTFwd_RToCCS_3_32f	N/A
ippgDFTFwd_RToCCS_3_64f	N/A
ippgDFTFwd_RToCCS_40_32f	N/A
ippgDFTFwd_RToCCS_40_64f	N/A
ippgDFTFwd_RToCCS_41_32f	N/A
ippgDFTFwd_RToCCS_41_64f	N/A
ippgDFTFwd_RToCCS_42_32f	N/A
ippgDFTFwd_RToCCS_42_64f	N/A
ippgDFTFwd_RToCCS_43_32f	N/A
ippgDFTFwd_RToCCS_43_64f	N/A
ippgDFTFwd_RToCCS_44_32f	N/A
ippgDFTFwd_RToCCS_44_64f	N/A
ippgDFTFwd_RToCCS_45_32f	N/A
ippgDFTFwd_RToCCS_45_64f	N/A
ippgDFTFwd_RToCCS_46_32f	N/A
ippgDFTFwd_RToCCS_46_64f	N/A
ippgDFTFwd_RToCCS_47_32f	N/A
ippgDFTFwd_RToCCS_47_64f	N/A
ippgDFTFwd_RToCCS_48_32f	N/A
ippgDFTFwd_RToCCS_48_64f	N/A
ippgDFTFwd_RToCCS_49_32f	N/A
ippgDFTFwd_RToCCS_49_64f	N/A
ippgDFTFwd_RToCCS_4_32f	N/A

ippgDFTFwd_RToCCS_4_64f	N/A
ippgDFTFwd_RToCCS_50_32f	N/A
ippgDFTFwd_RToCCS_50_64f	N/A
ippgDFTFwd_RToCCS_51_32f	N/A
ippgDFTFwd_RToCCS_51_64f	N/A
ippgDFTFwd_RToCCS_52_32f	N/A
ippgDFTFwd_RToCCS_52_64f	N/A
ippgDFTFwd_RToCCS_53_32f	N/A
ippgDFTFwd_RToCCS_53_64f	N/A
ippgDFTFwd_RToCCS_54_32f	N/A
ippgDFTFwd_RToCCS_54_64f	N/A
ippgDFTFwd_RToCCS_55_32f	N/A
ippgDFTFwd_RToCCS_55_64f	N/A
ippgDFTFwd_RToCCS_56_32f	N/A
ippgDFTFwd_RToCCS_56_64f	N/A
ippgDFTFwd_RToCCS_57_32f	N/A
ippgDFTFwd_RToCCS_57_64f	N/A
ippgDFTFwd_RToCCS_58_32f	N/A
ippgDFTFwd_RToCCS_58_64f	N/A
ippgDFTFwd_RToCCS_59_32f	N/A
ippgDFTFwd_RToCCS_59_64f	N/A
ippgDFTFwd_RToCCS_5_32f	N/A
ippgDFTFwd_RToCCS_5_64f	N/A
ippgDFTFwd_RToCCS_60_32f	N/A
ippgDFTFwd_RToCCS_60_64f	N/A
ippgDFTFwd_RToCCS_61_32f	N/A
ippgDFTFwd_RToCCS_61_64f	N/A
ippgDFTFwd_RToCCS_62_32f	N/A
ippgDFTFwd_RToCCS_62_64f	N/A
ippgDFTFwd_RToCCS_63_32f	N/A
ippgDFTFwd_RToCCS_63_64f	N/A

ippgDFTFwd_RToCCS_64_32f	N/A
ippgDFTFwd_RToCCS_64_64f	N/A
ippgDFTFwd_RToCCS_64f	N/A
ippgDFTFwd_RToCCS_6_32f	N/A
ippgDFTFwd_RToCCS_6_64f	N/A
ippgDFTFwd_RToCCS_7_32f	N/A
ippgDFTFwd_RToCCS_7_64f	N/A
ippgDFTFwd_RToCCS_8_32f	N/A
ippgDFTFwd_RToCCS_8_64f	N/A
ippgDFTFwd_RToCCS_9_32f	N/A
ippgDFTFwd_RToCCS_9_64f	N/A
ippgDFTFwd_RToPack_10_32f	N/A
ippgDFTFwd_RToPack_10_64f	N/A
ippgDFTFwd_RToPack_11_32f	N/A
ippgDFTFwd_RToPack_11_64f	N/A
ippgDFTFwd_RToPack_12_32f	N/A
ippgDFTFwd_RToPack_12_64f	N/A
ippgDFTFwd_RToPack_13_32f	N/A
ippgDFTFwd_RToPack_13_64f	N/A
ippgDFTFwd_RToPack_14_32f	N/A
ippgDFTFwd_RToPack_14_64f	N/A
ippgDFTFwd_RToPack_15_32f	N/A
ippgDFTFwd_RToPack_15_64f	N/A
ippgDFTFwd_RToPack_16_32f	N/A
ippgDFTFwd_RToPack_16_64f	N/A
ippgDFTFwd_RToPack_17_32f	N/A
ippgDFTFwd_RToPack_17_64f	N/A
ippgDFTFwd_RToPack_18_32f	N/A
ippgDFTFwd_RToPack_18_64f	N/A
ippgDFTFwd_RToPack_19_32f	N/A
ippgDFTFwd_RToPack_19_64f	N/A

ippgDFTFwd_RToPack_20_32f	N/A
ippgDFTFwd_RToPack_20_64f	N/A
ippgDFTFwd_RToPack_21_32f	N/A
ippgDFTFwd_RToPack_21_64f	N/A
ippgDFTFwd_RToPack_22_32f	N/A
ippgDFTFwd_RToPack_22_64f	N/A
ippgDFTFwd_RToPack_23_32f	N/A
ippgDFTFwd_RToPack_23_64f	N/A
ippgDFTFwd_RToPack_24_32f	N/A
ippgDFTFwd_RToPack_24_64f	N/A
ippgDFTFwd_RToPack_25_32f	N/A
ippgDFTFwd_RToPack_25_64f	N/A
ippgDFTFwd_RToPack_26_32f	N/A
ippgDFTFwd_RToPack_26_64f	N/A
ippgDFTFwd_RToPack_27_32f	N/A
ippgDFTFwd_RToPack_27_64f	N/A
ippgDFTFwd_RToPack_28_32f	N/A
ippgDFTFwd_RToPack_28_64f	N/A
ippgDFTFwd_RToPack_29_32f	N/A
ippgDFTFwd_RToPack_29_64f	N/A
ippgDFTFwd_RToPack_2_32f	N/A
ippgDFTFwd_RToPack_2_64f	N/A
ippgDFTFwd_RToPack_30_32f	N/A
ippgDFTFwd_RToPack_30_64f	N/A
ippgDFTFwd_RToPack_31_32f	N/A
ippgDFTFwd_RToPack_31_64f	N/A
ippgDFTFwd_RToPack_32_32f	N/A
ippgDFTFwd_RToPack_32_64f	N/A
ippgDFTFwd_RToPack_32f	N/A
ippgDFTFwd_RToPack_33_32f	N/A
ippgDFTFwd_RToPack_33_64f	N/A

ippgDFTFwd_RToPack_34_32f	N/A
ippgDFTFwd_RToPack_34_64f	N/A
ippgDFTFwd_RToPack_35_32f	N/A
ippgDFTFwd_RToPack_35_64f	N/A
ippgDFTFwd_RToPack_36_32f	N/A
ippgDFTFwd_RToPack_36_64f	N/A
ippgDFTFwd_RToPack_37_32f	N/A
ippgDFTFwd_RToPack_37_64f	N/A
ippgDFTFwd_RToPack_38_32f	N/A
ippgDFTFwd_RToPack_38_64f	N/A
ippgDFTFwd_RToPack_39_32f	N/A
ippgDFTFwd_RToPack_39_64f	N/A
ippgDFTFwd_RToPack_3_32f	N/A
ippgDFTFwd_RToPack_3_64f	N/A
ippgDFTFwd_RToPack_40_32f	N/A
ippgDFTFwd_RToPack_40_64f	N/A
ippgDFTFwd_RToPack_41_32f	N/A
ippgDFTFwd_RToPack_41_64f	N/A
ippgDFTFwd_RToPack_42_32f	N/A
ippgDFTFwd_RToPack_42_64f	N/A
ippgDFTFwd_RToPack_43_32f	N/A
ippgDFTFwd_RToPack_43_64f	N/A
ippgDFTFwd_RToPack_44_32f	N/A
ippgDFTFwd_RToPack_44_64f	N/A
ippgDFTFwd_RToPack_45_32f	N/A
ippgDFTFwd_RToPack_45_64f	N/A
ippgDFTFwd_RToPack_46_32f	N/A
ippgDFTFwd_RToPack_46_64f	N/A
ippgDFTFwd_RToPack_47_32f	N/A
ippgDFTFwd_RToPack_47_64f	N/A
ippgDFTFwd_RToPack_48_32f	N/A

ippgDFTFwd_RToPack_48_64f	N/A
ippgDFTFwd_RToPack_49_32f	N/A
ippgDFTFwd_RToPack_49_64f	N/A
ippgDFTFwd_RToPack_4_32f	N/A
ippgDFTFwd_RToPack_4_64f	N/A
ippgDFTFwd_RToPack_50_32f	N/A
ippgDFTFwd_RToPack_50_64f	N/A
ippgDFTFwd_RToPack_51_32f	N/A
ippgDFTFwd_RToPack_51_64f	N/A
ippgDFTFwd_RToPack_52_32f	N/A
ippgDFTFwd_RToPack_52_64f	N/A
ippgDFTFwd_RToPack_53_32f	N/A
ippgDFTFwd_RToPack_53_64f	N/A
ippgDFTFwd_RToPack_54_32f	N/A
ippgDFTFwd_RToPack_54_64f	N/A
ippgDFTFwd_RToPack_55_32f	N/A
ippgDFTFwd_RToPack_55_64f	N/A
ippgDFTFwd_RToPack_56_32f	N/A
ippgDFTFwd_RToPack_56_64f	N/A
ippgDFTFwd_RToPack_57_32f	N/A
ippgDFTFwd_RToPack_57_64f	N/A
ippgDFTFwd_RToPack_58_32f	N/A
ippgDFTFwd_RToPack_58_64f	N/A
ippgDFTFwd_RToPack_59_32f	N/A
ippgDFTFwd_RToPack_59_64f	N/A
ippgDFTFwd_RToPack_5_32f	N/A
ippgDFTFwd_RToPack_5_64f	N/A
ippgDFTFwd_RToPack_60_32f	N/A
ippgDFTFwd_RToPack_60_64f	N/A
ippgDFTFwd_RToPack_61_32f	N/A
ippgDFTFwd_RToPack_61_64f	N/A

ippgDFTFwd_RToPack_62_32f	N/A
ippgDFTFwd_RToPack_62_64f	N/A
ippgDFTFwd_RToPack_63_32f	N/A
ippgDFTFwd_RToPack_63_64f	N/A
ippgDFTFwd_RToPack_64_32f	N/A
ippgDFTFwd_RToPack_64_64f	N/A
ippgDFTFwd_RToPack_64f	N/A
ippgDFTFwd_RToPack_6_32f	N/A
ippgDFTFwd_RToPack_6_64f	N/A
ippgDFTFwd_RToPack_7_32f	N/A
ippgDFTFwd_RToPack_7_64f	N/A
ippgDFTFwd_RToPack_8_32f	N/A
ippgDFTFwd_RToPack_8_64f	N/A
ippgDFTFwd_RToPack_9_32f	N/A
ippgDFTFwd_RToPack_9_64f	N/A
ippgDFTFwd_RToPerm_10_32f	N/A
ippgDFTFwd_RToPerm_10_64f	N/A
ippgDFTFwd_RToPerm_11_32f	N/A
ippgDFTFwd_RToPerm_11_64f	N/A
ippgDFTFwd_RToPerm_12_32f	N/A
ippgDFTFwd_RToPerm_12_64f	N/A
ippgDFTFwd_RToPerm_13_32f	N/A
ippgDFTFwd_RToPerm_13_64f	N/A
ippgDFTFwd_RToPerm_14_32f	N/A
ippgDFTFwd_RToPerm_14_64f	N/A
ippgDFTFwd_RToPerm_15_32f	N/A
ippgDFTFwd_RToPerm_15_64f	N/A
ippgDFTFwd_RToPerm_16_32f	N/A
ippgDFTFwd_RToPerm_16_64f	N/A
ippgDFTFwd_RToPerm_17_32f	N/A
ippgDFTFwd_RToPerm_17_64f	N/A

ippgDFTFwd_RToPerm_18_32f	N/A
ippgDFTFwd_RToPerm_18_64f	N/A
ippgDFTFwd_RToPerm_19_32f	N/A
ippgDFTFwd_RToPerm_19_64f	N/A
ippgDFTFwd_RToPerm_20_32f	N/A
ippgDFTFwd_RToPerm_20_64f	N/A
ippgDFTFwd_RToPerm_21_32f	N/A
ippgDFTFwd_RToPerm_21_64f	N/A
ippgDFTFwd_RToPerm_22_32f	N/A
ippgDFTFwd_RToPerm_22_64f	N/A
ippgDFTFwd_RToPerm_23_32f	N/A
ippgDFTFwd_RToPerm_23_64f	N/A
ippgDFTFwd_RToPerm_24_32f	N/A
ippgDFTFwd_RToPerm_24_64f	N/A
ippgDFTFwd_RToPerm_25_32f	N/A
ippgDFTFwd_RToPerm_25_64f	N/A
ippgDFTFwd_RToPerm_26_32f	N/A
ippgDFTFwd_RToPerm_26_64f	N/A
ippgDFTFwd_RToPerm_27_32f	N/A
ippgDFTFwd_RToPerm_27_64f	N/A
ippgDFTFwd_RToPerm_28_32f	N/A
ippgDFTFwd_RToPerm_28_64f	N/A
ippgDFTFwd_RToPerm_29_32f	N/A
ippgDFTFwd_RToPerm_29_64f	N/A
ippgDFTFwd_RToPerm_2_32f	N/A
ippgDFTFwd_RToPerm_2_64f	N/A
ippgDFTFwd_RToPerm_30_32f	N/A
ippgDFTFwd_RToPerm_30_64f	N/A
ippgDFTFwd_RToPerm_31_32f	N/A
ippgDFTFwd_RToPerm_31_64f	N/A
ippgDFTFwd_RToPerm_32_32f	N/A

ippgDFTFwd_RToPerm_32_64f	N/A
ippgDFTFwd_RToPerm_32f	N/A
ippgDFTFwd_RToPerm_33_32f	N/A
ippgDFTFwd_RToPerm_33_64f	N/A
ippgDFTFwd_RToPerm_34_32f	N/A
ippgDFTFwd_RToPerm_34_64f	N/A
ippgDFTFwd_RToPerm_35_32f	N/A
ippgDFTFwd_RToPerm_35_64f	N/A
ippgDFTFwd_RToPerm_36_32f	N/A
ippgDFTFwd_RToPerm_36_64f	N/A
ippgDFTFwd_RToPerm_37_32f	N/A
ippgDFTFwd_RToPerm_37_64f	N/A
ippgDFTFwd_RToPerm_38_32f	N/A
ippgDFTFwd_RToPerm_38_64f	N/A
ippgDFTFwd_RToPerm_39_32f	N/A
ippgDFTFwd_RToPerm_39_64f	N/A
ippgDFTFwd_RToPerm_3_32f	N/A
ippgDFTFwd_RToPerm_3_64f	N/A
ippgDFTFwd_RToPerm_40_32f	N/A
ippgDFTFwd_RToPerm_40_64f	N/A
ippgDFTFwd_RToPerm_41_32f	N/A
ippgDFTFwd_RToPerm_41_64f	N/A
ippgDFTFwd_RToPerm_42_32f	N/A
ippgDFTFwd_RToPerm_42_64f	N/A
ippgDFTFwd_RToPerm_43_32f	N/A
ippgDFTFwd_RToPerm_43_64f	N/A
ippgDFTFwd_RToPerm_44_32f	N/A
ippgDFTFwd_RToPerm_44_64f	N/A
ippgDFTFwd_RToPerm_45_32f	N/A
ippgDFTFwd_RToPerm_45_64f	N/A
ippgDFTFwd_RToPerm_46_32f	N/A

ippgDFTFwd_RToPerm_46_64f	N/A
ippgDFTFwd_RToPerm_47_32f	N/A
ippgDFTFwd_RToPerm_47_64f	N/A
ippgDFTFwd_RToPerm_48_32f	N/A
ippgDFTFwd_RToPerm_48_64f	N/A
ippgDFTFwd_RToPerm_49_32f	N/A
ippgDFTFwd_RToPerm_49_64f	N/A
ippgDFTFwd_RToPerm_4_32f	N/A
ippgDFTFwd_RToPerm_4_64f	N/A
ippgDFTFwd_RToPerm_50_32f	N/A
ippgDFTFwd_RToPerm_50_64f	N/A
ippgDFTFwd_RToPerm_51_32f	N/A
ippgDFTFwd_RToPerm_51_64f	N/A
ippgDFTFwd_RToPerm_52_32f	N/A
ippgDFTFwd_RToPerm_52_64f	N/A
ippgDFTFwd_RToPerm_53_32f	N/A
ippgDFTFwd_RToPerm_53_64f	N/A
ippgDFTFwd_RToPerm_54_32f	N/A
ippgDFTFwd_RToPerm_54_64f	N/A
ippgDFTFwd_RToPerm_55_32f	N/A
ippgDFTFwd_RToPerm_55_64f	N/A
ippgDFTFwd_RToPerm_56_32f	N/A
ippgDFTFwd_RToPerm_56_64f	N/A
ippgDFTFwd_RToPerm_57_32f	N/A
ippgDFTFwd_RToPerm_57_64f	N/A
ippgDFTFwd_RToPerm_58_32f	N/A
ippgDFTFwd_RToPerm_58_64f	N/A
ippgDFTFwd_RToPerm_59_32f	N/A
ippgDFTFwd_RToPerm_59_64f	N/A
ippgDFTFwd_RToPerm_5_32f	N/A
ippgDFTFwd_RToPerm_5_64f	N/A

ippgDFTFwd_RToPerm_60_32f	N/A
ippgDFTFwd_RToPerm_60_64f	N/A
ippgDFTFwd_RToPerm_61_32f	N/A
ippgDFTFwd_RToPerm_61_64f	N/A
ippgDFTFwd_RToPerm_62_32f	N/A
ippgDFTFwd_RToPerm_62_64f	N/A
ippgDFTFwd_RToPerm_63_32f	N/A
ippgDFTFwd_RToPerm_63_64f	N/A
ippgDFTFwd_RToPerm_64_32f	N/A
ippgDFTFwd_RToPerm_64_64f	N/A
ippgDFTFwd_RToPerm_64f	N/A
ippgDFTFwd_RToPerm_6_32f	N/A
ippgDFTFwd_RToPerm_6_64f	N/A
ippgDFTFwd_RToPerm_7_32f	N/A
ippgDFTFwd_RToPerm_7_64f	N/A
ippgDFTFwd_RToPerm_8_32f	N/A
ippgDFTFwd_RToPerm_8_64f	N/A
ippgDFTFwd_RToPerm_9_32f	N/A
ippgDFTFwd_RToPerm_9_64f	N/A
ippgDFTInv_CCSToR_10_32f	N/A
ippgDFTInv_CCSToR_10_64f	N/A
ippgDFTInv_CCSToR_11_32f	N/A
ippgDFTInv_CCSToR_11_64f	N/A
ippgDFTInv_CCSToR_12_32f	N/A
ippgDFTInv_CCSToR_12_64f	N/A
ippgDFTInv_CCSToR_13_32f	N/A
ippgDFTInv_CCSToR_13_64f	N/A
ippgDFTInv_CCSToR_14_32f	N/A
ippgDFTInv_CCSToR_14_64f	N/A
ippgDFTInv_CCSToR_15_32f	N/A
ippgDFTInv_CCSToR_15_64f	N/A

ippgDFTInv_CCSToR_16_32f	N/A
ippgDFTInv_CCSToR_16_64f	N/A
ippgDFTInv_CCSToR_17_32f	N/A
ippgDFTInv_CCSToR_17_64f	N/A
ippgDFTInv_CCSToR_18_32f	N/A
ippgDFTInv_CCSToR_18_64f	N/A
ippgDFTInv_CCSToR_19_32f	N/A
ippgDFTInv_CCSToR_19_64f	N/A
ippgDFTInv_CCSToR_20_32f	N/A
ippgDFTInv_CCSToR_20_64f	N/A
ippgDFTInv_CCSToR_21_32f	N/A
ippgDFTInv_CCSToR_21_64f	N/A
ippgDFTInv_CCSToR_22_32f	N/A
ippgDFTInv_CCSToR_22_64f	N/A
ippgDFTInv_CCSToR_23_32f	N/A
ippgDFTInv_CCSToR_23_64f	N/A
ippgDFTInv_CCSToR_24_32f	N/A
ippgDFTInv_CCSToR_24_64f	N/A
ippgDFTInv_CCSToR_25_32f	N/A
ippgDFTInv_CCSToR_25_64f	N/A
ippgDFTInv_CCSToR_26_32f	N/A
ippgDFTInv_CCSToR_26_64f	N/A
ippgDFTInv_CCSToR_27_32f	N/A
ippgDFTInv_CCSToR_27_64f	N/A
ippgDFTInv_CCSToR_28_32f	N/A
ippgDFTInv_CCSToR_28_64f	N/A
ippgDFTInv_CCSToR_29_32f	N/A
ippgDFTInv_CCSToR_29_64f	N/A
ippgDFTInv_CCSToR_2_32f	N/A
ippgDFTInv_CCSToR_2_64f	N/A
ippgDFTInv_CCSToR_30_32f	N/A

ippgDFTInv_CCSToR_30_64f	N/A
ippgDFTInv_CCSToR_31_32f	N/A
ippgDFTInv_CCSToR_31_64f	N/A
ippgDFTInv_CCSToR_32_32f	N/A
ippgDFTInv_CCSToR_32_64f	N/A
ippgDFTInv_CCSToR_32f	N/A
ippgDFTInv_CCSToR_33_32f	N/A
ippgDFTInv_CCSToR_33_64f	N/A
ippgDFTInv_CCSToR_34_32f	N/A
ippgDFTInv_CCSToR_34_64f	N/A
ippgDFTInv_CCSToR_35_32f	N/A
ippgDFTInv_CCSToR_35_64f	N/A
ippgDFTInv_CCSToR_36_32f	N/A
ippgDFTInv_CCSToR_36_64f	N/A
ippgDFTInv_CCSToR_37_32f	N/A
ippgDFTInv_CCSToR_37_64f	N/A
ippgDFTInv_CCSToR_38_32f	N/A
ippgDFTInv_CCSToR_38_64f	N/A
ippgDFTInv_CCSToR_39_32f	N/A
ippgDFTInv_CCSToR_39_64f	N/A
ippgDFTInv_CCSToR_3_32f	N/A
ippgDFTInv_CCSToR_3_64f	N/A
ippgDFTInv_CCSToR_40_32f	N/A
ippgDFTInv_CCSToR_40_64f	N/A
ippgDFTInv_CCSToR_41_32f	N/A
ippgDFTInv_CCSToR_41_64f	N/A
ippgDFTInv_CCSToR_42_32f	N/A
ippgDFTInv_CCSToR_42_64f	N/A
ippgDFTInv_CCSToR_43_32f	N/A
ippgDFTInv_CCSToR_43_64f	N/A
ippgDFTInv_CCSToR_44_32f	N/A

ippgDFTInv_CCSToR_44_64f	N/A
ippgDFTInv_CCSToR_45_32f	N/A
ippgDFTInv_CCSToR_45_64f	N/A
ippgDFTInv_CCSToR_46_32f	N/A
ippgDFTInv_CCSToR_46_64f	N/A
ippgDFTInv_CCSToR_47_32f	N/A
ippgDFTInv_CCSToR_47_64f	N/A
ippgDFTInv_CCSToR_48_32f	N/A
ippgDFTInv_CCSToR_48_64f	N/A
ippgDFTInv_CCSToR_49_32f	N/A
ippgDFTInv_CCSToR_49_64f	N/A
ippgDFTInv_CCSToR_4_32f	N/A
ippgDFTInv_CCSToR_4_64f	N/A
ippgDFTInv_CCSToR_50_32f	N/A
ippgDFTInv_CCSToR_50_64f	N/A
ippgDFTInv_CCSToR_51_32f	N/A
ippgDFTInv_CCSToR_51_64f	N/A
ippgDFTInv_CCSToR_52_32f	N/A
ippgDFTInv_CCSToR_52_64f	N/A
ippgDFTInv_CCSToR_53_32f	N/A
ippgDFTInv_CCSToR_53_64f	N/A
ippgDFTInv_CCSToR_54_32f	N/A
ippgDFTInv_CCSToR_54_64f	N/A
ippgDFTInv_CCSToR_55_32f	N/A
ippgDFTInv_CCSToR_55_64f	N/A
ippgDFTInv_CCSToR_56_32f	N/A
ippgDFTInv_CCSToR_56_64f	N/A
ippgDFTInv_CCSToR_57_32f	N/A
ippgDFTInv_CCSToR_57_64f	N/A
ippgDFTInv_CCSToR_58_32f	N/A
ippgDFTInv_CCSToR_58_64f	N/A

ippgDFTInv_CCSToR_59_32f	N/A
ippgDFTInv_CCSToR_59_64f	N/A
ippgDFTInv_CCSToR_5_32f	N/A
ippgDFTInv_CCSToR_5_64f	N/A
ippgDFTInv_CCSToR_60_32f	N/A
ippgDFTInv_CCSToR_60_64f	N/A
ippgDFTInv_CCSToR_61_32f	N/A
ippgDFTInv_CCSToR_61_64f	N/A
ippgDFTInv_CCSToR_62_32f	N/A
ippgDFTInv_CCSToR_62_64f	N/A
ippgDFTInv_CCSToR_63_32f	N/A
ippgDFTInv_CCSToR_63_64f	N/A
ippgDFTInv_CCSToR_64_32f	N/A
ippgDFTInv_CCSToR_64_64f	N/A
ippgDFTInv_CCSToR_64f	N/A
ippgDFTInv_CCSToR_6_32f	N/A
ippgDFTInv_CCSToR_6_64f	N/A
ippgDFTInv_CCSToR_7_32f	N/A
ippgDFTInv_CCSToR_7_64f	N/A
ippgDFTInv_CCSToR_8_32f	N/A
ippgDFTInv_CCSToR_8_64f	N/A
ippgDFTInv_CCSToR_9_32f	N/A
ippgDFTInv_CCSToR_9_64f	N/A
ippgDFTInv_CToC_10_32fc	N/A
ippgDFTInv_CToC_10_64fc	N/A
ippgDFTInv_CToC_11_32fc	N/A
ippgDFTInv_CToC_11_64fc	N/A
ippgDFTInv_CToC_12_32fc	N/A
ippgDFTInv_CToC_12_64fc	N/A
ippgDFTInv_CToC_13_32fc	N/A
ippgDFTInv_CToC_13_64fc	N/A

ippgDFTInv_CToC_14_32fc	N/A
ippgDFTInv_CToC_14_64fc	N/A
ippgDFTInv_CToC_15_32fc	N/A
ippgDFTInv_CToC_15_64fc	N/A
ippgDFTInv_CToC_16_32fc	N/A
ippgDFTInv_CToC_16_64fc	N/A
ippgDFTInv_CToC_17_32fc	N/A
ippgDFTInv_CToC_17_64fc	N/A
ippgDFTInv_CToC_18_32fc	N/A
ippgDFTInv_CToC_18_64fc	N/A
ippgDFTInv_CToC_19_32fc	N/A
ippgDFTInv_CToC_19_64fc	N/A
ippgDFTInv_CToC_20_32fc	N/A
ippgDFTInv_CToC_20_64fc	N/A
ippgDFTInv_CToC_21_32fc	N/A
ippgDFTInv_CToC_21_64fc	N/A
ippgDFTInv_CToC_22_32fc	N/A
ippgDFTInv_CToC_22_64fc	N/A
ippgDFTInv_CToC_23_32fc	N/A
ippgDFTInv_CToC_23_64fc	N/A
ippgDFTInv_CToC_24_32fc	N/A
ippgDFTInv_CToC_24_64fc	N/A
ippgDFTInv_CToC_25_32fc	N/A
ippgDFTInv_CToC_25_64fc	N/A
ippgDFTInv_CToC_26_32fc	N/A
ippgDFTInv_CToC_26_64fc	N/A
ippgDFTInv_CToC_27_32fc	N/A
ippgDFTInv_CToC_27_64fc	N/A
ippgDFTInv_CToC_28_32fc	N/A
ippgDFTInv_CToC_28_64fc	N/A
ippgDFTInv_CToC_29_32fc	N/A

ippgDFTInv_CToC_29_64fc	N/A
ippgDFTInv_CToC_2_32fc	N/A
ippgDFTInv_CToC_2_64fc	N/A
ippgDFTInv_CToC_30_32fc	N/A
ippgDFTInv_CToC_30_64fc	N/A
ippgDFTInv_CToC_31_32fc	N/A
ippgDFTInv_CToC_31_64fc	N/A
ippgDFTInv_CToC_32_32fc	N/A
ippgDFTInv_CToC_32_64fc	N/A
ippgDFTInv_CToC_32fc	N/A
ippgDFTInv_CToC_33_32fc	N/A
ippgDFTInv_CToC_33_64fc	N/A
ippgDFTInv_CToC_34_32fc	N/A
ippgDFTInv_CToC_34_64fc	N/A
ippgDFTInv_CToC_35_32fc	N/A
ippgDFTInv_CToC_35_64fc	N/A
ippgDFTInv_CToC_36_32fc	N/A
ippgDFTInv_CToC_36_64fc	N/A
ippgDFTInv_CToC_37_32fc	N/A
ippgDFTInv_CToC_37_64fc	N/A
ippgDFTInv_CToC_38_32fc	N/A
ippgDFTInv_CToC_38_64fc	N/A
ippgDFTInv_CToC_39_32fc	N/A
ippgDFTInv_CToC_39_64fc	N/A
ippgDFTInv_CToC_3_32fc	N/A
ippgDFTInv_CToC_3_64fc	N/A
ippgDFTInv_CToC_40_32fc	N/A
ippgDFTInv_CToC_40_64fc	N/A
ippgDFTInv_CToC_41_32fc	N/A
ippgDFTInv_CToC_41_64fc	N/A
ippgDFTInv_CToC_42_32fc	N/A

ippgDFTInv_CToC_42_64fc	N/A
ippgDFTInv_CToC_43_32fc	N/A
ippgDFTInv_CToC_43_64fc	N/A
ippgDFTInv_CToC_44_32fc	N/A
ippgDFTInv_CToC_44_64fc	N/A
ippgDFTInv_CToC_45_32fc	N/A
ippgDFTInv_CToC_45_64fc	N/A
ippgDFTInv_CToC_46_32fc	N/A
ippgDFTInv_CToC_46_64fc	N/A
ippgDFTInv_CToC_47_32fc	N/A
ippgDFTInv_CToC_47_64fc	N/A
ippgDFTInv_CToC_48_32fc	N/A
ippgDFTInv_CToC_48_64fc	N/A
ippgDFTInv_CToC_49_32fc	N/A
ippgDFTInv_CToC_49_64fc	N/A
ippgDFTInv_CToC_4_32fc	N/A
ippgDFTInv_CToC_4_64fc	N/A
ippgDFTInv_CToC_50_32fc	N/A
ippgDFTInv_CToC_50_64fc	N/A
ippgDFTInv_CToC_51_32fc	N/A
ippgDFTInv_CToC_51_64fc	N/A
ippgDFTInv_CToC_52_32fc	N/A
ippgDFTInv_CToC_52_64fc	N/A
ippgDFTInv_CToC_53_32fc	N/A
ippgDFTInv_CToC_53_64fc	N/A
ippgDFTInv_CToC_54_32fc	N/A
ippgDFTInv_CToC_54_64fc	N/A
ippgDFTInv_CToC_55_32fc	N/A
ippgDFTInv_CToC_55_64fc	N/A
ippgDFTInv_CToC_56_32fc	N/A
ippgDFTInv_CToC_56_64fc	N/A

ippgDFTInv_CToC_57_32fc	N/A
ippgDFTInv_CToC_57_64fc	N/A
ippgDFTInv_CToC_58_32fc	N/A
ippgDFTInv_CToC_58_64fc	N/A
ippgDFTInv_CToC_59_32fc	N/A
ippgDFTInv_CToC_59_64fc	N/A
ippgDFTInv_CToC_5_32fc	N/A
ippgDFTInv_CToC_5_64fc	N/A
ippgDFTInv_CToC_60_32fc	N/A
ippgDFTInv_CToC_60_64fc	N/A
ippgDFTInv_CToC_61_32fc	N/A
ippgDFTInv_CToC_61_64fc	N/A
ippgDFTInv_CToC_62_32fc	N/A
ippgDFTInv_CToC_62_64fc	N/A
ippgDFTInv_CToC_63_32fc	N/A
ippgDFTInv_CToC_63_64fc	N/A
ippgDFTInv_CToC_64_32fc	N/A
ippgDFTInv_CToC_64_64fc	N/A
ippgDFTInv_CToC_64fc	N/A
ippgDFTInv_CToC_6_32fc	N/A
ippgDFTInv_CToC_6_64fc	N/A
ippgDFTInv_CToC_7_32fc	N/A
ippgDFTInv_CToC_7_64fc	N/A
ippgDFTInv_CToC_8_32fc	N/A
ippgDFTInv_CToC_8_64fc	N/A
ippgDFTInv_CToC_9_32fc	N/A
ippgDFTInv_CToC_9_64fc	N/A
ippgDFTInv_PackToR_10_32f	N/A
ippgDFTInv_PackToR_10_64f	N/A
ippgDFTInv_PackToR_11_32f	N/A
ippgDFTInv_PackToR_11_64f	N/A

ippgDFTInv_PackToR_12_32f	N/A
ippgDFTInv_PackToR_12_64f	N/A
ippgDFTInv_PackToR_13_32f	N/A
ippgDFTInv_PackToR_13_64f	N/A
ippgDFTInv_PackToR_14_32f	N/A
ippgDFTInv_PackToR_14_64f	N/A
ippgDFTInv_PackToR_15_32f	N/A
ippgDFTInv_PackToR_15_64f	N/A
ippgDFTInv_PackToR_16_32f	N/A
ippgDFTInv_PackToR_16_64f	N/A
ippgDFTInv_PackToR_17_32f	N/A
ippgDFTInv_PackToR_17_64f	N/A
ippgDFTInv_PackToR_18_32f	N/A
ippgDFTInv_PackToR_18_64f	N/A
ippgDFTInv_PackToR_19_32f	N/A
ippgDFTInv_PackToR_19_64f	N/A
ippgDFTInv_PackToR_20_32f	N/A
ippgDFTInv_PackToR_20_64f	N/A
ippgDFTInv_PackToR_21_32f	N/A
ippgDFTInv_PackToR_21_64f	N/A
ippgDFTInv_PackToR_22_32f	N/A
ippgDFTInv_PackToR_22_64f	N/A
ippgDFTInv_PackToR_23_32f	N/A
ippgDFTInv_PackToR_23_64f	N/A
ippgDFTInv_PackToR_24_32f	N/A
ippgDFTInv_PackToR_24_64f	N/A
ippgDFTInv_PackToR_25_32f	N/A
ippgDFTInv_PackToR_25_64f	N/A
ippgDFTInv_PackToR_26_32f	N/A
ippgDFTInv_PackToR_26_64f	N/A
ippgDFTInv_PackToR_27_32f	N/A

ippgDFTInv_PackToR_27_64f	N/A
ippgDFTInv_PackToR_28_32f	N/A
ippgDFTInv_PackToR_28_64f	N/A
ippgDFTInv_PackToR_29_32f	N/A
ippgDFTInv_PackToR_29_64f	N/A
ippgDFTInv_PackToR_2_32f	N/A
ippgDFTInv_PackToR_2_64f	N/A
ippgDFTInv_PackToR_30_32f	N/A
ippgDFTInv_PackToR_30_64f	N/A
ippgDFTInv_PackToR_31_32f	N/A
ippgDFTInv_PackToR_31_64f	N/A
ippgDFTInv_PackToR_32_32f	N/A
ippgDFTInv_PackToR_32_64f	N/A
ippgDFTInv_PackToR_32f	N/A
ippgDFTInv_PackToR_33_32f	N/A
ippgDFTInv_PackToR_33_64f	N/A
ippgDFTInv_PackToR_34_32f	N/A
ippgDFTInv_PackToR_34_64f	N/A
ippgDFTInv_PackToR_35_32f	N/A
ippgDFTInv_PackToR_35_64f	N/A
ippgDFTInv_PackToR_36_32f	N/A
ippgDFTInv_PackToR_36_64f	N/A
ippgDFTInv_PackToR_37_32f	N/A
ippgDFTInv_PackToR_37_64f	N/A
ippgDFTInv_PackToR_38_32f	N/A
ippgDFTInv_PackToR_38_64f	N/A
ippgDFTInv_PackToR_39_32f	N/A
ippgDFTInv_PackToR_39_64f	N/A
ippgDFTInv_PackToR_3_32f	N/A
ippgDFTInv_PackToR_3_64f	N/A
ippgDFTInv_PackToR_40_32f	N/A

ippgDFTInv_PackToR_40_64f	N/A
ippgDFTInv_PackToR_41_32f	N/A
ippgDFTInv_PackToR_41_64f	N/A
ippgDFTInv_PackToR_42_32f	N/A
ippgDFTInv_PackToR_42_64f	N/A
ippgDFTInv_PackToR_43_32f	N/A
ippgDFTInv_PackToR_43_64f	N/A
ippgDFTInv_PackToR_44_32f	N/A
ippgDFTInv_PackToR_44_64f	N/A
ippgDFTInv_PackToR_45_32f	N/A
ippgDFTInv_PackToR_45_64f	N/A
ippgDFTInv_PackToR_46_32f	N/A
ippgDFTInv_PackToR_46_64f	N/A
ippgDFTInv_PackToR_47_32f	N/A
ippgDFTInv_PackToR_47_64f	N/A
ippgDFTInv_PackToR_48_32f	N/A
ippgDFTInv_PackToR_48_64f	N/A
ippgDFTInv_PackToR_49_32f	N/A
ippgDFTInv_PackToR_49_64f	N/A
ippgDFTInv_PackToR_4_32f	N/A
ippgDFTInv_PackToR_4_64f	N/A
ippgDFTInv_PackToR_50_32f	N/A
ippgDFTInv_PackToR_50_64f	N/A
ippgDFTInv_PackToR_51_32f	N/A
ippgDFTInv_PackToR_51_64f	N/A
ippgDFTInv_PackToR_52_32f	N/A
ippgDFTInv_PackToR_52_64f	N/A
ippgDFTInv_PackToR_53_32f	N/A
ippgDFTInv_PackToR_53_64f	N/A
ippgDFTInv_PackToR_54_32f	N/A
ippgDFTInv_PackToR_54_64f	N/A

ippgDFTInv_PackToR_55_32f	N/A
ippgDFTInv_PackToR_55_64f	N/A
ippgDFTInv_PackToR_56_32f	N/A
ippgDFTInv_PackToR_56_64f	N/A
ippgDFTInv_PackToR_57_32f	N/A
ippgDFTInv_PackToR_57_64f	N/A
ippgDFTInv_PackToR_58_32f	N/A
ippgDFTInv_PackToR_58_64f	N/A
ippgDFTInv_PackToR_59_32f	N/A
ippgDFTInv_PackToR_59_64f	N/A
ippgDFTInv_PackToR_5_32f	N/A
ippgDFTInv_PackToR_5_64f	N/A
ippgDFTInv_PackToR_60_32f	N/A
ippgDFTInv_PackToR_60_64f	N/A
ippgDFTInv_PackToR_61_32f	N/A
ippgDFTInv_PackToR_61_64f	N/A
ippgDFTInv_PackToR_62_32f	N/A
ippgDFTInv_PackToR_62_64f	N/A
ippgDFTInv_PackToR_63_32f	N/A
ippgDFTInv_PackToR_63_64f	N/A
ippgDFTInv_PackToR_64_32f	N/A
ippgDFTInv_PackToR_64_64f	N/A
ippgDFTInv_PackToR_64f	N/A
ippgDFTInv_PackToR_6_32f	N/A
ippgDFTInv_PackToR_6_64f	N/A
ippgDFTInv_PackToR_7_32f	N/A
ippgDFTInv_PackToR_7_64f	N/A
ippgDFTInv_PackToR_8_32f	N/A
ippgDFTInv_PackToR_8_64f	N/A
ippgDFTInv_PackToR_9_32f	N/A
ippgDFTInv_PackToR_9_64f	N/A

ippgDFTInv_PermToR_10_32f	N/A
ippgDFTInv_PermToR_10_64f	N/A
ippgDFTInv_PermToR_11_32f	N/A
ippgDFTInv_PermToR_11_64f	N/A
ippgDFTInv_PermToR_12_32f	N/A
ippgDFTInv_PermToR_12_64f	N/A
ippgDFTInv_PermToR_13_32f	N/A
ippgDFTInv_PermToR_13_64f	N/A
ippgDFTInv_PermToR_14_32f	N/A
ippgDFTInv_PermToR_14_64f	N/A
ippgDFTInv_PermToR_15_32f	N/A
ippgDFTInv_PermToR_15_64f	N/A
ippgDFTInv_PermToR_16_32f	N/A
ippgDFTInv_PermToR_16_64f	N/A
ippgDFTInv_PermToR_17_32f	N/A
ippgDFTInv_PermToR_17_64f	N/A
ippgDFTInv_PermToR_18_32f	N/A
ippgDFTInv_PermToR_18_64f	N/A
ippgDFTInv_PermToR_19_32f	N/A
ippgDFTInv_PermToR_19_64f	N/A
ippgDFTInv_PermToR_20_32f	N/A
ippgDFTInv_PermToR_20_64f	N/A
ippgDFTInv_PermToR_21_32f	N/A
ippgDFTInv_PermToR_21_64f	N/A
ippgDFTInv_PermToR_22_32f	N/A
ippgDFTInv_PermToR_22_64f	N/A
ippgDFTInv_PermToR_23_32f	N/A
ippgDFTInv_PermToR_23_64f	N/A
ippgDFTInv_PermToR_24_32f	N/A
ippgDFTInv_PermToR_24_64f	N/A
ippgDFTInv_PermToR_25_32f	N/A

ippgDFTInv_PermToR_25_64f	N/A
ippgDFTInv_PermToR_26_32f	N/A
ippgDFTInv_PermToR_26_64f	N/A
ippgDFTInv_PermToR_27_32f	N/A
ippgDFTInv_PermToR_27_64f	N/A
ippgDFTInv_PermToR_28_32f	N/A
ippgDFTInv_PermToR_28_64f	N/A
ippgDFTInv_PermToR_29_32f	N/A
ippgDFTInv_PermToR_29_64f	N/A
ippgDFTInv_PermToR_2_32f	N/A
ippgDFTInv_PermToR_2_64f	N/A
ippgDFTInv_PermToR_30_32f	N/A
ippgDFTInv_PermToR_30_64f	N/A
ippgDFTInv_PermToR_31_32f	N/A
ippgDFTInv_PermToR_31_64f	N/A
ippgDFTInv_PermToR_32_32f	N/A
ippgDFTInv_PermToR_32_64f	N/A
ippgDFTInv_PermToR_32f	N/A
ippgDFTInv_PermToR_33_32f	N/A
ippgDFTInv_PermToR_33_64f	N/A
ippgDFTInv_PermToR_34_32f	N/A
ippgDFTInv_PermToR_34_64f	N/A
ippgDFTInv_PermToR_35_32f	N/A
ippgDFTInv_PermToR_35_64f	N/A
ippgDFTInv_PermToR_36_32f	N/A
ippgDFTInv_PermToR_36_64f	N/A
ippgDFTInv_PermToR_37_32f	N/A
ippgDFTInv_PermToR_37_64f	N/A
ippgDFTInv_PermToR_38_32f	N/A
ippgDFTInv_PermToR_38_64f	N/A
ippgDFTInv_PermToR_39_32f	N/A

ippgDFTInv_PermToR_39_64f	N/A
ippgDFTInv_PermToR_3_32f	N/A
ippgDFTInv_PermToR_3_64f	N/A
ippgDFTInv_PermToR_40_32f	N/A
ippgDFTInv_PermToR_40_64f	N/A
ippgDFTInv_PermToR_41_32f	N/A
ippgDFTInv_PermToR_41_64f	N/A
ippgDFTInv_PermToR_42_32f	N/A
ippgDFTInv_PermToR_42_64f	N/A
ippgDFTInv_PermToR_43_32f	N/A
ippgDFTInv_PermToR_43_64f	N/A
ippgDFTInv_PermToR_44_32f	N/A
ippgDFTInv_PermToR_44_64f	N/A
ippgDFTInv_PermToR_45_32f	N/A
ippgDFTInv_PermToR_45_64f	N/A
ippgDFTInv_PermToR_46_32f	N/A
ippgDFTInv_PermToR_46_64f	N/A
ippgDFTInv_PermToR_47_32f	N/A
ippgDFTInv_PermToR_47_64f	N/A
ippgDFTInv_PermToR_48_32f	N/A
ippgDFTInv_PermToR_48_64f	N/A
ippgDFTInv_PermToR_49_32f	N/A
ippgDFTInv_PermToR_49_64f	N/A
ippgDFTInv_PermToR_4_32f	N/A
ippgDFTInv_PermToR_4_64f	N/A
ippgDFTInv_PermToR_50_32f	N/A
ippgDFTInv_PermToR_50_64f	N/A
ippgDFTInv_PermToR_51_32f	N/A
ippgDFTInv_PermToR_51_64f	N/A
ippgDFTInv_PermToR_52_32f	N/A
ippgDFTInv_PermToR_52_64f	N/A

ippgDFTInv_PermToR_53_32f	N/A
ippgDFTInv_PermToR_53_64f	N/A
ippgDFTInv_PermToR_54_32f	N/A
ippgDFTInv_PermToR_54_64f	N/A
ippgDFTInv_PermToR_55_32f	N/A
ippgDFTInv_PermToR_55_64f	N/A
ippgDFTInv_PermToR_56_32f	N/A
ippgDFTInv_PermToR_56_64f	N/A
ippgDFTInv_PermToR_57_32f	N/A
ippgDFTInv_PermToR_57_64f	N/A
ippgDFTInv_PermToR_58_32f	N/A
ippgDFTInv_PermToR_58_64f	N/A
ippgDFTInv_PermToR_59_32f	N/A
ippgDFTInv_PermToR_59_64f	N/A
ippgDFTInv_PermToR_5_32f	N/A
ippgDFTInv_PermToR_5_64f	N/A
ippgDFTInv_PermToR_60_32f	N/A
ippgDFTInv_PermToR_60_64f	N/A
ippgDFTInv_PermToR_61_32f	N/A
ippgDFTInv_PermToR_61_64f	N/A
ippgDFTInv_PermToR_62_32f	N/A
ippgDFTInv_PermToR_62_64f	N/A
ippgDFTInv_PermToR_63_32f	N/A
ippgDFTInv_PermToR_63_64f	N/A
ippgDFTInv_PermToR_64_32f	N/A
ippgDFTInv_PermToR_64_64f	N/A
ippgDFTInv_PermToR_64f	N/A
ippgDFTInv_PermToR_6_32f	N/A
ippgDFTInv_PermToR_6_64f	N/A
ippgDFTInv_PermToR_7_32f	N/A
ippgDFTInv_PermToR_7_64f	N/A

ippgDFTInv_PermToR_8_32f	N/A
ippgDFTInv_PermToR_8_64f	N/A
ippgDFTInv_PermToR_9_32f	N/A
ippgDFTInv_PermToR_9_64f	N/A
ippgHartley_10_32f	N/A
ippgHartley_10_64f	N/A
ippgHartley_11_32f	N/A
ippgHartley_11_64f	N/A
ippgHartley_12_32f	N/A
ippgHartley_12_64f	N/A
ippgHartley_13_32f	N/A
ippgHartley_13_64f	N/A
ippgHartley_14_32f	N/A
ippgHartley_14_64f	N/A
ippgHartley_15_32f	N/A
ippgHartley_15_64f	N/A
ippgHartley_16_32f	N/A
ippgHartley_16_64f	N/A
ippgHartley_17_32f	N/A
ippgHartley_17_64f	N/A
ippgHartley_18_32f	N/A
ippgHartley_18_64f	N/A
ippgHartley_19_32f	N/A
ippgHartley_19_64f	N/A
ippgHartley_20_32f	N/A
ippgHartley_20_64f	N/A
ippgHartley_21_32f	N/A
ippgHartley_21_64f	N/A
ippgHartley_22_32f	N/A
ippgHartley_22_64f	N/A
ippgHartley_23_32f	N/A

ippgHartley_23_64f	N/A
ippgHartley_24_32f	N/A
ippgHartley_24_64f	N/A
ippgHartley_25_32f	N/A
ippgHartley_25_64f	N/A
ippgHartley_26_32f	N/A
ippgHartley_26_64f	N/A
ippgHartley_27_32f	N/A
ippgHartley_27_64f	N/A
ippgHartley_28_32f	N/A
ippgHartley_28_64f	N/A
ippgHartley_29_32f	N/A
ippgHartley_29_64f	N/A
ippgHartley_2_32f	N/A
ippgHartley_2_64f	N/A
ippgHartley_30_32f	N/A
ippgHartley_30_64f	N/A
ippgHartley_31_32f	N/A
ippgHartley_31_64f	N/A
ippgHartley_32_32f	N/A
ippgHartley_32_64f	N/A
ippgHartley_32f	N/A
ippgHartley_33_32f	N/A
ippgHartley_33_64f	N/A
ippgHartley_34_32f	N/A
ippgHartley_34_64f	N/A
ippgHartley_35_32f	N/A
ippgHartley_35_64f	N/A
ippgHartley_36_32f	N/A
ippgHartley_36_64f	N/A
ippgHartley_37_32f	N/A

ippgHartley_37_64f	N/A
ippgHartley_38_32f	N/A
ippgHartley_38_64f	N/A
ippgHartley_39_32f	N/A
ippgHartley_39_64f	N/A
ippgHartley_3_32f	N/A
ippgHartley_3_64f	N/A
ippgHartley_40_32f	N/A
ippgHartley_40_64f	N/A
ippgHartley_41_32f	N/A
ippgHartley_41_64f	N/A
ippgHartley_42_32f	N/A
ippgHartley_42_64f	N/A
ippgHartley_43_32f	N/A
ippgHartley_43_64f	N/A
ippgHartley_44_32f	N/A
ippgHartley_44_64f	N/A
ippgHartley_45_32f	N/A
ippgHartley_45_64f	N/A
ippgHartley_46_32f	N/A
ippgHartley_46_64f	N/A
ippgHartley_47_32f	N/A
ippgHartley_47_64f	N/A
ippgHartley_48_32f	N/A
ippgHartley_48_64f	N/A
ippgHartley_49_32f	N/A
ippgHartley_49_64f	N/A
ippgHartley_4_32f	N/A
ippgHartley_4_64f	N/A
ippgHartley_50_32f	N/A
ippgHartley_50_64f	N/A

ippgHartley_51_32f	N/A
ippgHartley_51_64f	N/A
ippgHartley_52_32f	N/A
ippgHartley_52_64f	N/A
ippgHartley_53_32f	N/A
ippgHartley_53_64f	N/A
ippgHartley_54_32f	N/A
ippgHartley_54_64f	N/A
ippgHartley_55_32f	N/A
ippgHartley_55_64f	N/A
ippgHartley_56_32f	N/A
ippgHartley_56_64f	N/A
ippgHartley_57_32f	N/A
ippgHartley_57_64f	N/A
ippgHartley_58_32f	N/A
ippgHartley_58_64f	N/A
ippgHartley_59_32f	N/A
ippgHartley_59_64f	N/A
ippgHartley_5_32f	N/A
ippgHartley_5_64f	N/A
ippgHartley_60_32f	N/A
ippgHartley_60_64f	N/A
ippgHartley_61_32f	N/A
ippgHartley_61_64f	N/A
ippgHartley_62_32f	N/A
ippgHartley_62_64f	N/A
ippgHartley_63_32f	N/A
ippgHartley_63_64f	N/A
ippgHartley_64_32f	N/A
ippgHartley_64_64f	N/A
ippgHartley_64f	N/A

ippgHartley_6_32f	N/A
ippgHartley_6_64f	N/A
ippgHartley_7_32f	N/A
ippgHartley_7_64f	N/A
ippgHartley_8_32f	N/A
ippgHartley_8_64f	N/A
ippgHartley_9_32f	N/A
ippgHartley_9_64f	N/A
ippgWHTGetBufferSize_32f	N/A
ippgWHTGetBufferSize_64f	N/A
ippgWHT_10_32f	N/A
ippgWHT_10_64f	N/A
ippgWHT_11_32f	N/A
ippgWHT_11_64f	N/A
ippgWHT_12_32f	N/A
ippgWHT_12_64f	N/A
ippgWHT_13_32f	N/A
ippgWHT_13_64f	N/A
ippgWHT_1_32f	N/A
ippgWHT_1_64f	N/A
ippgWHT_2_32f	N/A
ippgWHT_2_64f	N/A
ippgWHT_32f	N/A
ippgWHT_3_32f	N/A
ippgWHT_3_64f	N/A
ippgWHT_4_32f	N/A
ippgWHT_4_64f	N/A
ippgWHT_5_32f	N/A
ippgWHT_5_64f	N/A
ippgWHT_64f	N/A
ippgWHT_6_32f	N/A

ippgWHT_6_64f	N/A
ippgWHT_7_32f	N/A
ippgWHT_7_64f	N/A
ippgWHT_8_32f	N/A
ippgWHT_8_64f	N/A
ippgWHT_9_32f	N/A
ippgWHT_9_64f	N/A
ippgenGetLibVersion	N/A

Bibliography for Signal Processing

This bibliography provides a list of reference books and other sources of additional information that might be useful to the application programmer. This list is neither complete nor exhaustive, but serves as a starting point. Of all the references listed, [Mit93] will be the most useful to those readers who already have a basic understanding of signal processing. This reference collects the work of 27 experts in the field and has both great breadth and depth.

The books [Opp75], [Opp89], [Jack89], and [Zie83] are undergraduate signal processing texts. [Opp89] is a much revised edition of the classic [Opp75]; [Jack89] is more concise than the others; and [Zie83] also covers continuous-time systems.

[AMRWB+]	3GPP Technical Specification 26.290: Extended Adaptive Multi-Rate - Wideband (AMR-WB+) codec; Transcoding functions, v.6.3.0, rel.6, June-2005.
[Arn97]	1. Arnavut, S. Magliveras. Block-Sorting and Compression. IEEE Data Compression Conference, Snowbird, Utah, pp.181-190, March 1997.
[Ash94]	M.R. Asharif and F. Amano. Acoustic Echo Canceler Using the FBAF Algorithm. IEEE Trans. Comm., Vol. 42, No. 12, pp. 3090-3094, Dec. 1994.
[Berl68]	Elwin R. Berlekamp. Algebraic Coding Theory. McGraw-Hill Book Company, 1968
[Bla84]	Richard E. Blahut. Theory and Practice of Error Control Codes. Addison-Wesley Publishing Company, 1984
[Bris94]	1. Brislawn. Classification of Nonexpansive Symmetric Extension Transforms for Multirate Filter Banks. Los Alamos Report LA-UR-94-1747, 1994.
[Cap78]	1. Cappellini, A. G. Constantinides, and P. Emilani. Digital Filters and Their Applications. Academic Press, London, 1978.

[Cap94]	Cappé O. Elimination of the musical noise phenomenon with the Ephraim and Malah noise suppressor. IEEE Trans. Speech and Audio Processing, vol. 2(2), (1994).
[Cast93]	G.Castagnoli, S.Brauer, M.Herrmann. Optimization of cyclic redundancy-check codes with 24 and 32 parity bits. IEEE Transactions on Communications, Vol.41, No.6, June, 1993, pp.883-892.
[CCITT]	CCITT, Recommendation G.711. Pulse Code Modulation of Frequencies, 1984.
[Coh02]	1. Cohen and B. Berdugo. Noise Estimation by Minima Controlled Recursive Averaging for Robust Speech Enhancement. IEEE Signal Proc. Letters, Vol. 9, No. 1, Jan. 2002, pp. 12-15.
[Cro83]	1. a. Crochiere and L. R. Rabiner. Multirate Digital Signal Processing. Prentice Hall, Englewood Cliffs, New Jersey, 1983.
[Dau92]	1. Daubechies. Ten Lectures on Wavelets. Springer Verlag, Pennsylvania, 1992.
[Eph84]	1. Ephraim and D. Malah. Speech Enhancement Using a Minimum Mean-Square Error Short-Time Spectral Amplitude Estimator. IEEE Trans. ASSP, Vol. 32, No. 6, Dec. 1984, pp. 1109-1121.
[ETSI02]	ETSI TS 102 114:2002 (DTS Coherent Acoustics - Core and Extensions)(2002). http://www.etsi.org/services_products/freestandard/home.htm
[EC126]	ETSI TS 126 404 V6.0.0. UMTS. General audio codec audio processing functions; Enhanced aacPlus general audio codec; Encoder specification; SBR PART - 3GPP TS 26.404 version 6.0.0 Release 6 (09/2004).
[ES201]	ETSI ES 201 108 V1.1.2. ETSI Standard. Speech processing, Transmission and Quality aspects (STQ); Distributed speech recognition; Front-end feature extraction algorithm; Compression algorithms.
[ES202]	ETSI ES 202 050 V1.1.1. ETSI Standard. Speech processing, Transmission and Quality aspects (STQ); Distributed speech recognition; Advanced front-end feature extraction algorithm; Compression algorithms.
[Feig92]	1. Feig and S. Winograd. Fast algorithms for DCT. IEEE Transactions on Signal Processing, vol.40, No.9, 1992.
[Gal75]	1. Gallager, D.Van Voorhis. Optimal source codes for geometrically distributed integer alphabets. IEEE Transactions on Information Theory, vol. IT-21, No. 3, pp.228-230, 1975.

[Griff87]	G.Griffiths, G.C.Stones. The tea-leaf reader algorithm: an efficient implementation of CRC-16 and CRC-32. Communications of the ACM , vol.30, No.7, 1987, pp.617-620.
[Ham83]	R.W. Hamming. Digital Filters, Prentice-Hall, New Jersey 1983.
[Har78]	1. Harris. On the Use of Windows. Proceedings of the IEEE, vol. 66, No.1, IEEE, 1978.
[Hay91]	1. Haykin. Adaptive Filter Theory. Prentice Hall, Englewood Cliffs, New Jersey, 1991.
[ICCC]	Intel (R) C++ Compiler 11.1. User and Reference Guides.. Document number 304968-023US.
[ISO11172]	ISO/IEC 11172-3 - Information technology. Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s. Part 3: Audio (1993)
[ISO13818]	ISO/IEC 13818-3 - Information technology. Generic coding of moving pictures and associated audio information. Part 3: Audio (1998).
[ISO14496]	ISO/IEC 14496-3 - Information technology. Coding of audio-visual objects. Part 3: Audio (2001).
[ISO14496A]	ISO/IEC 14496-3/Amd.1:2003 - Bandwidth Extension (2003).
[ISO14496B]	ISO/IEC 14496-3/Amd.2:2004 - Parametric coding for high-quality audio (2004).
[ITU169]	ITU-T Recommendation G.169. Automatic Level Control Devices, (06/99).
[ITU224]	ITU-T Recommendation X.224 Annex D. Checksum Algorithms, (11/93), pp144,145.
[ITU722]	ITU-T Recommendation G.722.1 (09/99), Coding at 24 and 32 8 kbit/s for hand-free operation in systems with low frame loss.
[ITU723]	ITU-T Recommendation G.723.1 . Dual Rate speech coder for Multimedia Communications transmitting at 5.3 and 6.3 Kbit/s, (03/96).
[ITU723A]	ITU-T Recommendation G.723.1 Annex A. Silence compression scheme, (11/96).
[ITU728]	ITU-T Recommendation G.728. Coding of Speech at 16 kbits/s Using Low-Delay Code Excited Linear Prediction, 1992.
[ITU729]	ITU-T Recommendation G.729. Coding of Speech at 8 kbit/s Using Conjugate-Structure Algebraic-Code-Excited Linear-Prediction (CS-ACELP), (03/96).
[ITU729A]	ITU-T Recommendation G.729 Annex A. Reduced Complexity 8 kbit/s CS-ACELP speech codec, (11/96).

[ITU729B]	ITU-T Recommendation G.729 Annex B. A silence compression scheme for G.729 optimized for terminals conforming to Recommendation V.70, (10/96).
[ITU729B1]	ITU-T Recommendation G.729 Annex B Corrigendum 1, (02/98).
[ITU7291]	ITU-T Recommendation G.729.1. G729 Based Embeded Variable Bit-Rate Coder: a 8-32 kbits/s Scalable Wideband Coder Bitstream Interoperable with G.729, (06/06).
[ITUV34]	ITU-T Recommendation V.34. A modem operating at data signalling rates of up to 33 600 bit/s for use on the general switched telephone network and on leased point-to-point 2-wire telephone-type circuit. (02/98).
[Jack89]	Leland B. Jackson. Digital Filters and Signal Processing. Kluwer Academic Publishers, second edition, 1989.
[Kab86]	**1.** Kabal and P.Ramachandran. The Computation of line Spectral Frequencies Using Chebyshev Polynomials. IEEE transaction on acoustic, speech and signal processing, vol. ASSP-34, No.6, 1986.
[Lyn89]	Paul A. Lynn. Introductory Digital Signal Processing with Computer Applications. John Wiley&Sons, Inc., New York, 1993.
[Mar01]	**1.** Martin. Noise Power Spectral Density Estimation Based on Optimal Smoothing and Minimum Statistics. IEEE Trans. Speech and Audio, Vol. 9, No. 5, July 2001, pp. 504-512.
[Med91]	**1.** Medan, E. Yair, D. Chazan. Super Resolution Pitch Determination of Speech Signals. IEEE Transactions on Signal Processing, vol 39, No.1, 1991.
[Mit93]	Sanjit K. Mitra and James F. Kaiser editors. Handbook for Digital Signal Processing. John Wiley & Sons, Inc., New York, 1993
[Mit98]	Sanjit K. Mitra. Digital Signal Processing. McGraw Hill, 1998.
[Mor02]	Robert H.Morelos-Zaragoza. The Art of Error Correcting Coding. Wiley & Sons, Ltd., 2002.
[Nel92]	M.R.Nelson. File verification using CRC 32-bit cyclical redundancy check. Dr.Dobb’s Journal, vol. 17, Issue 5, 1992, p.64.
[NIC91]	Nam Ik Cho and Sang Uk Lee. Fast algorithm and implementation of 2D DCT. IEEE Transactions on Circuits and Systems, vol. 31, No.3, 1991.

[Opp75]	A.V. Oppenheim and R.W. Schafer. Digital Signal Processing. Prentice-Hall, Englewood Cliffs, New Jersey, 1975.
[Opp89]	A.V. Oppenheim and R.W. Schafer. Discrete-Time Signal Processing. Prentice Hall, Englewood Cliffs, New Jersey, 1989.
[Rab78]	L.R. Rabiner and R.W. Schafer. Digital Processing of Speech Signals. Prentice Hall, Englewood Cliffs, New Jersey, 1978.
[Rao90]	K.R. Rao and P. Yip. Discrete Cosine Transform. Algorithms, Advantages and Applications. Academic Press, San Diego, 1990.
[RFC1950]	P.Deutsch, J-L.Gailly. ZLIB Compressed Data Format Specification version 3.3, May, 1996. http://www.ietf.org/rfc/rfc1950.txt
[RFC1951]	P.Deutsch. DEFLATE Compressed Data Format Specification version 1.3, May, 1996. http://www.ietf.org/rfc/rfc1951.txt
[RFC1952]	P.Deutsch. GZIP file format specification version 4.3, May, 1996. http://www.ietf.org/rfc/rfc1952.txt
[Seg78]	1. Sedgewick. Implementing quicksort programs. Communications of the ACM, Vol. 21, No. 10, pp. 847-857, Oct. 1978.
[Storer82]	1. Storer, T.Szymanski. Data Compression via Textual Substitution. Journal of the Association for Computing Machinery (ACM), vol.19, No.4, pp.928-951, Oct.1982.
[Strang96]	1. Strang and T. Nguyen. Wavelet and Filter Banks. Wellesley-Cambridge Press, 1996, pp. 153-157.
[Tuc92]	1. Tucker. Voice Activity Detection Using a Periodicity Measure. IEEE Proceedings-I, vol. 139, No. 4, August 1992, pp. 377-380.
[Vai93]	1. a. Vaidyanathan. Multirate Systems and Filter Banks. Prentice Hall, Englewood Cliffs, New Jersey.
[Wid85]	1. Widrow and S.D. Stearns. Adaptive Signal Processing. Prentice-Hall, Englewood Cliffs, New Jersey, 1985.
[ZFP]	zfp & fpzip: Floating Point Compression. https://computation.llnl.gov/projects/floating-point-compression
[Zie83]	Rodger E. Ziemer, William H. Tranter and D. Ronald Fannin. Signals and Systems: Continuous and Discrete. Macmillan Publishing Co., New York, 1983.

[Ziv77]	**1.** Ziv and A. Lempel. A Universal Algorithm for Sequential Data Compression. IEEE Transactions on Information Theory, vol.23, No.3, pp.337-343, May 1977.
[ZLIB]	**1.** Gailly and M. Adler. Zlib 1.2.8 Manual, April 2013. http://www.zlib.net/manual.html

Glossary

adaptive filter	Colors specified by each pixel's coordinates in a color space. Intel® Integrated Performance Primitives for image processing use images with absolute colors.An adaptive filter varies its filter coefficients (taps) over time. Typically, the filter's coefficients are varied to make its output match a prototype "desired" signal as closely as possible. Non-adaptive filters do not vary their filter coefficients over time.
BQ	One of the modes, which indicates that the IIR initialization function initializes a cascade of biquads.
CCS	See *complex conjugate-symmetric*
companding functions	The functions that perform an operation of data compression by using a logarithmic encoder-decoder. Companding allows you to maintain the percentage error constant by logarithmically spacing the quantization levels.
complex conjugate-symmetric	A kind of symmetry that arises in the Fourier transform of real signals. A complex conjugate-symmetric signal has the property that `x(-n) = x(n)*`, where "*" denotes conjugation.
conjugate	The conjugate of a complex number `a + bj` is `a - bj`.
conjugate-symmetric	See *complex conjugate-symmetric*.
DCT	Acronym for the discrete cosine transform.
decimation	Filtering a signal followed by down-sampling. Filtering prevents aliasing distortion in the subsequent down-sampling. See *down-sampling*.
down-sampling	Down-sampling conceptually decreases a signal's sampling rate by removing samples from between neighboring samples of a signal. See *decimation*.
element-wise	An element-wise operation performs the same operation on each element of a vector, or uses the elements of the same position in multiple vectors as inputs to the operation. For example, the element-wise addition of the vectors *{x0, x1, x2}* and *{y0, y1, y2}* is performed as follows: *{x0, x1, x2} + {y0, y1, y2} = {x0 + y0, x1 + y1, x2 + y2}*.

FIR	Abbreviation for finite impulse response filter. Finite impulse response filters do not vary their filter coefficients (taps) over time.
FIR LMS	Abbreviation for least mean squares finite impulse response filter.
fixed-point data format	A format that assigns one bit for a sign and all other bits for fractional part. This format is used for optimized conversion operations with signed, purely fractional vectors. For example, S.31 format assumes a sign bit and 31 fractional bits; S15.16 assumes a sign bit, 15 integer bits, and 16 fractional bits.
IIR	Abbreviation for infinite impulse response filters.
in-place operation	A function that performs its operation in-place, takes its input from an array and returns its output to the same array. *See not-in-place operation*.
interpolation	Up-sampling a signal followed by filtering. The filtering gives the inserted samples a value close to the samples of their neighboring samples in the original signal. *See up-sampling*.
LMS	Abbreviation for **least mean square**, an algorithm frequently used as a measure of the difference between two signals. Also used as shorthand for an adaptive FIR filter employing the LMS algorithm for adaptation.
LTI	Abbreviation for linear time-invariant systems. In LTI systems, if an input consists of the sum of a number of signals, then the output is the sum of the system's responses to each signal considered separately.
MMX™ technology	An enhancement to Intel architecture aimed at better performance in multimedia and communications applications. The technology uses four additional data types, eight 64-bit MMX registers, and 57 additional instructions implementing the SIMD (single instruction, multiple data) technique.
MR	One of the modes, indicating the multi-rate variety of the function.
multi-rate	An operation or signal processing system involving signals with multiple sample rates. Decimation and interpolation are examples of multi-rate operations.
not-in-place operation	A function that performs its operation not-in-place takes its input from a source array and puts its output in a second, destination array.
polyphase	A computationally efficient method for multi-rate filtering. For example, interpolation or decimation.
CCS	A representation of a complex conjugate-symmetric sequence which is easier to use than the `Pack` or `Perm` formats.

`Pack`	A compact representation of a complex conjugate-symmetric sequence. The disadvantage of this format is that it is not the natural format used by the real FFT algorithms ("natural" in the sense that bit-reversed order is natural for radix-2 complex FFTs).
`Perm`	A format for storing the values for the FFT algorithm. The `Perm` format stores the values in the order in which the FFT algorithm uses them. That is, the real and imaginary parts of a given sample need not be adjacent.
saturation	Using saturation arithmetic, when a number exceeds the data-range limit for its data type, it saturates to the upper data-range limit. For example, a signed word greater than 7FFFh saturates to 7FFFh. When a number is less than the lower data-range limit, it saturates to the lower data-range. For example, a signed word less than 8000h saturates to 8000h.
sinusoid	See *tone*
Intel® Streaming SIMD Extensions	The major enhancement to Intel architecture instruction set. Incorporates a group of general-purpose floating-point instructions operating on packed data, additional packed integer instructions, together with cacheability control and state management instructions. These instructions significantly improve performance of applications using compute-intensive processing of floating-point and integer data.
tone	A sinusoid of a given frequency, phase, and magnitude. Tones are used as test signals and as building blocks for more complex signals.
up-sampling	Up-sampling conceptually increases the signal sampling rate by inserting zero-valued samples between neighboring samples of a signal.
window	A mathematical function by which a signal is multiplied to improve the characteristics of some subsequent analysis. Windows are commonly used in FFT-based spectral analysis.

Developer Reference: Image Processing and Computer Vision

The following are some important features of the image processing part of the Intel® Integrated Performance Primitives (Intel® IPP) library:

Computer Vision

Computer Vision functions provide feature detection, pattern recognition, distance transform, image segmentation, and other operations.

Image Color Conversion

Image Color Conversion functions include color model, color-gray scale, and format conversions, as well as color twist, color keying, gamma correction, and intensity transformation.

Image Geometry Transforms

Image Geometry Transforms offer functions for resizing, rotation, warping, and remapping of an image.

Filtering Functions

Filtering Functions perform convolution and deconvolution operations, as well as median, linear, separable, Wiener, and fixed filtering.

Image Statistics Functions

Image Statistics Functions compute different statistical parameters of an image, such as sum, integral, mean and standard deviation, intensity histogram, and others.

Image Arithmetic and Logical Operations

Image Arithmetic and Logical Operations modify pixel values of an image buffer using arithmetic and logical operations. This group also includes functions that perform image composition based on opacity (alpha-blending).

Image Data Exchange and Initialization Functions

Image Data Exchange and Initialization Functions perform image data manipulation, exchange, and initialization operations.

- Intel® Integrated Performance Primitives Concepts
- Support Functions
- Image Data Exchange and Initialization Functions
- Image Arithmetic and Logical Operations
- Image Color Conversion
- Threshold and Compare Operations
- Morphological Operations
- Filtering Functions
- Image Linear Transforms
- Image Statistics Functions
- Image Geometry Transforms
- Miscellaneous Image Transforms
- Wavelet Transforms
- Computer Vision
- 3D Data Processing Functions
- Appendix A: Handling of Special Cases
- Appendix B: Interpolation in Image Geometric Transform Functions
- Appendix C: Removed Functions for Image and Video Processing
- Bibliography for Image Processing
- Glossary

Intel® Integrated Performance Primitives Concepts

This set of topics explains the purpose and structure of the Intel® Integrated Performance Primitives (Intel® IPP) for Intel® Architecture software and looks over some of the basic concepts used in the image part of Intel IPP. It also describes the supported data formats and operation modes, and defines function naming conventions in the document.

- Function Naming
- Function Prototypes in Intel IPP
- Rounding Mode
- Integer Result Scaling
- Error Reporting
- Platform-Aware Functions for Image Processing
- Threading Layer Functions
- Structures and Enumerators
- Image Data Types and Ranges
- Major Operation Models

Function Naming

Naming conventions for the Intel IPP functions are similar for all covered domains.

Function names in Intel IPP have the following general format:

```
ipp<data-domain><name>_<datatypedescriptor>][<_extension>](<parameters>);
```

The elements of this format are explained in the sections that follow.

- Data-Domain
- Name
- Data Types
- Descriptors
- Parameters
- Extensions

Data-Domain

The `data-domain` is a single character that denotes the subset of functionality to which a given function belongs. The current version of Intel IPP supports the following data-domains:

`s`	for signals (expected data type is a 1D signal)
`i`	for images and video (expected data type is a 2D image)
`m`	for matrices (expected data type is a matrix)
`r`	for realistic rendering functionality and 3D data processing (expected data type depends on supported rendering techniques)
`g`	for signals of fixed length

For example, function names that begin with `ippi` signify that respective functions are used for image or video processing.

Name

The `name` field identifies what function does and has the following format:

```
<name> = <operation>[_ modifier]
```

The `operation` component is one ore more words, acronyms, and abbreviations that describe the core operation.

The `modifier` component, if present, is a word or abbreviation that denotes a slight modification or variation of the given function.

For example, names without modifiers: `Add`, `RGBToYCbCr`, `MorphAddGetSize`; with modifiers: `DCT8x8Inv_2x2`, `DCT8x8Inv_4x4`, `RGBToYCbCr_JPEG`.

Data Types

The `datatype` field indicates data types used by the function, in the following format:

`<bit depth><bit interpretation>,`

where

`bit depth = <1|8|16|32|64>`

and

`bit interpretation = <u|s|f>[c]`

Here `u` indicates "unsigned integer", `s` indicates "signed integer", `f` indicates "floating point", and `c` indicates "complex".

Intel IPP supports the following data types for image and video processing functions:

`8u`	8 bit, unsigned data
`8s`	8 bit, signed data
`16u`	16 bit, unsigned data
`16uc`	16-bit, complex unsigned short data †
`16s`	16 bit, signed data
`16sc`	16-bit, complex short data
`32u`	32 bit, unsigned data
`32s`	32 bit, signed data
`32sc`	32-bit, complex int data
`32f`	32-bit, single-precision real floating point data
`32fc`	32-bit, single-precision complex floating point data
`64s`	64-bit, quadword signed data
`64f`	64-bit, double-precision real floating point data

† - only partial support for intermediate result after transforms (in the so-called "time" domain).

NOTE For image processing functions that do not support `1u` data type, convert bitonal images to `8u` gray scale images using the `ippiConvert_1u8u_C1R` function.

The formats for complex data are represented in Intel IPP by structures defined as follows:

```
typedef struct {     Ipp16s re;     Ipp16s im; } Ipp16sc;
typedef struct {     Ipp32s re;     Ipp32s im; } Ipp32sc;
typedef struct {     Ipp32f re;     Ipp32f im; } Ipp32fc;
```

where `re`, `im` denote the real and imaginary parts, respectively.

Complex data formats are used by several arithmetic image processing functions. The `32fc` format is also used to store input/output data in some Fourier transform functions.

The 64-bit formats, `64s` and `64f`, are used for storing data computed by some image statistics functions.

For functions that operate on a single data type, the `datatype` field contains only one of the values listed above.

If a function operates on source and destination images that have different data types, the respective data type identifiers are listed in the function name in order of source and destination as follows:

```
<datatype> = <src1Datatype>[src2Datatype][dstDatatype]
```

For example, the function that converts 8-bit unsigned source image data to 32-bit floating point destination image data has the `8u32f` value for the `datatype` field.

NOTE In the lists of function parameters (arguments), the `Ipp` prefix is written in the data type. For example, the 8-bit unsigned data is denoted as `Ipp8u` type. These Intel IPP-specific data types are defined in the respective library header files.

Descriptors

The `descriptors` field further describes the operation. Descriptors are individual characters that indicate additional details of the operation.

The following descriptors are used in image and video processing functions:

Descriptor	Description	Example
`A`	Image data contains an alpha channel as the last channel, requires `C4`, alpha-channel is not processed	`ippiAddC_8u_AC4R`
`A0`	Image data contains an alpha channel as the first channel, requires `C4`, alpha-channel is not processed	`ippiLUTPalette_8u_C3A0C4R`
`C1`, `C2`, `C3`, `C4`	Image data is in pixel order and made up of 1, 2, 3 or 4 discrete interleaved channels	`ippiFilter_32f_C1RippiFilter_32f_C3RippiFilter_32f_C4RippiResizeCCRotate_8u_C2R`
`C`	Channel of interest (COI) is used in the operation	`ippiSet_8u_C3CR`
`I`	Operation is performed in-place - that is result of operation is written back into the source (default is not-in-place)	`ippiMulPack_32f_C3IR`
`M`	Operation uses a mask to determine pixels to be processed	`ippiCopy_8u_C1MR`
`P`	Function works with non-contiguous volume data (the location of the planes is passed to the function using pointers on the plane)	`ipprResize_8u_C1PV`
`P2`, `P3`, `P4`	Image data is in planar order and made up of 2, 3 or 4 discrete planar (non-interleaved) channels, with a separate pointer to each plane	`ippiResizeYUV420_8u_P2R`, `ippiGammaFwd_8u_P3R`, `ippiCopy_32f_P4R`

R	Function operates on a defined region of interest (ROI) for each source image	ippiMean_8u_C4R
Sfs	Saturation and fixed scaling mode is used	ippiSubC_8u_C1RSfs
s	Saturation and no scaling (default)	ippiConvert_16s16u_C1Rs
V	Function operates on a defined volume of interest (VOI) for each source image	ipprCopyConstBorder_8u_C1V_L

If more that one descriptor is used, they are presented in the function name in alphabetical order.

Every function that operates on image data has a channel count descriptor `Cn` (for interleaved image) or `Pn` (for planar). No default channel count is defined.

If input and output channel layouts are different, both source and destination layouts are listed. For example, the function `ippiHLSToBGR_8u_C3P3R` converts three-channel interleaved HLS image to the three-plane BGR image.

Parameters

The parameters in functions are in the following order: all source operands, all destination operands, all other operation-specific parameters.

Source parameters are named `Src` or `SrcN`, if there is more than one input image. Destination parameters are named `Dst`. For in-place operations, the input/output parameter contains the name `SrcDst`. All parameters defined as pointers start with lowercase `p` , for example, `pSrc`,`pMean, pSpec` .

Extensions

The `extension` field denotes an Intel IPP extension to which the function belongs. The following extensions are supported in Intel IPP Image Processing functions:

Extension	Description	Example
L	Intel IPP platform-aware functions	ippiAddC_8u_AC4R_L
LT	Intel IPP threading layer (TL) functions based on the Platform Aware API	ippiSubC_8u_C1RSfs_LT
T	Intel IPP threading layer (TL) functions based on the Classic API	ipprFilterMedian_64f_C1V_T

See Also

Platform-Aware Functions for Image Processing

Threading Layer Functions

Function Prototypes in Intel IPP

Function names in Intel IPP contain `datatype` and `descriptor` fields after the `name` field (see Function Naming). Most of the Intel IPP functions for image processing have a number of flavors that differ in data types associated with the operation, and in some additional parameters.

Each function flavor has its unique prototype used in function definition and for calling the function from the application program. For many flavors of a given function, these prototypes look quite similar.

To avoid listing all the similar prototypes in function description in this document, only different templates for such prototypes followed by the table of applicable data types and descriptors for each function may be given. For simplicity, in such cases the data type and descriptor fields in the function name are denoted as `mod`:

```
<mod> = <datatype>_<descriptors>
```

For example, the template for the prototype of the image dilation function that performs not-in-place operation, looks like this:

```
IppStatus ippiDilate_<mod>(const Ipp<datatype>*pSrc, intsrcStep, Ipp<datatype>*pDst,
intdstStep, IppiSizeroiSize);
```

where the supported values for `mod` are:

`8u_C1R`
`8u_C3R`
`8u_AC4R`

This notation means that the `ippiDilate` function has three flavors for a not-in-place operation, which process 8-bit unsigned data (of `Ipp8u` type) and differ in the number of channels in processed images. These flavors have the following prototypes:

```
IppStatus ippiDilate_8u_C1R(const Ipp8u* pSrc, int srcStep,     Ipp8u* pDst, int dstStep,
IppiSize roiSize);
```

```
IppStatus ippiDilate_8u_C3R(const Ipp8u* pSrc, int srcStep,     Ipp8u* pDst, int dstStep,
IppiSize roiSize);
```

```
IppStatus ippiDilate_8u_AC4R(const Ipp8u* pSrc, int srcStep,     Ipp8u* pDst, int dstStep,
IppiSize roiSize);
```

Thus, to obtain the full name and parameters list for the specific function flavor, not listed directly, do the following:

1. Choose the function operation mode (denoted in this document as **Case 1**, **2** and so on) and look in the table for the supported data types and descriptors.
2. Set the `mod` field in the function name as the concatenation of the chosen data type and descriptor, delimited by the underscore.
3. Use the respective template, substituting all the `datatype` fields in the parameters list with the chosen data type. Note that `Ipp` prefix is written before the `datatype` in the parameters list (see Data Types for details).

Example

To get the prototype for the ippiSet function flavor that sets each channel of a 3-channel destination image to 16-bit signed values, choose **Case 2: Setting each color channel to a specified value** and use `datatype = 16s`, `descriptors = C3R`.

After substituting the `mod` field with `16s_C3R`, obtain the required prototype as

```
ippiSet_16s_C3R(const Ipp16svalue[3], Ipp16*pDst, intdstStep, IppiSizeroiSize);
```

Rounding Mode

As many Intel IPP functions have to meet the bit-exact requirement, image processing functions use rounding. The default rounding mode for all functions can be described as "nearest even", that is the fixed point number `x = N + α`, `0 ≤ α < 0`, where `N` is an integer number, is rounded as given by:

$$\lceil x \rceil = \begin{cases} N, & 0 \leq \alpha < 0.5 \\ N+1, & 0.5 < \alpha < 1 \\ N, & \alpha = 0.5, \; N-even \\ N+1, & \alpha = 0.5, \; N-odd \end{cases}$$

For example, 1.5 is rounded to 2, and 2.5 to 2.

Some image processing functions have additional rounding modes, which are set by the parameter `roundMode`.

Integer Result Scaling

The default for image processing functions is to saturate the results without scaling them.

Some image processing functions operating on integer data use scaling of the internally computed output results by the integer `scaleFactor`, which is specified as one of the function parameters. These functions have the `Sfs` descriptor in their names.

The scale factor can be negative, positive, or zero. Scaling is applied because internal computations are generally performed with a higher precision than the data types used for input and output images.

NOTE The result of integer operations is always saturated to the destination data type range even when scaling is used.

The scaling of an integer result is done by multiplying the output pixel values by $2^{-scaleFactor}$ before the function returns. This helps retain either the output data range or its precision. Usually the scaling with a positive factor is performed by the shift operation. The result is rounded off to the nearest even integer number (see Rounding Mode).

For example, the integer `Ipp16s` result of the square operation `ippiSqr` for the input value 200 is equal to 32767 instead of 40000, that is, the result is saturated and the exact value cannot be restored. The scaling of the output value with the factor `scaleFactor = 1` yields the result 20000, which is not saturated, and the exact value can be restored as 20000*2. Thus, the output data range is retained.

The following example shows how the precision can be partially retained by means of scaling. The integer square root operation `ippiSqr` (without scaling) for the input value 2 gives the result equal to 1 instead of 1.414. Scaling of the internally computed output value with the factor `scaleFactor` = -3 gives the result 11, and permits the more precise value to be restored as $11*2^{-3} = 1.375$.

Error Reporting

The Intel IPP functions return status codes of the performed operation to report errors and warnings to the calling program. Thus, it is up to the application to perform error-related actions and/or recover from the error. The last value of the error status is not stored, and the user is to decide whether to check it or not as the function returns. The status codes are of `IppStatus` type and are global constant integers.

The status codes and corresponding messages reported by Intel IPP for image and video processing are listed in Table Status Codes and Messages.

Status Code	Message

ippStsNotSupportedModeErr	The requested mode is currently not supported.
ippStsDecimateFractionErr	Unsupported fraction in decimate.
ippStsWeightErr	Incorrect value for weight.
ippStsQualityIndexErr	Quality Index cannot be calculated for an image filled with a constant.
ippStsResizeNoOperationErr	One of the output image dimensions is less than 1 pixel.
ippStsBlockStepErr	Step for Block is less than 8.
ippStsMBStepErr	Step for MB is less than 16.
ippStsNoiseRangeErr	Noise value for Wiener Filter is out of range.
ippStsLPCCalcErr	Cannot evaluate linear prediction.
ippStsJPEG2KBadPassNumber	Pass number exceeds the limits of [0,nOfPasses-1].
ippStsJPEG2KDamagedCodeBlock	Codeblock for decoding is damaged.
ippStsH263CBPYCodeErr	Illegal Huffman code is detected during CBPY stream processing.
ippStsH263MCBPCInterCodeErr	Illegal Huffman code is detected during MCBPC Inter stream processing.
ippStsH263MCBPCIntraCodeErr	Illegal Huffman code is detected during MCBPC Intra stream processing.
ippStsNotEvenStepErr	Step value is not pixel multiple.
ippStsHistoNofLevelsErr	Number of levels for histogram is less than 2.
ippStsLUTNofLevelsErr	Number of levels for LUT is less than 2.
ippStsMP4BitOffsetErr	Incorrect value for bit offset.
ippStsMP4QPErr	Incorrect value for quantization parameter.
ippStsMP4BlockIdxErr	Incorrect value for block index.
ippStsMP4BlockTypeErr	Incorrect block type.
ippStsMP4MVCodeErr	Illegal Huffman code is detected during MV stream processing.
ippStsMP4VLCCodeErr	Illegal Huffman code is detected during VLC stream processing.
ippStsMP4DCCodeErr	Illegal code is detected during DC stream processing.
ippStsMP4FcodeErr	Incorrect value for fcode.
ippStsMP4AlignErr	Incorrect buffer alignment.
ippStsMP4TempDiffErr	Incorrect temporal difference.
ippStsMP4BlockSizeErr	Incorrect size of block or macroblock.

ippStsMP4ZeroBABErr	All BAB values are equal to zero.
ippStsMP4PredDirErr	Incorrect prediction direction.
ippStsMP4BitsPerPixelErr	Incorrect number of bits per pixel.
ippStsMP4VideoCompModeErr	Incorrect video component mode.
ippStsMP4LinearModeErr	Incorrect DC linear mode.
ippStsH263PredModeErr	Incorrect value for Prediction Mode.
ippStsH263BlockStepErr	The step value is less than 8.
ippStsH263MBStepErr	The step value is less than 16.
ippStsH263FrameWidthErr	The frame width is less than 8.
ippStsH263FrameHeightErr	The frame height is less than, or equal to zero.
ippStsH263ExpandPelsErr	The expand pixels number is less than 8.
ippStsH263PlaneStepErr	Step value is less than the plane width.
ippStsH263QuantErr	Quantizer value is less than, or equal to zero, or more than 31.
ippStsH263MVCodeErr	Illegal Huffman code is detetcted during MV stream processing.
ippStsH263VLCCodeErr	Illegal Huffman code is detetcted during VLC stream processing.
ippStsH263DCCodeErr	Illegal code is detetcted during DC stream processing.
ippStsH263ZigzagLenErr	Zigzag compact length is more than 64.
ippStsJPEGHuffTableErr	JPEG Huffman table is destroyed.
ippStsJPEGDCTRangeErr	JPEG DCT coefficient is out of range.
ippStsJPEGOutOfBufErr	An attempt to access out of the buffer.
ippStsChannelOrderErr	Incorrect order of the destination channels.
ippStsZeroMaskValueErr	All values of the mask are equal to zero.
ippStsRangeErr	Incorrect values for bounds: the lower bound is greater than the upper bound.
ippStsQPErr	Incorrect value for a quantizer parameter.
ippStsQuadErr	The quadrangle is nonconvex or degenerates into triangle, line, or point.
ippStsRectErr	Size of the rectangular region is less than, or equal to 1.
ippStsCoeffErr	Incorrect values for the transformation coefficients.
ippStsNoiseValErr	Incorrect value for the noise amplitude for dithering.
ippStsDitherLevelsErr	Number of dithering levels is out of range.

ippStsNumChannelsErr	Incorrect or unsupported number of channels.
ippStsDataTypeErr	Incorrect or unsupported data type.
ippStsCOIErr	COI is out of range.
ippStsOutOfRangeErr	Argument is out of range or point is outside the image.
ippStsDivisorErr	Divisor is equal to zero, function is aborted.
ippStsAlphaTypeErr	Illegal type of image composition operation.
ippStsGammaRangeErr	Gamma range bound is less than, or equal to zero.
ippStsGrayCoefSumErr	Sum of the conversion coefficients must be less than, or equal to 1.
ippStsChannelErr	Illegal channel number.
ippStsJaehneErr	Magnitude value is negative.
ippStsStepErr	Step value is less than, or equal to zero.
ippStsStrideErr	Stride value is less than the row length.
ippStsEpsValErr	Negative epsilon value.
ippStsScaleRangeErr	Scale bounds are out of range.
ippStsThresholdErr	Invalid threshold bounds.
ippStsWtOffsetErr	Invalid offset value for the wavelet filter.
ippStsAnchorErr	Anchor point is outside the mask.
ippStsMaskSizeErr	Invalid mask size.
ippStsShiftErr	Shift value is less than zero.
ippStsSampleFactorErr	Sampling factor is less than, or equal to zero.
ippStsResizeFactorErr	Resize factor(s) is less than, or equal to zero.
ippStsDivByZeroErr	An attempt to divide by zero.
ippStsInterpolationErr	Invalid interpolation mode.
ippStsMirrorFlipErr	Invalid flip mode.
ippStsMoment00ZeroErr	Moment value M(0,0) is too small to continue calculations.
ippStsThreshNegLevelErr	Negative value of the level in the threshold operation.
ippStsContextMatchErr	Context parameter does not match the operation.
ippStsFftFlagErr	Invalid value of the FFT flag parameter.
ippStsFftOrderErr	Invalid value of the FFT order parameter.
ippStsMemAllocErr	Not enough memory for the operation.
ippStsNullPtrErr	Pointer is NULL.

ippStsSizeErr	Wrong value for the data size.
ippStsBadArgErr	Invalid or bad argument.
ippStsNoErr	No errors.
ippStsNoOperation	No operation has been executed.
ippStsMisalignedBuf	Misaligned pointer in operation in which it must be aligned.
ippStsSqrtNegArg	Negative value(s) of the argument in the function Sqrt.
ippStsInvZero	INF result. Zero value was met by InvThresh with zero level.
ippStsEvenMedianMaskSize	Even size of the Median Filter mask was replaced by the odd number.
ippStsDivByZero	Zero value(s) of the divisor in the function Div .
ippStsLnZeroArg	Zero value(s) of the argument in the function Ln.
ippStsLnNegArg	Negative value(s) of the argument in the function Ln.
ippStsNanArg	Argument value is not a number.
ippStsDoubleSize	Sizes of image are not multiples of 2.
ippStsJPEGMarker	JPEG marker in the bitstream.
ippStsResFloor	All result values are floored.
ippStsAffineQuadChanged	The fourth ertex of destination quad is not equal to the customer's one.
ippStsWrongIntersectROI	Incorrect ROI that has no intersection with the source or destination image. No operation.
ippStsWrongIntersectQuad	Incorrect quadrangle that has no intersection with the source or destination ROI. No operation.
ippStsSymKernelExpected	The kernel is not symmetric.
ippStsEvenMedianWeight	Even weight of the Weighted Median Filter was replaced by the odd one.
ippStsNoAntialiasing	The mode does not support antialiasing.
ippStsAlgTypeErr	The algorithm type is not supported.
ippStsAccurateModeNotSupported	Accurate mode is not supported.

The status codes ending with Err (except for the ippStsNoErr status) indicate an error; the integer values of these codes are negative. When an error occurs, the function execution is interrupted.

The status code ippStsNoErr indicates no error. All other status codes indicate warnings. When a specific case is encountered, the function execution is completed and the corresponding warning status is returned. For example, if the function ippiDiv meets an attempt to divide a positive value by zero, the function execution is not aborted. The result of the operation is set to the maximum value that can be represented by the source data type, and the user is warned by the output status ippStsDivByZero. See appendix A Handling of Special Cases for more information.

Platform-Aware Functions for Image Processing

Intel® Integrated Performance Primitives (Intel® IPP) library provides so-called platform-aware functions. These functions use the special data type `IppSizeL` for object sizes. The `IppSizeL` data type represents memory-related quantities: it can be 32- or 64-bit wide depending on the target architecture.

While the rest of Intel IPP functions support only objects of 32-bit integer size, platform-aware functions can work with 64-bit object sizes if it is supported by the platform. The API of platform-aware functions is similar to the API of other Intel IPP functions and has only slight differences. You can distinguish platform-aware functions by the `L` suffix in the function name, for example, `ippiAdd_8u_C1RSfs_L`.

Currently, the following image processing functions have platform-aware APIs:

Function Group	Header	Function Name
Arithmetic Functions	`ippi.h/ippi_l.h`	`Add|AddC Sub|SubC Mul|MulC Div`
Linear Filters	`ippi.h/ippi_l.h`	`FilterBilateralBorderGetBufferSize FilterBilateralBorderInit FilterBilateralBorder`
Resize Transform Functions	`ippi.h/ippi_l.h`	`ResizeGetSize ResizeGetBufferSize ResizeGetBorderSize Resize{Nearest|Linear|Cubic|Lanczos|Super}Init Resize{Nearest|Linear|Cubic|Lanczos|Super} ResizeAntialiasing{Nearest|Linear|Cubic|Lanczos|Super}Init ResizeAntialiasing{Nearest|Linear|Cubic|Lanczos|Super}`
Support Functions	`ippi.h/ippi_l.h`	`Malloc`
Copy Functions	`ippi.h/ippi_l.h`	`Copy Copy{Replicate|Wrap|Const|Mirror}Border`
Transform Functions	`ippi.h/ippi_l.h`	`WarpGetBufferSize WarpAffineGetSize GetAffine{Bound|Quad|Transform|SrcRoi} GetRotate{Shift|Transform} WarpAffine{Nearest|Linear|Cubic}Init WarpAffine{Nearest|Linear|Cubic}`
3D General Filters with Border	`ippi.h/ippi_l.h`	`FilterBorderGetSize FilterBorderInit FilterBorder Copy{Replicate|Const}Border FilterBilateral`

Intel IPP platform-aware functions are documented as additional flavors to the existing functions declared in standard Intel IPP headers (without the `l` suffix). The `ippi_l.h` header is included into `ippi.h`.

Threading Layer Functions

Intel® Integrated Performance Primitives (Intel® IPP) library provides threading layer (TL) functions for image processing. It is a set of functions that are implemented as wrappers over Intel IPP platform-aware functions by using tiling and multithreading with OpenMP*. For implementation details, please see the corresponding source code files.

The API of TL functions is similar to the API of other Intel IPP functions and has only slight differences. You can distinguish Intel IPP TL functions by the `_LT` or `_T` suffix in the function name, for example, `ippiAdd_8u_C1RSfs_LT`. See Extensions for more information about function naming.

For more information about the TL functions usage, refer to Using Intel® Integrated Performance Primitives Threading Layer (TL) Functions.

Structures and Enumerators

This topic describes the structures and enumerators used by the Intel® Integrated Performance Primitives (Intel® IPP) for image processing.

The `IppStatus` constant enumerates the status code values returned by Intel IPP functions, indicating whether the operation was error-free or not.

See Error Reporting for more information on the set of valid status codes and corresponding error messages for image and video processing functions.

The structure `IppiPoint` for storing the geometric position of a point is defined as

```
typedef struct {
    int x;
    int y;
} IppiPoint;
```

where `x`, `y` denote the coordinates of the point.

The structure `IppPointPolar` for storing the geometric position of a point in polar coordinates is defined as

```
typedef struct {
    Ipp32f rho;
    Ipp32f theta;
} IppPointPolar;
```

where `rho` - a radial coordinate (radial distance from the origin), `theta` - an angular coordinate (counterclockwise angle from the `x` - axis .

The structure `IppiSize` for storing the size of a rectangle is defined as

```
typedef struct {
    int width;
    int height;
} IppiSize;
```

where `width` and `height` denote the dimensions of the rectangle in the x- and `y`– directions, respectively.

The structure `IppiRect` for storing the geometric position and size of a rectangle is defined as

```
typedef struct {
    int x;
    int y;
    int width;
    int height;
} IppiRect;
```

where x, y denote the coordinates of the top left corner of the rectangle that has dimensions width in the x - direction by height in the y - direction

The ippiConnectedComp structure used in Computer Vision functions defines the connected component as follows:

```
typedef struct _IppiConnectedComp {
    Ipp64f area;
    Ipp64f value[3];
    IppiRect rect;
} IppiConnectedComp;
```

where area - area of the segmented component; value[3] - gray scale value of the segmented component; rect - bounding rectangle of the segmented component.

The IppiMaskSize enumeration defines the neighborhood area for some morphological and filtering functions:

```
typedef enum {
    ippMskSize1x3 = 13,
    ippMskSize1x5 = 15,
    ippMskSize3x1 = 31,
    ippMskSize3x3 = 33,
    ippMskSize5x1 = 51,
    ippMskSize5x5 = 55
} IppiMaskSize;
```

The IppCmpOp enumeration defines the type of compare operation to be used in image comparison functions:

```
typedef enum {
    ippCmpLess,
    ippCmpLessEq,
    ippCmpEq,
    ippCmpGreaterEq,
    ippCmpGreater
} IppCmpOp;
```

The IppRoundMode enumeration defines the rounding mode to be used in some conversion, filtering and arithmetic functions:

```
typedef enum {
    ippRndZero,
    ippRndNear,
    ippRndFinancial,
       ippRndHintAccurate=0x10
} IppRoundMode;
```

The IppHintAlgorithm enumeration defines the type of code to be used in some image transform and statistics functions, that is, faster but less accurate, or vice-versa, more accurate but slower. For more information on using this enumerator, see Table Hint Arguments for Image Moment Functions.

```
typedef enum {
    ippAlgHintNone,
    ippAlgHintFast,
    ippAlgHintAccurate
} IppHintAlgorithm;
```

The types of interpolation used by geometric transform functions are defined as follows:

```
enum {
    IPPI_INTER_NN        = 1,
    IPPI_INTER_LINEAR    = 2,
    IPPI_INTER_CUBIC     = 4,
    IPPI_INTER_CUBIC2P_BSPLINE,
```

```
    IPPI_INTER_CUBIC2P_CATMULLROM,
    IPPI_INTER_CUBIC2P_B05C03,
    IPPI_INTER_SUPER     = 8,
    IPPI_INTER_LANCZOS   = 16,
    IPPI_ANTIALIASING    =(1 << 29)
    IPPI_SUBPIXEL_EDGE   =(1 << 30)
    IPPI_SMOOTH_EDGE     = IPP_MIN_32S
};
```

The `IppiAlphaType` enumeration defines the type of the compositing operation to be used in the alpha composition functions:

```
typedef enum {
    ippAlphaOver,
    ippAlphaIn,
    ippAlphaOut,
    ippAlphaATop,
    ippAlphaXor,
    ippAlphaPlus,
    ippAlphaOverPremul,
    ippAlphaInPremul,
    ippAlphaOutPremul,
    ippAlphaATopPremul,
    ippAlphaXorPremul,
    ippAlphaPlusPremul
} IppiAlphaType;
```

The `IppiDitherType` enumeration defines the type of dithering to be used by the ippiReduceBits function:

```
typedef enum {
    ippDitherNone,
    ippDitherFS,
    ippDitherJJN,
    ippDitherStucki,
    ippDitherBayer
} IppiDitherType;
```

The layout of the image slices used in some image format conversion functions is defined as follows:

```
enum {
     IPP_UPPER        = 1,
     IPP_LEFT         = 2,
     IPP_CENTER       = 4,
     IPP_RIGHT        = 8,
     IPP_LOWER        = 16,
     IPP_UPPER_LEFT   = 32,
     IPP_UPPER_RIGHT  = 64,
     IPP_LOWER_LEFT   = 128,
     IPP_LOWER_RIGHT  = 256
};
```

The `IppiAxis` enumeration defines the flip axes for the ippiMirror functions or direction of the image intensity ramp for the ippiImageRamp functions:

```
typedef enum {
    ippAxsHorizontal,
    ippAxsVertical,
    ippAxsBoth,
    ippAxs45,
    ippAxs135
} IppiAxis;
```

The IppiBorderType enumeration defines the border type that is used by some Separable Filters and Fixed Filters functions:

```
typedef enum _IppiBorderType {
    ippBorderRepl      =  1,
    ippBorderWrap      =  2,
    ippBorderMirror    =  3,
    ippBorderMirrorR   =  4,
    ippBorderDefault   =  5,
    ippBorderConst     =  6,
    ippBorderTransp    =  7,
    ippBorderInMemTop     =  0x0010,
    ippBorderInMemBottom  =  0x0020,
    ippBorderInMemLeft    =  0x0040,
    ippBorderInMemRight   =  0x0080,
    ippBorderFirstStageInMemTop     =  0x0100,
    ippBorderFirstStageInMemBottom  =  0x0200,
    ippBorderFirstStageInMemLeft    =  0x0400,
    ippBorderFirstStageInMemRight   =  0x0800,
} IppiBorderType;
```

The IppiFraction enumeration defines shapes of the structuring element used in some decimate filter functions:

```
typedef enum {
    ippPolyphase_1_2,
    ippPolyphase_3_5,
    ippPolyphase_2_3,
    ippPolyphase_7_10,
    ippPolyphase_3_4,
} IppiFraction;
```

The IppiNormOp enumeration defines the type of normalization that should be applied to the output data:

```
typedef enum {
    ippiNormNone           = 0x00000000, // default
    ippiNorm          = 0x00000100, // normalized form
    ippiNormCoefficient = 0x00000200, // correlation coefficient in the range [-1.0,…,1.0]
    ippiNormMask          = 0x0000FF00,
} IppiNormOp;
```

The IppiROIShape enumeration defines the window shape for the two-dimensional convolution-specific functions:

```
typedef enum {
   ippiROIFull   = 0x00000000,
   ippiROIValid  = 0x00010000,
   ippiROISame   = 0x00020000,
   ippiROIMask   = 0x00FF0000
} IppiROIShape;
```

The IppNormType enumeration defines the norm type that should be applied when computing the magnitude of the gradient:

```
typedef enum {
    ippNormInf  =   0x00000001, // Infinity norm
    ippNormL1   =   0x00000002, // L1 normalization
    ippNormL2   =   0x00000004  // L2 normalization
} IppNormType;
```

The `IppiHOGConfig` structure defines the configuration parameters for the HOG descriptor:

```
typedef struct {
   int   cvCompatible;  /* openCV compatible output format */
   int   cellSize;      /* squre cell size (pixels) */
   int   blockSize;     /* square block size (pixels) */
   int   blockStride;   /* block displacement (the same for x- and y- directions) */
   int   nbins;         /* required number of bins */
   Ipp32f   sigma;      /* gaussian factor of HOG block weights */
   Ipp32f   l2thresh;   /* normalization factor */
   IppiSize winSize;    /* detection window size (pixels) */
} IppiHOGConfig;
```

The code flags used by the FastN functions are defined as follows:

```
enum {
    IPP_FASTN_ORIENTATION = 0x0001,
    IPP_FASTN_NMS         = 0x0002,
    IPP_FASTN_CIRCLE      = 0X0004,
    IPP_FASTN_SCORE_MODE0 = 0X0020
};
```

The `IppiFastNSpec` specification structure is used by the FastN function:

```
struct FastNSpec;
typedef struct FastNSpec IppiFastNSpec;
```

The `IppiCornerFastN` structure used by the FastN2DToVec function stores the destination vector of structures:

```
typedef struct _IppiCornerFastN {
    int     x;
    int     y;
    int     cornerType;
    int     orientation;
    float   angle;
    float   score;
} IppiCornerFastN;
```

The `IppFGMModel` structure contains parameters for the Gaussian mixture-based segmentation algorithm:

```
typedef struct
{
  unsigned int   numFrames;   /* length of history */
  unsigned int   numGauss;    /* maximal number of gaussian components per pixel */
                              /* (numGauss<=maxNumGauss) */
  Ipp32f varInit;     /* initial value of variance for new gaussian component */
  Ipp32f varMin;      /* minimal bound of variance */
  Ipp32f varMax;      /* maximal bound of variance */
  Ipp32f varWBRatio;  /* background threshold */
  Ipp32f bckgThr;     /* background total weights sum threshold */
  Ipp32f varNGRatio;  /* threshold for adding new gaussian component to list */
  Ipp32f reduction;   /* speed of reduction non-active gaussian components */
  Ipp8u  shadowValue; /* returned shadow value */
  char   shadowFlag;  /* search shadows flag */
  Ipp32f shadowRatio; /* shadow threshold */
} IppFGMModel
```

The IppiMorphMode enumerator defines modes for mask processing at the second stage of advanced morphology operations:

```
typedef enum {
    IPP_MORPH_DEFAULT      = 0x0000,
    IPP_MORPH_MASK_NO_FLIP = 0x0001,
} IppiMorphMode;
```

The IppChannels enumerator defines the number of channels in the image:

```
typedef enum {
    ippC0   = 0,
    ippC1   = 1,
    ippC2   = 2,
    ippC3   = 3,
    ippC4   = 4,
    ippP2   = 5,
    ippP3   = 6,
    ippP4   = 7,
    ippAC1  = 8,
    ippAC4  = 9,
    ippA0C4 = 10,
    ippAP4  = 11
} IppChannels
```

The IppiFilterBilateralType enumerator defines the type of the bilateral filter that is used by some Filtering Functions:

```
typedef enum {
    ippiFilterBilateralGauss     = 100,
    ippiFilterBilateralGaussFast = 101
} IppiFilterBilateralType
```

The IppiWarpTransformType enumerator defines the type of the warp transform for some Warp Functions with Prior Initialization:

```
typedef enum {
    ippWarpAffine,
    ippWarpPerspective,
    ippWarpBilinear
} IppiWarpTransformType
```

The IppiDistanceMethodType structure stores the method of defining the difference in intensity between pixels. It is defines as:

```
typedef enum {
    ippDistNormL1 = 0x00000002;
    ippDistNormL2 = 0x00000004;
} IppiDistanceMethodType;
```

Structures for 3D Data Processing Functions

The IpprBorderType enumeration defines the border type that is used by some 3D Data Processing Functions:

```
typedef enum _IpprBorderType {
    ipprBorderRepl        = ippBorderRepl,
    ipprBorderConst       = ippBorderConst,


    /* Flags to use source image memory pixels from outside of the border in particular
directions */
    ipprBorderInMemTop     =  0x0010,
```

```
    ipprBorderInMemBottom  =  0x0020,
    ipprBorderInMemLeft    =  0x0040,
    ipprBorderInMemRight   =  0x0080,
    ipprBorderInMemFront   =  0x1000,
    ipprBorderInMemBack    =  0x2000,


    ipprBorderInMem        =  ipprBorderInMemLeft|ipprBorderInMemTop|ipprBorderInMemRight|
ipprBorderInMemBottom|ipprBorderInMemFront|ipprBorderInMemBack,
} IpprBorderType;
```

The `IpprVolumeL` structure stores the volume of a three-dimensional space. It is defined as:

```
typedef struct {
    int width;
    int height;
    int depth;
} IpprVolume;
```

The `IpprCuboid` structure stores the volume of interest of a three-dimensional space. It is defined as:

```
typedef struct {
    int x;
    int y;
    int z;
    int width;
    int height;
    int depth;
} IpprCuboid;
```

- Function Context Structures
- Structures and Enumerators for Platform-Aware Functions

Function Context Structures

Some Intel IPP functions use special structures to store function-specific (context) information. For example, the `IppiFFTSpec` structure stores twiddle factors and bit reverse indexes needed in computing the fast Fourier transform.

Two different kinds of structures are used: structures that are not modified during function operation - they have the suffix `Spec` in their names, and structures that are modified during operation - they have the suffix `State` in their names.

These context-related structures are not defined in the public headers, and their fields are not accessible. It was done because the function context interpretation is processor dependent. Thus, you may only use context-related functions and may not create a function context as an automatic variable.

Structures and Enumerators for Platform-Aware Functions

This topic describes the structures and enumerators used by the Intel IPP platform-aware functions for image processing.

The `IppiPointL` structure stores the geometric position of a point. It is defined as:

```
typedef struct {
    IppSizeL x;
    IppSizeL y;
} IppiPointL;
```

where `x`, `y` denote the coordinates of the point.

The IppiSizeL structure stores the size of a rectangle. It is defined as:

```
typedef struct {
    IppSizeL width;
    IppSizeL height;
} IppiSizeL;
```

where width and height denote the dimensions of the rectangle in the x- and y– directions, respectively.

The IppiRectL structure stores the geometric position and size of a rectangle. It is defined as:

```
typedef struct {
    IppSizeL x;
    IppSizeL y;
    IppSizeL width;
    IppSizeL height;
} IppiRectL;
```

where x, y denote the coordinates of the top left corner of the rectangle that has dimensions width in the x - direction by height in the y - direction

The IpprVolumeL structure stores the volume of a three-dimensional space. It is defined as:

```
typedef struct {
    IppSizeL width;
    IppSizeL height;
    IppSizeL depth;
} IpprVolumeL;
```

where width, height, and depth denote the dimensions of the three-dimensional space.

The IpprBorderType enumerator defines the border type that is used by some 3D Data Processing Functions:

```
typedef enum _IpprBorderType {
    ipprBorderRepl        = ippBorderRepl,
    ipprBorderConst       = ippBorderConst,


    ipprBorderInMemTop     =  0x0010,
    ipprBorderInMemBottom  =  0x0020,
    ipprBorderInMemLeft    =  0x0040,
    ipprBorderInMemRight   =  0x0080,
    ipprBorderInMemFront   =  0x1000,
    ipprBorderInMemBack    =  0x2000
} IpprBorderType;
```

Image Data Types and Ranges

The Intel IPP image processing functions support only absolute color images in which each pixel is represented by its channel intensities. The data storage for an image can be either pixel-oriented or plane-oriented (planar). For images in pixel order, all channel values for each pixel are clustered and stored consecutively, for example, RGBRGBRGB in case of an RGB image. The number of channels in a pixel-order image can be 1, 2, 3, or 4.

For images in planar order, all image data for each channel is stored contiguously followed by the next channel, for example, RRR…GGG…BBB.

Functions that operate on planar images are identified by the presence of Pn descriptor in their names. In this case, n pointers (one for each plane) may be specified.

The image data type is determined by the pixel depth in bits per channel, or bit depth. Bit depth for each channel can be 8, 16 or 32 and is included in the function name as one of these numbers (see Function Naming for details). The data may be signed (s), unsigned (u), or floating-point real (f). For some arithmetic

and FFT/DFT functions, data in complex format (`sc` or `fc`) can be used, where each channel value is represented by two numbers: real and imaginary part. All channels in an image must have the same data type.

For example, in an absolute color 24-bit RGB image, three consecutive bytes (24 bits) per pixel represent the three channel intensities in the pixel mode. This data type is identified in function names as the `8u_C3` descriptor, where `8u` represents 8-bit unsigned data for each channel and `C3` represents three channels.

Some functions operate with images in 16-bit packed RGB format (see RGB Image Formats in *Image Color Conversion* for more details). In this case data of all 3 channels are represented as `16u` data type.

For example, in an absolute color 16-bit packed RGB image, two consecutive bytes (16 bits) per pixel represent the three channel intensities in the pixel mode. This data type is identified in function names as the `16u_C3` descriptor, where `16u` represents 16-bit unsigned data (not a bit depth) for all packed channels together and `C3` stands for three channels.

If an alpha (opacity) channel is present in image data, the image must have four channels, with alpha channel being the last or the first one. This data type is indicated by the `AC4` or `A0C4` descriptors respectively. The presence of the alpha channel can modify the function's behavior. For such functions, Intel IPP provides versions with and without alpha. If an alpha channel is specified, the operation usually is not performed on that channel.

The range of values that can be represented by each data type lies between the lower and upper bounds. The following table lists data ranges and constant identifiers used in Intel IPP to denote the respective range bounds:

Data Type	Lower Bound Identifier	Lower Bound Value	Upper Bound Identifier	Upper Bound Value
8s	IPP_MIN_8S	-128	IPP_MAX_8S	127
8u		0	IPP_MAX_8U	255
16s	IPP_MIN_16S	-32768	IPP_MAX_16S	32767
16u		0	IPP_MAX_16U	65535
32s	IPP_MIN_32S	-2^{31}	IPP_MAX_32S	$2^{31}-1$
32u		0	IPP_MAX_32U	$2^{32}-1$
32f†	IPP_MINABS_32F	$1.175494351e^{-38}$	IPP_MAXABS_32F	$3.402823466e^{38}$

† The range for absolute values

Major Operation Models

Most Intel IPP image processing functions perform identical and independent operations on all channels of the processed image except for alpha channel. It means that the same operation is applied to each channel, and the computed results do not depend on values of other channels. Some exceptions include the ippiFilterMedianColor function and color conversion functions, which process three channels together.

Intel IPP image processing functions can be divided by two major models of operation:

- **Point operations**: functions operate on one pixel to compute the result, for example, ippiAdd
- **Neighborhood operations**: functions operate on a group of pixels, for example, ippiFilterBox

See Also

FilterMedianColor Filters an image using a color median filter.

Add Adds pixel values of two images.

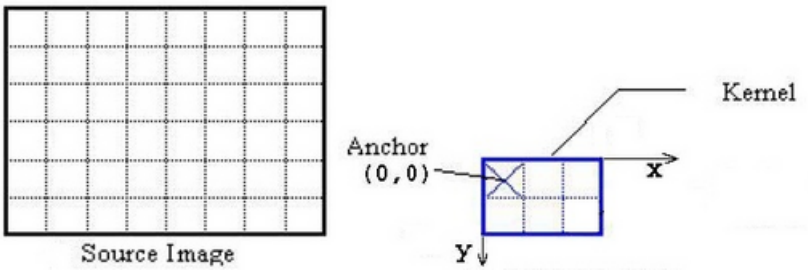
FilterBox Blurs an image using a simple box filter.

- Neighborhood Operations

- Regions of Interest in Intel IPP
- Tiled Image Processing

Neighborhood Operations

The result of a neighborhood operation is based on values of a certain group of pixels located near a given input pixel. The set of neighboring pixels is typically defined by a rectangular mask (or kernel) and anchor cell, specifying the mask alignment with respect to the position of the input pixel as shown in the following figure.



The anchor cell is a fixed cell within the kernel, which is used for positioning the kernel with respect to the currently processed pixel of the source image. The kernel is placed on the image in such a way that the anchor cell coincides with the input pixel. The anchor cell is specified by its coordinates `anchor.x` and `anchor.y` in the coordinate system associated with the top left corner of the kernel.

If position of the anchor cell is not specified explicitly in the function description, coordinates of the anchor are computed by the default formula:

```
anchor.x=(kernel.width-1)/2
anchor.y=(kernel.height-1)/2
```

where

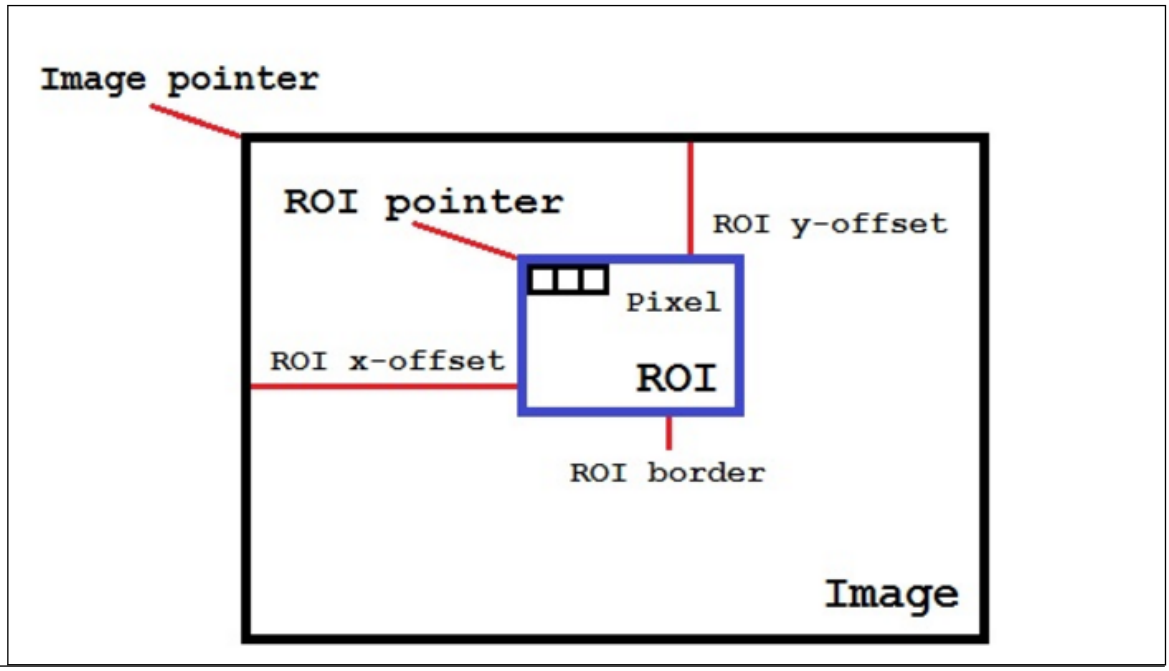
`kernel.width` and `kernel.height` is the width and height of the filter kernel, respectively.

Regions of Interest in Intel IPP

Most Intel IPP image processing functions operate not only on entire images but also on image areas. Image region of interest (ROI) is a rectangular area that may be either some part of the image or the whole image.

The Intel IPP functions with ROI support have an `R` descriptor in their names. ROI of an image is defined by the size and offset from the image origin as shown in figure Image, ROI, and Offsets. The origin of an image is in the top left corner, with `x` values increasing from left to right and `y` values increasing downwards.

Image, ROI, and Offsets



Both the source and destination images can have a ROI. In such cases the sizes of ROIs are assumed to be the same while offsets may differ. Image processing is performed on data of the source ROI, and the results are written to the destination ROI. In function call sequences, ROI is specified by:

- `roiSize` parameter of the `IppiSize` type
- `pSrc` and `pDst` pointers to the starts of source and destination ROI buffers
- `srcStep` and `dstStep` parameters that are equal to distances in bytes between the starting points of consecutive lines in source and destination images, respectively.

Thus, the `srcStep` and `dstStep` parameters set steps in bytes through image buffers to start processing a new line in the ROI of an image.

The following code example illustrates the use of the `dstStep` parameter in function calls:

Example

```
IppStatus roi( void ) {
   Ipp8u x[8*3] = {0};
   IppiSize roiSize = {3,2};
   IppiPoint roiPoint = {2,1};
   /// place the pointer to the ROI start position
   return ippiSet_8u_C1R( 7, x+8*roiPoint.y+roiPoint.x, 8, roiSize );
}
```

The resulting image `x` contains the following data:

```
00 00 00 00 00 00 00 00
00 00 07 07 07 00 00 00
00 00 07 07 07 00 00 00
```

If ROI is present:

- source and destination images can have different sizes;

- lines may have padding at the end for aligning the line sizes;
- application must correctly define the `pSrc`, `pDst`, and `roiSize` parameters.

The `pSrc` and `pDst` parameters are the shifted pointers to the image data. For example, in case of ROI operations on 3-bytes-per-pixel image data (`8u_C3R`), `pSrc` points to the start of the source ROI buffer and can be interpreted as follows:

```
pSrc=pSrcImg+ 3*(srcImgSize.width*srcRoiOffset.y+srcRoiOffset.x),
```

where

`pSrcImg`	points to the start of the source image buffer
`srcImgSize`	is the image size in pixels (of the `IppiSize` type)
`srcRoiOffset`	determines an offset of ROI relative to the start of the image as shown in Figure Image, ROI, and Offsets.

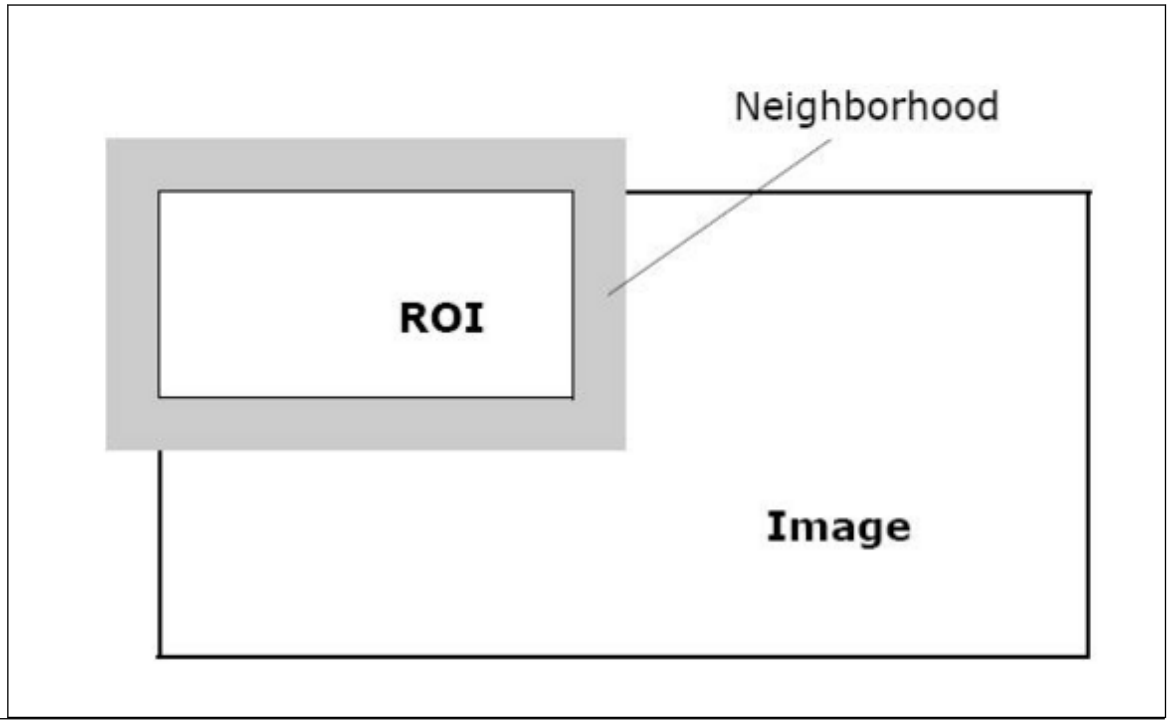
Another example for operations on four-channel image of 32-bit floating point data type `32f_AC4`:

```
pSrc= (Ipp32f*)((Ipp8u*)pSrcImg+srcImgStep*srcRoiOffset.y+
4*SizeOf(Ipp32f)*srcRoiOffset.x.
```

In this example the multiplier `4` is used because the AC4 pixel consists of 4 values - R, G, B, A. Pointer type conversion is required as in Intel IPP all image steps are always in bytes, and it is not recommended to use `step/SizeOf(Ipp32f)` as in a general case step value may be not a multiple of 4.

For functions using ROI with a neighborhood, you should correctly use values of the `pSrc` and `roiSize` parameters. These functions assume that the points in the neighborhood exist and that therefore `pSrc` is almost never equal to `pSrcImg`. Figure Using ROI with Neighborhood illustrates the case when neighborhood pixels can fall outside the source image.

Using ROI with Neighborhood



To ensure valid operation when image pixels are processed, the application should correctly define additional border pixels (see Borders in Neighborhood Operations).

NOTE
Warning
If the required border pixels are not defined prior to calling neighborhood functions that attempt to process such pixels, you may get memory violation errors.

The following code example shows how to process an image with ROI:

Example

```
IppStatus alignedLine( void
) {
   Ipp8u x[8*3] = {0};
   IppiSize imgSize = {5,3};
   /// The image is of size 5x3. Width 8 has been
   /// chosen by the user to align every line of the image
   return ippiSet_8u_C1R( 7, x, 8, imgSize);
}
```

The resulting image `x` contains the following data:

```
07 07 07 07 07 00 00 00
07 07 07 07 07 00 00 00
07 07 07 07 07 00 00 00
```

See Also

Borders in Neighborhood Operations

Tiled Image Processing

Intel IPP can process images composed from tiles, or tiled images.

Support Functions

This chapter describes the Intel® IPP support functions that are used to:

- retrieve information about the current Intel IPP software version
- get a brief explanation of the returned status codes
- allocate and free memory that is needed for the operation of other Intel IPP image and video processing functions
- execute Intel IPP Threading Layer service routines.

- Version Information Function
- Status Information Function
- Memory Allocation Functions
- Threading Layer Functions

Version Information Function

This function returns the version number and other information about the active Intel IPP image processing software.

- GetLibVersion

GetLibVersion

Returns information about the used version of Intel IPP software for image processing.

Syntax

```
const IppLibraryVersion* ippiGetLibVersion(void);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Description

This function returns a pointer to a static data structure `IppLibraryVersion` that contains information about the current version of Intel IPP for image processing. You need not release memory referenced by the returned pointer, as it points to a static variable. The following fields of the `IppLibraryVersion` structure are available:

`major`	the major number of the current library version
`minor`	the minor number of the current library version
`majorBuild`	the number of builds of the major version
`build`	current build number
`targetCpu[4]`	Intel® processor.
`Name`	the name of the current library version
`Version`	the library version string
`BuildDate`	the library version actual build date

For example, if the library version is 9.0, build revision number is 49671, library name is ippIP AVX2, target CPU is processor with Intel® Advanced Vector Extensions 2 (Intel® AVX2) and build date is "Dec 7 2015", then the fields in this structure are set as:

`major = 9`, `minor = 0`, `Name` = "ippIP AVX2", `Version` = "9.0.1 (r49671)", `targetCpu[4]` ="h9", `BuildDate` = "Dec 7 2015"

> **NOTE** Each sub-library that is used in the image processing domain has its own similar function to retrieve information about the active library version. These functions are: `ippiGetLibVersion`, `ippcvGetLibVersion`, and `ippccGetLibVersion`. They are declared in the following header files: `ippcore.h`, `ippcv.h`, `ippcc.h`, respectively, and have the same interface as the above described function.

Status Information Function

Use this function to get a brief description of the status code returned by the current Intel IPP software.

- ippGetStatusString

ippGetStatusString

Translates a status code into a message.

Syntax

```
const char* ippGetStatusString(IppStatus StsCode);
```

Include Files

`ippcore.h`

Parameters

`StsCode`	Code that indicates the status type (see Table Status Codes and Messages)

Description

This function returns a pointer to the text string associated with a status code `StsCode`. Use this function to produce error and warning messages. The returned pointer is a pointer to an internal static buffer and needs not be released.

The status information function translates this code into the corresponding message `Null Pointer Error`:

Example

A code example below shows how to use the `ippGetStatusString` function. If you call an Intel IPP function, in this example `ippiSet_8u_C1R`, with a `NULL` pointer, it returns an error code `-8`.

```
void statusInfo( void ) {
    IppiSize roi = {0};
    IppStatus st = ippiSet_8u_C1R(3, 0, 0, roi );
    printf( " %d : %s\n", st, ippGetStatusString( st ));
}
```

Output:

```
-8, Null Pointer Error
```

Memory Allocation Functions

This topic describes the Intel IPP functions that allocate aligned memory blocks for data of required type, or free previously allocated memory.

> **NOTE** The only function to free the memory allocated by any of these functions is `ippiFree()`.

- Malloc
- Free

Malloc

Allocates memory aligned to 64-byte boundary.

Syntax

Case 1: Memory allocation for blocks of 32-bit sizes

```
Ipp<datatype>* ippiMalloc_<mod>(int widthPixels, int heightPixels, int* pStepBytes);
```

Supported values for `mod`:

8u_C1	16u_C1	16s_C1	32s_C1	32f_C1	32sc_C1	32fc_C1
8u_C2	16u_C2	16s_C2	32s_C2	32f_C2	32sc_C2	32fc_C2
8u_C3	16u_C3	16s_C3	32s_C3	32f_C3	32sc_C3	32fc_C3
8u_C4	16u_C4	16s_C4	32s_C4	32f_C4	32sc_C4	32fc_C4
8u_AC4	16u_AC4	16s_AC4	32s_AC4	32f_AC4	32sc_AC4	32fc_AC4

Case 2: Memory allocation for platform-aware functions

```
Ipp<datatype>* ippiMalloc_<mod>(IppSizeL widthPixels, IppSizeL heightPixels, IppSizeL*
pStepBytes);
```

Supported values for `mod`:

8u_C1_L	16u_C1_L	16s_C1_L	32s_C1_L	32f_C1_L	32sc_C1_L	32fc_C1_L
8u_C2_L	16u_C2_L	16s_C2_L	32s_C2_L	32f_C2_L	32sc_C2_L	32fc_C2_L
8u_C3_L	16u_C3_L	16s_C3_L	32s_C3_L	32f_C3_L	32sc_C3_L	32fc_C3_L
8u_C4_L	16u_C4_L	16s_C4_L	32s_C4_L	32f_C4_L	32sc_C4_L	32fc_C4_L
8u_AC4_L	16u_AC4_L	16s_AC4_L	32s_AC4_L	32f_AC4_L	32sc_AC4_L	32fc_AC4_L

Include Files

`ippi.h`

Flavors with the `_L` suffix `ippi_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`widthPixels`	Width of an image, in pixels.
`heightPixels`	Height of an image, in pixels.
`pStepBytes`	Pointer to the distance, in bytes, between the starting points of consecutive lines in the image

Description

This function allocates a memory block aligned to 64-byte boundary for elements of different data types. Every line of the image is aligned in accordance with the `pStepBytes` parameter, which is calculated by the `Malloc` function and returned for further use.

The function `Malloc` allocates one continuous memory block. Functions that operate on planar images require an array of separate pointers (`IppType* plane[3]`) to each plane as an input. In this case, you should call the `Malloc` function three times.

The function `ippiMalloc_<dataType>` uses a 32-bit length, allowing it to allocate memory up to approximately 2GB. If a larger memory size is requested, the function will return a `NULL` pointer. To allocate memory exceeding this limit, use the platform-aware `Malloc` function (`ippiMalloc_<dataType>_L`).

Example

The code example below demonstrates how to construct an array and set correct values to the pointers to use the allocated memory block with the Intel IPP functions operating on planar images. You need to specify `pStepBytes` for each plane. The example is given for the `8u` data type.

```
int stepBytes[3];
Ipp8u* plane[3];
   plane[0] = ippiMalloc_8u_C1(widthPixels, heightPixels,
                               &(stepBytes [0]));
   plane[1] = ippiMalloc_8u_C1(widthPixels/2, heightPixels/2,
                               &(stepBytes [1]));
   plane[2] = ippiMalloc_8u_C1(widthPixels/2, heightPixels/2,
                               &(stepBytes [2]));
```

Return Values

The return value of `Malloc` function is a pointer to an aligned memory block.

If no memory is available in the system, the `NULL` value is returned.

To free the allocated memory block, use the `Free` function.

Free

Frees memory allocated by the function `ippiMalloc`.

Syntax

```
void ippiFree(void* ptr);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

ptr	Pointer to a memory block to be freed. This block must have been previously allocated by the function ippiMalloc.

Description

This function frees the aligned memory block allocated by the function ippiMalloc.

NOTE The function ippiFree cannot be used to free memory allocated by standard functions like malloc or calloc, nor can the memory allocated by ippiMalloc be freed by free.

Threading Layer Functions

This section described the Intel IPP Threading Layer (TL) functions for image processing. For more information about the TL functions usage, refer to Using Intel® Integrated Performance Primitives Threading Layer (TL) Functions.

- SplitUniform2D
- ParallelFor
- GetTilePointer
- GetTileParamsByIndex
- GetThreadingType
- GetThreadIdx

SplitUniform2D

Splits an image into tiles.

Syntax

Case 1: Operation with TL functions based on the Platform Aware API

```
IppStatus ippiSplitUniform2D_LT(IppiSizeL roiSize, IppSizeL minItemNumber, IppiPointL*
pSplit, IppiSizeL* pTileSize, IppiSizeL* pTailSize);
```

Case 2: Operation with TL functions based on the Classic API

```
IppStatus ippiSplitUniform2D_T(IppiSize roiSize, int minItemNumber, IppiPoint* pSplit,
IppiSize* pTileSize, IppiSize* pTailSize);
```

Include Files

ippi_tl.h

Parameters

roiSize	Image ROI size.
minItemNumber	Minimal size of a tile, in pixels.
pSplit	Number of split parts along x and y axes.
pTileSize	Size of a tile.

pTailSize	Size of the last corner tile.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function splits an image into tiles.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition when the pSplit, pTileSize, or pTailSize pointer is NULL.
ippStsSizeErr	Indicates an error condition when minItemNumber is less than zero.

ParallelFor

Performs parallel iterations for a processing function.

Syntax

Case 1: Operation with TL functions based on the Platform Aware API

```
IppStatus ippParallelFor_LT(IppSizeL numTiles, void* arg, functype_l func);
```

Case 2: Operation with TL functions based on the Classic API

```
IppStatus ippParallelFor_T(int numTiles, void* arg, functype func);
```

Include Files

ippcore_tl.h

Parameters

numTiles	Number of tiles.
arg	Pointer to the structure that contains arguments for the processing function.
func	Pointer to the processing function used in the “parallel for” loop.

Description

This function performs parallel iterations of a processing function, which is passed as an argument for each tile.

Return Values

ippStsNoErr	Indicates no error.

	Any IPP error that the processing function can return.

GetTilePointer

Returns a pointer to the specified image tile.

Syntax

Case 1: Operation with TL functions based on the Platform Aware API

```
IppStatus ippiGetTilePointer_32f_LT(const Ipp32f* pSrc, Ipp32f** pDst, IppSizeL srcStep, IppSizeL x, IppSizeL y, Ipp32s numChannels);
```

```
IppStatus ippiGetTilePointer_64f_LT(const Ipp64f* pSrc, Ipp64f** pDst, IppSizeL srcStep, IppSizeL x, IppSizeL y, Ipp32s numChannels);
```

Case 2: Operation with TL functions based on the Classic API

```
IppStatus ippiGetTilePointer_32f_T(const Ipp32f* pSrc, Ipp32f** pDst, int srcStep, int x, int y, Ipp32s numChannels);
```

```
IppStatus ippiGetTilePointer_64f_T(const Ipp64f* pSrc, Ipp64f** pDst, int srcStep, int x, int y, Ipp32s numChannels);
```

Include Files

`ippi_tl.h`

Parameters

`pSrc`	Pointer to the source image.
`pDst`	Pointer to the memory location of the pointer to the destination image.
`srcStep`	Distance in bytes between consecutive lines in the source image.
`x`	x coordinate of a tile, in pixels.
`y`	y coordinate of a tile, in pixels.
`numChannels`	Number of channels in the image. Possible values are 1, 3, and 4.

Description

This function returns a pointer to the specified image tile.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition when the `pSrc` or `pDst` pointer is `NULL`.

ippStsSizeErr	Indicates an error condition when x or y is less than zero.
ippStsNumChannelsErr	Indicates an error condition when numChannels has an illegal value.

GetTileParamsByIndex

Returns the offset and size of a tile by a given index.

Syntax

Case 1: Operation with TL functions based on the Platform Aware API

```
IppStatus ippiGetTileParamsByIndex_LT(IppSizeL index, IppiPointL splitImage, IppiSizeL tileSize, IppiSizeL tailSize, IppiPointL* pTileOffset, IppiSizeL* pTileSize);
```

Case 2: Operation with TL functions based on the Classic API

```
IppStatus ippiGetTileParamsByIndex_T(int index, IppiPoint splitImage, IppiSize tileSize, IppiSize tailSize, IppiPoint* pTileOffset, IppiSize* pTileSize);
```

Include Files

ippi_tl.h

Parameters

index	Ordinal index of a tile.
splitImage	Split of the image by x and y axis correspondingly.
tileSize	Size of a tile.
tailSize	Size of the last bottom right tile.
pTileOffset	Offset of the tile corresponding to the top left image corner.
pTileSize	Size of a tile.

Description

This function returns the offset and size of a tile by a given index.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition when the pTileOffset or pTileSize pointer is NULL.

GetThreadingType

Returns type of the threading layer.

Syntax

Case 1: Operation with TL functions based on the Platform Aware API

```
IppStatus ippGetThreadingType_LT(IppThreadingType* thrType);
```

Case 2: Operation with TL functions based on the Classic API

```
IppStatus ippGetThreadingType_T(IppThreadingType* thrType);
```

Include Files

`ippcore_tl.h`

Parameters

`thrType`	Pointer to the threading type.

Description

This function returns `OMP` if the OpenMP* Threading Layer and `TBB` if the TBB* Threading Layer is used.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition when the `thrType` pointer is `NULL`.

GetThreadIdx

Returns a unique thread identification number.

Syntax

Case 1: Operation with TL functions based on the Platform Aware API

```
IppStatus ippGetThreadIdx_LT(int* pThrIdx);
```

Case 2: Operation with TL functions based on the Classic API

```
IppStatus ippGetThreadIdx_T(int* pThrIdx);
```

Include Files

`ippcore_tl.h`

Parameters

`pThrIdx`	Pointer to the index of a thread.

Description

This function returns a unique thread identification number.

Return Values

`ippStsNoErr`	Indicates no error.

ippStsNullPtrErr	Indicates an error condition when the pThrIdx pointer is NULL.

Image Data Exchange and Initialization Functions

This chapter describes the Intel® IPP image processing functions that perform image data manipulation, exchange and initialization operations.

- Convert
- BinToGray, GrayToBin
- Scale
- ScaleC
- Set
- Copy
- CopyManaged
- CopyConstBorder
- CopyMirrorBorder
- CopyReplicateBorder
- CopyWrapBorder
- CopySubpix
- CopySubpixIntersect
- Dup
- Transpose
- SwapChannels
- AddRandUniform
- AddRandGauss
- ImageJaehne
- ImageRamp
- SampleLine
- ZigzagFwd8x8
- ZigzagInv8x8

Convert

Converts image pixel values from one data type to another.

Syntax

Case 1: Conversion to increase bit depth and change signed to unsigned type

```
IppStatus ippiConvert_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u16u_C1R	8u16s_C1R	8u32s_C1R	8u32f_C1R	8s32s_C1R
8u16u_C3R	8u16s_C3R	8u32s_C3R	8u32f_C3R	8s32s_C3R
8u16u_C4R	8u16s_C4R	8u32s_C4R	8u32f_C4R	8s32s_C4R
8s32f_C1R	16u32s_C1R	16u32f_C1R	16s32s_C1R	16s32f_C1R
8s32f_C3R	16u32s_C3R	16u32f_C3R	16s32s_C3R	16s32f_C3R
8s32f_C4R	16u32s_C4R	16u32f_C4R	16s32s_C4R	16s32f_C4R

8s8u_C1Rs	16s16u_C1Rs	16u32u_C1R	32s32u_C1Rs	32u32f_C1R
8s16u_C1Rs	16s32u_C1Rs		32s32f_C1R	
8s16s_C1R				
8s32u_C1Rs				
8s64f_C1R				

```
IppStatus ippiConvert_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

8u16u_AC4R	8u16s_AC4R	8u32s_AC4R	8u32f_AC4R	8s32s_AC4R
8s32f_AC4R	16u32s_AC4R	16u32f_AC4R	16s32s_AC4R	16s32f_AC4R

Case 2: Conversion to reduce bit depth and change unsigned to signed type: integer to integer type

```
IppStatus ippiConvert_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

16u8u_C1R	16s8u_C1R	32s8u_C1R	32s8s_C1R
16u8u_C3R	16s8u_C3R	32s8u_C3R	32s8s_C3R
16u8u_C4R	16s8u_C4R	32s8u_C4R	32s8s_C4R

```
IppStatus ippiConvert_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize, IppRoundMode roundMode, int
scaleFactor);
```

Supported values for `mod`:

8u8s_C1RSfs	16u8s_C1RSfs	32u8u_C1RSfs	32s16u_C1RSfs
	16s8s_C1RSfs	32u8s_C1RSfs	32s16s_C1RSfs
	16u16s_C1RSfs	32u16u_C1RSfs	
		32u16s_C1RSfs	
		32u32s_C1RSfs	

```
IppStatus ippiConvert_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

16u8u_AC4R	16s8u_AC4R	32s8u_AC4R	32s8s_AC4R

Floating point to integer type:

```
IppStatus ippiConvert_<mod>(const Ipp32f* pSrc, int srcStep, Ipp<dstDatatype>* pDst,
int dstStep, IppiSize roiSize, IppRoundMode roundMode);
```

Supported values for `mod`:

32f8u_C1R	32f8s_C1R	32f16u_C1R	32f16s_C1R
32f8u_C3R	32f8s_C3R	32f16u_C3R	32f16s_C3R
32f8u_C4R	32f8s_C4R	32f16u_C4R	32f16s_C4R

```
IppStatus ippiConvert_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize, IppRoundMode roundMode, int
scaleFactor);
```

Supported values for `mod`:

32f8u_C1RSfs	32f8s_C1RSfs	32f16u_C1RSfs	32f16s_C1RSfs	32f32u_C1RSfs	32f32s_C1RSfs
64f8u_C1RSfs	64f8s_C1RSfs	64f16u_C1RSfs	64f16s_C1RSfs		

```
IppStatus ippiConvert_32f32u_C1IRSfs(Ipp32u* pSrcDst, int srcDstStep, IppiSize roiSize,
IppRoundMode roundMode, int scaleFactor);
```

```
IppStatus ippiConvert_<mod>(const Ipp32f* pSrc, int srcStep, Ipp<dstDatatype>* pDst,
int dstStep, IppiSize roiSize, IppRoundMode roundMode);
```

Supported values for `mod`:

32f8u_AC4R	32f8s_AC4R	32f16u_AC4R	32f16s_AC4R

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance, in bytes, between the starting points of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI.
`pSrcDst`	Pointer to the source and destination image ROI for in-place operation.
`dstStep`	Distance, in bytes, between the starting points of consecutive lines in the destination image.
`srcDstStep`	Distance, in bytes, between the starting points of consecutive lines in the source and destination image for in-place operation.

roiSize	Size of the source and destination ROI in pixels.
scaleFactor	Scale factor (see Integer Result Scaling).
roundMode	Rounding mode, the following values are possible: • ippRndZero • specifies that floating-point values are truncated to zero, • ippRndNear • specifies that floating-point values are rounded to the nearest even • ippRndFinancial • specifies that floating-point values are rounded down to the nearest

Description

This function operates with ROI.

This function converts pixel values in the source image ROI pSrc to a different data type and writes them to the destination image ROI pDst.

The result of integer operations is always saturated to the destination data type range. It means that if the value of the source pixel is out of the data range of the destination image, the value of the corresponding destination pixel is set to the value of the lower or upper bound (minimum or maximum) of the destination data range:

```
x=pSrc[i,j]
if (x> MAX_VAL)x= MAX_VAL
if (x< MIN_VAL)x= MIN_VAL
pDst[i,j] = (CASTING)x
```

If you want to shift data from the signed range to the unsigned range and vice-versa, see "Application Notes" below.

The function flavors with the Sfs descriptor in their names perform scaling of the internally computed results in accordance with the scaleFactor parameter.

When converting from floating-point to integer type, rounding defined by roundMode is performed, and the result is saturated to the destination data type range.

NOTE The bit order of each byte in the source image is inverse to the pixel order. It means that the first pixel in a row represents the last (seventh) bit of the first byte in a row.

Application Notes

When data is converted from the signed integer to the corresponding unsigned integer and vice versa (8s –>8u, 16u –> 16s), the pixel information may be lost because all negative values will be set to zero (signed-unsigned conversion), or unsigned values from the high half of the range will be set to the maximum value of the signed range (unsigned - signed conversion).

If you need just to shift the data from the signed range to the unsigned range and vice versa, use the function `ippiXorC` with the parameter `value` specified in such a way that the most significant bit is set to 1, and all other bits are set to 0. For example, if you want to convert pixel values from `Ipps16s` type to `Ipp16u` type with the rage shift call the function:

ippiXorC_16u_C1R((Ipp16u *)pSrc, srcStep, 0x8000, pDst, dstStep, roiSize);

In this case the pixels values are converted as follows:

```
-32768 --> 0
-32767 --> 1
...
-1 --> 32767
0 --> 32768
1 --> 32769
...
32766 --> 65534
32767 --> 65535
```

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error when any of the specified pointers is `NULL`, with the exception of second mode in Case 4.
`ippStsSizeErr`	Indicates an error when `roiSize` has a field with zero or negative value, or `srcBitOffset`/ `dstBitOffset` is less than zero.
`ippStsStepErr`	Indicates an error when `srcStep` or `dstStep` has a zero or negative value.
`ippStsMemAllocErr`	Indicates an error when memory allocation fails.

Example

The code example below shows data conversion without scaling.

```
  IppStatus convert( void ) {


     IppiSize roi={5,4};
     Ipp32f x[5*4];


     Ipp8u y[5*4];
     ippiSet_32f_C1R( -1.0f, x, 5*sizeof(Ipp32f), roi );
           x[1] = 300; x[2] = 150;
```

```
return ippiConvert_32f8u_C1R( x, 5*sizeof(Ipp32f), y, 5, roi, ippRndNear );
}
```

The destination image `y` contains:

```
00 FF 96 00 00
00 00 00 00 00
00 00 00 00 00
00 00 00 00 00
```

See Also

Integer Result Scaling

Regions of Interest in Intel IPP

BinToGray, GrayToBin

Converts a bitonal image to a grayscale image and vice versa.

Syntax

Case 1: Conversion of a bitonal image to a grayscale image

```
IppStatus ippiBinToGray_1u<dstDataType>_C1R(const Ipp8u* pSrc, int srcStep, int
srcBitOffset, Ipp<dstDataType>* pDst, int dstStep, IppiSize roiSize, Ipp<dstDataType>
loVal, Ipp<dstDataType> hiVal);
```

Supported values for `dstDataType`:

8u	16u	16s	32f

Case 2: Conversion of a grayscale image to a bitonal image

```
IppStatus ippiGrayToBin_<srcDataType>1u_C1R(const Ipp<srcDataType>* pSrc, int srcStep,
Ipp8u* pDst, int dstStep, int dstBitOffset, IppiSize roiSize, Ipp<sourceDataType>
threshold);
```

Supported values for `srcDataType`:

8u	16u	16s	32f

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance, in bytes, between the starting points of consecutive lines in the source image.

srcBitOffset	Offset, in bits, from the first byte of the source image row.
pDst	Pointer to the destination image ROI.
dstStep	Distance, in bytes, between the starting points of consecutive lines in the destination image.
dstBitOffset	Offset, in bits, from the first byte of the destination image row.
roiSize	Size of the ROI in pixels.
loVal	Destination value that corresponds to the "0" value of the corresponding source element.
hiVal	Destination value that corresponds to the "1" value of the corresponding source element.
threshold	Threshold level.

Description

These functions operate with ROI.

The ippiBinToGray function converts a bitonal image to grayscale, and the ippiGrayToBin function converts a grayscale image to bitonal. The data type of the bitonal image is 8u. It means that each byte consists of eight consecutive pixels of the image (1 bit per pixel). You need to specify the start position of the ROI buffer in the srcBitOffset and dstBitOffset parameters.

The ippiBinToGray function transforms each bit of the source image into the pixel of the destination image in the following way:

- If the input pixel is equal to 0, the corresponding output pixel is set to loVal.
- If the input pixel is equal to 1, the corresponding output pixel is set to hiVal.

The ippiGrayToBin function transforms each pixel of the source image into the bit of the destination image in the following way:

- If the input pixel is more than the threshold value, the corresponding output bit is set to 1.
- If the input pixel is less than, or equal to the threshold value, the corresponding output bit is set to 0.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.
ippStsStepErr	Indicates an error when the srcStep or dstStep value is less than, or equal to zero.
ippStsSizeErr	Indicates an error when:

See Also

Regions of Interest in Intel IPP

Scale

Scales pixel values of an image and converts them to another bit depth.

Syntax

Case 1: Scaling with conversion to integer data of increased bit depth

```
IppStatus ippiScale_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp<dstDatatype>*
pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

8u16u_C1R	8u16s_C1R	8u32s_C1R
8u16u_C3R	8u16s_C3R	8u32s_C3R
8u16u_C4R	8u16s_C4R	8u32s_C4R

```
IppStatus ippiScale_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp<dstDatatype>*
pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

8u16u_AC4R	8u16s_AC4R	8u32s_AC4R

Case 2: Scaling with conversion to floating-point data

```
IppStatus ippiScale_<mod>(const Ipp8u* pSrc, int srcStep, Ipp32f* pDst, int dstStep,
IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);
```

Supported values for `mod`:

8u32f_C1R
8u32f_C3R
8u32f_C4R

```
IppStatus ippiScale_8u32f_AC4R(const Ipp8u* pSrc, int srcStep, Ipp32f* pDst, int
dstStep, IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);
```

Case 3: Scaling of integer data with conversion to reduced bit depth

```
IppStatus ippiScale_<mod> (const Ipp<srcDatatype>* pSrc, int srcStep, Ipp<dstDatatype>*
pDst, int dstStep, IppiSize roiSize, IppHintAlgorithm hint);
```

Supported values for `mod`:

16u8u_C1R	16s8u_C1R	32s8u_C1R
16u8u_C3R	16s8u_C3R	32s8u_C3R
16u8u_C4R	16s8u_C4R	32s8u_C4R

```
IppStatus ippiScale_<mod> (const Ipp<srcDatatype>* pSrc, int srcStep, Ipp<dstDatatype>*
pDst, int dstStep, IppiSize roiSize, IppHintAlgorithm hint);
```

Supported values for `mod`:

16u8u_AC4R	16s8u_AC4R	32s8u_AC4R

Case 4: Scaling of floating-point data with conversion to integer data type

```
IppStatus ippiScale_<mod>(const Ipp32f* pSrc, int srcStep, Ipp8u* pDst, int dstStep,
IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);
```

Supported values for `mod`:

`32f8u_C1R`
`32f8u_C3R`
`32f8u_C4R`

```
IppStatus ippiScale_32f8u_AC4R(const Ipp32f* pSrc, int srcStep, Ipp8u* pDst, int
dstStep, IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance, in bytes, between the starting points of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance, in bytes, between the starting bytes of consecutive lines in the destination image.
`roiSize`	Size of the source and destination ROI in pixels.
`vMin`, `vMax`	Minimum and maximum values of the input data.
`hint`	Option to select the algorithmic implementation of the function (see Hint Arguments for Image Moment Functions).

Description

This function operates with ROI.

This function converts pixel values of a source image ROI `pSrc` to the destination data type, using a linear mapping. The computation algorithm is specified by the `hint` argument. For conversion between integer data types, the whole range [`src_Min..src_Max`] of the input data type is mapped onto the range [`dst_Min..dst_Max`] of the output data type.

The source pixel `p` is mapped to the destination pixel `p'` by the following formula:

```
p'=dst_Min+k*(p-src_Min)
```

where

```
k= (dst_Max-dst_Min)/(src_Max-src_Min)
```

For conversions to and from floating-point data type, the user-defined floating-point data range [vMin..vMax] is mapped onto the source or destination data type range.

If the conversion is from Ipp32f type and some of the input floating-point values are outside the specified input data range [vMin..vMax], the corresponding output values saturate. To determine the actual floating-point data range in your image, use the ippiMinMax function.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with a zero or negative value.
ippStsStepErr	Indicates an error condition if srcStep or dstStep has a zero or negative value.
ippStsScaleRangeErr	Indicates an error condition if the input data bounds are incorrect, that is, vMax is less than or equal to vMin .

Example

The code example below shows how to use scaling to preserve the data range.

```
IppStatus scale( void ) {
   IppiSize roi = {5,4};
   Ipp32f x[5*4];
   Ipp8u y[5*4];
   ippiSet_32f_C1R( -1.0f, x, 5*sizeof(Ipp32f), roi );
   x[1] = 300; x[2] = 150;
   return ippiScale_32f8u_C1R( x, 5*sizeof(Ipp32f), y, 5, roi, -1, 300 );
}
```

The destination image y contains:

```
00 FF 80 00 00
00 00 00 00 00
00 00 00 00 00
00 00 00 00 00
```

See Also

Regions of Interest in Intel IPP

Image Moments

Intel® Integrated Performance Primitives Concepts

MinMax Computes the minimum and maximum of image pixel values.

ScaleC

Scales pixel values of an image and converts them to another bit depth.

Syntax

Case 1: Not-in-place operation

```
IppStatus ippiScaleC_<mod>_C1R(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp64f mVal,
Ipp64f aVal, Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize, IppHintAlgorithm
hint);
```

Supported values for `mod`:

8u	8u8s	8u16u	8u16s	8u32s	8u32f	8u64f
8s8u	8s	8s16u	8s16s	8s32s	8s32f	8s64f
16u8u	16u8s	16u	16u16s	16u32s	16u32f	16u64f
16s8u	16s8s	16s16u	16s	16s32s	16s32f	16s64f
32s8u	32s8s	32s16u	32s16s	32s	32s32f	32s64f
32f8u	32f8s	32f16u	32f16s	32f32s	32f	32f64f
64f8u	64f8s	64f16u	64f16s	64f32s	64f32f	64f

where the first value is `srcDatatype` and the second value is `dstDatatype`.

Case 2: In-place operation

```
IppStatus ippiScaleC_<mod>_C1IR(const Ipp<datatype>* pSrcDst, int srcDstStep, Ipp64f
mVal, Ipp64f aVal, IppiSize roiSize, IppHintAlgorithm hint);
```

Supported values for `mod`:

8u	8s	16u	16s	32s	32f	64f

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`pSrcDst`	Pointer to the source and destination buffer or an array of pointers to separate source and destination color planes for in-place operation.
`srcDstStep`	Distance, in bytes, between the starting points of consecutive lines in the source and destination image for in-place operation.
`srcStep`	Distance, in bytes, between the starting points of consecutive lines in the source image.
`mVal`	Value of the multiplier used for scaling.

aVal	Offset value for scaling.
pDst	Pointer to the destination image ROI.
dstStep	Distance, in bytes, between the starting bytes of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI, in pixels.
hint	Option to select the algorithmic implementation of the function. Supported values are ippAlgHintFast (default) and ippAlgHintAccurate.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function scales pixel values of the source image ROI and converts them to the destination data type according to the following formula:

```
dst = saturate_to_dstType(src * mVal + aVal)
```

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when roiSize has a field with a zero or negative value.
ippStsStepErr	Indicates an error when the step value is less than, or equal to zero.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

//    The code example below illustrates how to use the ippiScaleC function.

#include <stdio.h>
```

```
#include "ipp.h"

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define
check_sts(st)
                                  \
    if ((st) !=
ippStsNoErr)
                           \
        goto exitLine; /* Go to Exit if Intel(R) Integrated Performance Primitives (Intel(R)
IPP) function returned status different from \
                            ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main(void)
{
    IppStatus status = ippStsNoErr;

    Ipp32u seed = 0;
    IppiSize roiSize = {8, 8}; /* Image sizes */
    Ipp8u pSrc[8 * 8] = {0};   /* Pointers to source images */
    Ipp16s pDst[8 * 8];        /* Pointers to destination images */
    Ipp64f mVal, aVal;
    Ipp64f minSrc = IPP_MIN_8U, maxSrc = IPP_MAX_8U;
    Ipp64f minDst = IPP_MIN_16S, maxDst = IPP_MAX_16S;

    /* set random image  */
    check_sts(status = ippiAddRandUniform_8u_C1IR(pSrc, roiSize.width * sizeof(*pSrc), roiSize,
IPP_MIN_8U, IPP_MAX_8U, &seed));

    /* calculate coefficient for scaling */
    mVal = (maxDst - minDst) / (maxSrc - minSrc);
    aVal = minDst - minSrc * mVal;

    /* do scaling */
    check_sts(
        status = ippiScaleC_8u16s_C1R(pSrc, roiSize.width * sizeof(*pSrc), mVal, aVal, pDst,
roiSize.width * sizeof(*pDst), roiSize, ippAlgHintFast));

    EXIT_MAIN
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

See Also

Regions of Interest in Intel IPP

Set

Sets pixels of an array to a constant value.

Syntax

Case 1: Setting one-channel data to a value

```
IppStatus ippiSet_<mod>(Ipp<datatype> value, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

8u_C1R	16u_C1R	16s_C1R	32s_C1R	32f_C1R

Case 2: Setting each color channel to a specified value

```
IppStatus ippiSet_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

8u_C3R	16u_C3R	16s_C3R	32s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32s_AC4R	32f_AC4R

Case 3: Setting color channels and alpha channel to specified values

```
IppStatus ippiSet_<mod>(const Ipp<datatype> value[4], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

8u_C4R	16u_C4R	16s_C4R	32s_C4R	32f_C4R

Case 4: Setting masked one-channel data to a value

```
IppStatus ippiSet_<mod>(Ipp<datatype> value, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const Ipp8u* pMask, int maskStep);
```

Supported values for `mod`:

8u_C1MR	16u_C1MR	16s_C1MR	32s_C1MR	32f_C1MR

Case 5: Setting color channels of masked multi-channel data to specified values

```
IppStatus ippiSet_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const Ipp8u* pMask, int maskStep);
```

Supported values for `mod`:

8u_C3MR	16u_C3MR	16s_C3MR	32s_C3MR	32f_C3MR
8u_AC4MR	16u_AC4MR	16s_AC4MR	32s_AC4MR	32f_AC4MR

Case 6: Setting all channels of masked multi-channel data to specified values

```
IppStatus ippiSet_<mod>(const Ipp<datatype> value[4], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const Ipp8u* pMask, int maskStep);
```

Supported values for `mod`:

8u_C4MR	16u_C4MR	16s_C4MR	32s_C4MR	32f_C4MR

Case 7: Setting selected channel of multi-channel data to a value

```
IppStatus ippiSet_<mod>(Ipp<datatype> value, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

`8u_C3CR`	`16u_C3CR`	`16s_C3CR`	`32s_C3CR`	`32f_C3CR`
`8u_C4CR`	`16u_C4CR`	`16s_C4CR`	`32s_C4CR`	`32f_C4CR`

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`value`	Constant value to assign to each pixel in the destination image ROI.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance, in bytes, between the starting points of consecutive lines in the destination image.
`roiSize`	Size of the image ROI in pixels.
`pMask`	Pointer to the mask image buffer.
`maskStep`	Distance, in bytes, between the starting points of consecutive lines in the mask image buffer.

Description

This function operates with ROI.

This function sets pixels in the destination image ROI `pDst` to the `value` constant. Either all pixels in a rectangular ROI, or only those selected by the specified mask `pMask`, can be set to a value. In case of masked operation, the function sets pixel values in the destination buffer only if the spatially corresponding mask array value is non-zero. When a channel of interest is selected, that is only one channel of a multi-channel image must be set (see **Case 7**), the `pDst` pointer points to the start of ROI buffer in the required channel. If alpha channel is present in the source image data, the alpha components may be either skipped, or set to a value, depending on the chosen `ippiSet` function flavor.

This function supports negative step value.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error when any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize` has a field with zero or negative value.

ippStsStepErr	Indicates an error condition if dstStep or maskStep has a zero value.

Example

The code example below shows how to use the function ippiSet_8u_C1R.

```
void func_set()
{
    IppiSize roi = {5,4};
    Ipp8u x[8*4] = {0};


    ippiSet_8u_C1R(1, x, 8, roi);
}
```

Result:

```
01 01 01 01 01 00 00 00
01 01 01 01 01 00 00 00
01 01 01 01 01 00 00 00
01 01 01 01 01 00 00 00
```

See Also

Regions of Interest in Intel IPP

Copy

Copies pixel values between two buffers.

Syntax

Case 1: Copying all pixels of all color channels

```
IppStatus ippiCopy_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32s_C1R	32f_C1R
8u_C3R	16u_C3R	16s_C3R	32s_C3R	32f_C3R
8u_C4R	16u_C4R	16s_C4R	32s_C4R	32f_C4R
8u_AC4R	16u_AC4R	16s_AC4R	32s_AC4R	32f_AC4R
8u_C3AC4R	16u_C3AC4R	16s_C3AC4R	32s_C3AC4R	32f_C3AC4R
8u_AC4C3R	16u_AC4C3R	16s_AC4C3R	32s_AC4C3R	32f_AC4C3R

Case 2: Copying masked pixels only

```
IppStatus ippiCopy_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize roiSize,const Ipp8u* pMask, int maskStep);
```

Supported values for mod:

8u_C1MR	16u_C1MR	16s_C1MR	32s_C1MR	32f_C1MR

8u_C3MR	16u_C3MR	16s_C3MR	32s_C3MR	32f_C3MR
8u_C4MR	16u_C4MR	16s_C4MR	32s_C4MR	32f_C4MR
8u_AC4MR	16u_AC4MR	16s_AC4MR	32s_AC4MR	32f_AC4MR

Case 3: Copying a selected channel in a multi-channel image

```
IppStatus ippiCopy_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

8u_C3CR	16u_C3CR	16s_C3CR	32s_C3CR	32f_C3CR
8u_C4CR	16u_C4CR	16s_C4CR	32s_C4CR	32f_C4CR

Case 4: Copying a selected channel to a one-channel image

```
IppStatus ippiCopy_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

8u_C3C1R	16u_C3C1R	16s_C3C1R	32s_C3C1R	32f_C3C1R
8u_C4C1R	16u_C4C1R	16s_C4C1R	32s_C4C1R	32f_C4C1R

Case 5: Copying a one-channel image to a multi-channel image

```
IppStatus ippiCopy_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

8u_C1C3R	16u_C1C3R	16s_C1C3R	32s_C1C3R	32f_C1C3R
8u_C1C4R	16u_C1C4R	16s_C1C4R	32s_C1C4R	32f_C1C4R

Case 6: Splitting color image into separate planes

```
IppStatus ippiCopy_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* const
pDst[3], int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

8u_C3P3R	16u_C3P3R	16s_C3P3R	32s_C3P3R	32f_C3P3R

```
IppStatus ippiCopy_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* const
pDst[4], int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

8u_C4P4R	16u_C4P4R	16s_C4P4R	32s_C4P4R	32f_C4P4R

Case 7: Composing color image from separate planes

```
IppStatus ippiCopy_<mod>(const Ipp<datatype>* const pSrc[3], int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

8u_P3C3R	16u_P3C3R	16s_P3C3R	32s_P3C3R	32f_P3C3R

```
IppStatus ippiCopy_<mod>(const Ipp<datatype>* const pSrc[4], int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

8u_P4C4R	16u_P4C4R	16s_P4C4R	32s_P4C4R	32f_P4C4R

Case 8: Copying all pixels of all color channels with platform-aware functions

```
IppStatus ippiCopy_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>*
pDst, IppSizeL dstStep, IppiSizeL roiSize);
```

Supported values for `mod`:

8u_C1R_L	16s_C1R_L	16u_C1R_L	32s_C1R_L	32f_C1R_L
8u_C3R_L	16s_C3R_L	16u_C3R_L	32s_C3R_L	32f_C3R_L
8u_C4R_L	16s_C4R_L	16u_C4R_L	32s_C4R_L	32f_C4R_L
8u_AC4R_L	16s_AC4R_L	16u_AC4R_L	32s_AC4R_L	32f_AC4R_L

Include Files

`ippi.h`

Flavors with the `_L` suffix: `ippi_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI. The array storing pointers to the color planes of the source planar image.
`srcStep`	Distance, in bytes, between the starting points of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI. The array storing pointers to the color planes of the destination planar image.
`dstStep`	Distance, in bytes, between the starting points of consecutive lines in the destination image.
`roiSize`	Size of the source and destination ROI in pixels.
`pMask`	Pointer to the mask image buffer.
`maskStep`	Distance, in bytes, between the starting points of consecutive lines in the mask image buffer.

Description

This function operates with ROI.

This function copies data from the source image `pSrc` to the destination image `pDst`. Copying pixels selected by a mask `pMask` is supported as well.

For masked operation (**Case 2**), the function writes pixel values in the destination buffer only if the spatially corresponding mask array value is non-zero (as illustrated in the code example below).

Function flavors operating with the channel of interest (descriptor `C`) copy only one specified channel of a source multi-channel image to the channel of another multi-channel image (see **Case 3**). For these functions, the `pSrc` and `pDst` pointers point to the starts of ROI buffers in the specified channels of source and destination images, respectively.

Some function flavors add alpha channel to the 3-channel source image (flavors with the `_C3AC4R` descriptor), or discard alpha channel from the source image (flavors with the `_AC4C3R` descriptor) - see **Case 1**.

Special function flavors copy data from only one specified channel `pSrc` of a multi-channel image to a one-channel image `pDst` (see **Case 4**), as well as to copy data from a one-channel image `pSrc` to only one specified channel of a multi-channel image `pDst` (see **Case 5**).

You can also use the `ippiCopy` function to convert the interleaved color image into separate planes and vice versa (see **Case 6** and **Case 7**).

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error when any of the specified pointers is `NULL`, with the exception of second mode in **Case 4**.
`ippStsSizeErr`	Indicates an error condition if `roiSize` has a field with a zero or negative value.
`ippStsStepErr`	Indicates an error condition if `srcStep` or `dstStep` is less than `roiSize.width*<pixelSize>` for **Cases 4** and **5**.

Example

The code example below shows how to copy masked data.

```
   IppStatus copyWithMask( void ) {
   Ipp8u mask[3*3], x[5*4], y[5*4]={0};
   IppiSize imgroi={5,4}, mskroi={3,3};
   ippiSet_8u_C1R( 3, x, 5, imgroi );
   /// set mask with a hole in upper left corner
   ippiSet_8u_C1R( 1, mask, 3, mskroi );
   mask[0] = 0;
   /// copy roi with mask
   return ippiCopy_8u_C1MR( x, 5, y, 5, mskroi, mask, 3 );
}
```

The destination image `y` contains:

```
00 03 03 00 00
03 03 03 00 00
03 03 03 00 00
00 00 00 00 00
```

See Also

Regions of Interest in Intel IPP

CopyManaged

Copies pixel values between two images in accordance with the specified type of copying.

Syntax

```
IppStatus ippiCopyManaged_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, int flags);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance, in bytes, between the starting points of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance, in bytes, between the starting points of consecutive lines in the destination image.
`roiSize`	Size of the image ROI in pixels.
`flags`	Specifies the type of copying. Possible values are:

Description

This function operates with ROI.

This function copies data from a source image ROI `pSrc` to the destination image ROI `pDst`. The `flags` parameter specifies the type of copying that the function performs:

- When `flags` is set to `IPP_TEMPORAL_COPY`, the function is identical to the `ippiCopy_8u_C1R` function.
- When `flags` is set to `IPP_NONTEMPORAL_STORE`, the processor uses non-temporal store instructions. Copying is performed without caching the data of the destination image.
- When `flags` is set to `IPP_NONTEMPORAL_LOAD`, the processor uses non-temporal load instructions.

To achieve better performance, align data to 64-byte boundary.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when roiSize has a field with a zero or negative value.

See Also

Regions of Interest in Intel IPP

CopyConstBorder

Copies pixels values between two images and adds the border pixels with a constant value.

Syntax

Case 1: Not-in-place operation on one-channel data

```
IppStatus ippiCopyConstBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize
srcRoiSize, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, int topBorderHeight,
int leftBorderWidth, Ipp<datatype> value);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32s_C1R	32f_C1R

Case 2: In-place operation on one-channel data

```
IppStatus ippiCopyConstBorder_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize
srcRoiSize, IppiSize dstRoiSize, int topBorderHeight, int leftBorderWidth, const
Ipp<datatype> value);
```

Supported values for mod:

8u_C1IR	16u_C1IR	16s_C1IR	32s_C1IR	32f_C1IR

Case 3: Not-in-place operation on multi-channel data

```
IppStatus ippiCopyConstBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize
srcRoiSize, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, int topBorderHeight,
int leftBorderWidth, const Ipp<datatype> value[3]);
```

Supported values for mod:

8u_C3R	16u_C3R	16s_C3R	32s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32s_AC4R	32f_AC4R

```
IppStatus ippiCopyConstBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize
srcRoiSize, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, int topBorderHeight,
int leftBorderWidth, const Ipp<datatype> value[4]);
```

Supported values for mod:

8u_C4R	16u_C4R	16s_C4R	32s_C4R	32f_C4R

Case 4: In-place operation on multi-channel data

```
IppStatus ippiCopyConstBorder_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize
srcRoiSize, IppiSize dstRoiSize, int topBorderHeight, int leftBorderWidth, const
Ipp<datatype> value[3]);
```

Supported values for `mod`:

8u_C3IR	16u_C3IR	16s_C3IR	32s_C3IR	32f_C3IR
8u_AC4IR	16u_AC4IR	16s_AC4IR	32s_AC4IR	32f_AC4IR

```
IppStatus ippiCopyConstBorder_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize
srcRoiSize, IppiSize dstRoiSize, int topBorderHeight, int leftBorderWidth, const
Ipp<datatype> value[4]);
```

Supported values for `mod`:

8u_C4IR	16u_C4IR	16s_C4IR	32s_C4IR	32f_C4IR

Case 5: Not-in-place operation on one-channel data with platform-aware functions

```
IppStatus ippiCopyConstBorder_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep,
IppiSizeL srcRoiSize, Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL dstRoiSize,
IppSizeL topBorderHeight, IppSizeL leftBorderWidth, Ipp<datatype> value);
```

Supported values for `mod`:

8u_C1R_L	16u_C1R_L	16s_C1R_L	32s_C1R_L	32f_C1R_L

Case 6: In-place operation on one-channel data with platform-aware functions

```
IppStatus ippiCopyConstBorder_<mod>(Ipp<datatype>* pSrcDst, IppSizeL srcDstStep,
IppiSizeL srcRoiSize, IppiSizeL dstRoiSize, IppSizeL topBorderHeight, IppSizeL
leftBorderWidth, const Ipp<datatype> value);
```

Supported values for `mod`:

8u_C1IR_L	16u_C1IR_L	16s_C1IR_L	32s_C1IR_L	32f_C1IR_L

Case 7: Not-in-place operation on multi-channel data with platform-aware functions

```
IppStatus ippiCopyConstBorder_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep,
IppiSizeL srcRoiSize, Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL dstRoiSize,
IppSizeL topBorderHeight, IppSizeL leftBorderWidth, const Ipp<datatype> value[3]);
```

Supported values for `mod`:

8u_C3R_L	16u_C3R_L	16s_C3R_L	32s_C3R_L	32f_C3R_L
8u_AC4R_L	16u_AC4R_L	16s_AC4R_L	32s_AC4R_L	32f_AC4R_L

```
IppStatus ippiCopyConstBorder_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep,
IppiSizeL srcRoiSize, Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL dstRoiSize,
IppSizeL topBorderHeight, IppSizeL leftBorderWidth, const Ipp<datatype> value[4]);
```

Supported values for `mod`:

8u_C4R_L	16u_C4R_L	16s_C4R_L	32s_C4R_L	32f_C4R_L

Case 8: In-place operation on multi-channel data with platform-aware functions

```
IppStatus ippiCopyConstBorder_<mod>(Ipp<datatype>* pSrcDst, IppSizeL srcDstStep,
IppiSizeL srcRoiSize, IppiSizeL dstRoiSize, IppSizeL topBorderHeight, IppSizeL
leftBorderWidth, const Ipp<datatype> value[3]);
```

Supported values for `mod`:

`8u_C3IR_L`	`16u_C3IR_L`	`16s_C3IR_L`	`32s_C3IR_L`	`32f_C3IR_L`
`8u_AC4IR_L`	`16u_AC4IR_L`	`16s_AC4IR_L`	`32s_AC4IR_L`	`32f_AC4IR_L`

```
IppStatus ippiCopyConstBorder_<mod>(Ipp<datatype>* pSrcDst, IppSizeL srcDstStep,
IppiSizeL srcRoiSize, IppiSizeL dstRoiSize, IppSizeL topBorderHeight, IppSizeL
leftBorderWidth, const Ipp<datatype> value[4]);
```

Supported values for `mod`:

`8u_C4IR_L`	`16u_C4IR_L`	`16s_C4IR_L`	`32s_C4IR_L`	`32f_C4IR_L`

Include Files

`ippi.h`

Flavors with the `_L` suffix: `ippi_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

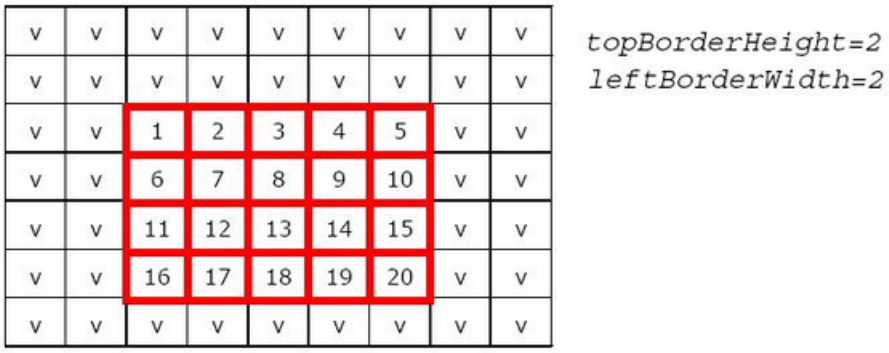
`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`pDst`	Pointer to the destination image.
`pSrcDst`	Pointer to the source/destination image (for in-place flavors).
`dstStep`	Distance, in bytes, between the starting points of consecutive lines in the destination image.
`srcDstStep`	Distance, in bytes, between the starting points of consecutive lines in the source/destination image (for in-place flavors).
`srcRoiSize`	Size of the source ROI, in pixels.
`dstRoiSize`	Size of the destination ROI, in pixels.
`topBorderHeight`	Height of the top border, in pixels.
`leftBorderWidth`	Width of the left border, in pixels.
`value`	The constant value to assign to the border pixels (constant vector in case of multi-channel images).

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function copies the source image `pSrc` to the destination image `pDst` and creates border outside the copied area; pixel values of the border are set to the specified constant value that is passed by the `value` argument. The parameters `topBorderHeight` and `leftBorderWidth` specify the position of the first pixel of the source ROI in the destination image ROI (see Figure Creating a Border of Pixels with Constant Value.) Squares marked in red correspond to pixels copied from the source image, that is, the source image ROI.

Creating a Border of Pixels with Constant Value |



The height (width) of the destination ROI cannot be less than the sum of the height (width) of source ROI and the `topBorderHeight` (`leftBorderWidth`) parameter.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error when any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `srcRoiSize` or `dstRoiSize` has a field with a zero or negative value, or `topBorderHeight` or `leftBorderWidth` is less than zero, or `dstRoiSize.width<srcRoiSize.width +leftBorderWidth`, or `dstRoiSize.height<srcRoiSize.height +topBorderHeight`.
`ippStsStepErr`	

Example

The code example below shows how to use the function `ippiCopyConstBorder_8u_C1R`.

```
Ipp8u src[8*4] = {3, 3, 3, 3, 3, 8, 8, 8,
                  3, 2, 1, 2, 3, 8, 8, 8,
                  3, 2, 1, 2, 3, 8, 8, 8,
                  3, 3, 3, 3, 3, 8, 8, 8};
```

```
Ipp8u dst[8*6];
IppiSize srcRoi = { 5, 4 };
IppiSize dstRoi = { 7, 6 };
int borderWidth  = 1;
int borderHeight = 1;
int borderVal = 0;


ippiCopyConstBorder_8u_C1R(src, 8, srcRoi, dst, 8, dstRoi, borderHeight, borderWidth, borderVal);
```

```
        Results
source image:
3 3 3 3 3 8 8 8
3 2 1 2 3 8 8 8
3 2 1 2 3 8 8 8   src
3 3 3 3 3 8 8 8


destination image:
0 0 0 0 0 0 0
0 3 3 3 3 3 0
0 3 2 1 2 3 0
0 3 2 1 2 3 0    dst
0 3 3 3 3 3 0
0 0 0 0 0 0 0
```

CopyMirrorBorder

Copies pixels values between two images and adds the mirrored border pixels.

Syntax

Case 1: Not-in-place operation

```
IppStatus ippiCopyMirrorBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize srcRoiSize, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, int topBorderHeight, int leftBorderWidth);
```

Supported values for `mod`:

8u_C1R	16u_C1R	16s_C1R	32s_C1R	32f_C1R
8u_C3R	16u_C3R	16s_C3R	32s_C3R	32f_C3R
8u_C4R	16u_C4R	16s_C4R	32s_C4R	32f_C4R

Case 2: In-place operation

```
IppStatus ippiCopyMirrorBorder_<mod>(const Ipp<datatype>* pSrc, int srcDstStep, IppiSize srcRoiSize, IppiSize dstRoiSize, int topBorderHeight, int leftBorderWidth);
```

Supported values for `mod`:

8u_C1IR	16u_C1IR	16s_C1IR	32s_C1IR	32f_C1IR
8u_C3IR	16u_C3IR	16s_C3IR	32s_C3IR	32f_C3IR
8u_C4IR	16u_C4IR	16s_C4IR	32s_C4IR	32f_C4IR

Case 3: Not-in-place operation with platform-aware functions

```
IppStatus ippiCopyMirrorBorder_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep,
IppiSizeL srcRoiSize, Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL dstRoiSize,
IppSizeL topBorderHeight, IppSizeL leftBorderWidth);
```

Supported values for `mod`:

8u_C1R_L	16u_C1R_L	16s_C1R_L	32s_C1R_L	32f_C1R_L
8u_C3R_L	16u_C3R_L	16s_C3R_L	32s_C3R_L	32f_C3R_L
8u_C4R_L	16u_C4R_L	16s_C4R_L	32s_C4R_L	32f_C4R_L

Case 4: In-place operation with platform-aware functions

```
IppStatus ippiCopyMirrorBorder_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcDstStep,
IppiSizeL srcRoiSize, IppiSizeL dstRoiSize, IppSizeL topBorderHeight, IppSizeL
leftBorderWidth);
```

Supported values for `mod`:

8u_C1IR_L	16u_C1IR_L	16s_C1IR_L	32s_C1IR_L	32f_C1IR_L
8u_C3IR_L	16u_C3IR_L	16s_C3IR_L	32s_C3IR_L	32f_C3IR_L
8u_C4IR_L	16u_C4IR_L	16s_C4IR_L	32s_C4IR_L	32f_C4IR_L

Include Files

`ippi.h`

Flavors with the `_L` suffix: `ippi_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance, in bytes, between the starting points of consecutive lines in the source image.
`pDst`	Pointer to the destination image.
`dstStep`	Distance, in bytes, between the starting points of consecutive lines in the destination image.
`srcDstStep`	Distance, in bytes, between the starting points of consecutive lines in the source and destination image.
`srcRoiSize`	Size of the source ROI, in pixels.
`dstRoiSize`	Size of the destination ROI, in pixels.
`topBorderHeight`	Height of the top border, in pixels.

leftBorderWidth	Width of the left border, in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function copies the source image pSrc to the destination image pDst and fills pixels outside the copied area (border pixels) in the destination image with the values of the source image pixels according to the scheme illustrated in the figure below. Squares marked in red correspond to pixels copied from the source image, that is, the source image ROI.

Creating a Mirrored Border |

13	12	11	12	13	14	15	14	13
8	7	6	7	8	9	10	9	8
3	2	1	2	3	4	5	4	3
8	7	6	7	8	9	10	9	8
13	12	11	12	13	14	15	14	13
18	17	16	17	18	19	20	19	18
13	12	11	12	13	14	15	14	13

topBorderWidth=2

topBorderHeight=2

NOTE In-place flavors actually add border pixels to the source image ROI, thus a destination image ROI is larger than the initial image.

The parameters topBorderHeight and leftBorderWidth specify the position of the first pixel of the source ROI in the destination image ROI.

The height (width) of the destination ROI cannot be less than the sum of the height (width) of source ROI and the topBorderHeight (leftBorderWidth) parameter.

NOTE If border width is greater than the image size in the corresponding dimension, multiple reflections are obtained for this border.

NOTE If you use this function for a tiled image, note that to perform correct mirroring, the size of a tile must be more than the size of the used border. For example, if the image referenced above is divided into two tiles of size 3x3 and 2x3 , the second tile (cells are highlighted in yellow) is extended with top, right, and bottom borders (highlighted in gray). The width of the right border is not less than the second tile width, so the pixels (blue) required for constructing the border of the tiled image are out of the processed tile. Therefore the last column (red) of the border extended image is computed incorrectly:



Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error when any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error in any of the following cases: • `srcRoiSize` or `dstRoiSize` has a field with a zero or negative value • `topBorderHeight` or `leftBorderWidth` is less than zero • `dstRoiSize.width < srcRoiSize.width + leftBorderWidth` • `dstRoiSize.height < srcRoiSize.height + topBorderHeight`
`ippStsStepErr`	Indicates an error when `srcStep` or `dstStep` has a zero or negative value.

Example

The code example below shows how to use the `ippiCopyMirrorBorder_8u_C1R` function.

```
Ipp8u src[8*4] = { 1,  2,  3,  8,  8,  8,  8,  8,
  10,  9,  8,  8,  8,  8,  8,  8,
  11, 12, 13,  8,  8,  8,  8,  8,
```

```
  19, 18, 17,  8,  8,  8,  8,  8};
Ipp8u dst[10*8];
IppiSize srcRoi = { 3,  4 }; IppiSize dstRoi = { 10, 8 };
int topBorderHeight  = 2;
int leftBorderWidth  = 2;


ippiCopyMirrorBorder_8u_C1R(src, 8, srcRoi, dst, 10, dstRoi, topBorderHeight, leftBorderWidth);
```

Result:

```
source image:
  1  2  3  8  8  8  8  8
 10  9  8  8  8  8  8  8
 11 12 13  8  8  8  8  8
 19 18 17  8  8  8  8  8




destination image:
13 12 11 12 13 12 11 12 13 12
 8  9 10  9  8  9 10  9  8  9
 3  2  1  2  3  2  1  2  3  2
 8  9 10  9  8  9 10  9  8  9
13 12 11 12 13 12 11 12 13 12
17 18 19 18 17 18 19 18 17 18
13 12 11 12 13 12 11 12 13 12
 8  9 10  9  8  9 10  9  8  9
```

See Also

Regions of Interest in Intel IPP

CopyReplicateBorder

Copies pixels values between two images and adds the replicated border pixels.

Syntax

Case 1: Not-in-place operation

```
IppStatus ippiCopyReplicateBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep,
IppiSize srcRoiSize, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, int
topBorderHeight, int leftBorderWidth);
```

Supported values for `mod`:

8u_C1R	16u_C1R	16s_C1R	32s_C1R	32f_C1R
8u_C3R	16u_C3R	16s_C3R	32s_C3R	32f_C3R
8u_C4R	16u_C4R	16s_C4R	32s_C4R	32f_C4R
8u_AC4R	16u_AC4R	16s_AC4R	32s_AC4R	32f_AC4R

Case 2: In-place operation

```
IppStatus ippiCopyReplicateBorder_<mod>(const Ipp<datatype>* pSrc, int srcDstStep,
IppiSize srcRoiSize, IppiSize dstRoiSize, int topBorderHeight, int leftBorderWidth);
```

Supported values for `mod`:

8u_C1IR	16u_C1IR	16s_C1IR	32s_C1IR	32f_C1IR
8u_C3IR	16u_C3IR	16s_C3IR	32s_C3IR	32f_C3IR
8u_C4IR	16u_C4IR	16s_C4IR	32s_C4IR	32f_C4IR
8u_AC4IR	16u_AC4IR	16s_AC4IR	32s_AC4IR	32f_AC4IR

Case 3: Not-in-place operation for platform-aware functions

```
IppStatus ippiCopyReplicateBorder_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep,
IppiSizeL srcRoiSize, Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL dstRoiSize,
IppSizeL topBorderHeight, IppSizeL leftBorderWidth);
```

Supported values for `mod`:

8u_C1R_L	16u_C1R_L	16s_C1R_L	32s_C1R_L	32f_C1R_L
8u_C3R_L	16u_C3R_L	16s_C3R_L	32s_C3R_L	32f_C3R_L
8u_C4R_L	16u_C4R_L	16s_C4R_L	32s_C4R_L	32f_C4R_L
8u_AC4R_L	16u_AC4R_L	16s_AC4R_L	32s_AC4R_L	32f_AC4R_L

Case 4: In-place operation for platform-aware functions

```
IppStatus ippiCopyReplicateBorder_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcDstStep,
IppiSizeL srcRoiSize, IppiSizeL dstRoiSize, IppSizeL topBorderHeight, IppSizeL
leftBorderWidth);
```

Supported values for `mod`:

8u_C1IR_L	16u_C1IR_L	16s_C1IR_L	32s_C1IR_L	32f_C1IR_L
8u_C3IR_L	16u_C3IR_L	16s_C3IR_L	32s_C3IR_L	32f_C3IR_L
8u_C4IR_L	16u_C4IR_L	16s_C4IR_L	32s_C4IR_L	32f_C4IR_L
8u_AC4IR_L	16u_AC4IR_L	16s_AC4IR_L	32s_AC4IR_L	32f_AC4IR_L

Include Files

`ippi.h`

Flavors with the `_L` suffix: `ippi_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.

pDst	Pointer to the destination image.
dstStep	Distance in bytes between starts of consecutive lines in destination image.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and the destination image.
srcRoiSize	Size of the source ROI in pixels.
dstRoiSize	Size of the destination ROI in pixels.
topBorderHeight	Height of the top border in pixels.
leftBorderWidth	Width of the left border in pixels.

Description

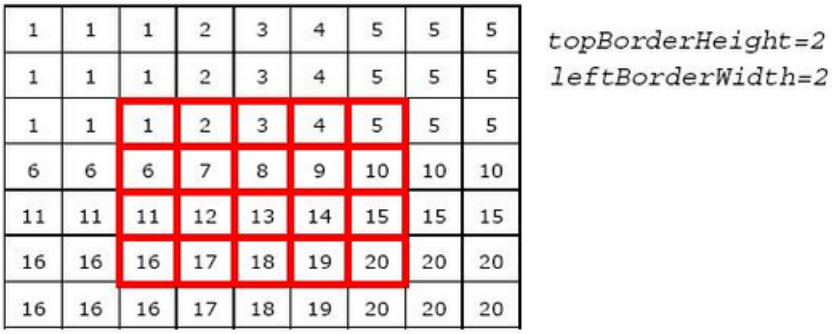
This function operates with ROI (see Regions of Interest in Intel IPP).

This function copies the source image pSrc to the destination image pDst and fills pixels (“border”) outside the copied area in the destination image with the values of the source image pixels according to the scheme illustrated in Figure Creating a Replicated Border. Squares marked in red correspond to pixels copied from the source image, that is, the source image ROI.

Note that the in-place function flavor actually adds border pixels to the source image ROI, thus a destination image ROI is larger than the initial image.

The parameters topBorderHeight and leftBorderWidth specify the position of the first pixel of the source ROI in the destination image ROI.

Creating a Replicated Border |



The height (width) of the destination ROI cannot be less than the sum of the height (width) of source ROI and the topBorderHeight (leftBorderWidth) parameter.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.

ippStsSizeErr	Indicates an error condition if srcRoiSize or dstRoiSize has a field with a zero or negative value, or topBorderHeight or leftBorderWidth is less than zero, or dstRoiSize.width<srcRoiSize.width +leftBorderWidth, or dstRoiSize.height<srcRoiSize.height +topBorderHeight.
ippStsStepErr	Indicates an error condition if srcStep or dstStep has a zero or negative value.

Example

The code example below shows how to use the ippiCopyReplicateBorder_8u_C1R function.

```
Ipp8u src[8*4] = {5, 4, 3, 4, 5, 8, 8, 8,
                  3, 2, 1, 2, 3, 8, 8, 8,
                  3, 2, 1, 2, 3, 8, 8, 8,
                  5, 4, 3, 4, 5, 8, 8, 8};
Ipp8u dst[9*8];
IppiSize srcRoi = { 5, 4 };
IppiSize dstRoi = { 9, 8 };
int topborderHeight  = 2;
int leftborderWidth  = 2;


ippiCopyReplicateBorder_8u_C1R(src, 8, srcRoi, dst, 9, dstRoi, topBorderHeight, leftBorderWidth);
```

```
        Results
source image:
5 4 3 4 5 8 8 8
3 2 1 2 3 8 8 8
3 2 1 2 3 8 8 8
5 4 3 4 5 8 8 8


destination image:
5 5 5 4 3 4 5 5 5
5 5 5 4 3 4 5 5 5
5 5 5 4 3 4 5 5 5
3 3 3 2 1 2 3 3 3
3 3 3 2 1 2 3 3 3
5 5 5 4 3 4 5 5 5
5 5 5 4 3 4 5 5 5
5 5 5 4 3 4 5 5 5
```

CopyWrapBorder

Copies pixels values between two images and adds the border pixels.

Syntax

Case 1: Not-in-place operation

```
IppStatus ippiCopyWrapBorder_32s_C1R(const Ipp32s* pSrc, int srcStep, IppiSize
srcRoiSize, Ipp32s* pDst, int dstStep, IppiSize dstRoiSize, int topBorderHeight, int
leftBorderWidth);

IppStatus ippiCopyWrapBorder_32f_C1R(const Ipp32f* pSrc, int srcStep, IppiSize
srcRoiSize, Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, int topBorderHeight, int
leftBorderWidth);
```

Case 2: In-place operation

```
IppStatus ippiCopyWrapBorder_32s_C1IR(const Ipp32s* pSrc, int srcDstStep, IppiSize
srcRoiSize, IppiSize dstRoiSize, int topBorderHeight, int leftBorderWidth);

IppStatus ippiCopyWrapBorder_32f_C1IR(const Ipp32f* pSrc, int srcDstStep, IppiSize
srcRoiSize, IppiSize dstRoiSize, int topBorderHeight, int leftBorderWidth);
```

Case 3: Not-in-place operation with platform-aware functions

```
IppStatus ippiCopyWrapBorder_32s_C1R_L(const Ipp32s* pSrc, IppSizeL srcStep, IppiSizeL
srcRoiSize, Ipp32s* pDst, IppSizeL dstStep, IppiSizeL dstRoiSize, IppSizeL
topBorderHeight, IppSizeL leftBorderWidth);

IppStatus ippiCopyWrapBorder_32f_C1R_L(const Ipp32f* pSrc, IppSizeL srcStep, IppiSizeL
srcRoiSize, Ipp32f* pDst, IppSizeL dstStep, IppiSizeL dstRoiSize, IppSizeL
topBorderHeight, IppSizeL leftBorderWidth);
```

Case 4: In-place operation with platform-aware functions

```
IppStatus ippiCopyWrapBorder_32s_C1IR_L(const Ipp32s* pSrc, IppSizeL srcDstStep,
IppiSizeL srcRoiSize, IppiSizeL dstRoiSize, IppSizeL topBorderHeight, IppSizeL
leftBorderWidth);

IppStatus ippiCopyWrapBorder_32f_C1IR_L(const Ipp32f* pSrc, IppSizeL srcDstStep,
IppiSizeL srcRoiSize, IppiSizeL dstRoiSize, IppSizeL topBorderHeight, IppSizeL
leftBorderWidth);
```

Include Files

`ippi.h`

Flavors with the `_L` suffix: `ippi_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image.

srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image for in-place flavor.
srcRoiSize	Size of the source ROI in pixels.
dstRoiSize	Size of the destination ROI in pixels.
topBorderHeight	Height of the top border in pixels.
leftBorderWidth	Width of the left border in pixels.

Description

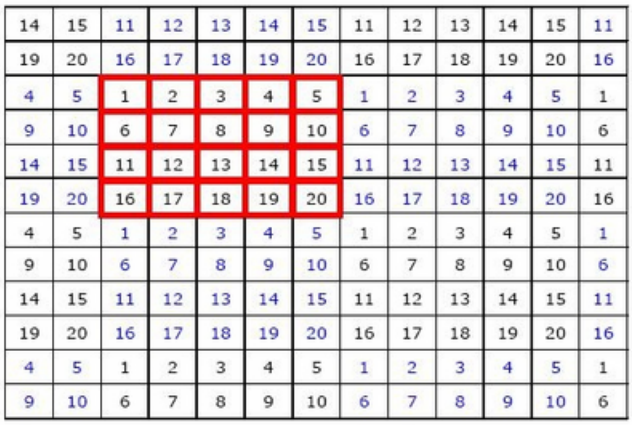
This function operates with ROI (see Regions of Interest in Intel IPP).

This function copies the source image pSrc to the destination image pDst and fills pixels ("border") outside the copied area in the destination image with the values of the source image pixels according to the scheme illustrated in Figure Creating a Border of Pixels by ippiCopyWrapBorder Function. Squares marked in red correspond to pixels copied from the source image.

Note that the in-place function flavor actually adds border pixels to the source image ROI, thus a destination image ROI is larger than the initial image.

The parameters topBorderHeight and leftBorderWidth specify the position of the first pixel of the source ROI in the destination image ROI.

Creating a Border of Pixels by ippiCopyWrapBorder Function |



The height (width) of the destination ROI cannot be less than the sum of the height (width) of source ROI and the topBorderHeight (leftBorderWidth) parameter.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.

ippStsSizeErr	Indicates an error condition if srcRoiSize or dstRoiSize has a field with a zero or negative value, or topBorderHeight or leftBorderWidth is less than zero, or dstRoiSize.width<srcRoiSize.width +leftBorderWidth, or dstRoiSize.height<srcRoiSize.height +topBorderHeight.
ippStsStepErr	Indicates an error condition if srcStep or dstStep has a zero or negative value.
ippStsNotEvenStepErr	Indicates an error condition if one of the step values is not divisible by 4 for floating point images, or by 2 for short integer images.

Example

The code example below shows how to use the ippiCopyWrapBorder_32s_C1R function.

```
Ipp32s src[8*4] = {
       5, 4, 3, 4, 5, 8, 8, 8,
       3, 2, 1, 2, 3, 8, 8, 8,
       3, 2, 1, 2, 3, 8, 8, 8,
       5, 4, 3, 4, 5, 8, 8, 8
       };
Ipp32s dst[9*8];
IppiSize srcRoi = { 5, 4 };
IppiSize dstRoi = { 9, 8 };
int topborderHeight  = 2;
int leftborderWidth = 2;


ippiCopyWrapBorder_32s_C1R (src, 8*sizeof(Ipp32s), srcRoi, dst,
                  9*sizeof(Ipp32s), dstRoi, topBorderHeight, leftBorderWidth);
```

```
        Results
source image:
   5 4 3 4 5 8 8 8
   3 2 1 2 3 8 8 8
   3 2 1 2 3 8 8 8
   5 4 3 4 5 8 8 8


destination image:
   2 3 3 2 1 2 3 3 2
   4 5 5 4 3 4 5 5 4
   4 5 5 4 3 4 5 5 4
   2 3 3 2 1 2 3 3 2
   2 3 3 2 1 2 3 3 2
   4 5 5 4 3 4 5 5 4
   4 5 5 4 3 4 5 5 4
   2 3 3 2 1 2 3 3 2
```

CopySubpix

Copies pixel values between two images with subpixel precision.

Syntax

Case 1: Copying without conversion or with conversion to floating point data

```
IppStatus ippiCopySubpix_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize, Ipp32f dx, Ipp32f dy);
```

Supported values for `mod`:

`8u_C1R`	`16u_C1R`	`32f_C1R`
`8u32f_C1R`	`16u32f_C1R`	

Case 2: Copying with conversion to integer data

```
IppStatus ippiCopySubpix_8u16u_C1R_Sfs(const Ipp8u* pSrc, int srcStep, Ipp16u* pDst,
int dstStep, IppiSize roiSize, Ipp32f dx, Ipp32f dy, int scaleFactor);
```

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image.
`roiSize`	Size of the source and destination ROI in pixels.
`dx`	Fractional part of the `x` - coordinate in the source image.
`dy`	Fractional part of the `y` - coordinate in the source image.
`scaleFactor`	Scale factor (see Integer Result Scaling).

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function computes the pixel value of the destination image using linear interpolation (see Linear Interpolation in Appendix B) in accordance with the following formula:

$$pDst_{j,i} = pSrc_{j+dx,i+dy}$$

where i= 0, ...roiSize.height- 1,j= 0, ...roiSize.width- 1.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with a zero or negative value.
ippStsStepErr	Indicates an error condition if srcStep or dstStep is less than roiSize.width*<pixelSize>.
ippStsNotEvenStepErr	Indicates an error condition if one of srcStep or dstStep is not divisible by 4 for floating point images, or by 2 for short integer images.

CopySubpixIntersect

Copies pixel values of the intersection with specified window with subpixel precision.

Syntax

Case 1: Copying without conversion or with conversion to floating point data

```
IppStatus ippiCopySubpixIntersect_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
IppiSize srcRoiSize, Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize,
IppiPoint_32f point, IppiPoint* pMin, IppiPoint* pMax);
```

Supported values for mod:

8u_C1R	16u_C1R	32f_C1R
8u32f_C1R	16u32f_C1R	

Case 2: Copying with conversion to integer data

```
IppStatus ippiCopySubpixIntersect_8u16u_C1R_Sfs(const Ipp8u* pSrc, int srcStep,
IppiSize srcRoiSize, Ipp16u* pDst, int dstStep, IppiSize dstRoiSize, IppiPoint_32f
point, IppiPoint* pMin, IppiPoint* pMax, int scaleFactor);
```

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
srcRoiSize	Size of the source image ROI in pixels.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the source and destination image.
dstRoiSize	Size of the destination ROI in pixels.
point	Center point of the window.
pMin	Pointer to coordinates of the top left pixel of the intersection in the destination image.
pMax	Pointer to coordinates of the bottom right pixel of the intersection in the destination image.
scaleFactor	Scale factor (see Integer Result Scaling).

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function determines the intersection of the source image and the window of size `dstRoiSize` centered in point `point`. The corresponding pixels of the destination image are calculated using linear interpolation (see Linear Interpolation in Appendix B) in accordance with the following formula:

$$pDst_{j,i} = pSrc_{Xsubpix(j),Ysubpix(i)}$$

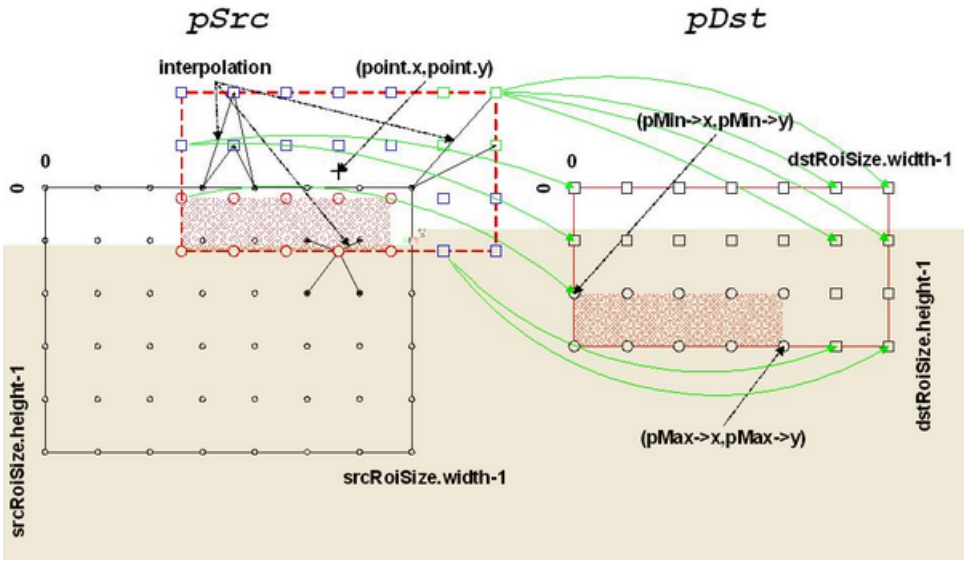
where `Xsubpix(j) = min(max(point.x + j` - 0.5*(`dstRoiSize.width - 1`), 0), `srcRoiSize.width - 1`),

`Ysubpix(i) = min(max(point.y + i` - 0.5*(`dstRoiSize.height - 1`), 0), `srcRoiSize.height - 1`),

`i= 0, ...dstRoiSize.height- 1,j= 0, ...dstRoiSize.width- 1.`

Minimal values of `j` and `i` (coordinates of the top left calculated destination pixel) are assigned to `pMin.x` and `pMin.y`, maximal values (coordinates of the top left calculated destination pixel) - to `pMin.x` and `pMin.y`. (See Figure Image Copying with Subpixel Precision Using `ippiCopySubpixIntersect` Function.)

Image Copying with Subpixel Precision Using `ippiCopySubpixIntersect` Function |



The height (width) of the destination ROI cannot be less than the sum of the height (width) of source ROI and the `topBorderHeight` (`leftBorderWidth`) parameter.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error when any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `srcRoiSize` or `dstRoiSize` has a field with a zero or negative value.
`ippStsStepErr`	Indicates an error condition if `srcStep` is less than `srcRoiSize.width*<pixelSize>`, or `dstStep` is less than `dstRoiSize.width*<pixelSize>`.

Example

The code example below shows how to use the function `ippiCopySubpixIntersect_8u_C1R`.

```
Ipp8u src[8*6] = {
      7, 7, 6, 6, 6, 6, 7, 7,
      6, 5, 5, 5, 5, 5, 5, 6,
      6, 5, 4, 3, 3, 4, 5, 6,
      6, 5, 4, 3, 3, 4, 5, 6,
      6, 5, 5, 5, 5, 5, 5, 6,
      6, 6, 6, 6, 6, 6, 6, 6
    };
Ipp8u dst[7*4];
IppiSize srcRoi = { 8, 6 };
IppiSize dstRoi = { 7, 4 };
IppiPoint_32f point = { 4, 1 };
IppiPoint min;
IppiPoint max;
```

```
ippiCopySubpixIntersect_8u_C1R (src, 8, srcRoi, dst, 7, dstRoi, point, &min, &max );
```

```
        Results
source image:
7 7 6 6 6 6 7 7
6 5 5 5 5 5 5 6
6 5 4 3 3 4 5 6
6 5 4 3 3 4 5 6
6 5 5 5 5 5 5 6
6 6 6 6 6 6 6 6


destination image:
7 6 6 6 6 7 7
6 6 6 6 6 6 7
5 5 4 4 5 5 6
5 4 3 3 4 5 6


min = { 0, 1 }
max = { 5, 3 }
```

Dup

Copies a gray scale image to all channels of the color image.

Syntax

`IppStatus ippiDup_8u_C1C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);`

`IppStatus ippiDup_8u_C1C4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);`

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image.

roiSize	Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function copies a one-channel (gray scale) image pSrc to each channel of the multi-channel image pDst.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if RoiSize has a field with a zero or negative value.

Transpose

Transposes a source image.

Syntax

Case 1: Not-in-place operation

```
IppStatus ippiTranspose_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32s_C1R	32f_C1R
8u_C3R	16u_C3R	16s_C3R	32s_C3R	32f_C3R
8u_C4R	16u_C4R	16s_C4R	32s_C4R	32f_C4R

Case 2: In-place operation

```
IppStatus ippiTranspose_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize
roiSize);
```

Supported values for mod:

8u_C1IR	16u_C1IR	16s_C1IR	32s_C1IR	32f_C1IR
8u_C3IR	16u_C3IR	16s_C3IR	32s_C3IR	32f_C3IR
8u_C4IR	16u_C4IR	16s_C4IR	32s_C4IR	32f_C4IR

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance, in bytes, between the starting points of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance, in bytes, between the starting points of consecutive lines in the destination image.
pSrcDst	Pointer to the source and destination ROI for in-place operation.
srcDstStep	Distance, in bytes, between the starting points of consecutive lines in the source and destination image buffer for the in-place operation.
roiSize	Size of the source ROI in pixels.

Description

This function operates with ROI.

This function transposes the source image pSrc(pSrcDst``for in-place flavors) and stores the result in ``pDst(pSrcDst). The destination image is obtained from the source image by transforming the columns to the rows: pDst(x, y) = pSrc(y, x)

The parameter roiSize is specified for the source image. The value of the roiSize.width parameter for the destination image is equal to roiSize.height for the source image, and roiSize.height for the destination image is equal to roiSize.width for the source image.

NOTE For in-place operations, roiSize.width must be equal to roiSize.height.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL, with the exception of second mode in Case 4.
ippStsSizeErr	Indicates an error when:

Example

The code example below shows how to use the ippiTranspose_8u_C1R function.

```
Ipp8u src[8*4] = {1, 2, 3, 4, 8, 8, 8, 8,
                  1, 2, 3, 4, 8, 8, 8, 8,
                  1, 2, 3, 4, 8, 8, 8, 8,
                  1, 2, 3, 4, 8, 8, 8, 8};
Ipp8u dst[4*4];
```

```
IppiSize srcRoi = { 4, 4 };


ippiTranspose_8u_C1R ( src, 8, dst, 4, srcRoi );
```

Result:

```
1 2 3 4 8 8 8 8
1 2 3 4 8 8 8 8    src
1 2 3 4 8 8 8 8




1 1 1 1
2 2 2 2
3 3 3 3      dst
4 4 4 4
```

See Also

Regions of Interest in Intel IPP

SwapChannels

Copies channels of the source image to the destination image.

Syntax

Case 1: Not-in-place operation

```
IppStatus ippiSwapChannels_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize roiSize, const int dstOrder[3]);
```

Supported values for `mod`:

8u_C3R	16u_C3R	16s_C3R	32s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32s_AC4R	32f_AC4R

```
IppStatus ippiSwapChannels_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize roiSize, const int dstOrder[4]);
```

Supported values for `mod`:

8u_C4R	16u_C4R	16s_C4R	32s_C4R	32f_C4R

Case 2: In-place operation

```
IppStatus ippiSwapChannels_8u_C3IR(Ipp8u* pSrcDst, int srcDstStep, IppiSize roiSize,
const int dstOrder[3]);
```

```
IppStatus ippiSwapChannels_8u_C4IR(Ipp8u* pSrcDst, int srcDstStep, IppiSize roiSize,
const int dstOrder[4]);
```

Case 3: Operation with converting 3-channel image to the 4-channel image

```
IppStatus ippiSwapChannels_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize roiSize, const int dstOrder[4], Ipp<datatype> val);
```

Supported values for `mod`:

8u_C3C4R	16u_C3C4R	16s_C3C4R	32s_C3C4R	32f_C3C4R

Case 4: Operation with converting 4-channel image to the 3-channel image

```
IppStatus ippiSwapChannels_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize roiSize, const int dstOrder[3]);
```

Supported values for `mod`:

8u_C4C3R	16u_C4C3R	16s_C4C3R	32s_C4C3R	32f_C4C3R

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image.
`pSrcDst`	Pointer to the source and destination ROI for in-place operation.
`srcDstStep`	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
`roiSize`	Size of the source and destination ROI in pixels.
`dstOrder`	Order of channels in the destination image.
`val`	Constant value.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function copies the data from specified channels of the source image ROI `pSrc` to the specified channels of the destination image ROI `pDst`.

The first channel in the destination image is determined by the first component of `dstOrder`. Its value lies in the range [0..2] for a 3-channel image, and [0..3] for a 4-channel image, and indicates the corresponding channel number of the source image. Other channels are specified in the similar way. For example, if the sequence of channels in the source 3-channel image is `A`, `B`, `C`, and `dstOrder[0]=2`,`dstOrder[1] = 0`,

dstOrder[2] = 1, then the order of channels in the 3-channel destination image is C, A, B. Some or all components of dstOrder may have the same values. It means that data from a certain channel of the source image may be copied to several channels of the destination image.

Some functions flavors convert a 3-channel source image to the 4-channel destination image (see Case 3). In this case an additional channel contains data from any specified source channel, or its pixel values are set to the specified constant value val (corresponding component dstOrder[n] should be set to 3), or its pixel values are not set at all (corresponding component dstOrder[n] should be set to an arbitrary value greater than 3). For example, the sequence of channels in the source 3-channel image is A, B, C, if dstOrder[0]=1,dstOrder[1] = 0, dstOrder[2]=1,dstOrder[3] = 2, then the order of channels in the 4-channel destination image will be B, A, B, C; if dstOrder[0]=4,dstOrder[1] = 0, dstOrder[2]=1,dstOrder[3] = 2, then the order of channels in the 4-channel destination image will be D, A, B, C, where D is a channel whose pixel values are not set.

The function flavors that support image with the alpha channel do not perform operation on it.

This function supports negative step values.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if srcStep or dstStep has a zero value.

Example

The code example below shows how to use the function ippiSwapChannels_8u_C3R.

```
Ipp8u src[12*3] = { 255, 0,  0, 255, 0,  0, 255, 0,  0, 255, 0,  0,
                    0, 255, 0,  0, 255, 0,  0, 255, 0,  0, 255, 0,
                    0,  0, 255, 0,  0, 255, 0,  0, 255, 0,  0, 255};
Ipp8u dst[12*3];
IppiSize roiSize = { 4, 3 };
int order[3] = { 2, 1, 0 }


ippiSwapChannels_8u_C3R ( src, 12, dst, 12, roiSize, order );
```

```
Result:
src    [rgb]


255 0  0 255 0  0 255 0  0 255 0  0
0  255 0  0 255 0  0 255 0  0 255 0
0   0 255 0  0 255 0  0 255 0  0 255


dst    [bgr]
```

```
0   0 255 0   0 255 0   0 255 0   0 255
0  255 0   0 255 0   0 255 0   0 255 0
255 0   0 255 0   0 255 0   0 255 0   0
```

AddRandUniform

Generates random samples with uniform distribution and adds them to an image data.

Syntax

```
IppStatus ippiAddRandUniform_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize
roiSize, Ipp<datatype> low, Ipp<datatype> high, unsigned int* pSeed);
```

Supported values for `mod`:

8u_C1IR	16u_C1IR	16s_C1IR	32f_C1IR
8u_C3IR	16u_C3IR	16s_C3IR	32f_C3IR
8u_C4IR	16u_C4IR	16s_C4IR	32f_C4IR
8u_AC4IR	16u_AC4IR	16s_AC4IR	32f_AC4IR

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrcDst`	Pointer to the source and destination image ROI.
`srcDstStep`	Distance in bytes between starts of consecutive lines in the source and destination image.
`roiSize`	Size of the image ROI in pixels.
`low`	The lower bound for the range of uniformly distributed values.
`high`	The upper bound for the range of uniformly distributed values.
`pSeed`	The initial seed value for the pseudo-random number generator.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

The function generates samples with uniform distribution over the range [`low`, `high`] and adds them to a source image pointed to by `pSrcDst`.

The resulting pixel values that exceed the image data range are saturated to the respective data-range limits. To obtain an image that contains pure noise with uniform distribution, call `ippiAddRandUniform` using a source image with zero data as input.

Example 4-11 Using `ippiAddRandUniform` function

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error when any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize` has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if `srcDstStep` has a zero or negative value.

Example

The code example below shows data conversion without scaling.

```
IppStatus randUniform( void ) {
   unsigned int seed = 123456;
   Ipp8u img[2048], mn, mx;
   IppiSize roi={2048,1};
   Ipp64f mean;
   IppStatus st;
   ippiSet_8u_C1R( 0, img, 2048, roi );
   st = ippiAddRandUniform_8u_C1IR(img, 2048, roi, 0, 255, &seed);
   ippiMean_8u_C1R( img, 2048, roi, &mean );
   ippiMinMax_8u_C1R( img, 2048, roi, &mn, &mx );
   printf( "[%d..%d], mean=%.3f\n", mn, mx, mean );
   return st;
}
```

AddRandGauss

Generates random samples with Gaussian distribution and adds them to an image data.

Syntax

```
IppStatus ippiAddRandGauss_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize
roiSize, Ipp<datatype> mean, Ipp<datatype> stDev, unsigned int* pSeed);
```

Supported values for `mod`:

`8u_C1IR`	`16u_C1IR`	`16s_C1IR`	`32f_C1IR`
`8u_C3IR`	`16u_C3IR`	`16s_C3IR`	`32f_C3IR`
`8u_C4IR`	`16u_C4IR`	`16s_C4IR`	`32f_C4IR`
`8u_AC4IR`	`16u_AC4IR`	`16s_AC4IR`	`32f_AC4IR`

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrcDst	Pointer to the source and destination image ROI.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image.
roiSize	Size of the image ROI in pixels.
mean	The mean of the Gaussian distribution.
stDev	The standard deviation of the Gaussian distribution.
pSeed	The initial seed value for the pseudo-random number generator.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

The function generates samples with Gaussian distribution that have the mean value mean and standard deviation stdev and adds them to a source image ROI pointed to by pSrcDst.

The resulting pixel values that exceed the image data range are saturated to the respective data-range limits. To obtain an image which contains pure noise with Gaussian distribution, call ippiAddRandGauss using a source image with zero data as input.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if pSrcDst or pSeed pointer is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if srcDstStep has a zero or negative value.

ImageJaehne

Creates Jaehne test image.

Syntax

```
IppStatus ippiImageJaehne_<mod>(Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_C4R	16u_C4R	16s_C4R	32f_C4R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the destination image ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function creates a specific one- or three-channel test image that has been first introduced to digital image processing by B.Jaehne (see [Jae95]).

The destination image pixel values are computed according to the following formula:

Dst(x,y) = A*sin(0.5*IPP_PI* (x_2^2 + y_2^2) /roiSize.height),

where x, y are pixel coordinates varying in the range

0≤x≤roiSize.width-1,0≤y≤roiSize.height-1;

IPP_PI is the library constant that stands for π value.

x2 = (x-roiSize.width+1)/2.0,

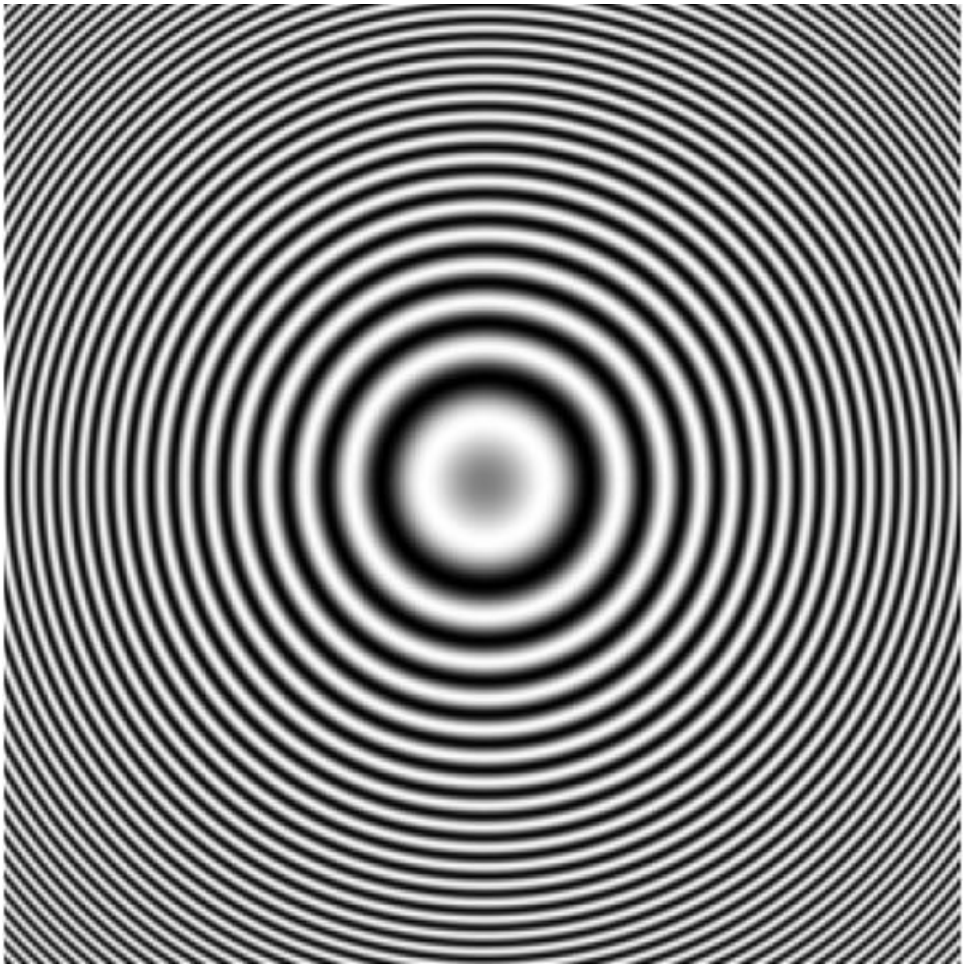
y2 = (y-roiSize.height+1)/2.0,

A is the constant value that depends upon the image type being created.

For the 32f floating point data, the pixel values in the created image can vary in the range between 0 (inclusive) and 1 (exclusive).

Figure Example of a Generated Jaehne's Test Image illustrates an example of a test image generated by the ippiImageJaehne function.

Example of a Generated Jaehne's Test Image |



These test images can be effectively used when you need to visualize and interpret the results of applying filtering functions, similarly to what is proposed in [Jae95].

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if the `pDst` pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize` has a field with a zero or negative value, or if `dstStep` is less than or equal to zero.

ImageRamp

Creates a test image that has an intensity ramp.

Syntax

```
IppStatus ippiImageRamp_<mod>(Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, float offset, float slope, IppiAxis axis);
```

Supported values for `mod`:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_C4R	16u_C4R	16s_C4R	32f_C4R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the destination image ROI in pixels.
offset	Offset value.
slope	Slope coefficient.
axis	Specifies the direction of the image intensity ramp; can be one of the following: * ippAxsHorizontal for the ramp in X - direction * ippAxsVertical for the ramp in Y - direction * ippAxsBoth for the ramp in both X and Y - directions

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

The function creates a one- or three-channel image that can be used as a test image to examine the effect of applying different image processing functions.

The destination image pixel values are computed according to one of the following formulas:

dst(x,y) = offset+slope* x, if axis = ippAxsHorizontal,

dst(x,y) = offset+slope* y, if axis = ippAxsVertical,

dst(x,y) = offset+slope* x * y, if axis = ippAxsBoth,

where x, y are pixel coordinates varying in the range

0≤x≤roiSize.width-1,0≤y≤roiSize.height-1;

Note that linear transform coefficients offset and slope have floating-point values for all function flavors. The computed pixel values that exceed the image data range are saturated to the respective data-range limits.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if the pDst pointer is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with a zero or negative value, or if dstStep is less than or equal to zero.

Example

The code example below illustrates how to use the ippiImageRamp function.

```
IppStatus ramp( void ){
   Ipp8u dst[8*4];
   IppiSize roiSize = { 8, 4 };
   return ippiImageRamp_8u_C1R( dst, 8, roiSize, 0.0f, 256.0f/7,  ippAxsHorizontal);
}
```

The destination image contains the following data:

```
00 25 49 6E 92 B7 DB FF
00 25 49 6E 92 B7 DB FF
00 25 49 6E 92 B7 DB FF
00 25 49 6E 92 B7 DB FF
```

SampleLine

Puts a raster line into buffer.

Syntax

```
IppStatus ippiSampleLine_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize
roiSize, Ipp<datatype>* pDst, IppiPoint pt1, IppiPoint pt2);
```

Supported values for mod:

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Pointer to the ROI in the source raster image.

srcStep	Distance in bytes between starts of consecutive lines in the raster image.
roiSize	Size of the image ROI in pixels.
pDst	Pointer to the destination buffer. The buffer is to store at least max(|pt2.x - pt1.x| + 1, |pt2.y - pt1.y| + 1) points.
pt1	Starting point of the line.
pt2	Ending point of the line.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function iterates through the points that belong to the raster line using the 8-point connected Bresenham algorithm, and puts the resulting pixels into the destination buffer.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize.width or roiSize.height is less than or equal to zero.
ippStsStepErr	Indicates an error condition if srcStep is less than roiSize.width*<pixelSize>.
ippStsNotEvenStepErr	Indicates an error when the step for the floating-point image cannot be divided by 4.
ippStsOutOfRangeErr	Indicates an error when any of the line ending points is outside the image.

Example

The code example below shows how to use the function ippiSampleLine_8u_C1R .

```
void func_sampleline()
{
    Ipp8u pSrc[5*4] = { 0, 1, 2, 3, 4,
                        5, 6, 7, 8, 9,
                        0, 9, 8, 7, 6,
                        5, 4, 3, 2, 1 };
    IppiSize roiSize = {5, 4};
    IppiPoint pt1 = {1, 1};
    IppiPoint pt2 = {2, 3};
    Ipp8u pDst[3];
    int srcStep = 5;


    ippiSampleLine_8u_C1R( pSrc, srcStep, roiSize, pDst, pt1, pt2 );
```

```
  printf("%Result: d,    %d, %d\n", pDst[0], pDst[1], pDst[2] ); // << this wrong line
  printf("%Result: %d, %d, %d\n", pDst[0], pDst[1], pDst[2] ); // this is correct line
}
```

Result: 6, 9, 3

ZigzagFwd8x8

Converts a conventional order to the zigzag order.

Syntax

```
IppStatus ippiZigzagFwd8x8_16s_C1(const Ipp16s* pSrc, Ipp16s* pDst);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source data.
`pDst`	Pointer to the destination data.

Description

This function rearranges data in an 8x8 block from a conventional order (left-to-right, top-to-bottom) to the zigzag sequence.

Figure Zigzag Sequence specifies the resulting zigzag sequence.

Zigzag Sequence |

0	1	5	6	14	15	27	28
2	4	7	13	16	26	29	42
3	8	12	17	25	30	41	43
9	11	18	24	31	40	44	53
10	19	23	32	39	45	52	54
20	22	33	38	46	51	55	60
21	34	37	47	50	56	59	61
35	36	48	49	57	58	62	63

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when any of the specified pointers is `NULL`.

Example

The code example below shows how to use the `ippiZigzagFwd8x8_16s_C1` function.

```
Ipp16s src[8*8] = {
                    0, 1, 5, 7, 9, 2, 4, 1,
                    5, 4, 8, 6, 3, 8, 0, 3,
                    6, 2, 6, 8, 1, 4, 2, 8,
                    4, 3, 2, 9, 3, 0, 6, 6,
                    7, 7, 3, 0, 4, 1, 0, 9,
                    5, 1, 9, 2, 5, 7, 1, 7,
                    0, 3, 5, 0, 7, 5, 9, 8,
                    2, 9, 1, 4, 6, 8, 2, 3
                  };
Ipp16s dst[8*8];


ippiZigzagFwd8x8_16s_C1 ( src, dst );


Result:
```

```
0 1 5 7 9 2 4 1
5 4 8 6 3 8 0 3
6 2 6 8 1 4 2 8
4 3 2 9 3 0 6 6     src  //conventional order
7 7 3 0 4 1 0 9
5 1 9 2 5 7 1 7
0 3 5 0 7 5 9 8
2 9 1 4 6 8 2 3




0 1 5 6 4 5 7 8
2 4 7 3 6 6 9 2
3 8 2 7 5 0 1 3
9 1 8 4 1 0 4 3      dst  //zigzag order
0 9 3 2 9 5 2 4
0 2 3 8 6 1 5 0
1 4 7 7 0 6 9 1
5 6 8 9 7 8 2 3
```

ZigzagInv8x8

Converts a zigzag order to the conventional order.

Syntax

```
IppStatus ippiZigzagInv8x8_16s_C1(const Ipp16s* pSrc, Ipp16s* pDst);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source data.
`pDst`	Pointer to the destination data.

Description

This function rearranges data in an 8x8 block from a zigzag sequence to the conventional order (left-to-right, top-to-bottom).

Figure Zigzag Sequence specifies the resulting zigzag sequence.

Return Values

`ippStsNoErr`	Indicates no error.

ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.

Image Arithmetic and Logical Operations

This chapter describes functions that modify pixel values of an image buffer using arithmetic or logical operations. It also includes functions that perform image compositing based on opacity (alpha-blending).

An additional suffix C, if present in the function name, indicates operation with a constant. Arithmetic functions that operate on integer data perform fixed scaling of the internally computed results. In case of overflow the result value is saturated to the destination data type range.

NOTE Most arithmetic and logical functions support data with 1-,3-, or 4-channel pixel values. In the alpha channel case (AC4), the alpha channels are not processed.

- Arithmetic Operations
- Logical Operations
- Alpha Composition

Arithmetic Operations

Functions described in this section perform arithmetic operations on pixel values. Arithmetic operations include addition, multiplication, subtraction, and division of pixel values of two images as well as similar operations on a single image and a constant. Computation of an absolute value, square, square root, exponential, and natural logarithm of pixels in an image buffer is also supported. Functions of this group perform operations on each pixel in the source buffer(s), and write the results into the destination buffer. Some functions also support processing of images with complex data.

- Add
- AddC
- AddSquare
- AddProduct
- AddWeighted
- Mul
- MulC
- MulC64f
- MulScale
- MulCScale
- Sub
- SubC
- Div
- Div_Round
- DivC
- Abs
- AbsDiff
- AbsDiffC
- Sqr
- Sqrt
- Ln
- Exp
- DotProd
- DotProdCol
- Complement

Add

Adds pixel values of two images.

Syntax

Case 1: Not-in-place operation on integer or complex data

```
IppStatus ippiAdd_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>*
pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int
scaleFactor);
```

Supported values for `mod`:

8u_C1RSfs	16u_C1RSfs	16s_C1RSfs
8u_C3RSfs	16u_C3RSfs	16s_C3RSfs
8u_C4RSfs	16u_C4RSfs	16s_C4RSfs
8u_AC4RSfs	16u_AC4RSfs	16s_AC4RSfs

Case 2: Not-in-place operation on floating point or complex data

```
IppStatus ippiAdd_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>*
pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

32f_C1R
32f_C3R
32f_C4R

```
IppStatus ippiAdd_32f_AC4R(const Ipp32f* pSrc1, int src1Step, const Ipp32f* pSrc2, int
src2Step, Ipp32f* pDst, int dstStep, IppiSize roiSize);
```

Case 3: In-place operation on integer or complex data

```
IppStatus ippiAdd_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pSrcDst,
int srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C1IRSfs	16u_C1IRSfs	16s_C1IRSfs
8u_C3IRSfs	16u_C3IRSfs	16s_C3IRSfs
8u_AC4IRSfs	16u_AC4IRSfs	16s_AC4IRSfs
8u_C4IRSfs	16u_C4IRSfs	16s_C4IRSfs

Case 4: In-place operation on floating point or complex data

```
IppStatus ippiAdd_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pSrcDst,
int srcDstStep, IppiSize roiSize);
```

Supported values for `mod`:

32f_C1IR

32f_C3IR
32f_AC4IR
32f_C4IR

Case 5: In-place operation using a floating point accumulator image

```
IppStatus ippiAdd_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for `mod`:

8u32f_C1IR	16u32f_C1IR

Case 6: Masked in-place operation using a floating point accumulator image

```
IppStatus ippiAdd_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, const Ipp8u* pMask, int maskStep, Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for `mod`:

8u32f_C1IMR	16u32f_C1IMR	32f_C1IMR

Case 7: Not-in-place operation on integer data with platform-aware functions

```
IppStatus ippiAdd_<mod>(const Ipp<datatype>* pSrc1, IppSizeL src1Step, const Ipp<datatype>* pSrc2, IppSizeL src2Step, Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C1RSfs_L	16u_C1RSfs_L	16s_C1RSfs_L
8u_C3RSfs_L	16u_C3RSfs_L	16s_C3RSfs_L
8u_C4RSfs_L	16u_C4RSfs_L	16s_C4RSfs_L
8u_AC4RSfs_L	16u_AC4RSfs_L	16s_AC4RSfs_L

Case 8: Not-in-place operation on floating point data with platform-aware functions

```
IppStatus ippiAdd_<mod>(const Ipp<datatype>* pSrc1, IppSizeL src1Step, const Ipp<datatype>* pSrc2, IppSizeL src2Step, Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize);
```

Supported values for `mod`:

32f_C1R_L
32f_C3R_L
32f_C4R_L
32f_AC4R_L

Case 9: In-place operation on integer data with platform-aware functions

```
IppStatus ippiAdd_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>* pSrcDst, IppSizeL srcDstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C1IRSfs_L	16u_C1IRSfs_L	16s_C1IRSfs_L
8u_C3IRSfs_L	16u_C3IRSfs_L	16s_C3IRSfs_L
8u_C4IRSfs_L	16u_C4IRSfs_L	16s_C4IRSfs_L
8u_AC4IRSfs_L	16u_AC4IRSfs_L	16s_AC4IRSfs_L

Case 10: In-place operation on floating point data with platform-aware functions

```
IppStatus ippiAdd_<mod>(const Ipp<datatype>* pSrc1, IppSizeL src1Step, Ipp<datatype>*
pDst, IppSizeL dstStep, IppiSizeL roiSize);
```

Supported values for `mod`:

32f_C1IR_L
32f_C3IR_L
32f_C4IR_L
32f_AC4IR_L

Case 11: Not-in-place operation on integer data with threading layer (TL) functions

```
IppStatus ippiAdd_<mod>(const Ipp<datatype>* pSrc1, IppSizeL src1Step, const
Ipp<datatype>* pSrc2, IppSizeL src2Step, Ipp<datatype>* pDst, IppSizeL dstStep,
IppiSizeL roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C1RSfs_LT	16u_C1RSfs_LT	16s_C1RSfs_LT
8u_C3RSfs_LT	16u_C3RSfs_LT	16s_C3RSfs_LT
8u_C4RSfs_LT	16u_C4RSfs_LT	16s_C4RSfs_LT
8u_AC4RSfs_LT	16u_AC4RSfs_LT	16s_AC4RSfs_LT

Case 12: Not-in-place operation on floating point data with TL functions

```
IppStatus ippiAdd_<mod>(const Ipp<datatype>* pSrc1, IppSizeL src1Step, const
Ipp<datatype>* pSrc2, IppSizeL src2Step, Ipp<datatype>* pDst, IppSizeL dstStep,
IppiSizeL roiSize);
```

Supported values for `mod`:

32f_C1R_LT
32f_C3R_LT
32f_C4R_LT
32f_AC4R_LT

Case 13: In-place operation on integer data with TL functions

```
IppStatus ippiAdd_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>*
pSrcDst, IppSizeL srcDstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C1IRSfs_LT	16u_C1IRSfs_LT	16s_C1IRSfs_LT
8u_C3IRSfs_LT	16u_C3IRSfs_LT	16s_C3IRSfs_LT
8u_C4IRSfs_LT	16u_C4IRSfs_LT	16s_C4IRSfs_LT
8u_AC4IRSfs_LT	16u_AC4IRSfs_LT	16s_AC4IRSfs_LT

Case 14: In-place operation on floating point data with TL functions

```
IppStatus ippiAdd_<mod>(const Ipp<datatype>* pSrc1, IppSizeL src1Step, Ipp<datatype>*
pDst, IppSizeL dstStep, IppiSizeL roiSize);
```

Supported values for `mod`:

32f_C1IR_LT
32f_C3IR_LT
32f_C4IR_LT
32f_AC4IR_LT

Case 15: In-place operation on integer data with TL functions based on classic API

```
IppStatus ippiAdd_<mod>(const Ipp<datatype>* pSrc, IppSize srcStep, Ipp<datatype>*
pSrcDst, IppSize srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for `mod`:

16s_C1IRSfs_T	32s_C1IRSfs_T
16s_C3IRSfs_T	
16s_C4IRSfs_T	

Include Files

`ippcv.h`

`ippi.h`

`ippi_l.h`

`ippi_tl.h`

Domain Dependencies

Flavors declared in `ippi.h`:

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Flavors declared in `ippcv.h`:

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Flavors declared in `ippi64x.h`:

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`, `ippcore_tl.lib`, `ippi_tl.lib`

Parameters

pSrc, pSrc1, pSrc2	Pointer to the ROI in the source images.
srcStep, src1Step, src2Step	Distance, in bytes, between the starting points of consecutive lines in the source images.
pDst	Pointer to the destination image ROI.
dstStep	Distance, in bytes, between the starting points of consecutive lines in the destination image.
pSrc	Pointer to the first source image ROI for the in-place operation.
srcStep	Distance, in bytes, between the starting points of consecutive lines in the first source image for the in-place operation.
pSrcDst	Pointer to the second source and destination image ROI for the in-place operation.
srcDstStep	Distance, in bytes, between the starting points of consecutive lines in the source and destination image for the in-place operation.
pMask	Pointer to the mask image ROI for the masked operation.
maskStep	Distance, in bytes, between the starting points of consecutive lines in the mask image for the masked operation.
roiSize	Size of the source and destination ROI in pixels.
scaleFactor	Scale factor (see Integer Result Scaling).

Description

This function operates with ROI.

This function adds corresponding pixel values of two source image buffers and places the results in a destination buffer. In case of operations on integer data, the resulting values are scaled by `scaleFactor`. For complex data, the function processes both real and imaginary parts of pixel values. Some function flavors add `8u`, `8s`, `16u`, or `32f` source image pixel values to a floating point accumulator image in-place. Addition of pixel values in case of a masked operation is performed only if the respective mask value is non-zero; otherwise, the accumulator pixel value remains unchanged.

NOTE For the functions that operate on complex data, step values must be positive. For the functions that use an accumulator image, step values must be no less than `roiSize.width*<pixelSize>`.

Functions with `AC4` descriptor do not process alpha channels.

Function flavors described in Case 5 and Case 6 are declared in the `ippcv.h`.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition when any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition when roiSize has a field with zero or negative value.
ippStsStepErr	Indicates an error condition in the following cases: • For functions that operate on complex data, if any of the specified step values is zero or negative. • For functions using an accumulator image, if any of the specified step values is less than roiSize.width * <pixelSize>.
ippStsNotEvenStepErr	Indicates an error condition when one of step values for floating-point images cannot be divided by 4.

Example

The code example below shows how to use the function ippiAdd_8u_C1RSfs.

```
Ipp8u src1[8*4] = {8, 4, 2, 1, 0, 0, 0, 0,
                   8, 4, 2, 1, 0, 0, 0, 0,
                   8, 4, 2, 1, 0, 0, 0, 0,
                   8, 4, 2, 1, 0, 0, 0, 0};
Ipp8u src2[8*4] = {4, 3, 2, 1, 0, 0, 0, 0,
                   4, 3, 2, 1, 0, 0, 0, 0,
                   4, 3, 2, 1, 0, 0, 0, 0,
                   4, 3, 2, 1, 0, 0, 0, 0};
Ipp8u dst[8*4];
IppiSize srcRoi = { 4, 4 };
Int scaleFactor = 1;    // later examples for 2 and -2 values


ippiAdd_8u_C1RSfs (src1, 8, src2, 8, dst, 4, srcRoi, scaleFactor );


Result:
    src1                src2
8 4 2 1 0 0 0 0    4 3 2 1 0 0 0 0
8 4 2 1 0 0 0 0    4 3 2 1 0 0 0 0
8 4 2 1 0 0 0 0    4 3 2 1 0 0 0 0
8 4 2 1 0 0 0 0    4 3 2 1 0 0 0 0


dst >>    scaleFactor = 1        scaleFactor = 2    ScaleFactor = -2
             6 4 2 1                3 2 1 0            48 28 16 8
             6 4 2 1                3 2 1 0            48 28 16 8
             6 4 2 1                3 2 1 0            48 28 16 8
             6 4 2 1                3 2 1 0            48 28 16 8
```

See Also

Regions of Interest in Intel IPP

AddC

Adds a constant to pixel values of an image.

Syntax

Case 1: Not-in-place operation on one-channel integer or complex data

```
IppStatus ippiAddC_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype> value,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C1RSfs	16u_C1RSfs	16s_C1RSfs

Case 2: Not-in-place operation on multi-channel integer or complex data

```
IppStatus ippiAddC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C3RSfs	16u_C3RSfs	16s_C3RSfs

```
IppStatus ippiAddC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[4], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C4RSfs	16u_C4RSfs	16s_C4RSfs

```
IppStatus ippiAddC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_AC4RSfs	16u_AC4RSfs	16s_AC4RSfs

Case 3: Not-in-place operation on one-channel floating-point or complex data

```
IppStatus ippiAddC_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype> value,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

32f_C1R

Case 4: Not-in-place operation on multi-channel floating-point or complex data

```
IppStatus ippiAddC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

32f_C3R

```
IppStatus ippiAddC_32f_C4R(const Ipp32f* pSrc, int srcStep, const Ipp32f value[4],
Ipp32f* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiAddC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

32f_AC4R

Case 5: In-place operation on one-channel integer or complex data

```
IppStatus ippiAddC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst, int srcDstStep,
IppiSize roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C1IRSfs	16u_C1IRSfs	16s_C1IRSfs

Case 6: In-place operation on multi-channel integer or complex data

```
IppStatus ippiAddC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pSrcDst, int
srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C3IRSfs	16u_C3IRSfs	16s_C3IRSfs
8u_AC4IRSfs	16u_AC4IRSfs	16s_AC4IRSfs

```
IppStatus ippiAddC_<mod>(const Ipp<datatype> value[4], Ipp<datatype>* pSrcDst, int
srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C4IRSfs	16u_C4IRSfs	16s_C4IRSfs

Case 7: In-place operation on one-channel floating-point or complex data

```
IppStatus ippiAddC_32f_C1IR(Ipp32f value, Ipp32f* pSrcDst, int srcDstStep, IppiSize
roiSize);
```

Case 8: In-place operation on multi-channel floating-point or complex data

```
IppStatus ippiAddC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pSrcDst, int
srcDstStep, IppiSize roiSize);
```

Supported values for `mod`:

32f_C3IR
32f_AC4IR

```
IppStatus ippiAddC_32f_C4IR(const Ipp32f value[4], Ipp32f* pSrcDst, int srcDstStep,
IppiSize roiSize);
```

Case 9: Not-in-place operation on one-channel integer data with platform-aware functions

```
IppStatus ippiAddC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>
value, Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C1RSfs_L	16u_C1RSfs_L	16s_C1RSfs_L

Case 10: Not-in-place operation on multi-channel integer data with platform-aware functions

```
IppStatus ippiAddC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, const
Ipp<datatype> value[3], Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int
scaleFactor);
```

Supported values for `mod`:

8u_C3RSfs_L	16u_C3RSfs_L	16s_C3RSfs_L
8u_AC4RSfs_L	16u_AC4RSfs_L	16s_AC4RSfs_L

```
IppStatus ippiAddC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, const
Ipp<datatype> value[4], Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int
scaleFactor);
```

Supported values for `mod`:

8u_C4RSfs_L	16u_C4RSfs_L	16s_C4RSfs_L

Case 11: Not-in-place operation on one-channel floating point data with platform-aware functions

```
IppStatus ippiAddC_32f_C1R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f value,
Ipp32f* pDst, IppSizeL dstStep, IppiSizeL roiSize);
```

Case 12: Not-in-place operation on multi-channel floating point data with platform-aware functions

```
IppStatus ippiAddC_<mod>(const Ipp32f* pSrc, IppSizeL srcStep, const Ipp32f value[3],
Ipp32f* pDst, IppSizeL dstStep, IppiSizeL roiSize);
```

Supported values for `mod`:

32f_C3R_L
32f_AC4R_L

```
IppStatus ippiAddC_32f_C4R_L(const Ipp32f* pSrc, IppSizeL srcStep, const Ipp32f
value[4], Ipp32f* pDst, IppSizeL dstStep, IppiSizeL roiSize);
```

Case 13: In-place operation on one-channel integer data with platform-aware functions

```
IppStatus ippiAddC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst, IppSizeL
srcDstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C1IRSfs_L	16u_C1IRSfs_L	16s_C1IRSfs_L

Case 14: In-place operation on multi-channel integer data with platform-aware functions

```
IppStatus ippiAddC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pSrcDst, IppSizeL
srcDstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C3IRSfs_L	16u_C3IRSfs_L	16s_C3IRSfs_L
8u_AC4IRSfs_L	16u_AC4IRSfs_L	16s_AC4IRSfs_L

```
IppStatus ippiAddC_<mod>(const Ipp<datatype> value[4], Ipp<datatype>* pSrcDst, IppSizeL
srcDstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C4IRSfs_L	16u_C4IRSfs_L	16s_C4IRSfs_L

Case 15: In-place operation on one-channel floating point data with platform-aware functions

```
IppStatus ippiAddC_32f_C1IR_L(Ipp32f value, Ipp32f* pSrcDst, IppSizeL srcDstStep,
IppiSizeL roiSize);
```

Case 16: In-place operation on multi-channel integer data with platform-aware functions

```
IppStatus ippiAddC_<mod>(const Ipp32f value[3], Ipp32f* pSrcDst, IppSizeL srcDstStep,
IppiSizeL roiSize);
```

Supported values for `mod`:

32f_C3IR_L
32f_AC4IR_L

```
IppStatus ippiAddC_32f_C4IR_L(const Ipp32f value[4], Ipp32f* pSrcDst, IppSizeL
srcDstStep, IppiSizeL roiSize);
```

Case 17: Not-in-place operation on one-channel integer data with threading layer (TL) functions

```
IppStatus ippiAddC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>
value, Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C1RSfs_LT	16u_C1RSfs_LT	16s_C1RSfs_LT

Case 18: Not-in-place operation on multi-channel integer data with TL functions

```
IppStatus ippiAddC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, const
Ipp<datatype> value[3], Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int
scaleFactor);
```

Supported values for `mod`:

8u_C3RSfs_LT	16u_C3RSfs_LT	16s_C3RSfs_LT
8u_AC4RSfs_LT	16u_AC4RSfs_LT	16s_AC4RSfs_LT

```
IppStatus ippiAddC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, const
Ipp<datatype> value[4], Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int
scaleFactor);
```

Supported values for `mod`:

8u_C4RSfs_LT	16u_C4RSfs_LT	16s_C4RSfs_LT

Case 19: Not-in-place operation on one-channel floating point data with TL functions

```
IppStatus ippiAddC_32f_C1R_LT(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f value,
Ipp32f* pDst, IppSizeL dstStep, IppiSizeL roiSize);
```

Case 20: Not-in-place operation on multi-channel floating point data with TL functions

```
IppStatus ippiAddC_<mod>(const Ipp32f* pSrc, IppSizeL srcStep, const Ipp32f value[3],
Ipp32f* pDst, IppSizeL dstStep, IppiSizeL roiSize);
```

Supported values for `mod`:

32f_C3R_LT

32f_AC4R_LT

```
IppStatus ippiAddC_32f_C4R_LT(const Ipp32f* pSrc, IppSizeL srcStep, const Ipp32f
value[4], Ipp32f* pDst, IppSizeL dstStep, IppiSizeL roiSize);
```

Case 21: In-place operation on one-channel integer data with TL functions

```
IppStatus ippiAddC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst, IppSizeL
srcDstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C1IRSfs_LT	16u_C1IRSfs_LT	16s_C1IRSfs_LT

Case 22: In-place operation on multi-channel integer data with TL functions

```
IppStatus ippiAddC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pSrcDst, IppSizeL
srcDstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C3IRSfs_LT	16u_C3IRSfs_LT	16s_C3IRSfs_LT
8u_AC4IRSfs_LT	16u_AC4IRSfs_LT	16s_AC4IRSfs_LT

```
IppStatus ippiAddC_<mod>(const Ipp<datatype> value[4], Ipp<datatype>* pSrcDst, IppSizeL
srcDstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C4IRSfs_LT	16u_C4IRSfs_LT	16s_C4IRSfs_LT

Case 23: In-place operation on one-channel floating point data with TL functions

```
IppStatus ippiAddC_32f_C1IR_LT(Ipp32f value, Ipp32f* pSrcDst, IppSizeL srcDstStep,
IppiSizeL roiSize);
```

Case 24: In-place operation on multi-channel integer data with TL functions

```
IppStatus ippiAddC_<mod>(const Ipp32f value[3], Ipp32f* pSrcDst, IppSizeL srcDstStep,
IppiSizeL roiSize);
```

Supported values for `mod`:

32f_C3IR_LT
32f_AC4IR_LT

```
IppStatus ippiAddC_32f_C4IR_LT(const Ipp32f value[4], Ipp32f* pSrcDst, IppSizeL
srcDstStep, IppiSizeL roiSize);
```

Include Files

`ippi.h`

Flavors with the `_LT` suffix: `ippi_tl.h`

Flavors with the `_L` suffix: `ippi_l.h`

Domain Dependencies

Flavors declared in `ippi.h`:

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Flavors declared in `ippi_tl.h`:

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`, `ippcore_tl.lib`, `ippi_tl.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`value`	The constant value to add to image pixel values (constant vector in case of multi-channel images).
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image.
`pSrcDst`	Pointer to the source and destination image ROI for the in-place operation.
`srcDstStep`	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
`roiSize`	Size of the source and destination ROI in pixels.
`scaleFactor`	Scale factor (see Integer Result Scaling).

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function changes the image intensity by adding `value` to image pixel values. For one-channel images, a positive `value` brightens the image (increases the intensity); a negative value darkens the image (decreases the intensity). For multi-channel images, the components of a constant vector `value` are added to pixel channel values. For complex data, the function processes both real and imaginary parts of pixel values.

NOTE Step values must be positive for functions that operate on complex data.

In case of operations on integer data, the resulting values are scaled by `scaleFactor`.

Note that the functions with `AC4` descriptor do not process alpha channels.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.

ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if any of the specified buffer step values is zero or negative for functions that operate on complex data.

AddSquare

Adds squared pixel values of a source image to floating-point pixel values of an accumulator image.

Syntax

Case 1: In-place operation

```
IppStatus ippiAddSquare_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp32f*
pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for mod:

8u32f_C1IR	16u32f_C1IR	32f_C1IR

Case 2: Masked in-place operation

```
IppStatus ippiAddSquare_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, const Ipp8u*
pMask, int maskStep, Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for mod:

8u32f_C1IMR	16u32f_C1IMR	32f_C1IMR

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pMask	Pointer to the mask image.
maskStep	Distance in bytes between starts of consecutive lines in the mask image.
pSrcDst	Pointer to the destination (accumulator) image ROI.
srcDstStep	Distance in bytes between starts of consecutive lines in the accumulator image.

roiSize	Size of the image ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function adds squared pixel values of the source image pSrc to floating-point pixel values of the accumulator image pSrcDst as follows:

$pSrcDst(x,y) = pSrcDst(x,y) + pSrc(x,y)^2$

Addition of the squared pixel values in case of a masked operation is performed only if the respective mask value is nonzero; otherwise, the accumulator pixel value remains unchanged.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when roiSize.width or roiSize.height is negative.
ippStsStepErr	Indicates an error when srcStep, maskStep, or srcDstStep is less than roiSize.width * <pixelSize>.
ippStsNotEvenStepErr	Indicates an error condition if one of step values for floating-point images cannot be divided by 4.

AddProduct

Adds product of pixel values of two source images to floating-point pixel values of an accumulator image.

Syntax

Case 1: In-place operation

```
IppStatus ippiAddProduct_<mod>(const Ipp<srcDatatype>* pSrc1, int src1Step, const
Ipp<srcDatatype>* pSrc2, int src2Step, Ipp32f* pSrcDst, int srcDstStep, IppiSize
roiSize);
```

Supported values for mod:

8u32f_C1IR	16u32f_C1IR	32f_C1IR

Case 2: Masked in-place operation

```
IppStatus ippiAddProduct_<mod>(const Ipp<srcDatatype>* pSrc1, int src1Step, const
Ipp<srcDatatype>* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep, Ipp32f*
pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for mod:

8u32f_C1IMR	16u32f_C1IMR	32f_C1IMR

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc1, pSrc2	Pointers to the source images ROI.
src1Step, src2Step	Distances in bytes between starts of consecutive lines in the source images.
pMask	Pointer to the mask image.
maskStep	Distance in bytes between starts of consecutive lines in the mask image.
pSrcDst	Pointer to the destination (accumulator) image ROI.
srcDstStep	Distance in bytes between starts of consecutive lines in the accumulator image.
roiSize	Size of the image ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function adds the product of pixel values of two source images pSrc1 and pSrc2 to floating-point pixel values of the accumulator image pSrcDst as given by:

```
pSrcDst(x,y) =pSrcDst(x,y) +pSrc1(x,y) *pSrc2(x,y)
```

The products of pixel values in case of a masked operation are added only if the respective mask value is nonzero; otherwise, the accumulator pixel value remains unchanged.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is null.
ippStsSizeErr	Indicates an error when roiSize.width or roiSize.height is negative.
ippStsStepErr	Indicates an error if src1Step, src2Step, maskStep, or srcDdstStep is less than roiSize.width * <pixelSize>.
ippStsNotEvenStepErr	Indicates an error condition if one of step values for floating-point images cannot be divided by 4.

AddWeighted

Adds weighted pixel values of a source image to floating-point pixel values of an accumulator image.

Syntax

Case 1: In-place operation

```
IppStatus ippiAddWeighted_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp32f*
pSrcDst, int srcDstStep, IppiSize roiSize, Ipp32f alpha);
```

Supported values for `mod`:

`8u32f_C1IR`	`16u32f_C1IR`	`32f_C1IR`

Case 2: Masked in-place operation

```
IppStatus ippiAddWeighted_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, const Ipp8u*
pMask, int maskStep, Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize, Ipp32f alpha);
```

Supported values for `mod`:

`8u32f_C1IMR`	`16u32f_C1IMR`	`32f_C1IMR`

Case 3: Not-in-place operation

```
IppStatus ippiAddWeighted_32f_C1R(const Ipp32f* pSrc1, int src1Step, const Ipp32f*
pSrc2, int src2Step, Ipp32f* pDst, int dstStep, IppiSize roiSize, Ipp32f alpha);
```

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the source image ROI for the in-place operation.
`pSrc1`, `pSrc2`	Pointers to the ROI in the source images.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image for the in-place operation.
`src1Step`, `src2Step`	Distance in bytes between starts of consecutive lines in the source images.
`pMask`	Pointer to the mask image.
`maskStep`	Distance in bytes between starts of consecutive lines in the mask image.
`pSrcDst`	Pointer to the destination (accumulator) image ROI for the in-place operation.
`srcDstStep`	Distance in bytes between starts of consecutive lines in the accumulator image.

pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the image ROI in pixels.
alpha	Weight a of the source image.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function adds pixel values of the source image pSrc1 multiplied by a weight factor alpha to pixel values of the image pSrc2 multiplied by (1-alpha) and stores result in the pDst as follows:

```
pDst(x,y) = pSrc1(x,y)* alpha +pSrc2(x,y)*(1- alpha).
```

The in-place flavors of the function adds pixel values of the source image pSrc multiplied by a weight factor alpha to floating-point pixel values of the accumulator image pSrcDst multiplied by (1-alpha) as follows:

```
pSrcDst(x,y) = pSrcDst(x,y)* (1-alpha) +pSrc(x,y)*alpha
```

Addition of the weighted pixel values in case of a masked operation is performed only if the respective mask value is nonzero; otherwise, the accumulator pixel value remains unchanged.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when roiSize.width or roiSize.height is equal to 0 or negative.
ippStsStepErr	Indicates an error when one of the step values is equal to zero, or is less than roiSize.width * <pixelSize>.
ippStsNotEvenStepErr	Indicates an error when one of step values for floating-point images cannot be divided by 4.

Mul

Multiplies pixel values of two images.

Case 1: Not-in-place operation on integer or complex data

```
IppStatus ippiMul_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>*
pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int
scaleFactor);
```

Supported values for mod:

8u_C1RSfs	16u_C1RSfs	16s_C1RSfs
8u_C3RSfs	16u_C3RSfs	16s_C3RSfs

8u_AC4RSfs	16u_AC4RSfs	16s_AC4RSfs
8u_C4RSfs	16u_C4RSfs	16s_C4RSfs

Case 2: Not-in-place operation on floating-point or complex data

```
IppStatus ippiMul_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>*
pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

32f_C1R	32f_AC4R
32f_C3R	32f_C4R

Case 3: In-place operation on integer or complex data

```
IppStatus ippiMul_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pSrcDst,
int srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C1IRSfs	16u_C1IRSfs	16s_C1IRSfs
8u_C3IRSfs	16u_C3IRSfs	16s_C3IRSfs
8u_AC4IRSfs	16u_AC4IRSfs	16s_AC4IRSfs
8u_C4IRSfs	16u_C4IRSfs	16s_C4IRSfs

Case 4: In-place operation on floating-point or complex data

```
IppStatus ippiMul_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pSrcDst,
int srcDstStep, IppiSize roiSize);
```

Supported values for `mod`:

32f_C1IR
32f_C3IR
32f_AC4IR
32f_C4IR

Case 5: Not-in-place operation on integer data with platform-aware functions

```
IppStatus ippiMul_<mod>(const Ipp<datatype>* pSrc1, IppSizeL src1Step, const
Ipp<datatype>* pSrc2, IppSizeL src2Step, Ipp<datatype>* pDst, IppSizeL dstStep,
IppiSizeL roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C1RSfs_L	16u_C1RSfs_L	16s_C1RSfs_L
8u_C3RSfs_L	16u_C3RSfs_L	16s_C3RSfs_L
8u_AC4RSfs_L	16u_AC4RSfs_L	16s_AC4RSfs_L
8u_C4RSfs_L	16u_C4RSfs_L	16s_C4RSfs_L

Case 6: Not-in-place operation on floating-point data with platform-aware functions

```
IppStatus ippiMul_<mod>(const Ipp<datatype>* pSrc1, IppSizeL src1Step, const
Ipp<datatype>* pSrc2, IppSizeL src2Step, Ipp<datatype>* pDst, IppSizeL dstStep,
IppiSizeL roiSize);
```

Supported values for `mod`:

32f_C1R_L
32f_C3R_L
32f_C4R_L
32f_AC4R_L

Case 7: In-place operation on integer data with platform-aware functions

```
IppStatus ippiMul_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>*
pSrcDst, IppSizeL srcDstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C1IRSfs_L	16u_C1IRSfs_L	16s_C1IRSfs_L
8u_C3IRSfs_L	16u_C3IRSfs_L	16s_C3IRSfs_L
8u_AC4IRSfs_L	16u_AC4IRSfs_L	16s_AC4IRSfs_L
8u_C4IRSfs_L	16u_C4IRSfs_L	16s_C4IRSfs_L

Case 8: In-place operation on floating-point data with platform-aware functions

```
IppStatus ippiMul_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>*
pSrcDst, IppSizeL srcDstStep, IppiSizeL roiSize);
```

Supported values for `mod`:

32f_C1IR_L
32f_C3IR_L
32f_AC4IR_L
32f_C4IR_L

Case 9: Not-in-place operation on integer data with threading layer (TL) functions

```
IppStatus ippiMul_<mod>(const Ipp<datatype>* pSrc1, IppSizeL src1Step, const
Ipp<datatype>* pSrc2, IppSizeL src2Step, Ipp<datatype>* pDst, IppSizeL dstStep,
IppiSizeL roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C1RSfs_LT	16u_C1RSfs_LT	16s_C1RSfs_LT
8u_C3RSfs_LT	16u_C3RSfs_LT	16s_C3RSfs_LT
8u_AC4RSfs_LT	16u_AC4RSfs_LT	16s_AC4RSfs_LT
8u_C4RSfs_LT	16u_C4RSfs_LT	16s_C4RSfs_LT

Case 10: Not-in-place operation on floating-point data with TL functions

```
IppStatus ippiMul_<mod>(const Ipp<datatype>* pSrc1, IppSizeL src1Step, const
Ipp<datatype>* pSrc2, IppSizeL src2Step, Ipp<datatype>* pDst, IppSizeL dstStep,
IppiSizeL roiSize);
```

Supported values for `mod`:

32f_C1R_LT

32f_C3R_LT
32f_C4R_LT
32f_AC4R_LT

Case 11: In-place operation on integer data with TL functions

```
IppStatus ippiMul_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>*
pSrcDst, IppSizeL srcDstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C1IRSfs_LT	16u_C1IRSfs_LT	16s_C1IRSfs_LT
8u_C3IRSfs_LT	16u_C3IRSfs_LT	16s_C3IRSfs_LT
8u_AC4IRSfs_LT	16u_AC4IRSfs_LT	16s_AC4IRSfs_LT
8u_C4IRSfs_LT	16u_C4IRSfs_LT	16s_C4IRSfs_LT

Case 12: In-place operation on floating-point data with TL functions

```
IppStatus ippiMul_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>*
pSrcDst, IppSizeL srcDstStep, IppiSizeL roiSize);
```

Supported values for `mod`:

32f_C1IR_LT
32f_C3IR_LT
32f_AC4IR_LT
32f_C4IR_LT

Include Files

`ippi.h`

Flavors with the `_LT` suffix: `ippi_tl.h`

Flavors with the `_L` suffix: `ippi_l.h`

Domain Dependencies

Flavors declared in `ippi.h`:

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Flavors declared in `ippi_tl.h`:

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`, `ippcore_tl.lib`, `ippi_tl.lib`

Parameters

`pSrc`, `pSrc1`, `pSrc2`	Pointers to the source images ROI.
`srcStep`, `src1Step`, `src2Step`	Distances in bytes between starts of consecutive lines in the source images.
`pDst`	Pointer to the destination image ROI.

dstStep	Distance in bytes between starts of consecutive lines in the destination image.
pSrcDst	Pointer to the source and destination image ROI for the in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
roiSize	Size of the source and destination ROI in pixels.
scaleFactor	Scale factor (see Integer Result Scaling).

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function multiplies corresponding pixel values of two source image buffers and places the results in a destination buffer. In case of operations on integer data, the resulting values are scaled by `scaleFactor`.

For complex data, the function processes both real and imaginary parts of pixel values.

NOTE Step values must be positive for functions that operate on complex data.

Note that the functions with `AC4` descriptor do not process alpha channels.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if any of the pointers is `NULL`.
ippStsSizeErr	Indicates an error condition if `roiSize` has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if any of the specified buffer step values is zero or negative for functions that operate on complex data.

MulC

Multiplies pixel values of an image by a constant.

Syntax

Case 1: Not-in-place operation on one-channel integer or complex data

```
IppStatus ippiMulC_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype> value,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C1RSfs	16u_C1RSfs	16s_C1RSfs

Case 2: Not-in-place operation on multi-channel integer or complex data

```
IppStatus ippiMulC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C3RSfs	16u_C3RSfs	16s_C3RSfs

```
IppStatus ippiMulC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[4], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C4RSfs	16u_C4RSfs	16s_C4RSfs

```
IppStatus ippiMulC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_AC4RSfs	16u_AC4RSfs	16s_AC4RSfs

Case 3: Not-in-place operation on one-channel floating-point or complex data

```
IppStatus ippiMulC_32f_C1R(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype> value,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Case 4: Not-in-place operation on multi-channel floating-point or complex data

```
IppStatus ippiMulC_32f_C3R(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);

IppStatus ippiMulC_32f_C4R(const Ipp32f* pSrc, int srcStep, const Ipp32f value[4],
Ipp32f* pDst, int dstStep, IppiSize roiSize);

IppStatus ippiMulC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

32f_AC4R

```
IppStatus ippiMulC_32f_C4R(const Ipp32f* pSrc, int srcStep, const Ipp32f value[4],
Ipp32f* pDst, int dstStep, IppiSize roiSize);
```

Case 5: In-place operation on one-channel integer or complex data

```
IppStatus ippiMulC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst, int srcDstStep,
IppiSize roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C1IRSfs	16u_C1IRSfs	16s_C1IRSfs

Case 6: In-place operation on multi-channel integer or complex data

```
IppStatus ippiMulC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pSrcDst, int
srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C3IRSfs	16u_C3IRSfs	16s_C3IRSfs

8u_AC4IRSfs	16u_AC4IRSfs	16s_AC4IRSfs

IppStatus ippiMulC_<mod>(const Ipp<datatype> value[4], Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);

Supported values for mod:

8u_C4IRSfs	16u_C4IRSfs	16s_C4IRSfs

Case 7: In-place operation on one-channel floating-point or complex data

IppStatus ippiMulC_32f_C1IR(Ipp32f value, Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);

Case 8: In-place operation on multi-channel floating-point or complex data

IppStatus ippiMulC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);

Supported values for mod:

32f_C3IR
32f_AC4IR

IppStatus ippiMulC_32f_C4IR(const Ipp32f value[4], Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);

Case 9: Not-in-place operation on one-channel integer data with platform-aware functions

IppStatus ippiMulC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype> value, Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int scaleFactor);

Supported values for mod:

8u_C1RSfs_L	16u_C1RSfs_L	16s_C1RSfs_L

Case 10: Not-in-place operation on multi-channel integer data with platform-aware functions

IppStatus ippiMulC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, const Ipp<datatype> value[3], Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int scaleFactor);

Supported values for mod:

8u_C3RSfs_L	16u_C3RSfs_L	16s_C3RSfs_L
8u_AC4RSfs_L	16u_AC4RSfs_L	16s_AC4RSfs_L

IppStatus ippiMulC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, const Ipp<datatype> value[4], Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int scaleFactor);

Supported values for mod:

8u_C4RSfs_L	16u_C4RSfs_L	16s_C4RSfs_L

Case 11: Not-in-place operation on one-channel floating point data with platform-aware functions

IppStatus ippiMulC_32f_C1R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f value, Ipp32f* pDst, IppSizeL dstStep, IppiSizeL roiSize);

Case 12: Not-in-place operation on multi-channel floating point data with platform-aware functions

```
IppStatus ippiMulC_<mod>(const Ipp32f* pSrc, IppSizeL srcStep, const Ipp32f value[3],
Ipp32f* pDst, IppSizeL dstStep, IppiSizeL roiSize);
```

Supported values for `mod`:

32f_C3R_L
32f_AC4R_L

```
IppStatus ippiMulC_32f_C4R_L(const Ipp32f* pSrc, IppSizeL srcStep, const Ipp32f
value[4], Ipp32f* pDst, IppSizeL dstStep, IppiSizeL roiSize);
```

Case 13: In-place operation on one-channel integer data with platform-aware functions

```
IppStatus ippiMulC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst, IppSizeL
srcDstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C1IRSfs_L	16u_C1IRSfs_L	16s_C1IRSfs_L

Case 14: In-place operation on multi-channel integer data with platform-aware functions

```
IppStatus ippiMulC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pSrcDst, IppSizeL
srcDstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C3IRSfs_L	16u_C3IRSfs_L	16s_C3IRSfs_L
8u_AC4IRSfs_L	16u_AC4IRSfs_L	16s_AC4IRSfs_L

```
IppStatus ippiMulC_<mod>(const Ipp<datatype> value[4], Ipp<datatype>* pSrcDst, IppSizeL
srcDstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C4IRSfs_L	16u_C4IRSfs_L	16s_C4IRSfs_L

Case 15: In-place operation on one-channel floating point data with platform-aware functions

```
IppStatus ippiMulC_32f_C1IR_L(Ipp32f value, Ipp32f* pSrcDst, IppSizeL srcDstStep,
IppiSizeL roiSize);
```

Case 16: In-place operation on multi-channel integer data with platform-aware functions

```
IppStatus ippiMulC_<mod>(const Ipp32f value[3], Ipp32f* pSrcDst, IppSizeL srcDstStep,
IppiSizeL roiSize);
```

Supported values for `mod`:

32f_C3IR_L
32f_AC4IR_L

```
IppStatus ippiMulC_32f_C4IR_L(const Ipp32f value[4], Ipp32f* pSrcDst, IppSizeL
srcDstStep, IppiSizeL roiSize);
```

Case 17: Not-in-place operation on one-channel integer data with threading layer (TL)functions

```
IppStatus ippiMulC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>
value, Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C1RSfs_LT	16u_C1RSfs_LT	16s_C1RSfs_LT

Case 18: Not-in-place operation on multi-channel integer data with TL functions

```
IppStatus ippiMulC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, const
Ipp<datatype> value[3], Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int
scaleFactor);
```

Supported values for `mod`:

8u_C3RSfs_LT	16u_C3RSfs_LT	16s_C3RSfs_LT
8u_AC4RSfs_LT	16u_AC4RSfs_LT	16s_AC4RSfs_LT

```
IppStatus ippiMulC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, const
Ipp<datatype> value[4], Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int
scaleFactor);
```

Supported values for `mod`:

8u_C4RSfs_LT	16u_C4RSfs_LT	16s_C4RSfs_LT

Case 19: Not-in-place operation on one-channel floating point data with TL functions

```
IppStatus ippiMulC_32f_C1R_LT(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f value,
Ipp32f* pDst, IppSizeL dstStep, IppiSizeL roiSize, int scaleFactor);
```

Case 20: Not-in-place operation on multi-channel floating point data with TL functions

```
IppStatus ippiMulC_<mod>(const Ipp32f* pSrc, IppSizeL srcStep, const Ipp32f value[3],
Ipp32f* pDst, IppSizeL dstStep, IppiSizeL roiSize);
```

Supported values for `mod`:

32f_C3R_LT
32f_AC4R_LT

```
IppStatus ippiMulC_32f_C4R_LT(const Ipp32f* pSrc, IppSizeL srcStep, const Ipp32f
value[4], Ipp32f* pDst, IppSizeL dstStep, IppiSizeL roiSize);
```

Case 21: In-place operation on one-channel integer data with TL functions

```
IppStatus ippiMulC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst, IppSizeL
srcDstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C1IRSfs_LT	16u_C1IRSfs_LT	16s_C1IRSfs_LT

Case 22: In-place operation on multi-channel integer data with TL functions

```
IppStatus ippiMulC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pSrcDst, IppSizeL
srcDstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C3IRSfs_LT	16u_C3IRSfs_LT	16s_C3IRSfs_LT

8u_AC4IRSfs_LT	16u_AC4IRSfs_LT	16s_AC4IRSfs_LT

```
IppStatus ippiMulC_<mod>(const Ipp<datatype> value[4], Ipp<datatype>* pSrcDst, IppSizeL srcDstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C4IRSfs_LT	16u_C4IRSfs_LT	16s_C4IRSfs_LT

Case 23: In-place operation on one-channel floating point data with TL functions

```
IppStatus ippiMulC_32f_C1IR_LT(Ipp32f value, Ipp32f* pSrcDst, IppSizeL srcDstStep, IppiSizeL roiSize);
```

Case 24: In-place operation on multi-channel integer data with TL functions

```
IppStatus ippiMulC_<mod>(const Ipp32f value[3], Ipp32f* pSrcDst, IppSizeL srcDstStep, IppiSizeL roiSize);
```

Supported values for `mod`:

32f_C3IR_LT
32f_AC4IR_LT

```
IppStatus ippiMulC_32f_C4IR_LT(const Ipp32f value[4], Ipp32f* pSrcDst, IppSizeL srcDstStep, IppiSizeL roiSize);
```

Include Files

`ippi.h`

Flavors with the `_LT` suffix: `ippi_tl.h`

Flavors with the `_L` suffix: `ippi_l.h`

Domain Dependencies

Flavors declared in `ippi.h`:

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Flavors declared in `ippi_tl.h`:

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`, `ippcore_tl.lib`, `ippi_tl.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance, in bytes, between the starting points of consecutive lines in the source image.
`value`	The constant value to add to image pixel values (constant vector in case of multi-channel images).
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance, in bytes, between the starting points of consecutive lines in the destination image.

pSrcDst	Pointer to the source and destination image ROI for the in-place operation.
srcDstStep	Distance, in bytes, between the starting points of consecutive lines in the source and destination image for the in-place operation.
roiSize	Size of the source and destination ROI in pixels.
scaleFactor	Scale factor (see Integer Result Scaling).

Description

This function operates with ROI (see Regions of Regions of Interest in Intel IPP).

This function multiplies pixel values of an image by a constant `value`. For multi-channel images, pixel channel values are multiplied by the components of a constant vector `value`. For complex data, the function processes both real and imaginary parts of pixel values.

NOTE Step values must be positive for functions that operate on complex data.

In case of operations on integer data, the resulting values are scaled by `scaleFactor`.

Note that the functions with `AC4` descriptor do not process alpha channels.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if any of the specified pointers is `NULL`.
ippStsSizeErr	Indicates an error condition if `roiSize` has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if any of the specified buffer step values is zero or negative for functions that operate on complex data.

MulC64f

Multiplies pixel values of an image by a constant array.

Syntax

Not-in-place operations

```
IppStatus ippiMulC64f_8u_C1R(const Ipp8u* pSrc, int srcStep, const Ipp64f value[1],
Ipp8u* pDst, int dstStep, IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode
rndMode);
```

```
IppStatus ippiMulC64f_8u_C3R(const Ipp8u* pSrc, int srcStep, const Ipp64f value[3],
Ipp8u* pDst, int dstStep, IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode
rndMode);
```

```
IppStatus ippiMulC64f_8u_C4R(const Ipp8u* pSrc, int srcStep, const Ipp64f value[4],
Ipp8u* pDst, int dstStep, IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode
rndMode);

IppStatus ippiMulC64f_16u_C1R(const Ipp16u* pSrc, int srcStep, const Ipp64f value[1],
Ipp16u* pDst, int dstStep, IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode
rndMode);

IppStatus ippiMulC64f_16u_C3R(const Ipp16u* pSrc, int srcStep, const Ipp64f value[3],
Ipp16u* pDst, int dstStep, IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode
rndMode);

IppStatus ippiMulC64f_16u_C4R(const Ipp16u* pSrc, int srcStep, const Ipp64f value[4],
Ipp16u* pDst, int dstStep, IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode
rndMode);

IppStatus ippiMulC64f_16s_C1R(const Ipp16s* pSrc, int srcStep, const Ipp64f value[1],
Ipp16s* pDst, int dstStep, IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode
rndMode);

IppStatus ippiMulC64f_16s_C3R(const Ipp16s* pSrc, int srcStep, const Ipp64f value[3],
Ipp16s* pDst, int dstStep, IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode
rndMode);

IppStatus ippiMulC64f_16s_C4R(const Ipp16s* pSrc, int srcStep, const Ipp64f value[4],
Ipp16s* pDst, int dstStep, IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode
rndMode);

IppStatus ippiMulC64f_32f_C1R(const Ipp32f* pSrc, int srcStep, const Ipp64f value[1],
Ipp32f* pDst, int dstStep, IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode
rndMode);

IppStatus ippiMulC64f_32f_C3R(const Ipp32f* pSrc, int srcStep, const Ipp64f value[3],
Ipp32f* pDst, int dstStep, IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode
rndMode);

IppStatus ippiMulC64f_32f_C4R(const Ipp32f* pSrc, int srcStep, const Ipp64f value[4],
Ipp32f* pDst, int dstStep, IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode
rndMode);
```

In-place operations

```
IppStatus ippiMulC64f_8u_C1IR(Ipp8u* pSrcDst, int srcDstStep, const Ipp64f value[1],
IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode rndMode);

IppStatus ippiMulC64f_8u_C3IR(Ipp8u* pSrcDst, int srcDstStep, const Ipp64f value[3],
IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode rndMode);

IppStatus ippiMulC64f_8u_C4IR(Ipp8u* pSrcDst, int srcDstStep, const Ipp64f value[4],
IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode rndMode);

IppStatus ippiMulC64f_16u_C1IR(Ipp16u* pSrcDst, int srcDstStep, const Ipp64f value[1],
IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode rndMode);

IppStatus ippiMulC64f_16u_C3IR(Ipp16u* pSrcDst, int srcDstStep, const Ipp64f value[3],
IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode rndMode);

IppStatus ippiMulC64f_16u_C4IR(Ipp16u* pSrcDst, int srcDstStep, const Ipp64f value[4],
IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode rndMode);
```

```
IppStatus ippiMulC64f_16s_C1IR(Ipp16s* pSrcDst, int srcDstStep, const Ipp64f value[1],
IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode rndMode);

IppStatus ippiMulC64f_16s_C3IR(Ipp16s* pSrcDst, int srcDstStep, const Ipp64f value[3],
IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode rndMode);

IppStatus ippiMulC64f_16s_C4IR(Ipp16s* pSrcDst, int srcDstStep, const Ipp64f value[4],
IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode rndMode);

IppStatus ippiMulC64f_32f_C1IR(Ipp32f* pSrcDst, int srcDstStep, const Ipp64f value[1],
IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode rndMode);

IppStatus ippiMulC64f_32f_C3IR(Ipp32f* pSrcDst, int srcDstStep, const Ipp64f value[3],
IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode rndMode);

IppStatus ippiMulC64f_32f_C4IR(Ipp32f* pSrcDst, int srcDstStep, const Ipp64f value[4],
IppiSize roiSize, IppHintAlgorithm hint, IppRoundMode rndMode);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image.
`pSrcDst`	Pointer to the source and destination image ROI for in-place operations.
`srcStep`	Distance, in bytes, between the starting points of consecutive lines in the source image.
`srcDstStep`	Distance, in bytes, between the starting points of consecutive lines in the source and destination image for in-place operations.
`value`	Constant vector to add to image pixel values.
`pDst`	Pointer to the destination image.
`dstStep`	Distance, in bytes, between the starting points of consecutive lines in the destination image.
`roiSize`	Size of the image ROI, in pixels.
`hint`	Option to select the algorithmic implementation of the function, the following values are supported: • `ippAlgHintAccurate` • All output pixels are exact; accuracy takes precedence • `ippAlgHintFast`, `ippAlgHintNone` • Function performance takes precedence over accuracy and some

rndMode	Rounding mode, the following values are supported: • ippRndZero • Floating-point values are truncated to zero. • ippRndNear • Floating-point values are rounded to the nearest even • ippRndFinancial • Floating-point values are rounded down to the nearest

Description

This function multiplies pixel values of the source image by the specified constant array and places the scaled results to the same image.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when width or height of the image is less than, or equal to zero.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

// A simple example of gradient flood filling of connected area
// using Intel(R) Integrated Performance Primitives (Intel(R) IPP) functions:
//     ippiMulC64f_8u_C1R
//

#include <stdio.h>
#include "ipp.h"

#define WIDTH  8 /* source image width */
#define HEIGHT 8 /* source image height */

/* Next two defines are created to simplify code reading and understanding */
```

```
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define check_sts(st)        \
    if ((st) != ippStsNoErr) \
        goto exitLine; /* Go to Exit if Intel(R) IPP function returned status different from
ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

static Ipp8u pImage[WIDTH * HEIGHT] = /* Pointer to ROI of initial image */
    {0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, 3, 4, 5, 6, 7,
     0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, 3, 4, 5, 6, 7,
     0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, 3, 4, 5, 6, 7,
     0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, 3, 4, 5, 6, 7,
     0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, 3, 4, 5, 6, 7,
     0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, 3, 4, 5, 6, 7,
     0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, 3, 4, 5, 6, 7,
     0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, 3, 4, 5, 6, 7};

int main(void)
{
    IppStatus status = ippStsNoErr;
    int imageStep = WIDTH; /* Steps, in bytes, through the source and destination images */
    Ipp8u pDstImage[WIDTH * HEIGHT];
    IppiSize roiSize = {WIDTH, HEIGHT}; /* Size of source ROI in pixels */
    Ipp64f value[1] = {0.5};
    /* Calculate size of temporary buffer for flood filling operation */

    /* Perform gradient flood filling of 4-connected area */
    check_sts(status = ippiMulC64f_8u_C1R(pImage, imageStep, value, pDstImage, imageStep,
roiSize, ippAlgHintFast, ippRndNear));

    EXIT_MAIN
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

MulScale

Multiplies pixel values of two images and scales the products.

Syntax

Case 1: Not-in-place operation

```
IppStatus ippiMulScale_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize
roiSize);
```

Supported values for `mod`:

8u_C1R	16u_C1R
8u_C3R	16u_C3R
8u_C4R	16u_C4R
8u_AC4R	16u_AC4R

Case 2: In-place operation

```
IppStatus ippiMulScale_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for `mod`:

`8u_C1IR`	`16u_C1IR`
`8u_C3IR`	`16u_C3IR`
`8u_C4IR`	`16u_C4IR`
`8u_AC4IR`	`16u_AC4IR`

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`, `pSrc1`, `pSrc2`	Pointers to the source images ROI.
`srcStep`, `src1Step`, `src2Step`	Distances in bytes between starts of consecutive lines in the source images.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image.
`pSrcDst`	Pointer to the source and destination image ROI for the in-place operation.
`srcDstStep`	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
`roiSize`	Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Regions of Interest in Intel IPP).

This function multiplies corresponding pixel values of two input buffers and scales the products using the following formula:

`dst_pixel=src1_pixel*src2_pixel/max_val,`

where `src1_pixel` and `src2_pixel` are pixel values of the source buffers, `dst_pixel` is the resultant pixel value, and `max_val` is the maximum value of the pixel data range (see Table “Image Data Types and Ranges” for details). The function is implemented for 8-bit and 16-bit unsigned data types only.

Note that the functions with `AC4` descriptor do not process alpha channelss.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if any of the specified buffer step values is zero or negative.

MulCScale

Multiplies pixel values of an image by a constant and scales the products.

Syntax

Case 1: Not-in-place operation on one-channel data

```
IppStatus ippiMulCScale_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>
value, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C1R	16u_C1R

Case 2: Not-in-place operation on multi-channel data

```
IppStatus ippiMulCScale_<mod>(const Ipp<datatype>* pSrc, int srcStep, const
Ipp<datatype> value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C3R	16u_C3R
8u_AC4R	16u_AC4R

```
IppStatus ippiMulCScale_<mod>(const Ipp<datatype>* pSrc, int srcStep, const
Ipp<datatype> value[4], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C4R	16u_C4R

Case 3: In-place operation on one-channel data

```
IppStatus ippiMulCScale_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst, int
srcDstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C1IR	16u_C1IR

Case 4: In-place operation on multi-channel data

```
IppStatus ippiMulCScale_<mod>(const Ipp<datatype> value[3], const Ipp<datatype>*
pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C3IR	16u_C3IR
8u_AC4IR	16u_AC4IR

```
IppStatus ippiMulCScale_<mod>(const Ipp<datatype> value[4], Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for `mod`:

8u_C4IR	16u_C4IR

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`value`	The constant value to multiply each pixel value in a source image (constant vector in case of 3- or four-channel images).
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image.
`pSrcDst`	Pointer to the source and destination image ROI for the in-place operation.
`srcDstStep`	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
`roiSize`	Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function multiplies pixel values in the input buffer by a constant `value` and scales the products using the following formula:

`dst_pixel=src_pixel*value/max_val,`

where `src_pixel` is a pixel values of the source buffer, `dst_pixel` is the resultant pixel value, and `max_val` is the maximum value of the pixel data range (see Table "Image Data Types and Ranges" for details).

The function is implemented for 8-bit and 16-bit unsigned data types only. It can be used to multiply pixel values by a number between 0 and 1.

Note that the functions with AC4 descriptor do not process alpha channelss.

Return Values

ippStsNoErr	ndicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if the roiSize has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if any of the specified buffer step values is zero or negative.

Example

The code example below shows how to use the function ippiMulCScale_8u_C1R.

```
void func_mulcscale()
{
    IppiSize ROI = {8,4};
    IppiSize ROI2 = {5,4};
    Ipp8u src[8*4];
    Ipp8u dst[8*4];
    Ipp8u v = 100;


    ippiSet_8u_C1R(100,src,8,ROI);
    ippiSet_8u_C1R(0,dst,8,ROI);
    ippiMulCScale_8u_C1R(src,8,v,dst,8,ROI2);
}
```

Result:

```
            src1                                dst
100 100 100 100 100 100 100 100         39 39 39 39 39 0 0 0
100 100 100 100 100 100 100 100         39 39 39 39 39 0 0 0
100 100 100 100 100 100 100 100         39 39 39 39 39 0 0 0
100 100 100 100 100 100 100 100         39 39 39 39 39 0 0 0
```

Sub

Subtracts pixel values of two images.

Syntax

Case 1: Not-in-place operation on integer or complex data

```
IppStatus ippiSub_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>*
pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int
scaleFactor);
```

Supported values for mod

8u_C1RSfs	16u_C1RSfs	16s_C1RSfs

8u_C3RSfs	16u_C3RSfs	16s_C3RSfs
8u_AC4RSfs	16u_AC4RSfs	16s_AC4RSfs
8u_C4RSfs	16u_C4RSfs	16s_C4RSfs

Case 2: Not-in-place operation on floating-point or complex data

```
IppStatus ippiSub_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>*
pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

32f_C1R
32f_C3R
32f_AC4R
32f_C4R

Case 3: In-place operation on integer or complex data

```
IppStatus ippiSub_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pSrcDst,
int srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C1IRSfs	16u_C1IRSfs	16s_C1IRSfs
8u_C3IRSfs	16u_C3IRSfs	16s_C3IRSfs
8u_AC4IRSfs	16u_AC4IRSfs	16s_AC4IRSfs
8u_C4IRSfs	16u_C4IRSfs	16s_C4IRSfs

Case 4: In-place operation on floating-point or complex data

```
IppStatus ippiSub_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pSrcDst,
int srcDstStep, IppiSize roiSize);
```

Supported values for `mod`:

32f_C1IR
32f_C3IR
32f_AC4IR
32f_C4IR

Case 5: Not-in-place operation on integer data with platform-aware functions

```
IppStatus ippiSub_<mod>(const Ipp<datatype>* pSrc1, IppSizeL src1Step, const
Ipp<datatype>* pSrc2, IppSizeL src2Step, Ipp<datatype>* pDst, IppSizeL dstStep,
IppiSizeL roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C1RSfs_L	16u_C1RSfs_L	16s_C1RSfs_L
8u_C3RSfs_L	16u_C3RSfs_L	16s_C3RSfs_L

8u_AC4RSfs_L	16u_AC4RSfs_L	16s_AC4RSfs_L
8u_C4RSfs_L	16u_C4RSfs_L	16s_C4RSfs_L

Case 6: Not-in-place operation on floating-point data with platform-aware functions

```
IppStatus ippiSub_<mod>(const Ipp<datatype>* pSrc1, IppSizeL src1Step, const
Ipp<datatype>* pSrc2, IppSizeL src2Step, Ipp<datatype>* pDst, IppSizeL dstStep,
IppiSizeL roiSize);
```

Supported values for `mod`:

32f_C1R_L
32f_C3R_L
32f_C4R_L
32f_AC4R_L

Case 7: In-place operation on integer data with platform-aware functions

```
IppStatus ippiSub_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>*
pSrcDst, IppSizeL srcDstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C1IRSfs_L	16u_C1IRSfs_L	16s_C1IRSfs_L
8u_C3IRSfs_L	16u_C3IRSfs_L	16s_C3IRSfs_L
8u_AC4IRSfs_L	16u_AC4IRSfs_L	16s_AC4IRSfs_L
8u_C4IRSfs_L	16u_C4IRSfs_L	16s_C4IRSfs_L

Case 8: In-place operation on floating-point data with platform-aware functions

```
IppStatus ippiSub_<mod>(const Ipp<datatype>* pSrc1, IppSizeL src1Step, Ipp<datatype>*
pDst, IppSizeL dstStep, IppiSizeL roiSize);
```

Supported values for `mod`:

32f_C1IR_L
32f_C3IR_L
32f_AC4IR_L
32f_C4IR_L

Case 9: Not-in-place operation on integer data with threading layer (TL) functions

```
IppStatus ippiSub_<mod>(const Ipp<datatype>* pSrc1, IppSizeL src1Step, const
Ipp<datatype>* pSrc2, IppSizeL src2Step, Ipp<datatype>* pDst, IppSizeL dstStep,
IppiSizeL roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C1RSfs_LT	16u_C1RSfs_LT	16s_C1RSfs_LT
8u_C3RSfs_LT	16u_C3RSfs_LT	16s_C3RSfs_LT

8u_AC4RSfs_LT	16u_AC4RSfs_LT	16s_AC4RSfs_LT
8u_C4RSfs_LT	16u_C4RSfs_LT	16s_C4RSfs_LT

Case 10: Not-in-place operation on floating-point data with TL functions

```
IppStatus ippiSub_<mod>(const Ipp<datatype>* pSrc1, IppSizeL src1Step, const
Ipp<datatype>* pSrc2, IppSizeL src2Step, Ipp<datatype>* pDst, IppSizeL dstStep,
IppiSizeL roiSize);
```

Supported values for `mod`:

32f_C1R_LT
32f_C3R_LT
32f_C4R_LT
32f_AC4R_LT

Case 11: In-place operation on integer data with TL functions

```
IppStatus ippiSub_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>*
pSrcDst, IppSizeL srcDstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C1IRSfs_LT	16u_C1IRSfs_LT	16s_C1IRSfs_LT
8u_C3IRSfs_LT	16u_C3IRSfs_LT	16s_C3IRSfs_LT
8u_AC4IRSfs_LT	16u_AC4IRSfs_LT	16s_AC4IRSfs_LT
8u_C4IRSfs_LT	16u_C4IRSfs_LT	16s_C4IRSfs_LT

Case 12: In-place operation on floating-point data with TL functions

```
IppStatus ippiSub_<mod>(const Ipp<datatype>* pSrc1, IppSizeL src1Step, Ipp<datatype>*
pDst, IppSizeL dstStep, IppiSizeL roiSize);
```

Supported values for `mod`:

32f_C1IR_LT
32f_C3IR_LT
32f_AC4IR_LT
32f_C4IR_LT

Include Files

`ippi.h`

Flavors with the `_LT` suffix: `ippi_tl.h`

Flavors with the `_L` suffix: `ippi_l.h`

Domain Dependencies

Flavors declared in `ippi.h`:

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Flavors declared in `ippi_tl.h`:

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`, `ippcore_tl.lib`, `ippi_tl.lib`

Parameters

`pSrc`, `pSrc1`, `pSrc2`	Pointers to the source images ROI.
`srcStep`, `src1Step`, `src2Step`	Distances in bytes between starts of consecutive lines in the source images.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image.
`pSrcDst`	Pointer to the source and destination image ROI for the in-place operation.
`srcDstStep`	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
`roiSize`	Size of the source and destination ROI in pixels.
`scaleFactor`	Scale factor (see Integer Result Scaling).

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function subtracts pixel values of the source buffer `pSrc1` from the corresponding pixel values of the buffer `pSrc2` and places the result in the destination buffer `pDst`. For in-place operations, the values in `pSrc` are subtracted from the values in `pSrcDst` and the results are placed into `pSrcDst`. For complex data, the function processes both real and imaginary parts of pixel values.

NOTE Step values must be positive for functions that operate on complex data.

In case of operations on integer data, the resulting values are scaled by `scaleFactor`.

Note that the functions with `AC4` descriptor do not process alpha channels.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize` has a field with zero or negative value.

ippStsStepErr	Indicates an error condition if any of the specified buffer step values is zero or negative for functions that operate on complex data.

SubC

Subtracts a constant from pixel values of an image.

Syntax

Case 1: Not-in-place operation on one-channel integer or complex data

```
IppStatus ippiSubC_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype> value,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C1RSfs	16u_C1RSfs	16s_C1RSfs

Case 2: Not-in-place operation on multi-channel integer or complex data

```
IppStatus ippiSubC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C3RSfs	16u_C3RSfs	16s_C3RSfs

```
IppStatus ippiSubC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[4], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C4RSfs	16u_C4RSfs	16s_C4RSfs

```
IppStatus ippiSubC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_AC4RSfs	16u_AC4RSfs	16s_AC4RSfs

Case 3: Not-in-place operation on one-channel floating-point or complex data

```
IppStatus ippiSubC_32f_C1R(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype> value,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Case 4: Not-in-place operation on multi-channel floating-point or complex data

```
IppStatus ippiSubC_32f_C3R(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);

IppStatus ippiSubC_32f_C4R(const Ipp32f* pSrc, int srcStep, const Ipp32f value[4],
Ipp32f* pDst, int dstStep, IppiSize roiSize);

IppStatus ippiSubC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

32f_AC4R

Case 5: In-place operation on one-channel integer or complex data

```
IppStatus ippiSubC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst, int srcDstStep,
IppiSize roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C1IRSfs	16u_C1IRSfs	16s_C1IRSfs

Case 6: In-place operation on multi-channel integer or complex data

```
IppStatus ippiSubC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pSrcDst, int
srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C3IRSfs	16u_C3IRSfs	16s_C3IRSfs
8u_AC4IRSfs	16u_AC4IRSfs	16s_AC4IRSfs

```
IppStatus ippiSubC_<mod>(const Ipp<datatype> value[4], Ipp<datatype>* pSrcDst, int
srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C4IRSfs	16u_C4IRSfs	16s_C4IRSfs

Case 7: In-place operation on one-channel floating-point or complex data

```
IppStatus ippiSubC_32f_C1IR(Ipp<datatype> value, Ipp<datatype>* pSrcDst, int
srcDstStep, IppiSize roiSize);
```

Case 8: In-place operation on multi-channel floating-point or complex data:

```
IppStatus ippiSubC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pSrcDst, int
srcDstStep, IppiSize roiSize);
```

Supported values for `mod`:

32f_C3IR
32f_AC4IR

```
IppStatus ippiSubC_32f_C4IR(const Ipp32f value[4], Ipp32f* pSrcDst, int srcDstStep,
IppiSize roiSize);
```

Case 9: Not-in-place operation on one-channel integer data with platform-aware functions

```
IppStatus ippiSubC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>
value, Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C1RSfs_L	16u_C1RSfs_L	16s_C1RSfs_L

Case 10: Not-in-place operation on multi-channel integer data with platform-aware functions

```
IppStatus ippiSubC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, const
Ipp<datatype> value[3], Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int
scaleFactor);
```

Supported values for `mod`:

8u_C3RSfs_L	16u_C3RSfs_L	16s_C3RSfs_L
8u_AC4RSfs_L	16u_AC4RSfs_L	16s_AC4RSfs_L

```
IppStatus ippiSubC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, const
Ipp<datatype> value[4], Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int
scaleFactor);
```

Supported values for `mod`:

8u_C4RSfs_L	16u_C4RSfs_L	16s_C4RSfs_L

Case 11: Not-in-place operation on one-channel floating point data with platform-aware functions

```
IppStatus ippiSubC_32f_C1R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f value,
Ipp32f* pDst, IppSizeL dstStep, IppiSizeL roiSize);
```

Case 12: Not-in-place operation on multi-channel floating point data with platform-aware functions

```
IppStatus ippiSubC_<mod>(const Ipp32f* pSrc, IppSizeL srcStep, const Ipp32f value[3],
Ipp32f* pDst, IppSizeL dstStep, IppiSizeL roiSize);
```

Supported values for `mod`:

32f_C3R_L
32f_AC4R_L

```
IppStatus ippiSubC_32f_C4R_L(const Ipp32f* pSrc, IppSizeL srcStep, const Ipp32f
value[4], Ipp32f* pDst, IppSizeL dstStep, IppiSizeL roiSize);
```

Case 13: In-place operation on one-channel integer data with platform-aware functions

```
IppStatus ippiSubC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst, IppSizeL
srcDstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C1IRSfs_L	16u_C1IRSfs_L	16s_C1IRSfs_L

Case 14: In-place operation on multi-channel integer data with platform-aware functions

```
IppStatus ippiSubC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pSrcDst, IppSizeL
srcDstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C3IRSfs_L	16u_C3IRSfs_L	16s_C3IRSfs_L
8u_AC4IRSfs_L	16u_AC4IRSfs_L	16s_AC4IRSfs_L

```
IppStatus ippiSubC_<mod>(const Ipp<datatype> value[4], Ipp<datatype>* pSrcDst, IppSizeL
srcDstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C4IRSfs_L	16u_C4IRSfs_L	16s_C4IRSfs_L

Case 15: In-place operation on one-channel floating point data with platform-aware functions

```
IppStatus ippiSubC_32f_C1IR_L(Ipp32f value, Ipp32f* pSrcDst, IppSizeL srcDstStep,
IppiSizeL roiSize);
```

Case 16: In-place operation on multi-channel integer data with platform-aware functions

```
IppStatus ippiSubC_<mod>(const Ipp32f value[3], Ipp32f* pSrcDst, IppSizeL srcDstStep,
IppiSizeL roiSize);
```

Supported values for `mod`:

32f_C3IR_L
32f_AC4IR_L

```
IppStatus ippiSubC_32f_C4IR_L(const Ipp32f value[4], Ipp32f* pSrcDst, IppSizeL
srcDstStep, IppiSizeL roiSize);
```

Case 17: Not-in-place operation on one-channel integer data with threading layer (TL) functions

```
IppStatus ippiSubC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>
value, Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C1RSfs_LT	16u_C1RSfs_LT	16s_C1RSfs_LT

Case 18: Not-in-place operation on multi-channel integer data with TL functions

```
IppStatus ippiSubC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, const
Ipp<datatype> value[3], Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int
scaleFactor);
```

Supported values for `mod`:

8u_C3RSfs_LT	16u_C3RSfs_LT	16s_C3RSfs_LT
8u_AC4RSfs_LT	16u_AC4RSfs_LT	16s_AC4RSfs_LT

```
IppStatus ippiSubC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, const
Ipp<datatype> value[4], Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int
scaleFactor);
```

Supported values for `mod`:

8u_C4RSfs_LT	16u_C4RSfs_LT	16s_C4RSfs_LT

Case 19: Not-in-place operation on one-channel floating point data with TL functions

```
IppStatus ippiSubC_32f_C1R_LT(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f value,
Ipp32f* pDst, IppSizeL dstStep, IppiSizeL roiSize);
```

Case 20: Not-in-place operation on multi-channel floating point data with TL functions

```
IppStatus ippiSubC_<mod>(const Ipp32f* pSrc, IppSizeL srcStep, const Ipp32f value[3],
Ipp32f* pDst, IppSizeL dstStep, IppiSizeL roiSize);
```

Supported values for `mod`:

32f_C3R_LT
32f_AC4R_LT

```
IppStatus ippiSubC_32f_C4R_LT(const Ipp32f* pSrc, IppSizeL srcStep, const Ipp32f
value[4], Ipp32f* pDst, IppSizeL dstStep, IppiSizeL roiSize);
```

Case 21: In-place operation on one-channel integer data with TL functions

```
IppStatus ippiSubC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst, IppSizeL
srcDstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C1IRSfs_LT	16u_C1IRSfs_LT	16s_C1IRSfs_LT

Case 22: In-place operation on multi-channel integer data with TL functions

```
IppStatus ippiSubC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pSrcDst, IppSizeL
srcDstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C3IRSfs_LT	16u_C3IRSfs_LT	16s_C3IRSfs_LT
8u_AC4IRSfs_LT	16u_AC4IRSfs_LT	16s_AC4IRSfs_LT

```
IppStatus ippiSubC_<mod>(const Ipp<datatype> value[4], Ipp<datatype>* pSrcDst, IppSizeL
srcDstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C4IRSfs_LT	16u_C4IRSfs_LT	16s_C4IRSfs_LT

Case 23: In-place operation on one-channel floating point data with TL functions

```
IppStatus ippiSubC_32f_C1IR_LT(Ipp32f value, Ipp32f* pSrcDst, IppSizeL srcDstStep,
IppiSizeL roiSize);
```

Case 24: In-place operation on multi-channel integer data with TL functions

```
IppStatus ippiSubC_<mod>(const Ipp32f value[3], Ipp32f* pSrcDst, IppSizeL srcDstStep,
IppiSizeL roiSize);
```

Supported values for `mod`:

32f_C3IR_LT
32f_AC4IR_LT

```
IppStatus ippiSubC_32f_C4IR_LT(const Ipp32f value[4], Ipp32f* pSrcDst, IppSizeL
srcDstStep, IppiSizeL roiSize);
```

Include Files

`ippi.h`

Flavors with the `_LT` suffix: `ippi_tl.h`

Flavors with the `_L` suffix: `ippi_l.h`

Domain Dependencies

Flavors declared in `ippi.h`:

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Flavors declared in `ippi_tl.h`:

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`, `ippcore_tl.lib`, `ippi_tl.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`value`	The constant value to subtract from each pixel value in a source image (constant vector in case of multi-channel images).
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image.
`pSrcDst`	Pointer to the source and destination image ROI for the in-place operation.
`srcDstStep`	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
`roiSize`	Size of the source and destination ROI in pixels.
`scaleFactor`	Scale factor (see Integer Result Scaling).

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function changes image intensity by subtracting the constant `value` from pixel values of an image buffer. For multi-channel images, the components of a constant vector `value` are subtracted from pixel channel values. For complex data, the function processes both real and imaginary parts of pixel values.

NOTE Step values must be positive for functions that operate on complex data.

In case of operations on integer data, the resulting values are scaled by `scaleFactor`.

Note that the functions with the `AC4` descriptor do not process alpha channels.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize` has a field with zero or negative value.

ippStsStepErr	Indicates an error condition if any of the specified buffer step values is zero or negative for functions that operate on complex data.

Div

Divides pixel values of an image by pixel values of another image.

Syntax

Case 1: Not-in-place operation on integer or complex data

```
IppStatus ippiDiv_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>*
pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int
scaleFactor);
```

Supported values for `mod`:

8u_C1RSfs	16u_C1RSfs	16s_C1RSfs
8u_C3RSfs	16u_C3RSfs	16s_C3RSfs
8u_AC4RSfs	16u_AC4RSfs	16s_AC4RSfs
8u_C4RSfs	16u_C4RSfs	16s_C4RSfs

Case 2: Not-in-place operation on floating-point or complex data

```
IppStatus ippiDiv_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>*
pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

32f_C1R
32f_C3R
32f_AC4R
32f_C4R

Case 3: In-place operation on integer or complex data

```
IppStatus ippiDiv_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pSrcDst,
int srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C1IRSfs	16u_C1IRSfs	16s_C1IRSfs
8u_C3IRSfs	16u_C3IRSfs	16s_C3IRSfs
8u_AC4IRSfs	16u_AC4IRSfs	16s_AC4IRSfs
8u_C4IRSfs	16u_C4IRSfs	16s_C4IRSfs

Case 4: In-place operation on floating-point or complex data

```
IppStatus ippiDiv_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pSrcDst,
int srcDstStep, IppiSize roiSize);
```

Supported values for `mod`:

32f_C1IR
32f_C3IR
32f_AC4IR
32f_C4IR

Case 5: Not-in-place operation on integer data with platform-aware functions

```
IppStatus ippiDiv_<mod>(const Ipp<datatype>* pSrc1, IppSizeL src1Step, const
Ipp<datatype>* pSrc2, IppSizeL src2Step, Ipp<datatype>* pDst, IppSizeL dstStep,
IppiSizeL roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C1RSfs_L	16u_C1RSfs_L	16s_C1RSfs_L
8u_C3RSfs_L	16u_C3RSfs_L	16s_C3RSfs_L
8u_AC4RSfs_L	16u_AC4RSfs_L	16s_AC4RSfs_L
8u_C4RSfs_L	16u_C4RSfs_L	16s_C4RSfs_L

Case 6: Not-in-place operation on floating-point data with platform-aware functions

```
IppStatus ippiDiv_<mod>(const Ipp<datatype>* pSrc1, IppSizeL src1Step, const
Ipp<datatype>* pSrc2, IppSizeL src2Step, Ipp<datatype>* pDst, IppSizeL dstStep,
IppiSizeL roiSize);
```

Supported values for `mod`:

32f_C1R_L
32f_C3R_L
32f_AC4R_L
32f_C4R_L

Case 7: In-place operation on integer data with platform-aware functions

```
IppStatus ippiDiv_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>*
pSrcDst, IppSizeL srcDstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C1IRSfs_L	16u_C1IRSfs_L	16s_C1IRSfs_L
8u_C3IRSfs_L	16u_C3IRSfs_L	16s_C3IRSfs_L
8u_AC4IRSfs_L	16u_AC4IRSfs_L	16s_AC4IRSfs_L
8u_C4IRSfs_L	16u_C4IRSfs_L	16s_C4IRSfs_L

Case 8: In-place operation on floating-point data with platform-aware functions

```
IppStatus ippiDiv_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>*
pSrcDst, IppSizeL srcDstStep, IppiSizeL roiSize);
```

Supported values for `mod`:

32f_C1IR_L
32f_C3IR_L
32f_AC4IR_L
32f_C4IR_L

Case 9: Not-in-place operation on integer data with threading layer (TL) functions

```
IppStatus ippiDiv_<mod>(const Ipp<datatype>* pSrc1, IppSizeL src1Step, const
Ipp<datatype>* pSrc2, IppSizeL src2Step, Ipp<datatype>* pDst, IppSizeL dstStep,
IppiSizeL roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C1RSfs_LT	16u_C1RSfs_LT	16s_C1RSfs_LT
8u_C3RSfs_LT	16u_C3RSfs_LT	16s_C3RSfs_LT
8u_AC4RSfs_LT	16u_AC4RSfs_LT	16s_AC4RSfs_LT
8u_C4RSfs_LT	16u_C4RSfs_LT	16s_C4RSfs_LT

Case 10: Not-in-place operation on floating-point data with TL functions

```
IppStatus ippiDiv_<mod>(const Ipp<datatype>* pSrc1, IppSizeL src1Step, const
Ipp<datatype>* pSrc2, IppSizeL src2Step, Ipp<datatype>* pDst, IppSizeL dstStep,
IppiSizeL roiSize);
```

Supported values for `mod`:

32f_C1R_LT
32f_C3R_LT
32f_C4R_LT
32f_AC4R_LT

Case 11: In-place operation on integer data with TL functions

```
IppStatus ippiDiv_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>*
pSrcDst, IppSizeL srcDstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C1IRSfs_LT	16u_C1IRSfs_LT	16s_C1IRSfs_LT
8u_C3IRSfs_LT	16u_C3IRSfs_LT	16s_C3IRSfs_LT
8u_AC4IRSfs_LT	16u_AC4IRSfs_LT	16s_AC4IRSfs_LT
8u_C4IRSfs_LT	16u_C4IRSfs_LT	16s_C4IRSfs_LT

Case 12: In-place operation on floating-point data with TL functions

```
IppStatus ippiDiv_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>*
pSrcDst, IppSizeL srcDstStep, IppiSizeL roiSize);
```

Supported values for `mod`:

32f_C1IR_LT
32f_C3IR_LT
32f_AC4IR_LT
32f_C4IR_LT

Include Files

ippi.h

Flavors with the _LT suffix: ippi_tl.h

Flavors with the _L suffix: ippi_l.h

Domain Dependencies

Flavors declared in ippi.h:

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Flavors declared in ippi_tl.h:

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib, ippcore_tl.lib, ippi_tl.lib

Parameters

pSrc, pSrc1, pSrc2	Pointers to the source images ROI.
srcStep, src1Step, src2Step	Distances in bytes between starts of consecutive lines in the source images.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
pSrcDst	Pointer to the source and destination image ROI for the in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
roiSize	Size of the source and destination ROI in pixels.
scaleFactor	Scale factor (see Integer Result Scaling).

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function divides pixel values of the source buffer pSrc2 by the corresponding pixel values of the buffer pSrc1 and places the result in a destination buffer pDst. For in-place operations, the values in pSrcDst are divided by the values in pSrc and placed into pSrcDst. For complex data, the function processes both real and imaginary parts of pixel values. In case of operations on integer data, the resulting values are scaled by

scaleFactor and rounded (not truncated). When the function encounters a zero divisor value, the execution is not interrupted. The function returns the warning message and corresponding result value (see appendix " Handling of Special Cases " for more information).

Note that the functions with the AC4 descriptor do not process alpha channels.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if any of the specified buffer step values is zero or negative.
ippStsDivByZero	Indicates a warning that a divisor value is zero. The function execution is continued.

Example

The code example below illustrates how the function ippiDiv can beused.

```
IppStatus div32f( void ) {
   Ipp32f a[4*3], b[4*3];
   IppiSize roi = {2,2};
   int i;
   for( i=0; i<4*3; ++i ) a[i] = b[i] = (float)i;
   return ippiDiv_32f_C1IR( a, 4*sizeof(Ipp32f), b,
                            4*sizeof(Ipp32f), roi );
}
```

The destination image b contains

```
-1.#IND  +1.000  +2.000  +3.000
+1.000   +1.000  +6.000  +7.000
+8.000   +9.000  +10.00  +11.00
```

Console output:

```
-- warning in div32f:
(6) Zero value(s) of divisor in the function Div
```

Div_Round

Divides pixel values of an image by pixel values of another image with different rounding modes.

Syntax

Case 1: Not-in-place operation on integer data

```
IppStatus ippiDiv_Round_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
IppRoundMode rndMode, int scaleFactor);
```

Supported values for `mod`:

`8u_C1RSfs`	`16u_C1RSfs`	`16s_C1RSfs`
`8u_C3RSfs`	`16u_C3RSfs`	`16s_C3RSfs`
`8u_AC4RSfs`	`16u_AC4RSfs`	`16s_AC4RSfs`
`8u_C4RSfs`	`16u_C4RSfs`	`16s_C4RSfs`

Case 2: In-place operation on integer data

```
IppStatus ippiDiv_Round_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pSrcDst, int srcDstStep, IppiSize roiSize, IppRoundMode rndMode, int scaleFactor);
```

Supported values for `mod`:

`8u_C1IRSfs`	`16u_C1IRSfs`	`16s_C1IRSfs`
`8u_C3IRSfs`	`16u_C3IRSfs`	`16s_C3IRSfs`
`8u_AC4IRSfs`	`16u_AC4IRSfs`	`16s_AC4IRSfs`
`8u_C4IRSfs`	`16u_C4IRSfs`	`16s_C4IRSfs`

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`, `pSrc1`, `pSrc2`	Pointers to the source images ROI.
`srcStep`, `src1Step`, `src2Step`	Distances in bytes between starts of consecutive lines in the source images.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image.
`pSrcDst`	Pointer to the source and destination image ROI for the in-place operation.
`srcDstStep`	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
`roiSize`	Size of the source and destination ROI in pixels.
`roundMode`	Rounding mode, the following values are possible: • `ippRndZero`

	• specifies that floating-point values are truncated toward zero, • ippRndNear • specifies that floating-point values are rounded to the nearest even • ippRndFinancial • specifies that floating-point values are rounded down to the nearest
scaleFactor	Scale factor (see Integer Result Scaling).

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function divides pixel values of the source buffer pSrc2 by the corresponding pixel values of the buffer pSrc1 and places the result in a destination buffer pDst. For in-place operations, the values in pSrcDst are divided by the values in pSrc and placed into pSrcDst. The resulting values are scaled by scaleFactor and rounded using the rounding method specified by the parameter roundMode. When the function encounters a zero divisor value, the execution is not interrupted. The function returns the warning message and corresponding result value (see appendix " Handling of Special Cases " for more information).

Note that the functions with the AC4 descriptor do not process alpha channelss.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if any of the specified buffer step values is zero or negative.
ippStsDivByZero	Indicates a warning that a divisor value is zero. The function execution is continued.
ippStsStsRoundModeNotSupportedErr	Indicates an error condition if the roundMode has an illegal value.

DivC

Divides pixel values of an image by a constant.

Syntax

Case 1: Not-in-place operation on one-channel integer or complex data

```
IppStatus ippiDivC_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype> value,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for mod:

8u_C1RSfs	16u_C1RSfs	16s_C1RSfs

Case 2: Not-in-place operation on multi-channel integer or complex data

```
IppStatus ippiDivC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C3RSfs	16u_C3RSfs	16s_C3RSfs	16s_AC4RSfs

```
IppStatus ippiDivC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[4], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C4RSfs	16u_C4RSfs	16s_C4RSfs

```
IppStatus ippiDivC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_AC4RSfs	16u_AC4RSfs

Case 3: Not-in-place operation on one-channel floating-point or complex data

```
IppStatus ippiDivC_32f_C1R(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype> value,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Case 4: Not-in-place operation on multi-channel floating-point or complex data

```
IppStatus ippiDivC_32f_C3R(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);

IppStatus ippiDivC_32f_C4R(const Ipp32f* pSrc, int srcStep, const Ipp32f value[4],
Ipp32f* pDst, int dstStep, IppiSize roiSize);

IppStatus ippiDivC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

32f_AC4R

```
IppStatus ippiDivC_32f_C4R(const Ipp32f* pSrc, int srcStep, const Ipp32f value[4],
Ipp32f* pDst, int dstStep, IppiSize roiSize);
```

Case 5: In-place operation on one-channel integer or complex data

```
IppStatus ippiDivC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst, int srcDstStep,
IppiSize roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C1IRSfs	16u_C1IRSfs	16s_C1IRSfs

Case 6: In-place operation on multi-channel integer or complex data

```
IppStatus ippiDivC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pSrcDst, int
srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C3IRSfs	16u_C3IRSfs	16s_C3IRSfs
8u_AC4IRSfs	16u_AC4IRSfs	16s_AC4IRSfs

```
IppStatus ippiDivC_<mod>(const Ipp<datatype> value[4], Ipp<datatype>* pSrcDst, int
srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C4IRSfs	16u_C4IRSfs	16s_C4IRSfs

Case 7: In-place operation on one-channel floating-point or complex data

```
IppStatus ippiDivC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst, int srcDstStep,
IppiSize roiSize);
```

Supported values for `mod`:

32f_C1IR

Case 8: In-place operation on multi-channel floating-point or complex data

```
IppStatus ippiDivC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pSrcDst, int
srcDstStep, IppiSize roiSize);
```

Supported values for `mod`:

32f_C3IR
32f_AC4IR

```
IppStatus ippiDivC_32f_C4IR(const Ipp32f value[4], Ipp32f* pSrcDst, int srcDstStep,
IppiSize roiSize);
```

Case 9: Not-in-place operation on one-channel integer data with platform-aware functions

```
IppStatus ippiDivC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>
value, Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C1RSfs_L	16u_C1RSfs_L	16s_C1RSfs_L

Case 10: Not-in-place operation on multi-channel integer data with platform-aware functions

```
IppStatus ippiDivC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, const
Ipp<datatype> value[3], Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int
scaleFactor);
```

Supported values for `mod`:

8u_C3RSfs_L	16u_C3RSfs_L	16s_C3RSfs_L
8u_AC4RSfs_L	16u_AC4RSfs_L	16s_AC4RSfs_L

```
IppStatus ippiDivC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, const
Ipp<datatype> value[4], Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int
scaleFactor);
```

Supported values for `mod`:

8u_C4RSfs_L	16u_C4RSfs_L	16s_C4RSfs_L

Case 11: Not-in-place operation on one-channel floating point data with platform-aware functions

```
IppStatus ippiDivC_32f_C1R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f value,
Ipp32f* pDst, IppSizeL dstStep, IppiSizeL roiSize);
```

Case 12: Not-in-place operation on multi-channel floating point data with platform-aware functions

```
IppStatus ippiDivC_<mod>(const Ipp32f* pSrc, IppSizeL srcStep, const Ipp32f value[3],
Ipp32f* pDst, IppSizeL dstStep, IppiSizeL roiSize);
```

Supported values for `mod`:

`32f_C3R_L`
`32f_AC4R_L`

```
IppStatus ippiDivC_32f_C4R_L(const Ipp32f* pSrc, IppSizeL srcStep, const Ipp32f
value[4], Ipp32f* pDst, IppSizeL dstStep, IppiSizeL roiSize);
```

Case 13: In-place operation on one-channel integer data with platform-aware functions

```
IppStatus ippiDivC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst, IppSizeL
srcDstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for `mod`:

`8u_C1IRSfs_L`	`16u_C1IRSfs_L`	`16s_C1IRSfs_L`

Case 14: In-place operation on multi-channel integer data with platform-aware functions

```
IppStatus ippiDivC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pSrcDst, IppSizeL
srcDstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for `mod`:

`8u_C3IRSfs_L`	`16u_C3IRSfs_L`	`16s_C3IRSfs_L`
`8u_AC4IRSfs_L`	`16u_AC4IRSfs_L`	`16s_AC4IRSfs_L`

```
IppStatus ippiDivC_<mod>(const Ipp<datatype> value[4], Ipp<datatype>* pSrcDst, IppSizeL
srcDstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for `mod`:

`8u_C4IRSfs_L`	`16u_C4IRSfs_L`	`16s_C4IRSfs_L`

Case 15: In-place operation on one-channel floating point data with platform-aware functions

```
IppStatus ippiDivC_32f_C1IR_L(Ipp32f value, Ipp32f* pSrcDst, IppSizeL srcDstStep,
IppiSizeL roiSize);
```

Case 16: In-place operation on multi-channel integer data with platform-aware functions

```
IppStatus ippiDivC_<mod>(const Ipp32f value[3], Ipp32f* pSrcDst, IppSizeL srcDstStep,
IppiSizeL roiSize);
```

Supported values for `mod`:

`32f_C3IR_L`
`32f_AC4IR_L`

```
IppStatus ippiDivC_32f_C4IR_L(const Ipp32f value[4], Ipp32f* pSrcDst, IppSizeL
srcDstStep, IppiSizeL roiSize);
```

Case 17: Not-in-place operation on one-channel integer data with threading layer (TL) functions

```
IppStatus ippiDivC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, Ipp<datatype>
value, Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C1RSfs_LT	16u_C1RSfs_LT	16s_C1RSfs_LT

Case 18: Not-in-place operation on multi-channel integer data with TL functions

```
IppStatus ippiDivC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, const
Ipp<datatype> value[3], Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int
scaleFactor);
```

Supported values for `mod`:

8u_C3RSfs_LT	16u_C3RSfs_LT	16s_C3RSfs_LT
8u_AC4RSfs_LT	16u_AC4RSfs_LT	16s_AC4RSfs_LT

```
IppStatus ippiDivC_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep, const
Ipp<datatype> value[4], Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, int
scaleFactor);
```

Supported values for `mod`:

8u_C4RSfs_LT	16u_C4RSfs_LT	16s_C4RSfs_LT

Case 19: Not-in-place operation on one-channel floating point data with TL functions

```
IppStatus ippiDivC_32f_C1R_LT(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f value,
Ipp32f* pDst, IppSizeL dstStep, IppiSizeL roiSize);
```

Case 20: Not-in-place operation on multi-channel floating point data with TL functions

```
IppStatus ippiDivC_<mod>(const Ipp32f* pSrc, IppSizeL srcStep, const Ipp32f value[3],
Ipp32f* pDst, IppSizeL dstStep, IppiSizeL roiSize);
```

Supported values for `mod`:

32f_C3R_LT
32f_AC4R_LT

```
IppStatus ippiDivC_32f_C4R_LT(const Ipp32f* pSrc, IppSizeL srcStep, const Ipp32f
value[4], Ipp32f* pDst, IppSizeL dstStep, IppiSizeL roiSize);
```

Case 21: In-place operation on one-channel integer data with TL functions

```
IppStatus ippiDivC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst, IppSizeL
srcDstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C1IRSfs_LT	16u_C1IRSfs_LT	16s_C1IRSfs_LT

Case 22: In-place operation on multi-channel integer data with TL functions

```
IppStatus ippiDivC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pSrcDst, IppSizeL
srcDstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for `mod`:

`8u_C3IRSfs_LT`	`16u_C3IRSfs_LT`	`16s_C3IRSfs_LT`
`8u_AC4IRSfs_LT`	`16u_AC4IRSfs_LT`	`16s_AC4IRSfs_LT`

```
IppStatus ippiDivC_<mod>(const Ipp<datatype> value[4], Ipp<datatype>* pSrcDst, IppSizeL
srcDstStep, IppiSizeL roiSize, int scaleFactor);
```

Supported values for `mod`:

`8u_C4IRSfs_LT`	`16u_C4IRSfs_LT`	`16s_C4IRSfs_LT`

Case 23: In-place operation on one-channel floating point data with TL functions

```
IppStatus ippiDivC_32f_C1IR_LT(Ipp32f value, Ipp32f* pSrcDst, IppSizeL srcDstStep,
IppiSizeL roiSize);
```

Case 24: In-place operation on multi-channel integer data with TL functions

```
IppStatus ippiDivC_<mod>(const Ipp32f value[3], Ipp32f* pSrcDst, IppSizeL srcDstStep,
IppiSizeL roiSize);
```

Supported values for `mod`:

`32f_C3IR_LT`
`32f_AC4IR_LT`

```
IppStatus ippiDivC_32f_C4IR_LT(const Ipp32f value[4], Ipp32f* pSrcDst, IppSizeL
srcDstStep, IppiSizeL roiSize);
```

Include Files

`ippi.h`

Flavors with the `_LT` suffix: `ippi_tl.h`

Flavors with the `_L` suffix: `ippi_l.h`

Domain Dependencies

Flavors declared in `ippi.h`:

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Flavors declared in `ippi_tl.h`:

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`, `ippcore_tl.lib`, `ippi_tl.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.

value	The constant value to divide each pixel value in a source buffer (constant vector in case of 3- or four-channel images).
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
pSrcDst	Pointer to the source and destination image ROI for the in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
roiSize	Size of the source and destination ROI in pixels.
scaleFactor	Scale factor (see In Integer Result Scaling).

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function changes image intensity by dividing pixel values of an image buffer by the constant `value`. For multi-channel images, pixel channel values are divided by the components of a constant vector `value`. For complex data, the function processes both real and imaginary parts of pixel values. In case of operations on integer data, the resulting values are scaled by `scaleFactor` and rounded (not truncated).

When the divisor value is zero, the function execution is aborted and the `ippStsDivByZeroErr` error status is set. Note that in the alpha channel case (`AC4`), the alpha channels are not processed.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is `NULL`.
ippStsSizeErr	Indicates an error condition if `roiSize` has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if `srcStep`, `dstStep`, or `srcDstStep` has a zero or negative value.
ippStsDivByZeroErr	Indicates an error condition if the divisor value is zero.

Abs

Computes absolute pixel values of a source image and places them into the destination image.

Case 1: Not-in-place operation

```
IppStatus ippiAbs_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

16s_C1R	32f_C1R
16s_C3R	32f_C3R
16s_C4R	32f_C4R
	32f_AC4R

```
IppStatus ippiAbs_16s_AC4R(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize roiSize);
```

Case 2: In-place operation

```
IppStatus ippiAbs_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for `mod`:

16s_C1IR	32f_C1IR
16s_C3IR	32f_C3IR
16s_C4IR	32f_C4IR
16s_AC4IR	32f_AC4IR

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image.
`pSrcDst`	Pointer to the source and destination image ROI for the in-place operation.
`srcDstStep`	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
`roiSize`	Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function takes the absolute value of each channel in each pixel of the source image ROI and places the result into a destination image ROI. It operates on signed data only. Note that the functions with the `AC4` descriptor do not process alpha channels.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize` has a field with zero or negative value.

AbsDiff

Calculates absolute difference between two images.

Syntax

```
IppStatus ippiAbsDiff_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

`8u_C1R`	`16u_C1R`	`32f_C1R`
`8u_C3R`		

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc1`	Pointer to the first source image.
`src1Step`	Distance in bytes between starts of consecutive lines in the first source image.
`pSrc2`	Pointer to second source image.
`src2Step`	Distance in bytes between starts of consecutive lines in the second source image.
`pDst`	Pointer to the destination image.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image.

roiSize	Size of the image ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function calculates the absolute pixel-wise difference between two images by the formula:

pDst(x,y) = abs(pSrc1(x,y) -pSrc2(x,y)).

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value.
ippStsStepErr	Indicates an error when src1Step, src2Step or dstStep is less than roiSize.width * <pixelSize>.
ippStsNotEvenStepErr	Indicates an error condition if one of step values for floating-point images cannot be divided by 4.

AbsDiffC

Calculates absolute difference between image and scalar value.

Syntax

```
IppStatus ippiAbsDiffC_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize roiSize, int value);
```

Supported values for mod:

8u_C1R	16u_C1R

```
IppStatus ippiAbsDiffC_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int
dstStep, IppiSize roiSize, Ipp32f value);
```

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Pointer to the source image.

srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the image ROI in pixels.
value	Scalar value used to decrement each element of the source image.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function calculates the absolute pixel-wise difference between the source image `pSrc` and the scalar `value` by the formula:

`pDst(x,y) = abs(pSrc(x,y) -value.`

The function clips the `value` to the range [0, 255] for the `8u` data type, and to the range [0, 65535] for the `16u` data type.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is `NULL`.
ippStsSizeErr	Indicates an error condition if `roiSize` has a field with zero or negative value.
ippStsStepErr	Indicates an error when `srcStep` or `dstStep` is less than `roiSize.width * <pixelSize>`.
ippStsNotEvenStepErr	Indicates an error condition if one of step values for floating-point images cannot be divided by 4.

Sqr

Squares pixel values of an image and writes them into the destination image.

Syntax

Case 1: Not-in-place operation on integer data

```
IppStatus ippiSqr_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C1RSfs	16u_C1RSfs	16s_C1RSfs
8u_C3RSfs	16u_C3RSfs	16s_C3RSfs

8u_C4RSfs	16u_C4RSfs	16s_C4RSfs
8u_AC4RSfs	16u_AC4RSfs	16s_AC4RSfs

Case 2: Not-in-place operation on floating-point data

```
IppStatus ippiSqr_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep,
IppiSize roiSize);
```

Supported values for `mod`:

32f_C1R
32f_C3R
32f_C4R
32f_AC4R

Case 3: In-place operation on integer data

```
IppStatus ippiSqr_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, int
scaleFactor);
```

Supported values for `mod`:

8u_C1IRSfs	16u_C1IRSfs	16s_C1IRSfs
8u_C3IRSfs	16u_C3IRSfs	16s_C3IRSfs
8u_C4IRSfs	16u_C4IRSfs	16s_C4IRSfs
8u_AC4IRSfs	16u_AC4IRSfs	16s_AC4IRSfs

Case 4: In-place operation on floating-point data

```
IppStatus ippiSqr_<mod>(Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for `mod`:

32f_C1IR
32f_C3IR
32f_C4IR
32f_AC4IR

Case 5: Threading layer functions based on classic API

```
IppStatus ippiSqr_16s_C1IRSfs_T(Ipp16s* pSrcDst, int srcDstStep, IppiSize roiSize, int
scaleFactor);
```

```
IppStatus ippiSqr_16s32s_C1RSfs_T(const Ipp16s* pSrc, int srcStep, Ipp32s* pDst, int
dstStep, IppiSize roiSize, int scaleFactor);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance, in bytes, between the starting points of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance, in bytes, between the starting points of consecutive lines in the destination image.
`pSrcDst`	Pointer to the source and destination image ROI for the in-place operation.
`srcDstStep`	Distance, in bytes, between the starting points of consecutive lines in the source and destination image for the in-place operation.
`roiSize`	Size of the source and destination ROI in pixels.
`scaleFactor`	Scale factor (see Integer Result Scaling).

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function squares pixel values of the source image ROI and writes them to the destination image ROI. The function flavors operating on integer data apply fixed scaling defined by `scaleFactor` to the internally computed values, and saturate the results before writing them to the destination image ROI.

Note that the functions with the `AC4` descriptor do not process alpha channels.

Return Values

`ippStsNoErr`	Indicates an error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize` has a field with zero or negative value.

Sqrt

Computes square roots of pixel values of a source image and writes them into the destination image.

Syntax

Case 1: Not-in-place operation on integer data

```
IppStatus ippiSqrt_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C1RSfs	16u_C1RSfs	16s_C1RSfs
8u_C3RSfs	16u_C3RSfs	16s_C3RSfs
8u_AC4RSfs	16u_AC4RSfs	16s_AC4RSfs

Case 2: Not-in-place operation on floating-point data

```
IppStatus ippiSqrt_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep,
IppiSize roiSize);
```

Supported values for `mod`:

32f_C1R
32f_C3R
32f_AC4R

Case 3: In-place operation on integer data

```
IppStatus ippiSqrt_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, int
scaleFactor);
```

Supported values for `mod`:

8u_C1IRSfs	16u_C1IRSfs	16s_C1IRSfs
8u_C3IRSfs	16u_C3IRSfs	16s_C3IRSfs
8u_AC4IRSfs	16u_AC4IRSfs	16s_AC4IRSfs

Case 4: In-place operation on floating-point data

```
IppStatus ippiSqrt_<mod>(Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for `mod`:

32f_C1IR
32f_C3IR
32f_C4IR
32f_AC4IR

Case 5: Threading layer functions based on classic API

```
IppStatus ippiSqrt_16s_C1IRSfs_T(Ipp16s* pSrcDst, int srcDstStep, IppiSize roiSize, int
scaleFactor);
```

```
IppStatus ippiSqrt_16s32s_C1RSfs_T(const Ipp16s* pSrc, int srcStep, Ipp32s* pDst, int
dstStep, IppiSize roiSize, int scaleFactor);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance, in bytes, between the starting points of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance, in bytes, between the starting points of consecutive lines in the destination image.
pSrcDst	Pointer to the source and destination image ROI for the in-place operation.
srcDstStep	Distance, in bytes, between the starting points of consecutive lines in the source and destination image for the in-place operation.
roiSize	Size of the source and destination ROI in pixels.
scaleFactor	Scale factor (see Integer Result Scaling).

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function computes square roots of pixel values of the source image ROI and writes them into the destination image ROI. The function flavors operating on integer data apply fixed scaling defined by scaleFactor to the internally computed values, and saturate the results before writing them to the destination image ROI. If a source pixel value is negative, the function issues a warning and continues execution with the corresponding result value (see appendix " Handling of Special Cases " for more information).

Note that the functions with the AC4 descriptor do not process alpha channels.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value.
ippStsSqrtNegArg	Indicates a warning that a source pixel has a negative value.

Ln

Computes the natural logarithm of pixel values in a source image and writes the results into the destination image.

Syntax

Case 1: Not-in-place operation on integer data

```
IppStatus ippiLn_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C1RSfs	16u_C1RSfs	16s_C1RSfs
8u_C3RSfs	16u_C3RSfs	16s_C3RSfs

Case 2: Not-in-place operation on floating-point data

```
IppStatus ippiLn_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

32f_C1R
32f_C3R

Case 3: In-place operation on integer data

```
IppStatus ippiLn_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C1IRSfs	16u_C1IRSfs	16s_C1IRSfs
8u_C3IRSfs	16u_C3IRSfs	16s_C3IRSfs

Case 4: In-place operation on floating-point data

```
IppStatus ippiLn_<mod>(Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for `mod`:

32f_C1IR
32f_C3IR

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.

pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
pSrcDst	Pointer to the source and destination image ROI for the in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
roiSize	Size of the source and destination ROI in pixels.
scaleFactor	Scale factor (see Integer Result Scaling).

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function computes natural logarithms of pixel values of the source image ROI and writes the resultant values to the destination image ROI. The function flavors operating on integer data apply fixed scaling defined by scaleFactor to the internally computed values, and saturate the results before writing them to the destination image ROI.

If a source pixel value is zero or negative, the function issues a corresponding warning and continues execution with the corresponding result value (see appendix " Handling of Special Cases " for more information).

When several inputs have zero or negative value, the status code returned by the function corresponds to the first encountered case as illustrated in the code example below.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if srcStep, dstStep, or srcDstStep has a zero or negative value.
ippStsLnZeroArg	Indicates a warning that a source pixel has a zero value.
ippStsLnNegArg	Indicates a warning that a source pixel has a negative value.

Example

The code example below shows how to use Ln function.

```
IppStatus ln( void ) {
   Ipp32f img[8*8];
   IppiSize roi = { 8, 8 };
```

```
   IppStatus st;
   ippiSet_32f_C1R( (float)IPP_E, img, 8*4, roi );
   img[0] = -0;
   img[1] = -1;
   st = ippiLn_32f_C1IR( img, 8*sizeof(Ipp32f), roi );
   printf( "%f %f %f\n", img[0], img[1], img[2] );
   return st;
}
```

Output values:

```
-1.#INF00 -1.#IND00 1.000000
```

Status value and message:

```
(7) Zero value(s) of argument in the Ln function
```

Exp

Computes the exponential of pixel values in a source image and writes the results into the destination image.

Syntax

Case 1: Not-in-place operation on integer data

```
IppStatus ippiExp_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize roiSize, int scaleFactor);
```

Supported values for `mod`:

8u_C1RSfs	16u_C1RSfs	16s_C1RSfs
8u_C3RSfs	16u_C3RSfs	16s_C3RSfs

Case 2: Not-in-place operation on floating-point data

```
IppStatus ippiExp_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep,
IppiSize roiSize);
```

Supported values for `mod`:

32f_C1R
32f_C3R

Case 3: In-place operation on integer data

```
IppStatus ippiExp_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, int
scaleFactor);
```

Supported values for `mod`:

8u_C1IRSfs	16u_C1IRSfs	16s_C1IRSfs
8u_C3IRSfs	16u_C3IRSfs	16s_C3IRSfs

Case 4: In-place operation on floating-point data

```
IppStatus ippiExp_<mod>(Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for `mod`:

32f_C1IR
32f_C3IR

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
pSrcDst	Pointer to the source and destination image ROI for the in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
roiSize	Size of the source and destination ROI in pixels.
scaleFactor	Scale factor (see Integer Result Scaling).

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function computes e to the power of pixel values of the source image ROI and writes the resultant values into the destination image ROI. The function flavors operating on integer data apply fixed scaling defined by scaleFactor to the internally computed values, and saturate the results before writing them to the destination image ROI.

When the overflow occurs, the resultant value is determined in accordance with the data type (see appendix " Handling of Special Cases " for more information).

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.

ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if srcStep, dstStep, or srcDstStep has a zero or negative value.

DotProd

Computes the dot product of pixel values of two source images.

Syntax

Case 1: Operation on one-channel integer data

```
IppStatus ippiDotProd_<mod>(const Ipp<srcDatatype>* pSrc1, int src1Step, const
Ipp<srcDatatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pDp);
```

Supported values for mod:

8u64f_C1R	16u64f_C1R	16s64f_C1R	32u64f_C1R	32s64f_C1R

Case 2: Operation on three-channel integer data

```
IppStatus ippiDotProd_<mod>(const Ipp<srcDatatype>* pSrc1, int src1Step, const
Ipp<srcDatatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f pDp[3]);
```

Supported values for mod:

8u64f_C3R	16u64f_C3R	16s64f_C3R	32u64f_C3R	32s64f_C3R

Case 3: Operation on four-channel integer data

```
IppStatus ippiDotProd_<mod>(const Ipp<srcDatatype>* pSrc1, int src1Step, const
Ipp<srcDatatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f pDp[4]);
```

Supported values for mod:

8u64f_C4R	16u64f_C4R	16s64f_C4R	32u64f_C4R	32s64f_C4R

```
IppStatus ippiDotProd_<mod>(const Ipp<srcDatatype>* pSrc1, int src1Step, const
Ipp<srcDatatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f pDp[3]);
```

Supported values for mod:

8u64f_AC4R	16u64f_AC4R	32u64f_AC4R
	16s64f_AC4R	32s64f_AC4R

Case 4: Operation on floating-point data

```
IppStatus ippiDotProd_32f64f_C1R(const Ipp32f* pSrc1, int src1Step, const Ipp32f*
pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pDp, IppHintAlgorithm hint);

IppStatus ippiDotProd_32f64f_C3R(const Ipp32f* pSrc1, int src1Step, const Ipp32f*
pSrc2, int src2Step, IppiSize roiSize, Ipp64f pDp[3], IppHintAlgorithm hint);

IppStatus ippiDotProd_32f64f_C4R(const Ipp32f* pSrc1, int src1Step, const Ipp32f*
pSrc2, int src2Step, IppiSize roiSize, Ipp64f pDp[4], IppHintAlgorithm hint);

IppStatus ippiDotProd_32f64f_AC4R(const Ipp32f* pSrc1, int src1Step, const Ipp32f*
pSrc2, int src2Step, IppiSize roiSize, Ipp64f pDp[3], IppHintAlgorithm hint);
```

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc1, pSrc2	Pointer to the ROI in the source images.
src1Step, src2Step	Distance, in bytes, between the starting points of consecutive lines in the source images.
roiSize	Size of the source image ROI.
pDp	Pointer to the dot product or to the array containing the computed dot products for multi-channel images.
hint	Option to select the algorithmic implementation of the function, see Table " Hint Arguments for Image Moment Functions ".

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function computes the dot product of pixel values of two source images pSrc1 and pSrc2 using algorithm indicated by the hint argument (see Table " Hint Arguments for Image Moment Functions ") and stores the result in pDp. In case of multi-channel images, the dot product is computed for each channel separately and stored in the array pDp.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if one of the step values has zero or negative value.

DotProdCol

Calculates the dot product of taps vector and columns of the specified set of rows.

Syntax

```
IppStatus ippiDotProdCol_32f_L2(const Ipp32f* const ppSrcRow[], const Ipp32f* pTaps,
int tapsLen, Ipp32f* pDst, int width);
```

```
IppStatus ippiDotProdCol_64f_L2(const Ipp64f* const ppSrcRow[], const Ipp64f* pTaps,
int tapsLen, Ipp64f* pDst, int width);
```

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

ppSrcRow	Pointer to the set of rows.
pTaps	Pointer to the taps vector.
tapsLen	Length of taps vector, is equal to the number of rows.
pDst	Pointer to the destination row.
width	Width of the source and destination rows.

Description

This function calculates the dot product of taps vector pTaps and columns of the specified set of the rows ppSrcRow. It is useful for external vertical filtering pipeline implementation.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if width is less than or equal to 0.

Complement

Converts pixel values from two’s complement to binary representation.

Syntax

Case 1: Not-in-place operation

```
IppStatus ippiComplement_<mod>(Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize roiSize);
```

Supported values for mod:

8s_C1R	16s_C1R	32s_C1R

8s_C3R	16s_C3R	32s_C3R
8s_C4R	16s_C4R	32s_C4R
8s_AC4R	16s_AC4R	32s_AC4R

Case 2: In-place operation

```
IppStatus ippiComplement_<mod>(Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize);
```

Supported values for `mod`:

8s_C1IR	16s_C1IR	32s_C3IR
8s_C3IR	16s_C3IR	32s_C3IR
8s_C4IR	16s_C4IR	32s_C4IR
8s_AC4IR	16s_AC4IR	32s_AC4IR

Include Files

`ippi.h`

Parameters

`pSrc`, `pDst`, `pSrcDst`	Pointers to the start of the array with the source image.
`srcStep`, `dstStep`, `srcDstStep`	Distances in bytes between starts of adjacent lines in the source image.
`roiSize`	Size of the image ROI in pixels.

Description

This function converts the value of each pixel from the two's complement representation to the binary representation. This function takes the data from `pSrc` and saves the result in `pDst`.

In-place functions convert the value of each pixel by rewriting the source data. These functions take the data from `pSrcDst` and saves the result in `pSrcDst`.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when any of the specified pointers is `null`.
`ippStsSizeErr`	Indicates an error when the value for the data size is incorrect.
`ippStsStepErr`	Indicates an error when step value is less than, or equal to zero.

Logical Operations

Functions described in this section perform bitwise operations on pixel values. The operations include logical AND, NOT, inclusive OR, exclusive OR, and bit shifts.

- And
- AndC
- Or
- OrC
- Xor
- XorC
- Not
- RShiftC
- LShiftC

And

Performs a bitwise AND operation between corresponding pixels of two images.

Syntax

Case 1: Not-in-place operation

```
IppStatus ippiAnd_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>*
pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

8u_C1R	16u_C1R	32s_C1R
8u_C3R	16u_C3R	32s_C3R
8u_C4R	16u_C4R	32s_C4R
8u_AC4R	16u_AC4R	32s_AC4R

Case 2: In-place operation

```
IppStatus ippiAnd_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pSrcDst,
int srcDstStep, IppiSize roiSize);
```

Supported values for `mod`:

8u_C1IR	16u_C1IR	32s_C1IR
8u_C3IR	16u_C3IR	32s_C3IR
8u_C4IR	16u_C4IR	32s_C4IR
8u_AC4IR	16u_AC4IR	32s_AC4IR

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

pSrc, pSrc1, pSrc2	Pointers to the source images ROI.
srcStep, src1Step, src2Step	Distances in bytes between starts of consecutive lines in the source images.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
pSrcDst	Pointer to the source and destination image ROI for the in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination imager for the in-place operation.
roiSize	Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function performs a bitwise AND operation between the values of corresponding pixels of two source image ROIs, and writes the result into a destination image ROI.

Note that the functions with the AC4 descriptor do not process alpha channelss.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if any of the specified buffer step values is zero or negative.

AndC

Performs a bitwise AND operation of each pixel with a constant.

Syntax

Case 1: Not-in-place operation on one-channel data

```
IppStatus ippiAndC_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype> value,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C1R	16u_C1R	32s_C1R

Case 2: Not-in-place operation on multi-channel data

```
IppStatus ippiAndC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

8u_C3R	16u_C3R	32s_C3R
8u_AC4R	16u_AC4R	32s_AC4R

```
IppStatus ippiAndC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[4], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

8u_C4R	16u_C4R	32s_C4R

Case 3: In-place operation on one-channel data

```
IppStatus ippiAndC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst, int srcDstStep,
IppiSize roiSize);
```

Supported values for `mod`:

8u_C1IR	16u_C1IR	32s_C1IR

Case 4: In-place operation on multi-channel data

```
IppStatus ippiAndC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pSrcDst, int
srcDstStep, IppiSize roiSize);
```

Supported values for `mod`:

8u_C3IR	16u_C3IR	32s_C3IR
8u_AC4IR	16u_AC4IR	32s_AC4IR

```
IppStatus ippiAndC_<mod>(const Ipp<datatype> value[4], Ipp<datatype>* pSrcDst, int
srcDstStep, IppiSize roiSize);
```

Supported values for `mod`:

8u_C4IR	16u_C4IR	32s_C4IR

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.

value	The constant value to perform the bitwise AND operation on each pixel of the source image ROI (constant vector in case of multi-channel images).
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
pSrcDst	Pointer to the source and destination image ROI for the in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image buffer for the in-place operation.
roiSize	Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function performs a bitwise AND operation between each pixel value of a source image ROI and constant `value`.

Note that the functions with the `AC4` descriptor do not process alpha channels.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is `NULL`.
ippStsSizeErr	Indicates an error condition if `roiSize` has a field with a zero or negative value.
ippStsStepErr	Indicates an error condition if `srcStep`, `dstStep`, or `srcDstStep` has a zero or negative value.

Or

Performs bitwise inclusive OR operation between pixels of two source buffers.

Syntax

Case 1: Not-in-place operation

```
IppStatus ippiOr_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>*
pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

8u_C1R	16u_C1R	32s_C1R
8u_C3R	16u_C3R	32s_C3R

8u_C4R	16u_C4R	32s_C4R
8u_AC4R	16u_AC4R	32s_AC4R

Case 2: In-place operation

```
IppStatus ippiOr_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pSrcDst,
int srcDstStep, IppiSize roiSize);
```

Supported values for `mod`:

8u_C1IR	16u_C1IR	32s_C1IR
8u_C3IR	16u_C3IR	32s_C3IR
8u_C4IR	16u_C4IR	32s_C4IR
8u_AC4IR	16u_AC4IR	32s_AC4IR

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`, `pSrc1`, `pSrc2`	Pointers to the source images ROI.
`srcStep`, `src1Step`, `src2Step`	Distances in bytes between starts of consecutive lines in the source images.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image.
`pSrcDst`	Pointer to the source and destination image ROI for the in-place operation.
`srcDstStep`	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
`roiSize`	Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function performs a bitwise inclusive OR operation between the values of corresponding pixels of two source image ROIs, and writes the result into a destination image ROI. Note that the functions with the `AC4` descriptor do not process alpha channels.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if any of the specified buffer step values is zero or negative.

Example

The code example below show how to use the function ippiOr_8u_C1R.

```
void func_or()
{
    IppiSize Src1ROI = {8,4};
    IppiSize Src2ROI = {8,4};
    IppiSize DstROI = {5,4};
    Ipp8u src1[8*4];
    Ipp8u src2[8*4];
    Ipp8u dst[8*4];


    ippiSet_8u_C1R(0,dest,8,Src1ROI);
    ippiSet_8u_C1R(5,src1,8,Src1ROI);
    ippiSet_8u_C1R(6,src2,8,Src2ROI);
    ippiOr_8u_C1R(src1,8,src2,8,dst,8,DstROI);


}
```

Result:

```
         src1                        src2                        dst
05 05 05 05 05 05 05 05    06 06 06 06 06 06 06 06    07 07 07 07 07 00 00 00
05 05 05 05 05 05 05 05    06 06 06 06 06 06 06 06    07 07 07 07 07 00 00 00
05 05 05 05 05 05 05 05    06 06 06 06 06 06 06 06    07 07 07 07 07 00 00 00
05 05 05 05 05 05 05 05    06 06 06 06 06 06 06 06    07 07 07 07 07 00 00 00
```

OrC

Performs a bitwise inclusive OR operation between each pixel of a buffer and a constant.

Syntax

Case 1: Not-in-place operation on one-channel data

```
IppStatus ippiOrC_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype> value,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C1R	16u_C1R	32s_C1R

Case 2: Not-in-place operation on multi-channel data

```
IppStatus ippiOrC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

8u_C3R	16u_C3R	32s_C3R
8u_AC4R	16u_AC4R	32s_AC4R

```
IppStatus ippiOrC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[4], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

8u_C4R	16u_C4R	32s_C4R

Case 3: In-place operation on one-channel data

```
IppStatus ippiOrC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst, int srcDstStep,
IppiSize roiSize);
```

Supported values for `mod`:

8u_C1IR	16u_C1IR	32s_C1IR

Case 4: In-place operation on multi-channel data

```
IppStatus ippiOrC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pSrcDst, int
srcDstStep, IppiSize roiSize);
```

Supported values for `mod`:

8u_C3IR	16u_C3IR	32s_C3IR
8u_AC4IR	16u_AC4IR	32s_AC4IR

```
IppStatus ippiOrC_<mod>(const Ipp<datatype> value[4], Ipp<datatype>* pSrcDst, int
srcDstStep, IppiSize roiSize);
```

Supported values for `mod`:

8u_C4IR	16u_C4IR	32s_C4IR

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.

value	The constant value to perform the bitwise OR operation on each pixel of the source image (constant vector in case of multi-channel images).
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
pSrcDst	Pointer to the source and destination image ROI for the in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
roiSize	Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function performs a bitwise inclusive OR operation between each pixel value of a source image ROI and constant `value`.

Note that the functions with the `AC4` descriptor do not process alpha channels.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is `NULL`.
ippStsSizeErr	Indicates an error condition if `roiSize` has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if `srcStep`, `dstStep`, or `srcDstStep` has a zero or negative value.

Xor

Performs bitwise exclusive OR operation between pixels of two source buffers.

Syntax

Case 1: Not-in-place operation

```
IppStatus ippiXor_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp<datatype>*
pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

8u_C1R	16u_C1R	32s_C1R
8u_C3R	16u_C3R	32s_C3R

8u_C4R	16u_C4R	32s_C4R
8u_AC4R	16u_AC4R	32s_AC4R

Case 2: In-place operation

```
IppStatus ippiXor_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pSrcDst,
int srcDstStep, IppiSize roiSize);
```

Supported values for `mod`:

8u_C1IR	16u_C1IR	32s_C1IR
8u_C3IR	16u_C3IR	32s_C3IR
8u_C4IR	16u_C4IR	32s_C4IR
8u_AC4IR	16u_AC4IR	32s_AC4IR

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`, `pSrc1`, `pSrc2`	Pointers to the source images ROI.
`srcStep`, `src1Step`, `src2Step`	Distances in bytes between starts of consecutive lines in the source images.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image.
`pSrcDst`	Pointer to the source and destination image ROI for the in-place operation.
`srcDstStep`	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
`roiSize`	Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function performs a bitwise exclusive OR operation between the values of corresponding pixels of two source image ROIs, and writes the result into a destination image ROI.

Note that the functions with the `AC4` descriptor do not process alpha channels.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if any of the specified buffer step values is zero or negative.

XorC

Performs a bitwise exclusive OR operation between each pixel of a buffer and a constant.

Syntax

Case 1: Not-in-place operation on one-channel data

```
IppStatus ippiXorC_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype> value,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C1R	16u_C1R	32s_C1R

Case 2: Not-in-place operation on multi-channel data

```
IppStatus ippiXorC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C3R	16u_C3R	32s_C3R
8u_AC4R	16u_AC4R	32s_AC4R

```
IppStatus ippiXorC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp<datatype>
value[4], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C4R	16u_C4R	32s_C4R

Case 3: In-place operation on one-channel data

```
IppStatus ippiXorC_<mod>(Ipp<datatype> value, Ipp<datatype>* pSrcDst, int srcDstStep,
IppiSize roiSize);
```

Supported values for mod:

8u_C1IR	16u_C1IR	32s_C1IR

Case 4: In-place operation on multi-channel data

```
IppStatus ippiXorC_<mod>(const Ipp<datatype> value[3], Ipp<datatype>* pSrcDst, int
srcDstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C3IR	16u_C3IR	32s_C3IR
8u_AC4IR	16u_AC4IR	32s_AC4IR

```
IppStatus ippiXorC_<mod>(const Ipp<datatype> value[4], Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for `mod`:

8u_C4IR	16u_C4IR	32s_C4IR

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`value`	The constant value to perform the bitwise XOR operation on each pixel of the source image ROI (constant vector in case of multi-channel images).
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image.
`pSrcDst`	Pointer to the source and destination image ROI for the in-place operation.
`srcDstStep`	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
`roiSize`	Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function performs a bitwise exclusive OR operation between each pixel value of a source image ROI and constant `value`.

Note that the functions with the `AC4` descriptor do not process alpha channels.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if srcStep, dstStep, or srcDstStep has a zero or negative value.

Not

Performs a bitwise NOT operation on each pixel of a source buffer.

Syntax

Case 1: Not-in-place operation

```
IppStatus ippiNot_<mod>(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep,
IppiSize roiSize);
```

Supported values for mod:

8u_C1R
8u_C3R
8u_C4R
8u_AC4R

Case 2: In-place operation

```
IppStatus ippiNot_<mod>(Ipp8u* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C1IR
8u_C3IR
8u_C4IR
8u_AC4IR

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
pSrcDst	Pointer to the source and destination image ROI for the in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
roiSize	Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function performs a bitwise NOT operation on each pixel value of a source image ROI.

Note that the functions with the AC4 descriptor do not process alpha channels.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if srcStep, dstStep, or srcDstStep has a zero or negative value.

RShiftC

Shifts bits in pixel values to the right.

Case 1: Not-in-place operation on one-channel data

```
IppStatus ippiRShiftC_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp32u value,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32s_C1R

Case 2: Not-in-place operation on multi-channel data

```
IppStatus ippiRShiftC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp32u
value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

8u_C3R	16u_C3R	16s_C3R	32s_C3R

```
IppStatus ippiRShiftC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp32u
value[4], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

8u_C4R	16u_C4R	16s_C4R	32s_C4R

```
IppStatus ippiRShiftC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp32u
value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

8u_AC4R	16u_AC4R	16s_AC4R	32s_AC4R

Case 3: In-place operation on one-channel data

```
IppStatus ippiRShiftC_<mod>(Ipp32u value, Ipp<datatype>* pSrcDst, int srcDstStep,
IppiSize roiSize);
```

Supported values for `mod`:

8u_C1IR	16u_C1IR	16s_C1IR	32s_C1IR

Case 4: In-place operation on multi-channel data

```
IppStatus ippiRShiftC_<mod>(const Ipp32u value[3], Ipp<datatype>* pSrcDst, int
srcDstStep, IppiSize roiSize);
```

Supported values for `mod`:

8u_C3IR	16u_C3IR	16s_C3IR	32s_C3IR
8u_AC4IR	16u_AC4IR	16s_AC4IR	32s_AC4IR

```
IppStatus ippiRShiftC_<mod>(const Ipp32u value[4], Ipp<datatype>* pSrcDst, int
srcDstStep, IppiSize roiSize);
```

Supported values for `mod`:

8u_C4IR	16u_C4IR	16s_C4IR	32s_C4IR

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`value`	The number of bits to shift (constant vector in case of multi-channel images).

pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
pSrcDst	Pointer to the source and destination image ROI for the in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
roiSize	Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function decreases the intensity of pixels in the source image ROI by shifting the bits in each pixel value by value bits to the right. The positions vacated after shifting the bits are filled with the sign bit. In case of multi-channel data, each color channel can have its own shift value. This operation is equivalent to dividing the pixel values by a constant power of 2.

Note that the functions with the AC4 descriptor do not process alpha channels.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with a zero or negative value.
ippStsStepErr	Indicates an error condition if srcStep, dstStep, or srcDstStep has a zero or negative value.

LShiftC

Shifts bits in pixel values to the left.

Syntax

Case 1: Not-in-place operation on one-channel data

```
IppStatus ippiLShiftC_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp32u value,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C1R	16u_C1R	32s_C1R

Case 2: Not-in-place operation on multi-channel data

```
IppStatus ippiLShiftC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp32u
value[3], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

8u_C3R	16u_C3R	32s_C3R	32s_AC4R
8u_AC4R	16u_AC4R		

```
IppStatus ippiLShiftC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp32u value[4], Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

8u_C4R	16u_C4R	32s_C4R

Case 3: In-place operation on one-channel data

```
IppStatus ippiLShiftC_<mod>(Ipp32u value, Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for `mod`:

8u_C1IR	16u_C1IR	32s_C1IR

Case 4: In-place operation on multi-channel data

```
IppStatus ippiLShiftC_<mod>(const Ipp32u value[3], Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for `mod`:

8u_C3IR	16u_C3IR	32s_C3IR
8u_AC4IR	16u_AC4IR	32s_AC4IR

```
IppStatus ippiLShiftC_<mod>(const Ipp32u value[4], Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for `mod`:

8u_C4IR	16u_C4IR	32s_C4IR

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`value`	The number of bits to shift (constant vector in case of multi-channel images).
`pDst`	Pointer to the destination image ROI.

dstStep	Distance in bytes between starts of consecutive lines in the destination image.
pSrcDst	Pointer to the source and destination image ROI for the in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
roiSize	Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function changes the intensity of pixels in the source image ROI by shifting the bits in each pixel value by value bits to the left. In case of multi-channel data, each color channel can have its own shift value. The positions vacated after shifting the bits are filled with zeros. Values obtained as a result of left shift operations are not saturated. To get saturated values, use multiplication functions instead.

Note that the functions with the AC4 descriptor do not process alpha channels.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if srcStep, dstStep, or srcDstStep has a zero or negative value.

Example

The code example below illustrates the use of left shift function.

```
IppStatus lshift( void ) {
   Ipp8u img[8*8] = { 1, 0x7F, 0xFE };
   IppiSize roi = { 8, 8 };
   IppStatus st = ippiLShiftC_8u_C1IR( 1, img, 8, roi );
   printf( "%02x %02x %02x\n", img[0], img[1], img[2] );
   return st;
}
```

Output values:

```
02 fe fc
```

Alpha Composition

The Intel IPP provides functions that composite two image buffers using either the opacity (alpha) channel in the images or provided alpha values.

These functions operate on image buffers with 8-bit or 16-bit data in RGB or RGBA format. In all compositing operations a resultant pixel in destination buffer `pDst` is created by overlaying a pixel from the foreground image buffer `pSrc1` over a pixel from the background image buffer `pSrc2`. The supported types of images combining by using alpha values are listed in the table below.

Type	Output Pixel Color Components	Output Pixel Alpha Value	Description in Imaging Terms
OVER	$\alpha_A * A + (1- \alpha_A) * \alpha_B * B$	$\alpha_A +(1-\alpha_A) * \alpha_B$	A occludes B
IN	$\alpha_A * A * \alpha_B$	$\alpha_A * \alpha_B$	A within B. A acts as a matte for B. A shows only where B is visible.
OUT	$\alpha_A * A * (1-\alpha_B)$	$\alpha_A * (1-\alpha_B)$	A outside B. NOT-`B` acts as a matte for A. A shows only where B is not visible.
ATOP	$\alpha_A * A * \alpha_B+(1- \alpha_A) * \alpha_B * B$	$\alpha_A * \alpha_B + (1- \alpha_A) * \alpha_B$	Combination of (A IN B) and (B OUT A). B is both back-ground and matte for A.
XOR	$\alpha_A * A * (1-\alpha_B) + (1- \alpha_A) * \alpha_B * B$	$\alpha_A * (1-\alpha_B) + (1- \alpha_A) * \alpha_B$	Combination of (A OUT B) and (B OUT A). A and B mutually exclude each other.
PLUS	$\alpha_A * A + \alpha_B * B$	$\alpha_A +\alpha_B$	Blend without precedence

In the formulas above, the input image buffers are denoted as `A` and `B` for simplicity. The Greek letter α with subscripts denotes the normalized (scaled) alpha value in the range 0 to 1. It is related to the integer alpha value *alpha* as:

`α= alpha / max_val`

where *max_val* is 255 for 8-bit or 65535 for 16-bit unsigned pixel data.

For the `ippiAlphaComp` function that operates on 4-channel RGBA buffers only, α_A and α_B are the normalized alpha values of the two input image buffers, respectively.

For the `ippiAlphaCompC` function, α_A and α_B are the normalized constant alpha values that are passed as parameters to the function.

Note that in formulas for computing the resultant color channel values, `A` and `B` stand for the pixel color components of the respective input image buffers.

To save a significant amount of computation for some of the alpha compositing operations, use functions AlphaPremul , AlphaPremulC for pre-multiplying color channel values by the alpha values. This reduces the number of multiplications required in the compositing operations, which is especially efficient for repeated compositing of an image.

The type of composition operation that is performed by the function `AlphaComp` and `AlphaCompC` is specified by the parameter `alphaType`, the table below lists its possible values.

Operation types	Parameter Value	Parameter Value (`Premul`)
OVER	*ippAlphaOver*	*ippAlphaOverPremul*
IN	*ippAlphaIn*	*ippAlphaInPremul*
OUT	*ippAlphaOut*	*ippAlphaOutPremul*

ATOP	*ippAlphaATop*	*ippAlphaATopPremul*
XOR	*ippAlphaXor*	*ippAlphaXorPremul*
PLUS	*ippAlphaPlus*	*ippAlphaPlusPremul*

- AlphaComp
- AlphaCompC
- AlphaPremul
- AlphaPremulC

AlphaComp

Combines two images using alpha (opacity) values of both images.

Syntax

Case 1: Not-in-place operation

```
IppStatus ippiAlphaComp_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
IppiAlphaType alphaType);
```

Supported values for `mod`:

8u_AC1R	16u_AC1R	16s_AC4R	32u_AC1R	32s_AC1R	32f_AC1R
8u_AC4R	16u_AC4R	16s_AC1R	32u_AC4R	32s_AC4R	32f_AC4R

```
IppStatus ippiAlphaComp_<mod>(const Ipp<datatype>* const pSrc1[4], int src1Step, const
Ipp<datatype>* const pSrc2[4], int src2Step, Ipp<datatype>* const pDst[4], int dstStep,
IppiSize roiSize, IppiAlphaType alphaType);
```

Supported values for `mod`:

8u_AP4R	16u_AP4R

Case 2: In-place operation

```
IppStatus ippiAlphaComp_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pSrcDst, int srcDstStep, IppiSize roiSize, IppiAlphaType alphaType);
```

Supported values for `mod`:

8u_AC4IR	16u_AC4IR	16s_AC4IR	32u_AC4IR	32s_AC4IR	32f_AC4IR

```
IppStatus ippiAlphaComp_<mod>(const Ipp<datatype>* const pSrc[4], int srcStep,
Ipp<datatype>* const pSrcDst[4], int srcDstStep, IppiSize roiSize, IppiAlphaType
alphaType);
```

Supported values for `mod`:

8u_AP4IR	16u_AP4IR

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`, `pSrc1`, `pSrc2`	Pointers to the source image ROI for pixel-order data. An array of pointers to ROI in the separate source color planes in case of planar data.
`srcStep`, `src1Step`, `src2Step`	Distances, in bytes, between the starting points of consecutive lines in the source images.
`pSrcDst`	Pointer to the source and destination buffer or an array of pointers to separate source and destination color planes for in-place operation.
`pDst`	Pointer to the destination image ROI for pixel-order data. An array of pointers to ROI in the separate destination color planes in case of planar data.
`srcDstStep`	Distance, in bytes, between the starting points of consecutive lines in the source and destination image for in-place operation.
`dstStep`	Distance, in bytes, between the starting points of consecutive lines in the destination image.
`roiSize`	Size of the source and destination ROI, in pixels.
`alphaType`	The composition type to perform. See Table "Possible Values of the Parameter alphaType" for the type value and description.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function performs an image compositing operation on RGBA images using alpha values of both images. The compositing is done by overlaying pixels (r_A, g_A, b_A, α_A) from the foreground image `pSrc1` with pixels (r_B, g_B, b_B, α_B) from the background image `pSrc2` to produce pixels (r_C, g_C, b_C, α_C) in the resultant image `pDst`. The alpha values are assumed to be normalized to the range [0..1].

The type of the compositing operation is indicated by the `alphaType` parameter. Use Table "Possible Values of the Parameter alphaType" to choose a valid `alphaType` value depending on the required composition type. For example, the resulting pixel color components for the OVER operation (see Table "Types of Image Composing Operations") are computed as follows:

$r_C = \alpha_A * r_A + (1 - \alpha_A) * \alpha_B * r_B$

$g_C = \alpha_A * g_A + (1 - \alpha_A) * \alpha_B * g_B$

$b_C = \alpha_A * b_A + (1 - \alpha_A) * \alpha_B * b_B$

The resulting (normalized) alpha value is computed as

$\alpha_C = \alpha_A + (1 - \alpha_A) * \alpha_B$

This function can be used for unsigned pixel data only.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value.

AlphaCompC

Combines two images using constant alpha values.

Syntax

Case 1: Not-in-place operation

```
IppStatus ippiAlphaCompC_<mod>(const Ipp<datatype>* pSrc1, int src1Step, Ipp<datatype>
alpha1, const Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype> alpha2, Ipp<datatype>*
pDst, int dstStep, IppiSize roiSize, IppiAlphaType alphaType);
```

Supported values for mod:

8u_C1R	8u_C3R	8u_C4R	8u_AC4R
16u_C1R	16u_C3R	16u_C4R	16u_AC4R
16s_C1R			
32u_C1R			
32s_C1R			
32f_C1R			

```
IppStatus ippiAlphaCompC_<mod>(const Ipp<datatype>* const pSrc1[4], int src1Step,
Ipp<datatype> alpha1, const Ipp<datatype>* const pSrc2[4], int src2Step, Ipp<datatype>
alpha2, Ipp<datatype>* const pDst[4], int dstStep, IppiSize roiSize, IppiAlphaType
alphaType);
```

Supported values for mod:

8u_AP4R	16u_AP4R

Case 2: In-place operation

```
IppStatus ippiAlphaCompC_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>
alpha1, Ipp<datatype>* pSrcDst, int srcDstStep, Ipp<datatype> alpha2, IppiSize roiSize,
IppiAlphaType alphaType);
```

Supported values for mod:

8u_C1IR	16u_C1IR	16s_C1IR	32s_C1IR	32u_C1IR	32f_C1IR
8u_C3IR	16u_C3IR				
8u_C4IR	16u_C4IR				

8u_AC4IR	16u_AC4IR

```
IppStatus ippiAlphaCompC_<mod>(const Ipp<datatype>* const pSrc[4], int srcStep,
Ipp<datatype> alpha1, Ipp<datatype>* const pSrcDst[4], int srcDstStep, Ipp<datatype>
alpha2, IppiSize roiSize, IppiAlphaType alphaType);
```

Supported values for `mod`:

8u_AP4IR	16u_AP4IR

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc1`, `pSrc2`	Pointers to the source image ROI for pixel-order data. An array of pointers to ROI in the separate source color planes in case of planar data.
`src1Step`, `src2Step`	Distances, in bytes, between the starting points of consecutive lines in the source images.
`pSrcDst`	Pointer to the source and destination buffer or an array of pointers to separate source and destination color planes for in-place operation.
`pDst`	Pointer to the destination image ROI for pixel-order data. An array of pointers to ROI in the separate destination color planes in case of planar data.
`srcDstStep`	Distance, in bytes, between the starting points of consecutive lines in the source and destination image for in-place operation.
`dstStep`	Distance, in bytes, between the starting points of consecutive lines in the destination image.
`roiSize`	Size of the source and destination ROI in pixels.
`alpha1`, `alpha2`	Constant alpha values to use for the compositing operation.
`alphaType`	The composition type to perform. See Table "Possible Values of the Parameter alphaType" for the type value and description.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function performs an image compositing operation on one-channel image buffers, three-channel RGB and four-channel RGBA image buffers and on planar images, using constant alpha values `alpha1` and `alpha2`. These values are passed to the function as parameters.

The compositing is done by overlaying pixels from the foreground image ROI `pSrc1` with pixels from the background image ROI `pSrc2` to produce pixels in the resultant image ROI `pDst`. The alpha values are normalized to the range [0..1].

The type of the compositing operation is indicated by the `alphaType` parameter. Use Table "Possible Values of the Parameter alphaType" to choose a valid `alphaType` value depending on the required composition type. For example, the resulting pixel color components for the OVER operation (see Table "Types of Image Composing Operations") are computed as follows:

$r_C = \alpha_1 * r_A + (1 - \alpha_1) * \alpha_2 * r_B$

$g_C = \alpha_1 * g_A + (1 - \alpha_1) * \alpha_2 * g_B$

$b_C = \alpha_1 * b_A + (1 - \alpha_1) * \alpha_2 * b_B$

where α_1, α_2 are the normalised alpha values `alpha1`, `alpha2`.

This function can be used for unsigned pixel data only.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize` has a field with zero or negative value.

Example

The code example below shows how to use alpha composition function.

```
IppStatus acomp(void) {
   Ipp8u imga[8*8], imgb[8*8], imgc[8*8];
   IppiSize roi = { 8, 8 };
   IppStatus st;

   ippiImageRamp_8u_C1R(imga, 8, roi, 0, 1, ippAxsHorizontal);
   ippiImageRamp_8u_C1R(imgb, 8, roi, 0, 2, ippAxsHorizontal);

   st = ippiAlphaCompC_8u_C1R(imga, 8, 255/3, imgb, 8, 255,
                              imgc, 8, roi, ippAlphaOver);

   printf("over: a=%d,A=255/3; b=%d,B=255; c=%d c=a*A+b*(1-A)*B\n",
          imga[6], imgb[6], imgc[6]);

   return st;
}
```

Output

```
over: a=6,A=255/3; b=12,B=255; c=10 // c=a*A+b*B*(1-A)
```

AlphaPremul

Pre-multiplies pixel values of an image by its alpha values.

Syntax

Case 1: Not-in-place operation

```
IppStatus ippiAlphaPremul_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

`8u_AC4R`	`16u_AC4R`

```
IppStatus ippiAlphaPremul_<mod>(const Ipp<datatype>* const pSrc[4], int srcStep,
Ipp<datatype>* const pDst[4], int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

`8u_AP4R`	`16u_AP4R`

Case 2: In-place operation

```
IppStatus ippiAlphaPremul_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize
roiSize);
```

Supported values for `mod`:

`8u_AC4IR`	`16u_AC4IR`

```
IppStatus ippiAlphaPremul_<mod>(Ipp<datatype>* const pSrcDst[4], int srcDstStep,
IppiSize roiSize);
```

Supported values for `mod`:

`8u_AP4IR`	`16u_AP4IR`

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI for pixel-order data. An array of pointers to ROI in the separate source color planes in case of planar data.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI for pixel-order data. An array of pointers to ROI in the separate destination color planes in case of planar data.

dstStep	Distance in bytes between starts of consecutive lines in the destination image.
pSrcDst	Pointer to the source and destination buffer or an array of pointers to separate source and destination color planes for the in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
roiSize	Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts a RGBA source image (pixel order or planar) to the pre-multiplied alpha form. If (r,g,b,a) are the red, green, blue, and alpha values of a pixel, then new pixel values are (r* α, g* α, b* α, a) after execution of this function. Here α is the pixel normalized alpha value in the range 0 to 1.

The function `ippiAlphaPremul` can be used for unsigned pixel data only.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is `NULL`.
ippStsSizeErr	Indicates an error condition if `roiSize` has a field with zero or negative value.

AlphaPremulC

Pre-multiplies pixel values of an image using constant alpha (opacity) values.

Syntax

Case 1: Not-in-place operation

```
IppStatus ippiAlphaPremulC_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>
alpha, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod` :

8u_C1R	16u_C1R
8u_C3R	16u_C3R
8u_C4R	16u_C4R
8u_AC4R	16u_AC4R

```
IppStatus ippiAlphaPremulC_<mod>(const Ipp<datatype>* const pSrc[4], int srcStep,
Ipp<datatype> alpha, Ipp<datatype>* const pDst[4], int dstStep, IppiSize roiSize);
```

Supported values for `mod` :

8u_AP4R	16u_AP4R

Case 2: In-place operation

```
IppStatus ippiAlphaPremulC_<mod>(Ipp<datatype> alpha, Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for `mod` :

8u_C1IR	16u_C1IR
8u_C3IR	16u_C3IR
8u_C4IR	16u_C4IR
8u_AC4IR	16u_AC4IR

```
IppStatus ippiAlphaPremulC_<mod>(Ipp<datatype> alpha, Ipp<datatype>* const pSrcDst[4], int srcDstStep, IppiSize roiSize);
```

Supported values for `mod` :

8u_AP4IR	16u_AP4IR

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI for pixel-order data. An array of pointers to ROI in the separate source color planes in case of planar data.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI for pixel-order data. An array of pointers to separate ROI in the destination color planes in case of planar data.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image.
`pSrcDst`	Pointer to the source and destination image ROI or an array of pointers to ROI in the separate source and destination color planes for the in-place operation.
`srcDstStep`	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.

roiSize	Size of the source and destination ROI in pixels.
alpha	Global alpha value used for pre-multiplying pixel values.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts either a one-, three-, four-channel image or planar RGBA image to the pre-multiplied alpha form, using the global alpha value alpha. For one-, three-, four-channel image buffers, pixel values in each channel are multiplied by α; for RGBA (pixel order and planar) images with (r,g,b,a) pixel values, new pixel values are (r* α, g* α, b* α, alpha) after execution of this function. Here α is the normalized alpha value in the range 0 to 1.

The function ippiAlphaPremulC can be used for unsigned pixel data only.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value.

Image Color Conversion

This chapter describes image processing functions that perform different types of image color conversion. The Intel IPP software supports the following image color conversions:

- Color models conversion
- Conversion from color to gray scale and vice versa
- Different types of format conversion:
  - from pixel-order to planar format and vice versa
  - changing number of channels or planes
  - changing sampling format
  - altering order of samples or planes
- Gamma correction
- Reduction from high bit resolution color to low bit resolution color
- Intensity transformation using lookup tables
- Color twist
- Color keying

All Intel IPP color conversion functions perform point operations on pixels of the source image. For a given destination pixel, the resultant channel values are computed using channel values of the corresponding source pixel only, and not involving any neighborhood pixels. Thus, the rectangular region of interest (ROI, see Regions of Interest in Intel IPP) used in function operations may extend to the size of the whole image.

This chapter starts with introductory material that discusses color space models essential for understanding of the Intel IPP color conversion functions.

For more information about color spaces and color conversion techniques, see Jack01, Rog85, and Foley90.

- Gamma Correction
- CIE Chromaticity Diagram and Color Gamut

- Color Models
- Image Downsampling
- RGB Image Formats
- Pixel and Planar Image Formats
- Color Model Conversion
- Color - Gray Scale Conversions
- Format Conversion
- Color Twist
- Color Keying
- Gamma Correction
- Intensity Transformation
- Image Bit Conversion

Gamma Correction

Gamma correction of images is used to optimize the usage of data type depth when encoding an image by taking advantage of the non-linear manner in which humans perceive light and color. This non-linearity must be compensated to achieve correct color reproduction. To do this, luminance of each of the linear red, green, and blue components is reduced to a non-linear form using an inverse transformation. This process is called *gamma correction*.

The Intel IPP functions use the following basic equations to convert an RGB image to a gamma-corrected R'G'B' image:

for $R, G, B < 0.018$

$R' = 4.5R$

$G' = 4.5G$

$B' = 4.5B$

for $R, G, B \geq 0.018$

$R' = 1.099R^{0.45} - 0``.099``$

$G' = 1.099G^{0.45} - 0``.099``$

$B' = 1.099B^{0.45} - 0``.099``$

Note that the channel intensity values are normalized to fit in the range [0..1]. The gamma value is equal to 1/0.45 = 2.22 in conformity with ITU Rec.709 specification (see ITU709).

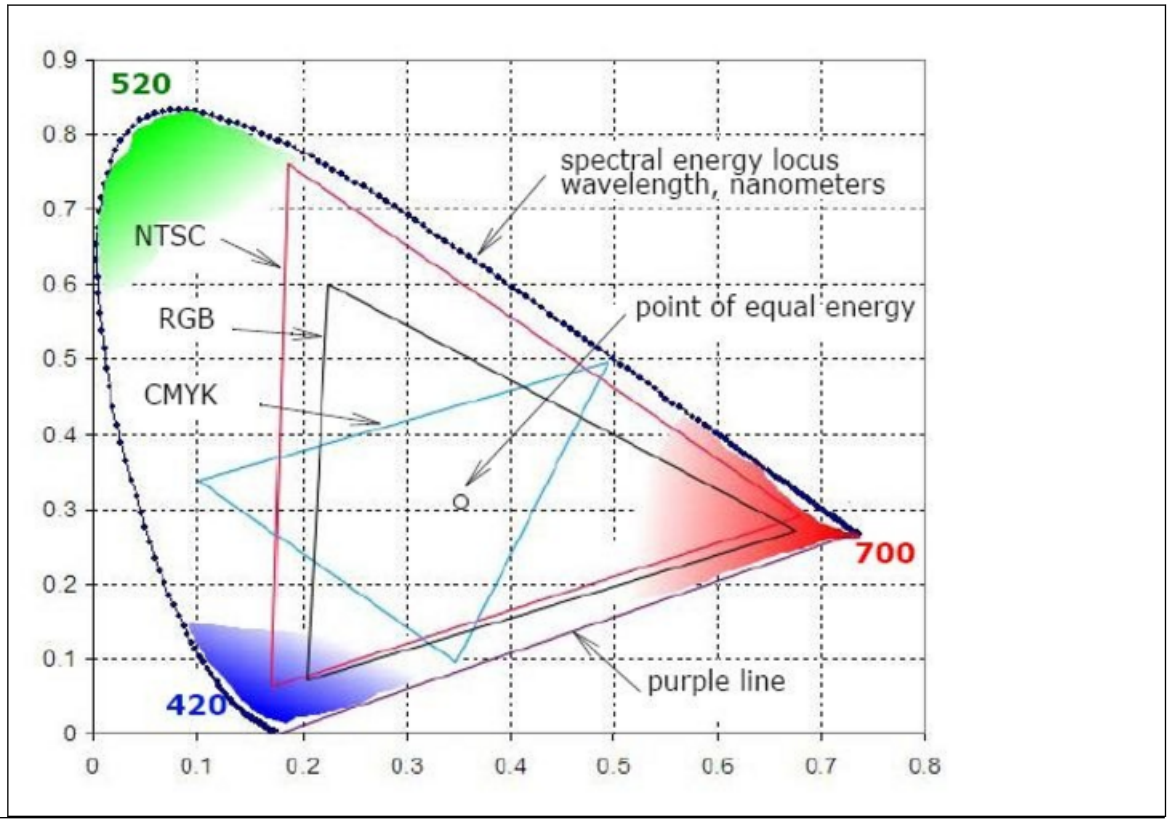
CIE Chromaticity Diagram and Color Gamut

Figure CIE xyY Chromaticity Diagram and Color Gamut presents a diagram of all visible colors. It was developed as a result of the experimental investigations performed by CIE (International Commission on Illumination, http://members.eunet.at/cie). The diagram presents visible colors as a function of x (red) and y (green) components called chromaticity coordinates. Positions of various spectrum colors (from violet to red) are indicated as the points of a tongue-shaped curve called spectrum locus. The straight line connecting the ends of the curve is called the purple line. The point of equal energy represents the CIE standard for white light. Any point within the diagram represents some mixture of spectrum colors. The pure or fully saturated colors lie on the spectrum locus. A straight-line segment joining any two points in the diagram defines all color variations that can be obtained by additively combining these two colors. A triangle with vertices at any three points determine the gamut of colors that can be obtained by combining corresponding three colors.

The structure of the human eye that distinguishes three different stimuli, establishes the three-dimensional nature of color. The color may be described with a set of three parameters called tristimulus values, or components. These values may, for example, be dominant wavelength, purity, and luminance, or so-called primary colors: red, green, and blue.

The chromaticity diagram exhibits that the gamut of any three fixed colors cannot enclose all visible colors. For example, Figure CIE `xyY` Chromaticity Diagram and Color Gamut shows schematically the gamut of reproducible colors for the RGB primaries of a typical color CRT monitor, CMYK color printing, and for the NTSC television.

CIE xyY Chromaticity Diagram and Color Gamut



Color Models

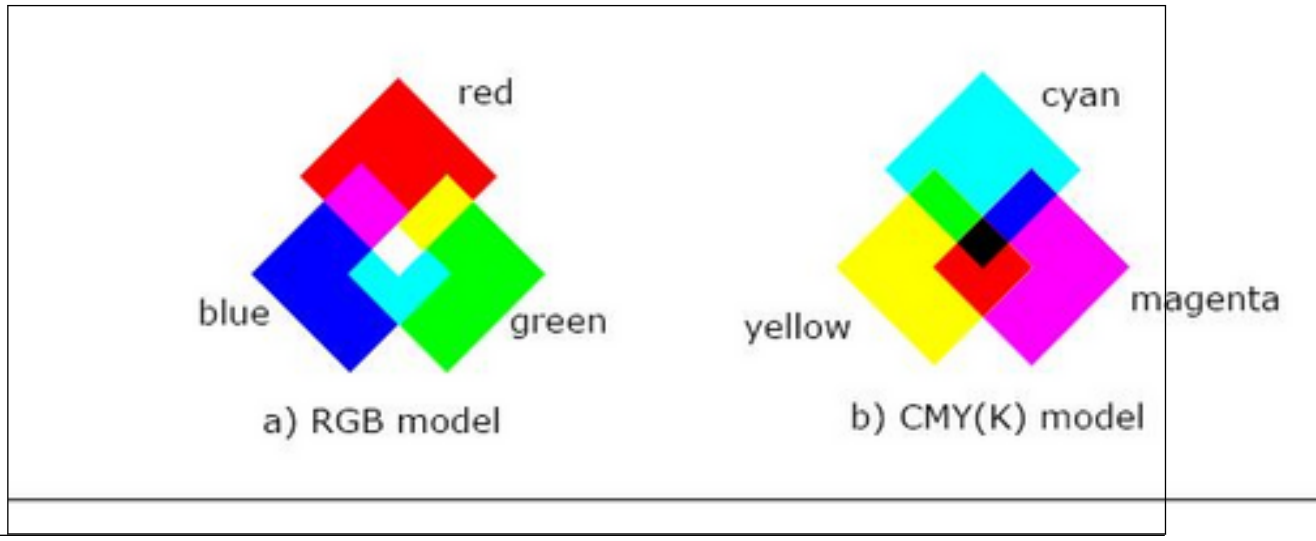
The purpose of a color model is to facilitate the specification of colors in some standard generally accepted way. In essence, a color model is a specification of a 3-D coordinate system and a subspace within that system where each color is represented by a single point.

Each industry that uses color employs the most suitable color model. For example, the RGB color model is used in computer graphics, YUV or YCbCr are used in video systems, PhotoYCC* is used in PhotoCD* production and so on. Transferring color information from one industry to another requires transformation from one set of values to another. Intel IPP provides a wide number of functions to convert different color spaces to RGB and vice versa.

RGB Color Model

In the RGB model, each color appears as a combination of red, green, and blue. This model is called additive, and the colors are called primary colors. The primary colors can be added to produce the secondary colors of light (see Primary and Secondary Colors for RGB and CMYK Models) - magenta (red plus blue), cyan (green plus blue), and yellow (red plus green). The combination of red, green, and blue at full intensities makes white.

Primary and Secondary Colors for RGB and CMYK Models



The color subspace of interest is a cube shown in *Figure RGB and CMY Color Models* (RGB values are normalized to 0..1), in which RGB values are at three corners; cyan, magenta, and yellow are the three other corners, black is at their origin; and white is at the corner farthest from the origin.

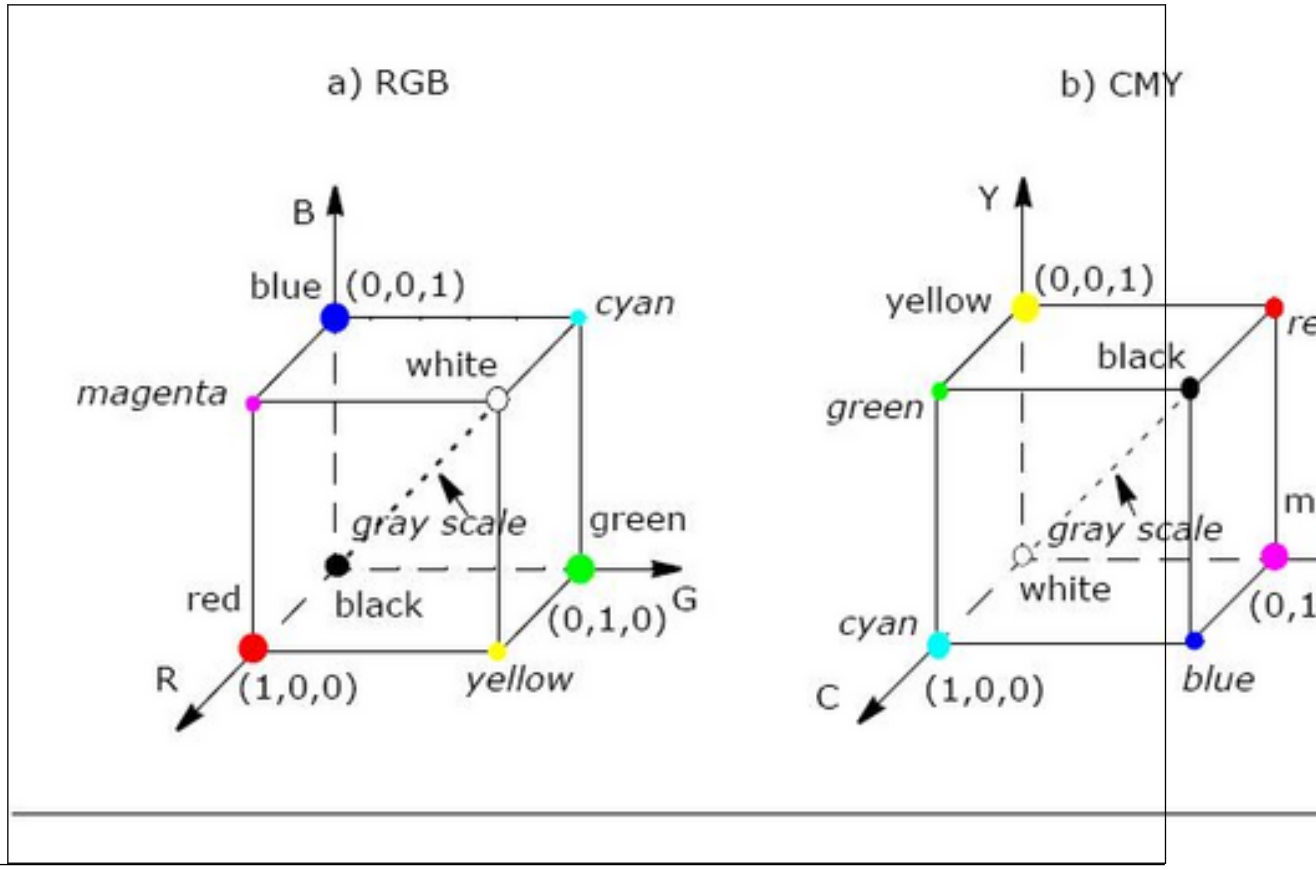
The gray scale extends from black to white along the diagonal joining these two points. The colors are the points on or inside the cube, defined by vectors extending from the origin.

Thus, images in the RGB color model consist of three independent image planes, one for each primary color.

As a rule, the Intel IPP color conversion functions operate with non-linear gamma-corrected images R'G'B'.

The importance of the RGB color model is that it relates very closely to the way that the human eye perceives color. RGB is a basic color model for computer graphics because color displays use red, green, and blue to create the desired color. Therefore, the choice of the RGB color space simplifies the architecture and design of the system. Besides, a system that is designed using the RGB color space can take advantage of a large number of existing software routines, because this color space has been around for a number of years.

RGB and CMY Color Models



However, RGB is not very efficient when dealing with real-world images. To generate any color within the RGB color cube, all three RGB components need to be of equal pixel depth and display resolution. Also, any modification of the image requires modification of all three planes.

CMYK Color Model

The CMYK color model is a subset of the RGB model and is primarily used in color print production. CMYK is an acronym for cyan, magenta, and yellow along with black (noted as K). The CMYK color space is subtractive, meaning that cyan, magenta yellow, and black pigments or inks are applied to a white surface to subtract some color from white surface to create the final color. For example (see Figure Primary and Secondary Colors for RGB and CMYK Models), cyan is white minus red, magenta is white minus green, and yellow is white minus blue. Subtracting all colors by combining the CMY at full saturation should, in theory, render black. However, impurities in the existing CMY inks make full and equal saturation impossible, and some RGB light does filter through, rendering a muddy brown color. Therefore, the black ink is added to CMY. The CMY cube is shown in Figure RGB and CMY Color Models, in which CMY values are at three corners; red, green, and blue are the three other corners, white is at the origin; and black is at the corner farthest from the origin.

YUV Color Model

The YUV color model is the basic color model used in analogue color TV broadcasting. Initially YUV is the re-coding of RGB for transmission efficiency (minimizing bandwidth) and for downward compatibility with black-and white television. The YUV color space is "derived" from the RGB space. It comprises the luminance (Y) and two color difference (U, V) components. The luminance can be computed as a weighted sum of red, green and blue components; the color difference, or chrominance, components are formed by subtracting luminance from blue and from red.

The principal advantage of the YUV model in image processing is decoupling of luminance and color information. The importance of this decoupling is that the luminance component of an image can be processed without affecting its color component. For example, the histogram equalization of the color image in the YUV format may be performed simply by applying histogram equalization to its Y component.

There are many combinations of YUV values from nominal ranges that result in invalid RGB values, because the possible RGB colors occupy only part of the YUV space limited by these ranges. Figure RGB Colors Cube in the YUV Color Space shows the valid color block in the YUV space that corresponds to the RGB color cube RGB values that are normalized to [0..1]).

The `Y'U'V'` notation means that the components are derived from gamma-corrected `R'G'B'`. Weighted sum of these non-linear components forms a signal representative of luminance that is called luma `Y'`. (Luma is often loosely referred to as luminance, so you need to be careful to determine whether a particular author assigns a linear or non-linear interpretation to the term luminance).

The Intel IPP functions use the following basic equation (Jack01) to convert between gamma-corrected `R'G'B'` and `Y'U'V'` models:

```
Y'= 0.299*R' + 0.587*G' + 0.114*B'

U'= -0.147*R' - 0.289*G' + 0.436*B' = 0.492*(B'- Y')

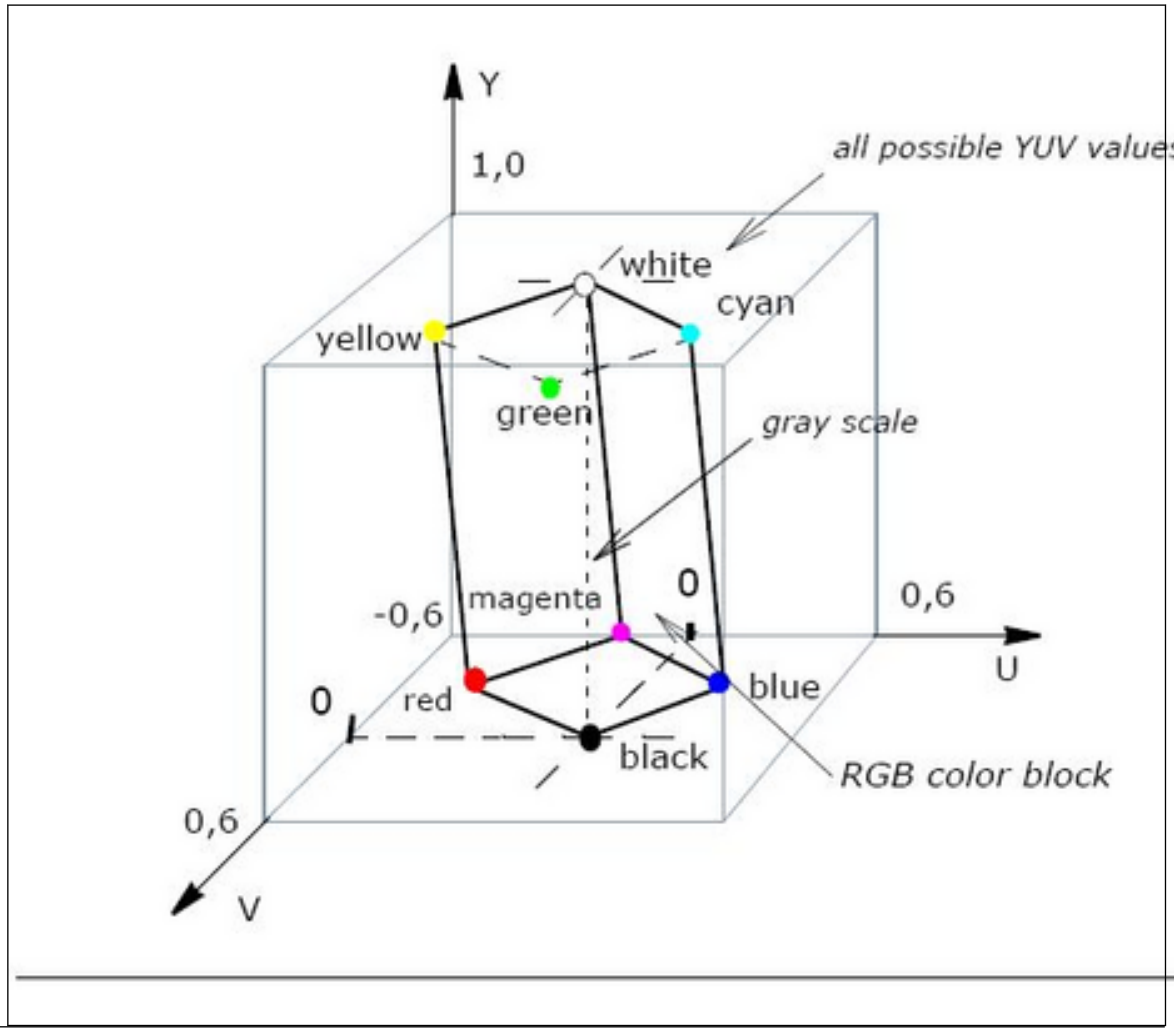
V'= 0.615*R' - 0.515*G' - 0.100*B' = 0.877*(R'- Y')

R' = Y' + 1.140*V'

G' = Y' - 0.394*U' - 0.581*V'
```

```
B' = Y' + 2.032*U'
```

RGB Colors Cube in the YUV Color Space



There are several YUV sampling formats such as 4:4:4, 4:2:2, and 4:2:0 that are supported by the Intel IPP color conversion functions and are described in Image Downsampling.

YCbCr and YCCK Color Models

The YCbCr color space is used for component digital video and was developed as part of the ITU-R BT.601 Recommendation. YCbCr is a scaled and offset version of the YUV color space.

The Intel IPP functions use the following basic equations Jack01 to convert between `R'G'B'` in the range 0-255 and `Y'Cb'Cr'` (this notation means that all components are derived from gamma-corrected `R'G'B'`):

```
Y' = 0.257*R' + 0.504*G' + 0.098*B' + 16

Cb' = -0.148*R' - 0.291*G' + 0.439*B' + 128

Cr' = 0.439*R' - 0.368*G' - 0.071*B' + 128

R' = 1.164*(Y'-16) + 1.596*(Cr'-128)

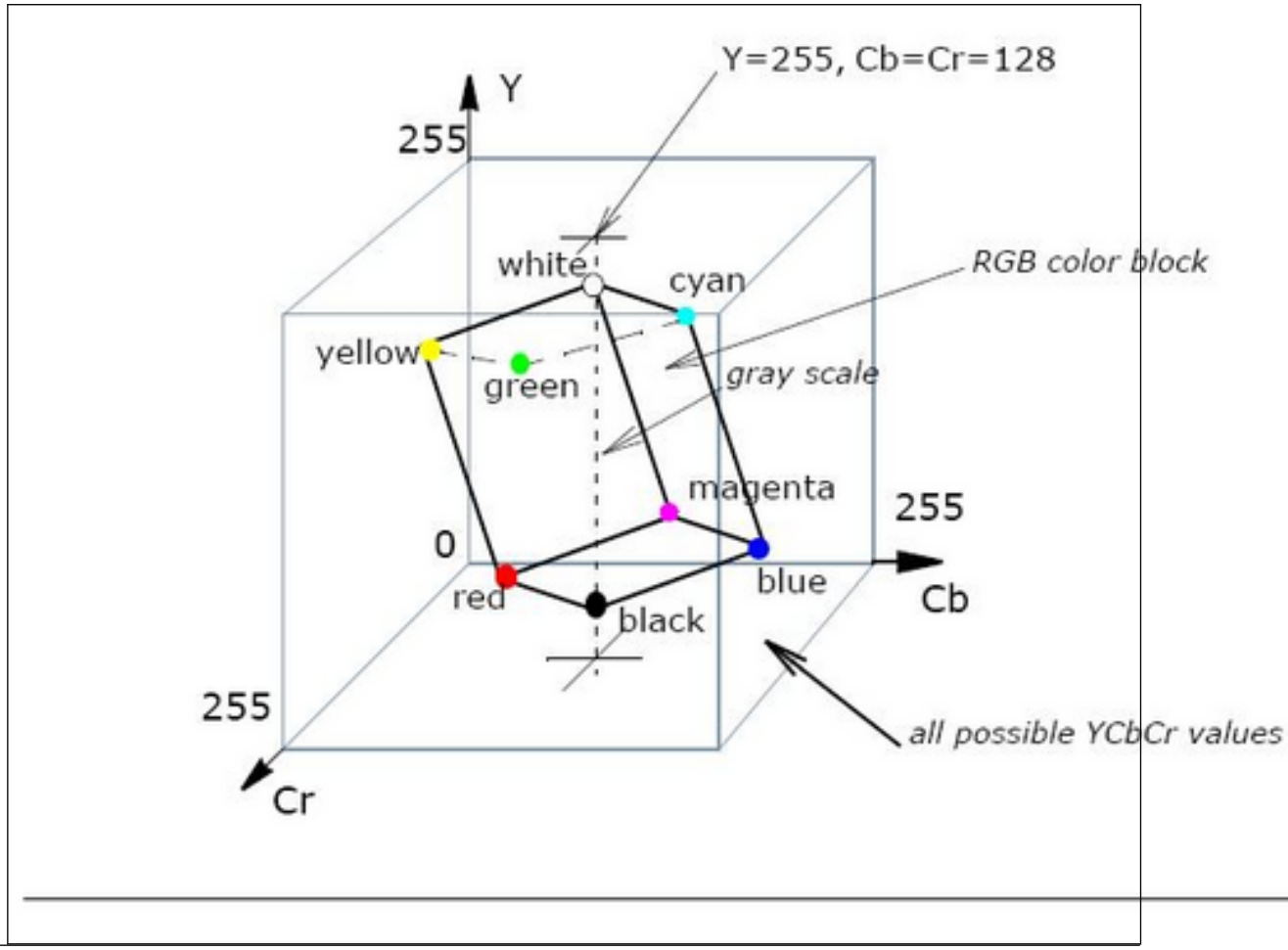
G' = 1.164*(Y'-16) - 0.813*(Cr'-128) - 0.392*(Cb'-128)

B' = 1.164*(Y'-16) + 2.017*(Cb'-128)
```

Possible RGB colors occupy only part of the YCbCr color space (see Figure RGB Colors Cube in the YCbCr Space) limited by the nominal ranges, therefore there are many YCbCr combinations that result in invalid RGB values.

There are several YCbCr sampling formats such as 4:4:4, 4:2:2, 4:1:1, and 4:2:0, which are supported by the Intel IPP color conversion functions and are described in Image Downsampling.

RGB Colors Cube in the YCbCr Space



PhotoYCC Color Model

The Kodak* PhotoYCC* was developed for encoding Photo CD* image data. It is based on both the ITU Recommendations 601 and 709, using luminance-chrominance representation of color like in BT.601 YCbCr and BT.709 (ITU709). This model comprises luminance (Y) and two color difference, or chrominance (C1, C2) components. The PhotoYCC is optimized for the color photographic material, and provides a color gamut that is greater than the one that can currently be displayed.

The Intel IPP functions use the following basic equations Jack01 to convert non-linear gamma-corrected `R'G'B'` to `Y'C'C'`:

```
Y' = 0.213*R' + 0.419*G' + 0.081*B'
```

```
C1' = -0.131*R' - 0.256*G' + 0.387*B'+ 0.612
```

```
C2' = 0.373*R' - 0.312*R' - 0.061*B' + 0.537
```

The equations above are given on the assumption that `R',G',` and `B'` values are normalized to the range [0..1].

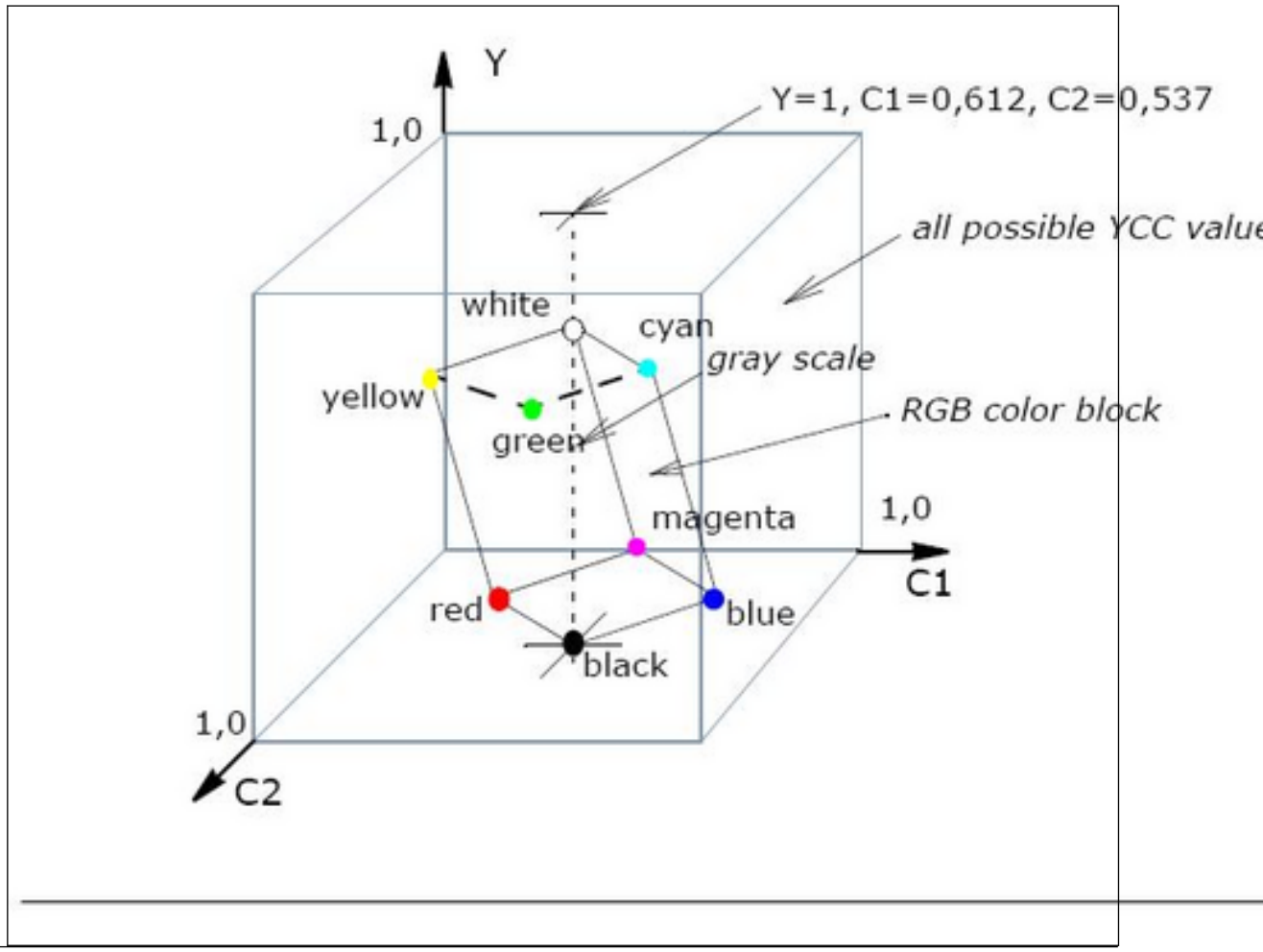
Since the PhotoYCC model attempts to preserve the dynamic range of film, decoding PhotoYCC images requires selection of a color space and range appropriate for the output device. Thus, the decoding equations are not always the exact inverse of the encoding equations. The following equations Jack01 are used in Intel IPP to generate `R'G'B'` values for driving a CRT display and require a unity relationship between the luma in the encoded image and the displayed image:

```
R' = 0.981 * Y + 1.315 * (C2 - 0.537)
G' = 0.981 * Y - 0.311 * (C1 - 0.612)- 0.669 * (C2 - 0.537)
B' = 0.981 * Y + 1.601 * (C1 - 0.612)
```

The equations above are given on the assumption that source `Y`, `C1` and `C2` values are normalized to the range [0..1], and the display primaries have the chromaticity values in accordance with ITU709 specifications.

The possible RGB colors occupy only part of the YCC color space (see RGB Colors in the YCC Color Space) limited by the nominal ranges, therefore there are many YCC combinations that result in invalid RGB values.

RGB Colors in the YCC Color Space



YCoCg Color Models

The YCoCg color model was developed to increase the effectiveness of the image compression Malvar03. This color model comprises the luminance (`Y`) and two color difference components (`Co - offset` orange, `Cg - offset` green).

The Intel IPP functions use the following simple basic equations Malvar03 to convert between RGB and YCoCg:

```
Y = R/4 + G/2 + B/4

Co = R/2 - B/2

Cg = -R/4 + G/2 - B/4

R = Y + Co - Cg

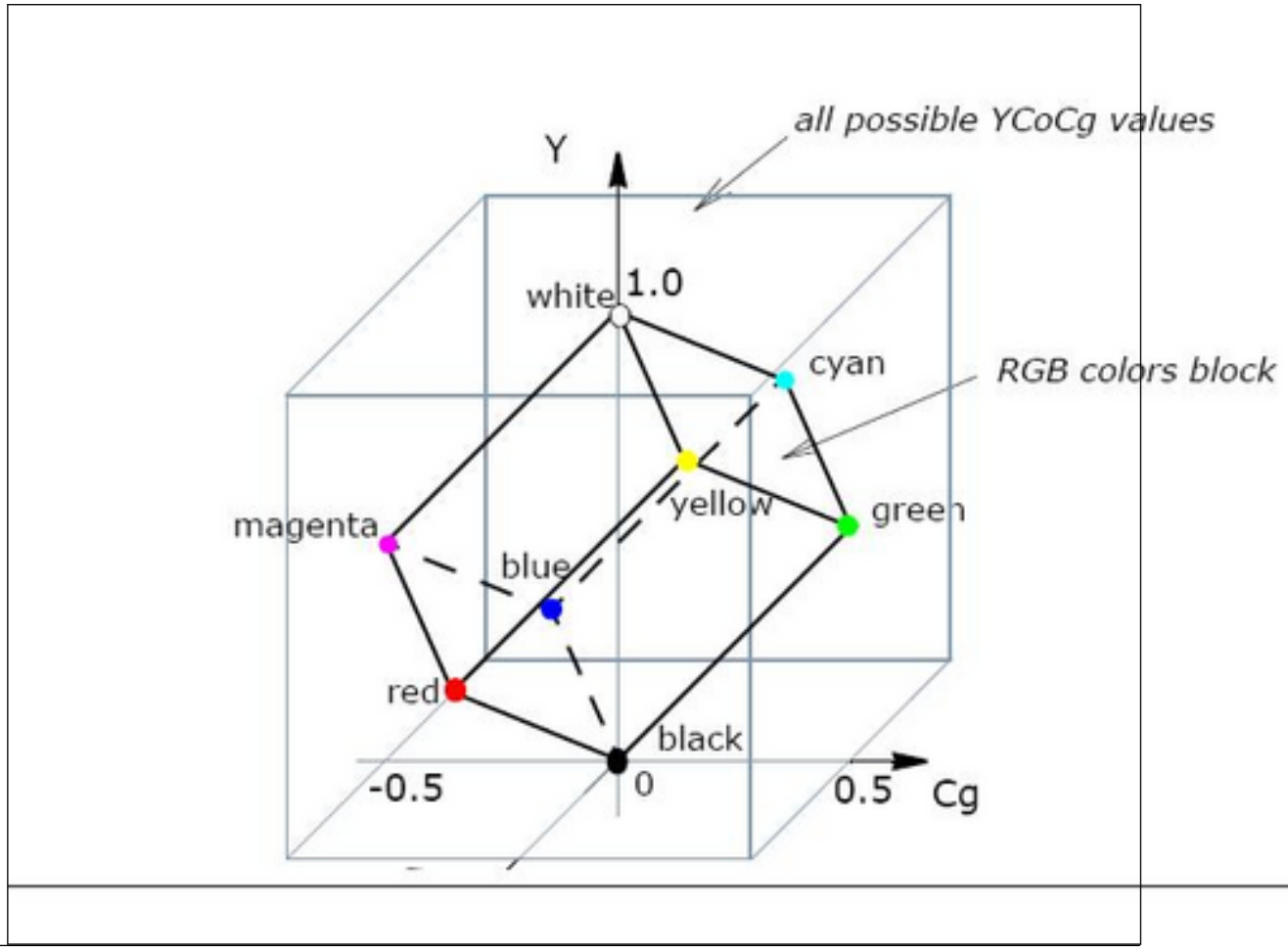
G = Y + Cg

B = Y - Co - Cg
```

A variation of this color space which is called YCoCg-R, enables transformation reversibility with a smaller dynamic range requirements than does YCoCg Malvar03-1.

The possible RGB colors occupy only part of the YCoCg color space (see RGB Color Cube in the YCoCg Color Space) limited by the nominal ranges, therfore there are many YCoCg combinations that result in invalid RGB values.

RGB Color Cube in the YCoCg Color Space



HSV, and HLS Color Models

The HLS (hue, lightness, saturation) and HSV (hue, saturation, value) color models were developed to be more “intuitive” in manipulating with color and were designed to approximate the way humans perceive and interpret color.

Hue defines the color itself. The values for the hue axis vary from 0 to 360 beginning and ending with red and running through green, blue and all intermediary colors.

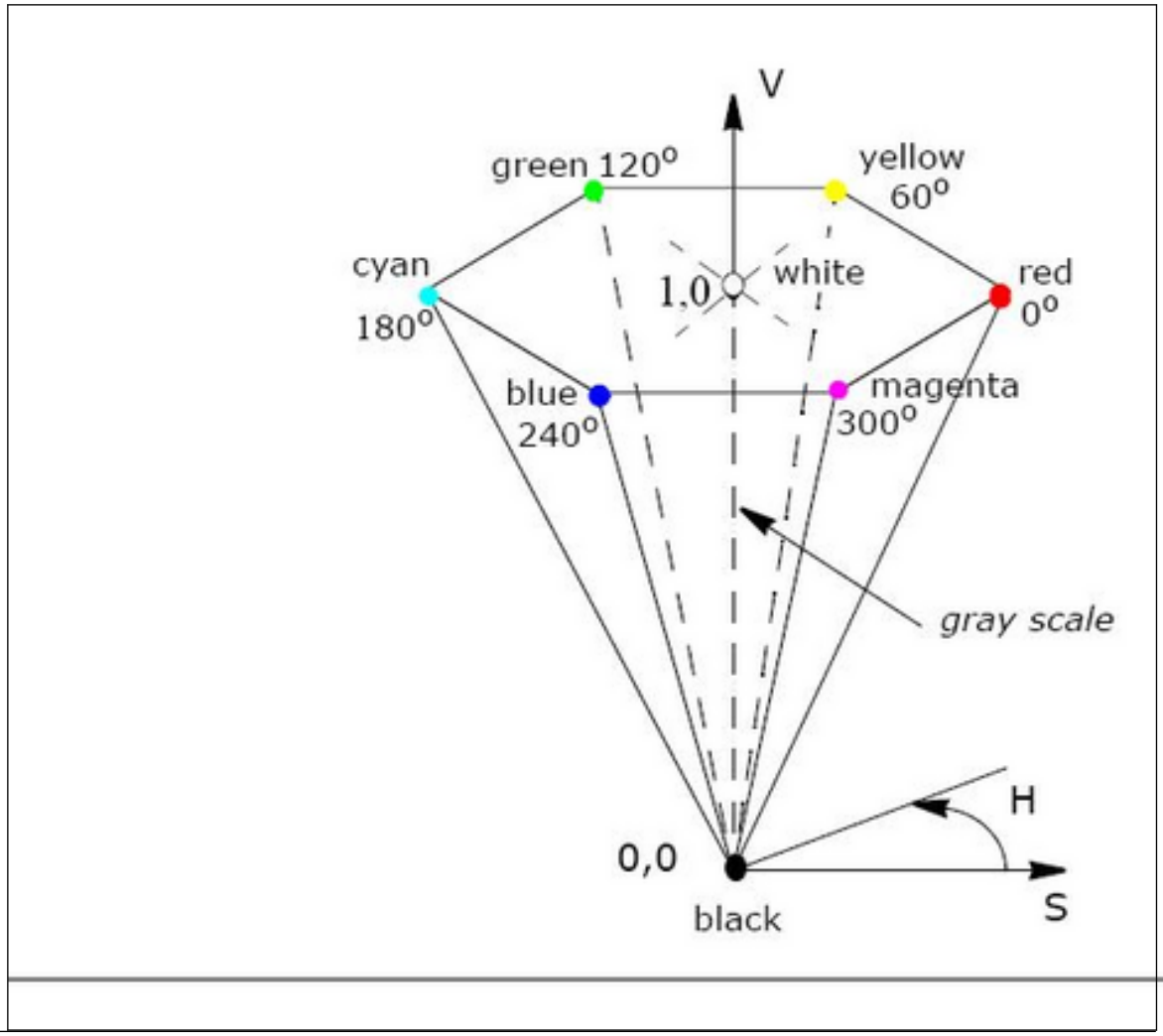
Saturation indicates the degree to which the hue differs from a neutral gray. The values run from 0, which means no color saturation, to 1, which is the fullest saturation of a given hue at a given illumination.

Intensity component - lightness (HLS) or value (HSV), indicates the illumination level. Both vary from 0 (black, no light) to 1 (white, full illumination). The difference between the two is that maximum saturation of hue (S=1) is at *value* V=1 (full illumination) in the HSV color model, and at *lightness* L=0.5 in the HLS color model.

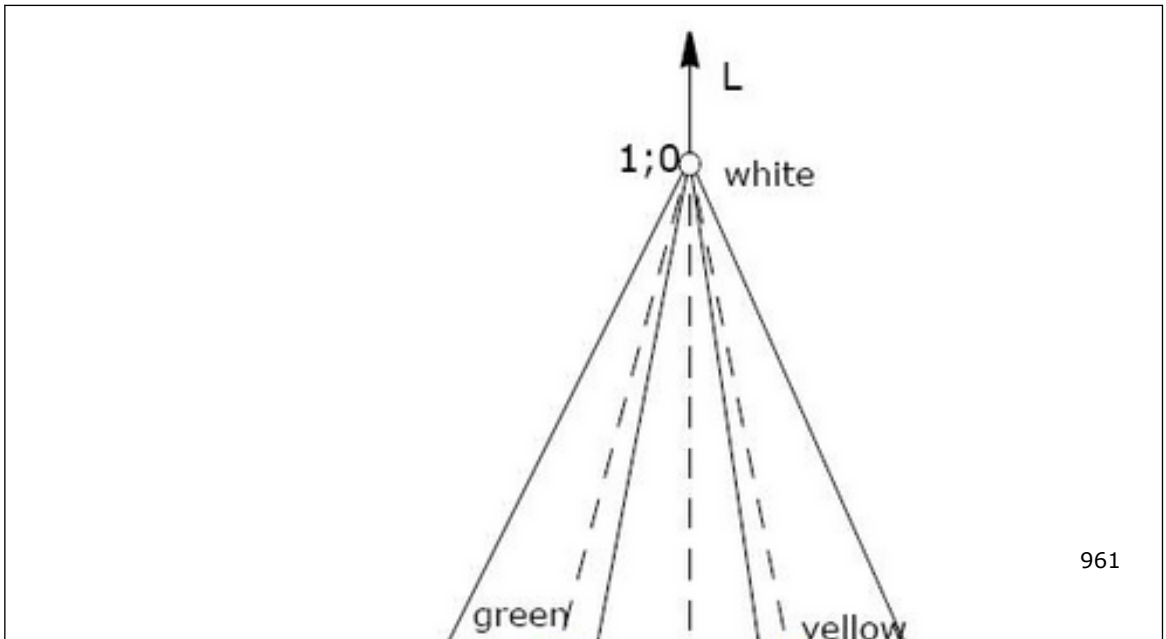
The HSV color space is essentially a cylinder, but usually it is represented as a cone or hexagonal cone (hexcone) as shown in the Figure HSV Solid, because the hexcone defines the subset of the HSV space with valid RGB values. The *value* V is the vertical axis, and the vertex V=0 corresponds to black color. Similarly, a color solid, or 3D-representation, of the HLS model is a double hexcone :ref:(**Figure HSV Solid <FIG6-9>**) with lightness as the axis, and the vertex of the second hexcone corresponding to white.

Both color models have intensity component decoupled from the color information. The HSV color space yields a greater dynamic range of saturation. Conversions from RGBToHSV/RGBToHSV and vice-versa in Intel IPP are performed in accordance with the respective pseudocode algorithms Rog85 given in the descriptions of corresponding conversion functions.

HSV Solid



HLS Solid

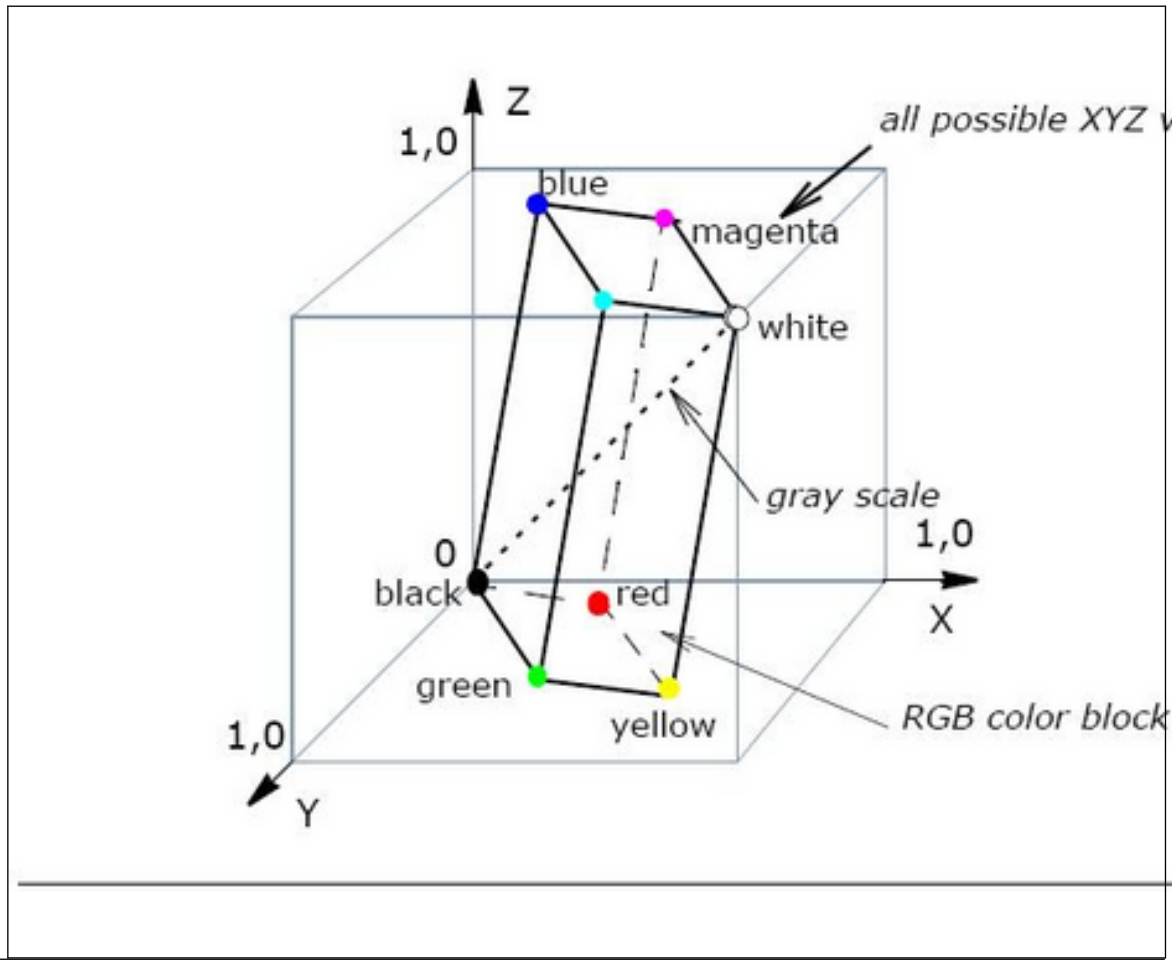


CIE XYZ Color Model

The XYZ color space is an international standard developed by the CIE (Commission Internationale de l'Eclairage). This model is based on three hypothetical primaries, XYZ, and all visible colors can be represented by using only positive values of X, Y, and Z. The CIE XYZ primaries are hypothetical because they do not correspond to any real light wavelengths. The Y primary is intentionally defined to match closely to luminance, while X and Z primaries give color information. The main advantage of the CIE XYZ space (and any color space based on it) is that this space is completely device-independent. The chromaticity diagram in **Figure CIE `:ref:`xyY <c_cie-chromaticity-diagram-and-color-gamut>** is in fact a two-dimensional projection of the CIE XYZ sub-space. Note that arbitrarily combining X, Y, and Z values within nominal ranges can easily lead to a color outside of the visible color spectrum.

The position of the block of RGB-representable colors in the XYZ space is shown in RGB Colors Cube in the XYZ Color Space.

RGB Colors Cube in the XYZ Color Space



Intel IPP functions use the following basic equations Rog85, to convert between `RGB` and `CIE XYZ` models:

```
X = 0.412453*R + 0.35758 *G + 0.180423*B
Y = 0.212671*R + 0.71516 *G + 0.072169*B
Z = 0.019334*R + 0.119193*G + 0.950227*B
```

The equations for `X,Y,Z` calculation are given on the assumption that `R,G`, and `B` values are normalized to the range [0..1].

```
R = 3.240479 * X - 1.53715 * Y - 0.498535 * Z
```

```
G = -0.969256 * X + 1.875991 * Y + 0.041556 * Z

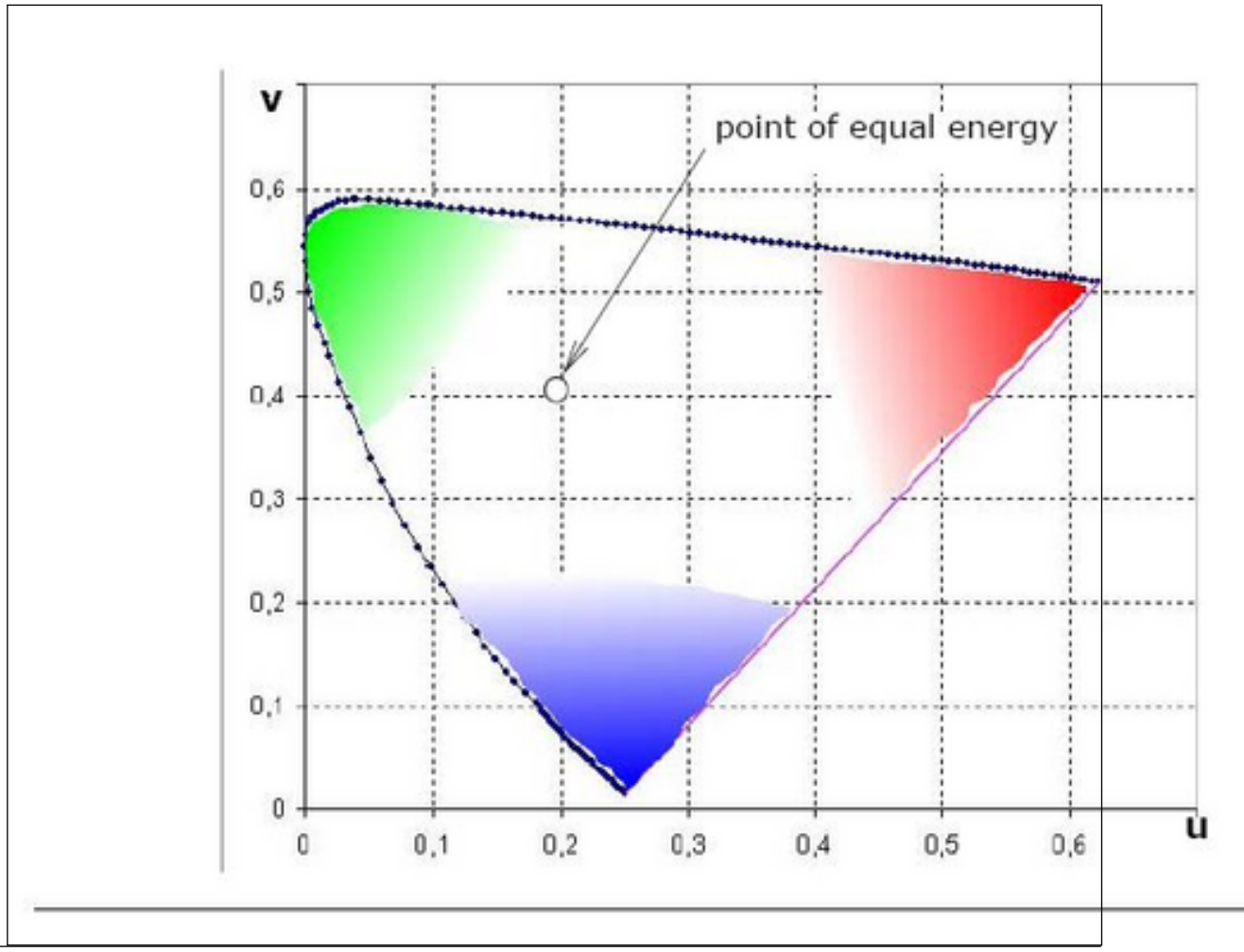
B = 0.055648 * X - 0.204043 * Y + 1.057311 * Z
```

The equations for `R`, `G`, and `B` calculation are given on the assumption that `X`, `Y`, and `Z` values are in the range [0..1].

CIE LUV and CIE Lab Color Models

The CIE LUV and CIE Lab color models are considered to be perceptually uniform and are referred to as uniform color models. Both are uniform derivations from the standard CIE XYZ space. "Perceptually uniform" means that two colors that are equally distant in the color space are equally distant perceptually. To accomplish this approach, a uniform chromaticity scale (UCS) diagram was proposed by CIE (Figure CIE u', v' Uniform Chromaticity Scale Diagram). The UCS diagram uses a mathematical formula to transform the XYZ values or x, y coordinates (**Figure CIE `:ref:`xyY <c_cie-chromaticity-diagram-and-color-gamut>**), to a new set of values that present a visually more accurate two-dimensional model. The Y lightness scale is replaced with a new scale called L that is approximately uniformly spaced but is more indicative of the actual visual differences. Chrominance components are U and V for CIE LUV, and `a` and `b` (referred to also respectively as red/blue and yellow/blue chrominances) in CIE Lab. Both color spaces are derived from the CIE XYZ color space.

CIE u', v' Uniform Chromaticity Scale Diagram



The CIE LUV color space is derived from CIE XYZ as follows (Rog85),

$L = 116. * (Y/Y_n)^{1/3.} - 16``.``$

$U = 13. * L * (u - u_n)$

$V = 13. * L * (v - v_n)$

where

$u = 4.*X / (X + 15.*Y + 3.*Z)$

$v = 9.*Y / (X + 15.*Y + 3.*Z)$

$u_n = 4``.*x``_n / (-2.*x_n + 12``.*y``_n + 3``. )``$

$v_n = 9``.*y``_n / (-2.*x_n + 12``.*y``_n + 3``. )``$

Inverse conversion is performed in accordance with equations:

$Y = Y_n * ((L + 16.) / 116.)^{3.}$

$X = -9.* Y * u / ((u - 4.)* v - u * v)$

$Z = (9.* Y - 15*v*Y - v*X) / 3. * v$

where

$u = U / (13.* L) + u_n$

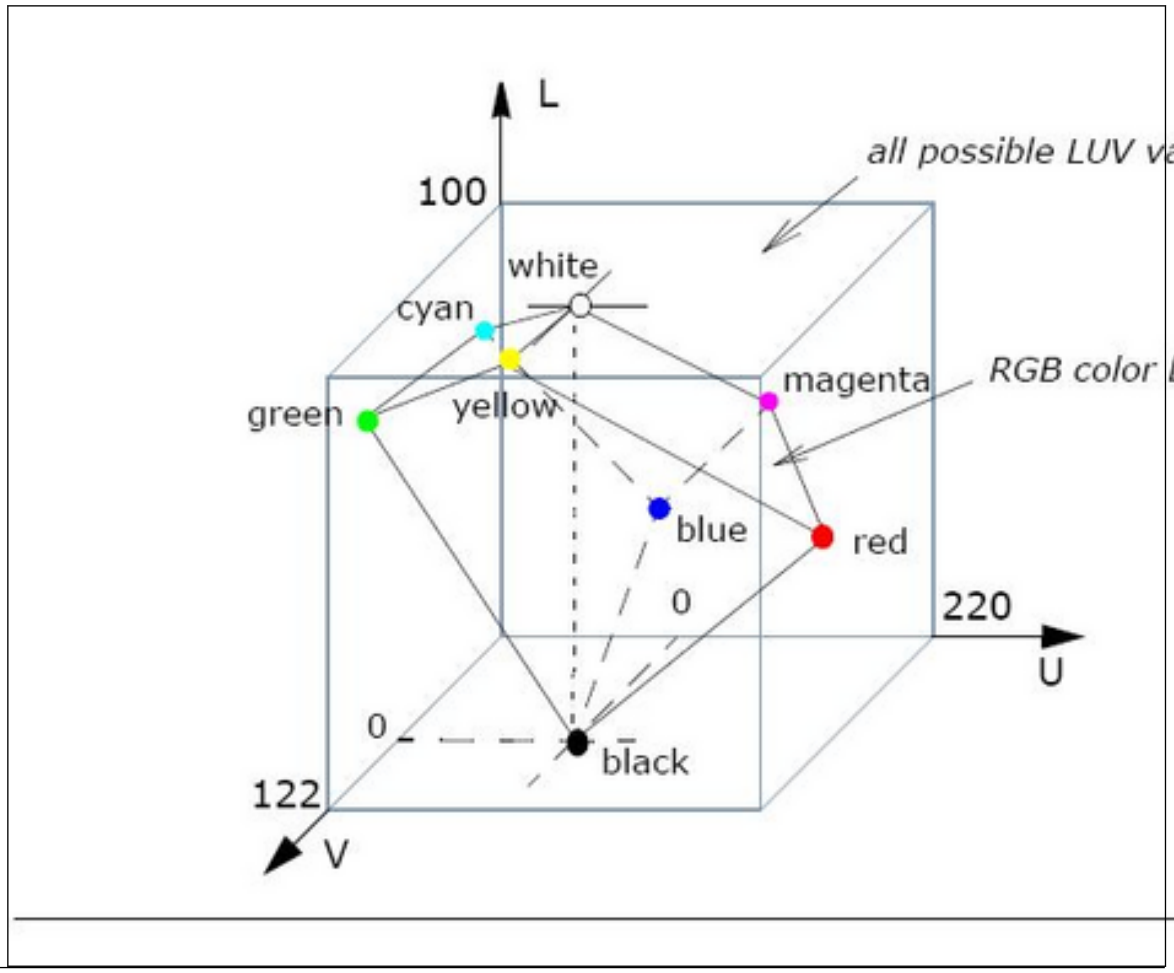
$v = V / (13.* L) + v_n$

and u_n, v_n are defined above.

Here x_n = 0.312713, y_n = 0.329016 are the CIE chromaticity coordinates of the D65 white point (ITU709), and Y_n = 1.0 is the luminance of the D65 white point. The values of the L component are in the range [0..100], U component in the range [-134..220], and V component in the range [-140..122].

The RGB-representable colors occupy only part of the LUV color space (see RGB Color Cube in the CIE LUV Color Space) limited by the nominal ranges, therefore there are many LUV combinations that result in invalid RGB values.

RGB Color Cube in the CIE LUV Color Space



The CIE Lab color space is derived from CIE XYZ as follows:

$L = 116. * (Y/Y_n)^{1/3.} - 16$ for $Y/Y_n > 0``.008856``$

$L = 903.3 * (Y/Y_n)^{1/3.}$ for $Y/Y_n \leq 0.008856$

$a = 500. * [f(X/X_n)-f(Y/Y_n)]$

$b = 200. * [f(Y/Y_n)-f(Z/Z_n)]$

where

$f(t) = t^{1/3.} - 16$ for $t > 0.008856$

$f(t) = 7.787*t + 16/116$ for $t \leq 0.008856$

Here $Y_n = 1.0$ is the luminance, and $X_n = 0.950455$, $Z_n = 1.088753$ are the chrominances for the D65 white point.

The values of the L component are in the range [0..100], a and b component values are in the range [-128..127].

Inverse conversion is performed in accordance with equations:

$Y = Y_n * P^{3.}$

$X = X_n * (P + a/500.)^{3.}$

$Z = Z_n * (P - b/200.)^{3.}$

where

$P = (L + 16)/116.$

Image Downsampling

Conventionally, digital color images are represented by setting specific values of the color space coordinates for each pixel. Color spaces with decoupled luminance and chrominance coordinates (YUV type) allow the number of bits required for acceptable color description of an image to be reduced. This reduction is based on greater sensitivity of the human eye to changes in luminance than to changes in chrominance. The idea behind this approach is to set individual value of luminance component to each pixel, while assigning the same color (chrominance components) to certain groups of pixels (sometimes called macropixels) in accordance with some specific rules. This process is called downsampling and there are different sampling formats depending on the underlying scheme.

The following sampling formats are supported by the Intel IPP image processing functions (excluding the JPEG functions):

`4:4:4 YUV (YCbCr)` - conventional format, no downsampling, Y, U(Cb), V(Cr) components are sampled at every pixel. If each component takes 8 bits, than every pixel requires 24 bits. This format is often denoted as YUV (YCbCr) with the `4:4:4` descriptor omitted.

`4:2:2 YUV (YCbCr)` - uses the 2:1 horizontal downsampling. It means that the Y component is sampled at each pixel, while U(Cb) and V(Cr) components are sampled every 2 pixels in horizontal direction. If each component takes 8 bits, the pair of pixels requires 32 bits.

`4:1:1 YCbCr` - uses the 4:1 horizontal downsampling. It means that the Y component is sampled at each pixel, while Cb and Cr components are sampled every 4 pixels horizontally. If each component takes 8 bits, each four horizontal pixels require 48 bits.

`4:2:0 YUV (YCbCr)` - uses the 2:1 horizontal downsampling and the 2:1 vertical downsampling. Y is sampled at each pixel, U(Cb) and V(Cr) are sampled at every block of 2x2 pixels. If each component takes 8 bits, each four-pixel block requires 48 bits.

In JPEG compression, downsampling has specific distinctive features and is denoted in a slightly different way. In JPEG domain, sampling formats determine the structure of minimal coded units, or MCUs. Therefore, the Intel IPP functions specific for a JPEG codec, support the following sampling formats:

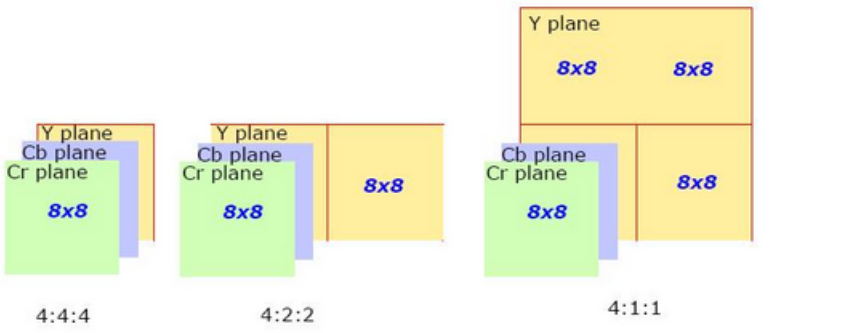
`4:4:4 YCbCr` - for every 8x8 block of Y samples, there is one 8x8 block of each Cb and Cr samples.

`4:2:2 YCbCr` - for every two horizontal 8x8 blocks of Y samples, there is one 8x8 block of each Cb and Cr samples.

`4:1:1 YCbCr` - for every four (two in horizontal and two in vertical direction) 8x8 blocks of Y samples, there is one 8x8 block of each Cb and Cr samples.

Structure of the corresponding MCU for each of these sampling formats is shown in Figure MCU Structure for Different JPEG Sampling Formats.

MCU Structure for Different JPEG Sampling Formats |



RGB Image Formats

In addition to the 24-bit-per-pixel RGB/BGR image formats, the Intel IPP color conversion functions support 32-bit-per-pixel RGB/BGR formats, which include three RGB channels plus alpha channel. For 24-bit formats, each color is one byte, every pixel is three bytes. For 32-bit formats, each color is one byte and alpha component is one byte, which yields four bytes per pixel. Memory layout for these formats is given in Table "Pixel-Order Image Formats".

For 16-bit formats, every pixel is two bytes and each color occupies a specified number of bits. The figure below shows all the supported 16-bit-per-pixel formats and their memory layout (bit order):

16-bit pixel formats |

16-bit pixel formats

	High-order byte	Low-order byte
RGB565	B4 B3 B2 B1 B0 G5 G4 G3	G2 G1 G0 R4 R3 R2 R1 R0
RGB555	X B4 B3 B2 B1 B0 G4 G3	G2 G1 G0 R4 R3 R2 R1 R0
RGB444	X X X X B3 B2 B1 B0	G3 G2 G1 G0 R3 R2 R1 R0
BGR565	R4 R3 R2 R1 R0 G5 G4 G3	G2 G1 G0 B4 B3 B2 B1 B0
BGR555	X R4 R3 R2 R1 R0 G4 G3	G2 G1 G0 B4 B3 B2 B1 B0
BGR444	X X X X R3 R2 R1 R0	G3 G2 G1 G0 B3 B2 B1 B0

R - Red, G - Green, B - Blue, X - free bit

Pixel and Planar Image Formats

Data storage for an image can be pixel-oriented or planar-oriented (planar). For images in pixel order, all channel values for each pixel are clustered and stored consecutively. Their layout depends on the color model and downsampling scheme.

Table Pixel-Order Image Formats lists all pixel-order image formats that are supported by the Intel IPP color conversion functions and shows the corresponding channel values order (here, group of underlined symbols represents one pixel and symbol A denotes alpha-channel value). The last column of this table gives an example of an Intel IPP color conversion function that uses the respective image format.

Pixel-Order Image Formats

Image Format	Number of Channels	Channel Values Order	Example
RGB RGB444 RGB555 RGB565	3	R0 G0 B0 R1 G1 B1 R2 G2 B2	ippiRGBTToYUV_8u_C3R ippiYCbCrToRGB444_8u16u_C3R ippiYCbCrToRGB555_8u16u_C3R ippiYCbCrToRGB565_8u16u_C3R
RGB	4	R0 G0 B0 A0 R1 G1 B1 A1	ippiRGBToYUV_8u_AC4R
BGRBGR444BGR555BGR565	3	B0 G0 R0 B1 G1 R1 B2 G2 R2	ippiYCbCrToBGR_8u_P3C3R ippiYCbCrToBGR444_8u16u_C3R ippiYCbCrToBGR555_8u16u_C3R ippiYCbCrToBGR565_8u16u_C3R
BGR	4	B0 G0 R0 A0 B1 G1 R1 A1	ippiBGRToHLS_8u_AC4R
YUV	3	Y0 U0 V0 Y1 U1 V1 Y2 U2 V2	ippiYUVToRGB_8u_C3R
YUV	4	Y0 U0 V0 A0 Y1 U1 V1 A1	ippiYUVToRGB_8u_AC4R
4:2:2 YUV	2	Y0 U0 Y1 V0 Y2 U1 Y3 V1	ippiYUV422T0RGB_8u_C2C3R
YCbCr	3	Y0 Cb0 Cr0 Y1 Cb1 Cr1	ippiYCbCrToRGB_8u_C3R
YCbCr	4	Y0 Cb0 Cr0 A0 Y1 Cb1 Cr1 A1	ippiYCbCrToRGB_8u_AC4R
4:2:2 YCbCr	2	Y0 Cb0 Y1 Cr0 Y2 Cb1 Y3 Cr1	ippiYCbCr422ToRGB_8u_C2C3R
4:2:2 YCrCb	2	Y0 Cr0 Y1 Cb0 Y2 Cr1 Y3 Cb1	ippiYCrCb422ToYCbCr422_8u_C2P3R
4:2:2 CbYCr	2	Cb0 Y0 Cr0 Y1 Cb1 Y2 Cr1 Y3	ippiCbYCr422ToRGB_8u_C2C3R
XYZ	3	X0 Y0 Z0 X1 Y1 Z1 X2 Y2 Z2	ippiXYZToRGB_8u_C3R
XYZ	4	X0 Y0 Z0 X1 Y1 Z1 X2 Y2 Z2	ippiXYZToRGB_16u_AC4R
LUV	3	L0 U0 V0 L1 U1 V1 L2 U2 V2	ippiLUVToRGB_16s_C3R

LUV	4	L0 U0 V0 A0 L1 U1 V1 A1	ippiLUVToRGB_32f_AC4R
YCC	3	Y0 C0 C0 Y1 C1 C1 Y2 C2 C2	ippiYCCToRGB_8u_C3R
YCC	4	Y0 C0 C0 A0 Y1 C1 C1 A1	ippiYCCToRGB_8u_AC4R
HLS	3	H0 L0 S0 H1 L1 S1 H2 L2 S2	ippiHLSToRGB_16u_C3R
HLS	4	H0 L0 S0 A0 H1 L1 S1 A0	ippiHLSToRGB_16u_AC4R
HSV	3	H0 S0 V0 H1 S1 V1 H2 S2 V2	ippiHSVToRGB_16s_C3R
HSV	4	H0 S0 V0 A0 H1 S1 C1 A1	ippiHSVToRGB_16s_AC4R

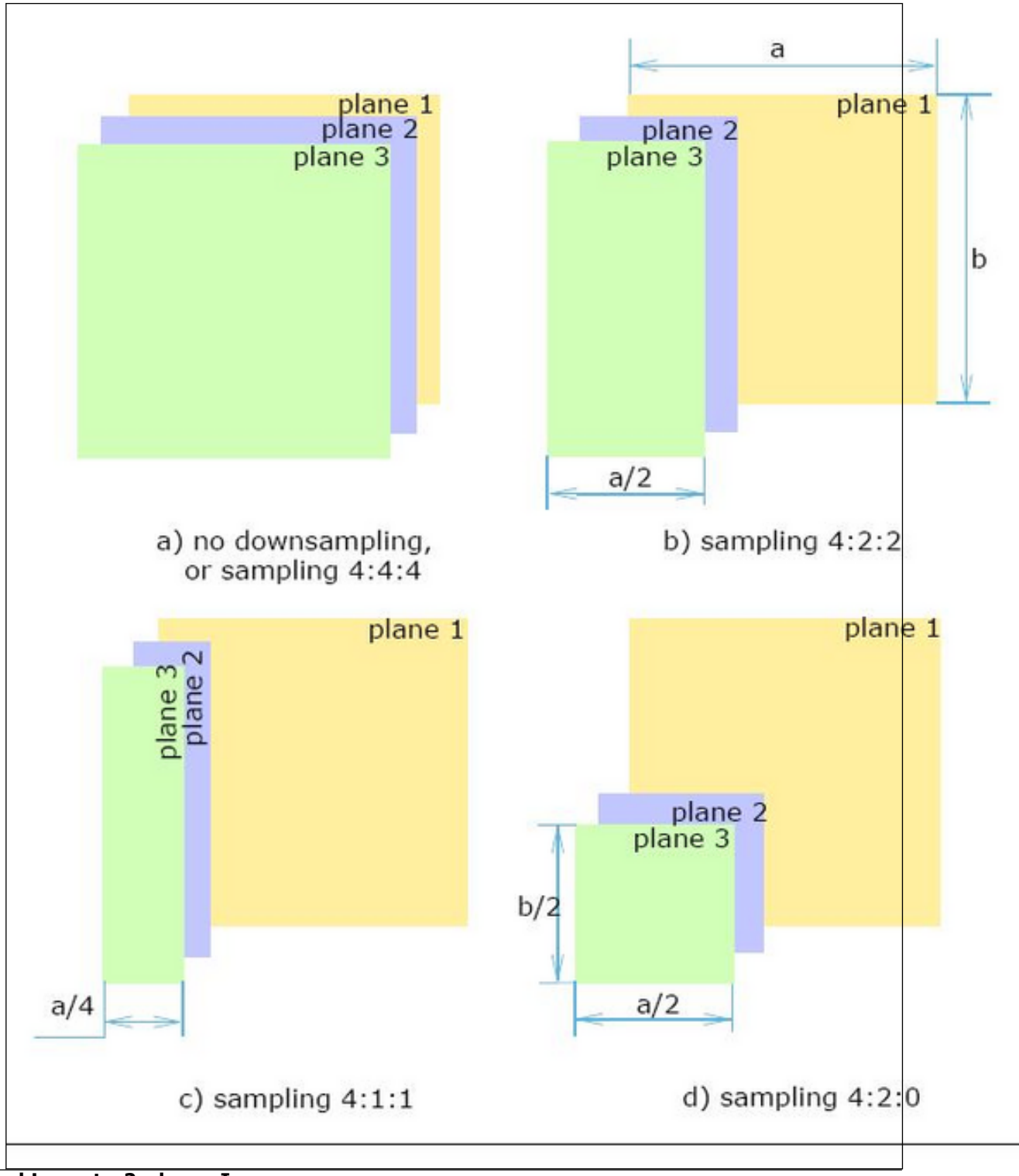
Planar image formats supported by the Intel IPP color conversion functions are listed in Table “Planar Image Formats” along with examples of the Intel IPP functions using that format. Planes layout and their relative sizes are shown in Figure Plane Size and Layout: 3-planes Images and Figure Plane Size and Layout: 2-planes Images.

Image Format	Number of Planes	Plane 0	Plane 1	Plane 2	Figure	Example
RGB	3	R	G	B	see Figure below, a	ippiRGBToYUV_8u_P3R
YUV	3	Y	U	V	see Figure below, a	ippiYUVToRGB_8u_P3R
4:2:2 YUV	3	Y	U	V	see Figure below, b	ippiYUV422ToRGB_8u_P3
4:2:0 YUV	3	Y	U	V	see Figure below, d	ippiYUV420ToRGB_8u_P3C3R
YCbCr	3	Y	Cb	Cr	see Figure below, a	ippiYCbCrToRGB_8u_P3R
4:2:2 YCbCr	3	Y	Cb	Cr	see Figure below, b	ippiYCbCr422ToRGB_8u_P3C2R
4:1:1 YCbCr	3	Y	Cb	Cr	see Figure below, c	ippiYCbCr411ToRGB_8u_P3C2R
4:1:1 YCbCr	2	Y	CbCr	•	see Figure below for 2-planes images, a	ippiYCbCr411ToRGB_8u_P2R

4:2:0 YCbCr	3	Y	Cb	Cr	see Figure below, d	ippiYCbCr420ToRGB_8u_P3C3R
4:2:0 YCbCr	2	Y	CbCr	•	see Figure below for 2-planes images, b	ippiYCbCr420ToRGB_8u_P2R

4:2:0 YCrCb	3	Y	Cr	Cb	see Figure below, d	ippiYCrCb420ToYCbCr422_8u_P3R

Plane Size and Layout - 3-planes Images



Plane Size and Layout - 2-planes Images



Color Model Conversion

- RGBToYUV
- YUVToRGB
- RGBToYUV422
- YUV422ToRGB
- RGBToYUV420
- YUV420ToRGB
- BGRToYUV420
- YUV420ToBGR
- YUV422v210ToRGB, YUV422v210ToBGR
- YUV422v210ToGray
- RGBToYCbCr
- YCbCrToRGB
- YCbCrToBGR
- YCbCrToBGR_709CSC
- RGBToYCbCr422
- YCbCr422ToRGB
- RGBToYCrCb422
- YCrCb422ToRGB, YCrCb422ToBGR
- BGRToYCbCr422
- YCbCr422ToBGR
- YCbCr422ToGray
- RGBToCbYCr422, RGBToCbYCr422Gamma
- CbYCr422ToRGB
- BGRToCbYCr422
- BGRToCbYCr422_709HDTV
- CbYCr422ToBGR
- CbYCr422ToBGR_709HDTV
- RGBToYCbCr420
- YCbCr420ToRGB, YCbCr420ToBGR
- RGBToYCrCb420
- YCrCb420ToRGB, YCrCb420ToBGR
- BGRToYCbCr420
- BGRToYCbCr420_709CSC
- BGRToYCbCr420_709HDTV
- BGRToYCrCb420_709CSC
- YCbCr420ToBGR
- YCbCr420ToBGR_709CSC
- YCbCr420ToBGR_709HDTV
- BGRToYCrCb420
- BGRToYCbCr411
- YCbCr411ToBGR
- RGBToXYZ
- XYZToRGB
- RGBToLUV, BGRToLUV
- LUVToRGB, LUVToBGR
- BGRToLab, RGBToLab
- LabToBGR, LabToRGB
- RGBToYCC
- YCCToRGB
- RGBToHLS
- HLSToRGB
- BGRToHLS
- HLSToBGR
- RGBToHSV
- HSVToRGB
- RGBToYCoCg

- YCoCgToRGB
- BGRToYCoCg
- SBGRToYCoCg
- YCoCgToBGR
- YCoCgToSBGR
- BGRToYCoCg_Rev
- SBGRToYCoCg_Rev
- YCoCgToBGR_Rev
- YCoCgToSBGR_Rev

RGBToYUV

Converts an RGB image to the YUV color model.

Syntax

Case 1: Operation on pixel-order data

```
IppStatus ippiRGBToYUV_<mod>(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep,
IppiSize roiSize);
```

Supported values for `mod`:

`8u_C3R`	`8u_AC4R`

Case 2: Operation on planar data

```
IppStatus ippiRGBToYUV_8u_P3R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst[3], int
dstStep, IppiSize roiSize);
```

Case 3: Conversion from pixel-order to planar data

```
IppStatus ippiRGBToYUV_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3],int
dstStep, IppiSize roiSize);
```

```
IppStatus ippiRGBToYUV_8u_AC4P4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[4], int
dstStep, IppiSize roiSize);
```

Include Files

`ippcc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the source image ROI for pixel-order data. An array of pointers to the source image ROI in separate color planes in case of planar data.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.

pDst	Pointer to the destination ROI for pixel-order data. An array of pointers to destination buffers in separate color planes in case of planar data.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the gamma-corrected R'G'B' image pSrc to the Y'U'V' image pDst (see Figure Converting an RGB image to YUV) according to the following formulas:

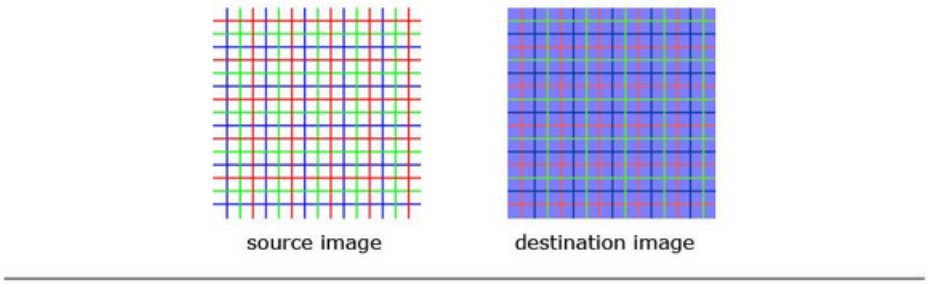
```
Y' = 0.299*R' + 0.587*G' + 0.114*B'

U' = -0.147*R' - 0.289*G' + 0.436*B' = 0.492*(B'-Y')

V' = 0.615*R' - 0.515*G' - 0.100*B' = 0.877*(R'-Y')
```

For digital RGB values in the range [0..255], Y' has the range [0..255], U varies in the range [-112..+112], and V in the range [-157..+157]. To fit in the range of [0..255], a constant value 128 is added to computed U and V values, and V is then saturated.

Converting an RGB image to YUV



Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if pSrc or pDst pointer is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with a zero or negative value.

Example

The code example below shows how to use the function ippiRGBToYUV_8u_C3R.

```
# define nChannels 3


int main () {
```

```
      Ipp 8 u src [3*3* nChannels ] = {
         255, 0, 0, 255, 0, 0, 255, 0, 0,
         0, 255, 0, 0, 255, 0, 0, 255, 0,
         0, 0, 255, 0, 0, 255, 0, 0, 255};
      Ipp 8 u dst [3*3* nChannels ];
      IppiSize roiSize = { 3, 3 };
      IppStatus st = ippStsNoErr ;
      int srcStep = 3* nChannels ;
      int dstStep = 3* nChannels ;
      st = ippiRGBToYUV _8 u _ C 3 R ( src , srcStep , dst , dstStep , roiSize );
      if ( st == ippStsNoErr){
            printf("\n *************  passed ****************\n");
      }else{
            printf("\n *************  failed ****************\t");
      }
      return 0;
}
```

Result:

```
255   0   0 255   0   0 255   0   0
  0 255   0   0 255   0   0 255   0    src
  0   0 255   0   0 255   0   0 255


 76  90 255  76  90 255  76  90 255
149  54   0 149  54   0 149  54   0    dst
 29 239 102  29 239 102  29 239 102
```

YUVToRGB

Converts a YUV image to the RGB color model.

Syntax

Case 1: Operation on pixel-order data

```
IppStatus ippiYUVToRGB_<mod>(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep,
IppiSize roiSize);
```

Supported values for `mod`:

`8u_C3R`
`8u_AC4R`

```
IppStatus ippiYUVToRGB_8u_C3C4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int
dstStep, IppiSize roiSize, Ipp8u aval);
```

Case 2: Operation on planar data

```
IppStatus ippiYUVToRGB_8u_P3R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst[3], int
dstStep, IppiSize roiSize);
```

Case 3: Conversion from planar to pixel-order data

```
IppStatus ippiYUVToRGB_8u_P3C3R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst, int
dstStep, IppiSize roiSize);
```

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Pointer to the source buffer for pixel-order data. An array of pointers to separate source color planes in case of planar data.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination buffer for pixel-order data. An array of pointers to separate destination color planes in case of planar data.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
aval	Constant value to create the fourth channel.
roiSize	Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the Y'U'V' image pSrc to the gamma-corrected R'G'B' image pDst according to the following formulas:

```
R' = Y' + 1.140*V'
G' = Y' - 0.394*U' - 0.581*V'
B' = Y' + 2.032*U'
```

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if pSrc or pDst is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with a zero or negative value.

RGBToYUV422

Converts an RGB image to the YUV color model; uses 4:2:2 sampling.

Syntax

Case 1: Operation on pixel-order data

```
IppStatus ippiRGBToYUV422_8u_C3C2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Case 2: Operation on planar data with ROI

```
IppStatus ippiRGBToYUV422_8u_P3R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

Case 3: Operation on planar data without ROI

```
IppStatus ippiRGBToYUV422_8u_P3(const Ipp8u* pSrc[3], Ipp8u* pDst[3], IppiSize imgSize);
```

Case 4: Conversion from pixel-order to planar data with ROI

```
IppStatus ippiRGBToYUV422_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

Case 5: Conversion from pixel-order to planar data without ROI

```
IppStatus ippiRGBToYUV422_8u_C3P3(const Ipp8u* pSrc, Ipp8u* pDst[3], IppiSize imgSize);
```

Include Files

`ippcc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the source image buffer for pixel-order image. An array of pointers to the source image buffer in each color plane for planar image.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image for operations with ROI.
`pDst`	Pointer to the destination image buffer for pixel-order image. An array of pointers to the destination image buffer in each color plane for planar image.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image for operations with ROI. An array of three values for planar image.
`roiSize`	Size of the source and destination ROI in pixels.
`imgSize`	Size of the source and destination images in pixels for operations without ROI.

Description

This function converts the gamma-corrected R'G'B' image pSrc to the Y'U'V' image pDst with 4:2:2 sampling format, according to the same formulas as the function ippiRGBToYUV does. For more details on this sampling format, see Table "Pixel-Order Image Formats" and Table "Planar Image Formats".

For digital RGB values in the range [0..255], Y' has the range [0..255], U varies in the range [-112..+112], and V in the range [-157..+157]. To fit in the range of [0..255], the constant value 128 is added to computed U and V values, and V is then saturated.

Some function flavors operates with ROI (see Regions of Interest in Intel IPP). The function flavors that does not use ROI operate on the assumption that both the source and destination images have the same size and occupy a contiguous memory area, which means that image rows are not padded with zeroes. In this case the step parameters are not needed.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if pSrc or pDst pointer is NULL.
ippStsSizeErr	Indicates an error condition if roiSize or imgSize has a field with a zero or negative value.

YUV422ToRGB

Converts a YUV image with the 4:2:2 sampling to the RGB color model.

Syntax

Case 1: Operation on pixel-order data

```
IppStatus ippiYUV422ToRGB_8u_C2C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Case 2: Operation on planar data with ROI

```
IppStatus ippiYUV422ToRGB_8u_P3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
```

Case 3: Operation on planar data without ROI

```
IppStatus ippiYUV422ToRGB_8u_P3(const Ipp8u* pSrc[3], Ipp8u* pDst[3], IppiSize imgSize);
```

Case 4: Conversion from planar to pixel-order data with ROI

```
IppStatus ippiYUV422ToRGB_<mod>(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_P3C3R	8u_P3AC4R

Case 5: Conversion from planar to pixel-order data without ROI

```
IppStatus ippiYUV422ToRGB_8u_P3C3(const Ipp8u* pSrc[3], Ipp8u* pDst, IppiSize imgSize);
```

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Pointer to the source image buffer for pixel-order image. An array of pointers to the source image buffer in each color plane for planar image.
srcStep	Distance in bytes between starts of consecutive lines in the source image for operations with ROI. An array of three values in case of planar image.
pDst	Pointer to the destination image buffer for pixel-order image. An array of pointers to the destination image buffers in each color plane for planar image.
dstStep	Distance in bytes between starts of consecutive lines in the destination image for operations with ROI.
roiSize	Size of the source and destination ROI in pixels.
imgSize	Size of the source and destination images in pixels for operations without ROI.

Description

This function converts the Y'U'V' image pSrc to the gamma-corrected R'G'B' image pDst according to the same formulas as the function ippiYUVToRGB does. The difference is that ippiYUV422ToRGB 4:2:0 sampling the input data to be in 4:2:2 sampling format (see Table "Pixel-Order Image Formats" and Table "Planar Image Formats" for more details).

The function ippiYUV422ToRGB_P3AC4R additionally creates an alpha channel in the destination image with alpha values set to zero.

Some function flavors operates with ROI (see Regions of Interest in Intel IPP). The function flavors that do not use ROI operate on the assumption that both the source and destination images have the same size and occupy a contiguous memory area, which means that image rows are not padded with zeroes. In this case the step arguments are not needed.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if pSrc or pDst is NULL.
ippStsSizeErr	ndicates an error condition if roiSize or imgSize has a field with a zero or negative value.

RGBToYUV420

Converts an RGB image to the 4:2:0 YUV image.

Syntax

Case 1: Operation on planar data with ROI

```
IppStatus ippiRGBToYUV420_8u_P3R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

Case 2: Operation on planar data without ROI

```
IppStatus ippiRGBToYUV420_8u_P3(const Ipp8u*pSrc[3], Ipp8u* pDst[3], IppiSize imgSize);
```

Case 3: Conversion from pixel-order to planar data with ROI

```
IppStatus ippiRGBToYUV420_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

Case 4: Conversion from pixel-order to planar data without ROI

```
IppStatus ippiRGBToYUV420_8u_C3P3(const Ipp8u* pSrc, Ipp8u* pDst[3], IppiSize imgSize);
```

Include Files

`ippcc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the source image buffer for pixel-order image. An array of pointers to the source image buffers in each color plane for planar image.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image for operations with ROI.
`pDst`	An array of pointers to the destination image buffers in each color plane.
`dstStep`	An array of distances in bytes between starts of consecutive lines in each plane of the destination image for operations with ROI.
`roiSize`	Size of the source and destination ROI in pixels.
`imgSize`	Size of the source and destination images in pixels for operations without ROI.

Description

This function converts the gamma-corrected `R'G'B'` image `pSrc` to the Y'U'V' image `pDst` with the 4:2:0 sampling (see Table "Planar Image Formats" for more details). The conversion is performed in the accordance with the same formulas as the function `ippiRGBToYUV` does.

For digital RGB values in the range [0..255], `Y'` has the range [0..255], `U` varies in the range [-112..+112], and `V` in the range [-157..+157]. To fit in the range of [0..255], a constant value 128 is added to computed `U` and `V` values, and `V` is then saturated.

Some function flavors operates with ROI see Regions of Interest in Intel IPP).

The function flavors that does not use ROI operate on the assumption that both the source and destination images have the same size and occupy a contiguous memory area, which means that image rows are not padded with zeroes. In this case the step parameters are not needed.

`roiSize.width(imgSize.width)` and `roiSize.height` (`imgSize.height`) should be multiples of 2. Otherwise, the function reduces their original values to the nearest multiples of 2, performs operation, and returns warning message.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if `pSrc` or `pDst` is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize` or `imgSize` has a field with a zero or negative value.
`ippStsDoubleSize`	Indicates a warning if `roiSize` or `imgSize` has a field that is not a multiple of 2.

YUV420ToRGB

Converts a YUV image that has 4:2:0 sampling format to the RGB image.

Syntax

Case 1: Operation on planar data with ROI

```
IppStatus ippiYUV420ToRGB_8u_P3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst[3],
int dstStep, IppiSize roiSize);
```

Case 2: Operation on planar data without ROI

```
IppStatus ippiYUV420ToRGB_8u_P3(const Ipp8u* pSrc[3], Ipp8u* pDst[3], IppiSize
imgSize);
```

Case 3: Conversion from planar to pixel-order data with ROI

```
IppStatus ippiYUV420ToRGB_<mod>(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst, int
dstStep, IppiSize roiSize);
```

Supported values for `mod`:

`8u_P3C3R`	`8u_P3AC4R`

Case 4: Conversion from planar to pixel-order data without ROI

```
IppStatus ippiYUV420ToRGB_8u_P3C3(const Ipp8u* pSrc[3], Ipp8u* pDst, IppiSize imgSize);
```

Include Files

`ippcc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	An array of pointers to the source image buffers in each color plane.
`srcStep`	An array of distances in bytes between starts of consecutive lines in each source image planes for operations with ROI.
`pDst`	Pointer to the destination image buffer for pixel-order images. An array of pointers to the destination image buffers in each color plane for planar images.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image for operations with ROI.
`roiSize`	Size of the source and destination ROI in pixels.
`imgSize`	Size of the source and destination images in pixels for operations without ROI.

Description

This function converts the Y'U'V' image `pSrc` to the gamma-corrected `R'G'B'` image `pDst` according to the same formulas as the function `ippiYUVToRGB` does. The difference is that `ippiYUV420ToRGB` 4:2:0 sampling the input data to be in 4:2:2 sampling format (see Table "Planar Image Formats" for more details).

The function `ippiYUV420ToRGB_P3AC4R` additionally creates an alpha channel in the destination image with alpha values set to zero.

Some function flavors operates with ROI see Regions of Interest in Intel IPP).

The function flavors that does not use ROI operate on the assumption that both the source and destination images have the same size and occupy a contiguous memory area, which means that image rows are not padded with zeroes. In this case the step parameters are not needed.

`roiSize.width(imgSize.width)` and `roiSize.height` (`imgSize.height`) should be multiples of 2. Otherwise, the function reduces their original values to the nearest multiples of 2, performs operation, and returns warning message.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if `pSrc` or `pDst` pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize` or `imgSize` has a field with a zero or negative value.

ippStsDoubleSize	Indicates a warning if roiSize or imgSize has a field that is not a multiple of 2.

BGRToYUV420

Converts an BGR image to the YUV color model; uses 4:2:0 sampling

Syntax

```
IppStatus ippiBGRToYUV420_8u_AC4P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	An array of pointers to the destination image buffers in each color plane.
dstStep	An array of distances in bytes between starts of consecutive lines in each plane of the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the gamma-corrected B'G'R' image pSrc to the Y'U'V' image pDst with the 4:2:0 sampling (see Table "Planar Image Formats" for more details). The function uses the same formulas as the function ippiRGBToYUV does.

For digital BGR values in the range [0..255], Y' varies in the range [0..255], U – in the range [-112..+112], and V – in the range [-157..+157]. To fit in the range of [0..255], a constant value 128 is added to the computed U and V values, and V is then saturated.

roiSize.width and roiSize.height should be multiples of 2. If not the function reduces their original values to the nearest multiples of 2, performs operation, and returns warning message.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if pSrc or pDst is NULL.
ippStsSizeErr	Indicates an error condition if roiSize or imgSize has a field with a zero or negative value.
ippStsDoubleSize	Indicates a warning if roiSize or imgSize has a field that is not a multiple of 2.

YUV420ToBGR

Converts a YUV image that has 4:2:0 sampling to the BGR image.

Syntax

```
IppStatus ippiYUV420ToBGR_8u_P3C3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst,
int dstStep, IppiSize roiSize);
```

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	An array of pointers to ROI in each color plane in the source image.
srcStep	An array of distances in bytes between starts of consecutive lines in the source image planes.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the Y'U'V' image pSrc to the gamma-corrected B'G'R' image pDst according to the same formulas as the function ippiYUVToRGB does. The input data must be presented in the 4:2:0 sampling format (see Table "Planar Image Formats" for more details).

roiSize.width and roiSize.height should be multiples of 2. Otherwise, the function reduces their original values to the nearest multiples of 2, performs operation, and returns warning message.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if pSrc or pDst is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with a zero or negative value.
ippStsDoubleSize	Indicates a warning if roiSize has a field that is not a multiple of 2.

YUV422v210ToRGB, YUV422v210ToBGR

Converts a YUV422 (v210) image to a RGB/BGR image for ITU-R BT.709 HDTV signal.

Syntax

```
IppStatus ippiYUV422v210ToRGB_709HDTV_32u16u_C3(const Ipp32u* pSrc, int srcStep,
Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiYUV422v210ToBGR_709HDTV_32u16u_C3(const Ipp32u* pSrc, int srcStep,
Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance, in bytes, between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance, in bytes, between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the YUV image pSrc, packed in the 4:2:2 sampling format, to the gamma-corrected RGB/BGR image pDst for digital component video signals in compliance with the ITU-R BT.709 Recommendation ITU709 for high-definition TV (HDTV). The source YUV image has the following sequence of bytes: UYV|YUY|VYU|YVY, … . The conversion is performed according to the following formulas:

```
R = Y + 1.540*(V - 512)
G = Y - 0.459*(V - 512) - 0.183*(U - 512)
B = Y + 1.816*(U - 512)
```

The output RGB/BGR values are saturated to the range R [0..31], G [0..63], B [0..31].

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if `pSrc` or `pDst` is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize` has a field with a zero or negative value.

YUV422v210ToGray

Converts a YUV422 (v210) image to a grayscale image for ITU-R BT.709 HDTV signal

Syntax

```
IppStatus ippiYUV422v210ToGray_709HDTV_32u16u_C3C1(const Ipp32u* pSrc, int srcStep,
Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

Include Files

`ippcc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance, in bytes, between starts of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance, in bytes, between starts of consecutive lines in the destination image.
`roiSize`	Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the YUV image pSrc, packed in the 4:2:2 sampling format, to the grayscale 16U_C1 image pDst for digital component video signals in compliance with the ITU-R BT.709 Recommendation ITU709 for high-definition TV (HDTV). The source YUV image has the following sequence of bytes: UYV|YUY|VYU|YVY, … .

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if pSrc or pDst is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with a zero or negative value.

RGBToYCbCr

Converts an RGB image to the YCbCr color model.

Syntax

Case 1: Operation on pixel-order data

```
IppStatus ippiRGBToYCbCr_<mod>(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C3R	8u_AC4R

Case 2: Operation on planar data

```
IppStatus ippiRGBToYCbCr_8u_P3R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
```

Case 3: Conversion from pixel-order to planar data

```
IppStatus ippiRGBToYCbCr_<mod>(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C3P3R	8u_AC4P3R

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Pointer to the source image ROI for a pixel-order image. An array of pointers to ROI in each separate source color planes for planar images.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination imnage ROI. Array of pointers to ROI in the separate destination color planes for planar images.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the gamma-corrected R'G'B' image pSrc with values in the range [0..255] to the Y'Cb'Cr' image pDst according to the following formulas:

```
Y' = 0.257*R' + 0.504*G' + 0.098*B' + 16
Cb' = -0.148*R' - 0.291*G' + 0.439*B' + 128
Cr' = 0.439*R' - 0.368*G' - 0.071*B' + 128
```

In the YCbCr model, Y is defined to have a nominal range [16..235], while Cb and Cr are defined to have a range [16..240], with the value of 128 as corresponding to zero.

Both the source and destination images have the same bit depth.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if pSrc or pDst is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with a zero or negative value.

YCbCrToRGB

Converts a YCbCr image to the RGB color model.

Syntax

Case 1: Operation on pixel-order data

```
IppStatus ippiYCbCrToRGB_<mod>(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C3R	8u_AC4R

Case 2: Operation on planar data

```
IppStatus ippiYCbCrToRGB_8u_P3R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
```

```
IppStatus ippiYCbCrToRGB_8u_P3C3R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Case 3: Conversion from planar to pixel-order data

```
IppStatus ippiYCbCrToRGB_8u_P3C4R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst , int dstStep, IppiSize roiSize, Ipp8u aval);
```

Include Files

`ippcc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the source image ROI for pixel-order image. Array of pointers to the ROI in each separate source color planes for planar images.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI. Array of pointers to the ROI in the separate destination color planes for planar images.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image.
`roiSize`	Size of the source and destination ROI in pixels.
`aval`	Constant value to create the fourth channel.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the Y'Cb'Cr' image `pSrc` to the 24-bit gamma-corrected `R'G'B'` image `pDst`. The following formulas are used for conversion:

```
R' = 1.164*(Y' - 16) + 1.596*(Cr' - 128)
```

```
G' = 1.164*(Y' - 16) - 0.813*(Cr' - 128) - 0.392*(Cb' - 128)
```

```
B' = 1.164*(Y' - 16) + 2.017*(Cb' - 128)
```

The output `R'G'B'` values are saturated to the range [0..255].

The fourth channel is created by setting channel values to the constant value `aval`.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if pSrc or pDst pointer is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with a zero or negative value.

YCbCrToBGR

Converts a YCbCr image to the BGR color model.

Syntax

```
IppStatus ippiYCbCrToBGR_8u_P3C3R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiYCbCrToBGR_8u_P3C4R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);
```

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	An array of pointers to ROI in each separate source color planes.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.
aval	Constant value to create the fourth channel.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the Y'Cb'Cr' image pSrc to the 24-bit gamma-corrected B'G'R' image pDst according to the same formulas as the function ippiYCbCrToRGB does. The output B'G'R' values are saturated to the range [0..255].

The fourth channel is created by setting channel values to the constant value aval.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if pSrc or pDst is NULL.
ippStsSizeErr	Indicates an error condition if roiSize.width is less than 4 or roiSize.height is less than 1.

YCbCrToBGR_709CSC

Converts a YCbCr image to the BGR image for ITU-R BT.709 CSC signal.

Syntax

```
IppStatus ippiYCbCrToBGR_709CSC_8u_P3C3R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiYCbCrToBGR_709CSC_8u_P3C4R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);
```

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	An array of pointers to ROI in separate planes of the source image.
srcStep	An array of distances in bytes between starts of consecutive lines in the source image planes.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.
aval	Constant value to create fourth channel.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts a planar Y'Cb'Cr' image pSrc to the three- or four-channel gamma-corrected B'G'R' image pDst for digital component video signals complied with the ITU-R BT.709 Recommendation ITU709 for computer systems consideration (CSC). The conversion is performed according to the following formulas Jack01:

```
R' = 1.164*(Y' - 16) + 1.793*(Cr' - 128)
G' = 1.164*(Y' - 16) - 0.534*(Cr' - 128) - 0.213*(Cb' - 128)
B' = 1.164*(Y' - 16) + 2.115*(Cb' - 128)
```

The output `R'G'B'` values are saturated to the range [0..255].

The fourth channel is created by setting channel values to the constant value `aval`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if `pSrc` or `pDst` is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize` has a field with a zero or negative value.

RGBToYCbCr422

Converts an RGB image to the YCbCr image with 4:2:2 sampling.

Syntax

Case 1: Operation on pixel-order data

```
IppStatus ippiRGBToYCbCr422_8u_C3C2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Case 2; Conversion from pixel-order to planar data

```
IppStatus ippiRGBToYCbCr422_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

Case 2: Conversion from planar to pixel-order data

```
IppStatus ippiRGBToYCbCr422_8u_P3C2R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Include Files

`ippcc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the source image ROI for pixel-order image. An array of pointers to ROI in each separate source color planes for planar images.

srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI for pixel-order image. An array of pointer to ROI in each separate planes for the planar destination image.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP). This function converts the gamma-corrected `R'G'B'` image `pSrc` to the Y'Cb'Cr' image `pDst` with 4:2:2 sampling (see Table "Pixel-Order Image Formats" and Table "Planar Image Formats" for more details). The conversion is performed according to the same formulas as the function ippiRGBToYCbCr does.

The converted buffer for pixel-order image has the reduced bit depth of a 16 bits per pixel, whereas the source buffer has 24 bit depth.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if `pSrc` or `pDst` is `NULL`.
ippStsSizeErr	Indicates an error condition if `roiSize` has a field with a zero or negative value.

YCbCr422ToRGB

Converts an YCbCr image with the 4:2:2 sampling to the RGB image.

Syntax

Case 1: Operation on pixel-order data

```
IppStatus ippiYCbCr422ToRGB_8u_C2C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiYCbCr422ToRGB_8u_C2C4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);
```

Case 2: Conversion from pixel-order to planar data

```
IppStatus ippiYCbCr422ToRGB_8u_C2P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
```

Case 3: Conversion from planar to pixel-order data

```
IppStatus ippiYCbCr422ToRGB_8u_P3C3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Pointer to the source image ROI for pixel-order image. An array of pointers to ROI in each separate source planes for planar images.
srcStep	Distance, in bytes, between the starting points of consecutive lines in the source image.
pDst	Pointer to the ROI in the destination pixel-order image. An array of pointers to ROI in each planes of the destination planar image.
dstStep	Distance, in bytes, between the starting points of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.
aval	Constant value to create the fourth channel.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the Y'Cb'Cr' image pSrc with the 4:2:2 sampling (see Table "Pixel-Order Image Formats" and Table "Planar Image Formats" for more details) to the gamma-corrected R'G'B' image pDst according to the same formulas as the function ippiYCbCrToRGB does. The output R'G'B' values are saturated to the range [0..255].

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if pSrc or pDst is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with a zero or negative value.

Example

The code example below demonstrates how to use the ippiYCbCr422ToRGB_8u_C2C4R function.

```
const int WIDTH  = 2;
const int HEIGHT = 2;


Ipp8u pSrc[WIDTH * HEIGHT * 2] =
{
```

```
    236,50,236,80,
    236,50,236,80,
};
Ipp8u pDstRGB[(WIDTH * HEIGHT) * 4];
int srcStep = WIDTH * 2, dstStep  = WIDTH * 4;
IppiSize roiSize = {WIDTH, HEIGHT};
Ipp8u alphaValue = 0xFF;
IppStatus status = ippiYCbCr422ToRGB_8u_C2C4R(pSrc, srcStep, pDstRGB, dstStep, roiSize,
alphaValue);
if ( status == ippStsNoErr)
    printf("\n *************  passed ****************\n");
else
    printf("\n *************  failed ****************\n");
```

RGBToYCrCb422

Converts 24-bit per pixel RGB image to 16-bit per pixel YCrCb image

Syntax

Case 1: Operation on pixel-order data

```
IppStatus ippiRGBToYCrCb422_8u_C3C2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int
dstStep, IppiSize roiSize);
```

Case 2: Conversion from planar to pixel-order data

```
IppStatus ippiRGBToYCrCb422_8u_P3C2R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst,
int dstStep, IppiSize roiSize);
```

Include Files

`ippcc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the source image ROI for pixel-order image. An array of pointers to ROI in each separate source color planes for planar images.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image.
`roiSize`	Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP). This function converts the gamma-corrected `R'G'B'` image `pSrc` to the Y'Cb'Cr' image `pDst` according to the same formulas as the function `ippiRGBToYCbCr` ippiRGBToYCbCr does. The difference is that `ippiRGBToYCrCb422` uses 4:2:2 sampling format for the converted image (see Table "Pixel-Order Image Formats" and Table "Planar Image Formats" for more details).

The converted buffer has the reduced bit depth of 16 bits per pixel, whereas the source buffer has 24 bit depth.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if `pSrc` or `pDst` is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize` has a field with a zero or negative value.

YCrCb422ToRGB, YCrCb422ToBGR

Convert 16-bit per pixel YCrCb image to 24 or 32-bit per pixel RGB or BGR image.

Syntax

Case 1: Operation on pixel-order data

```
IppStatus ippiYCrCb422ToRGB_8u_C2C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiYCrCb422ToRGB_8u_C2C4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);
```

```
IppStatus ippiYCrCb422ToBGR_8u_C2C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiYCrCb422ToBGR_8u_C2C4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);
```

Case 2: Conversion from pixel-order to planar data

```
IppStatus ippiYCrCb422ToRGB_8u_C2P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
```

Include Files

`ippcc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

pSrc	Pointer to the source image ROI for pixel-order image. An array of pointers to ROI in each separate source plane for planar images.
srcStep	Distance, in bytes, between the starting points of consecutive lines in the source image.
pDst	Pointer to the ROI in the destination pixel-order image. An array of pointers to ROI in each plane of the destination planar image.
dstStep	Distance, in bytes, between the starting points of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.
aval	Constant value to create the fourth channel.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the Y'Cr'Cb' image pSrc, packed in 4:2:2 sampling format (see Table "Pixel-Order Image Formats" and Table "Planar Image Formats" for more details) to the 24-bit gamma-corrected R'G'B' or B'G'R' image pDst according to the same formulas as the function ippiYCbCrToRGB does. The output R'G'B' values are saturated to the range [0..255]. Y'Cr'Cb' image with 4:2:2 sampling is also known as YVYU format.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if pSrc or pDst is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with a zero or negative value.

Example

The code example below demonstrates how to use the ippiYCrCb422ToRGB_8u_C2C4R function.

```
#define WIDTH 2
#define HEIGHT 2


Ipp8u pSrc[WIDTH * HEIGHT * 2] =
{
    236,50,236,80,
    236,50,236,80,
};
Ipp8u pDstRGB[(WIDTH * HEIGHT) * 4];
int srcStep = WIDTH * 2, dstStep  = WIDTH * 4;
IppiSize roiSize = {WIDTH, HEIGHT};
Ipp8u alphaValue = 0xFF;
IppStatus status = ippiYCrCb422ToRGB_8u_C2C4R(pSrc, srcStep, pDstRGB, dstStep, roiSize,
```

```
alphaValue);
if ( status == ippStsNoErr)
    printf("\n *************  passed ***************\n");
else
    printf("\n *************  failed ***************\n");
```

BGRToYCbCr422

Converts 24-bit per pixel BGR image to 16-bit per pixel YCbCr image.

Syntax

Case 1: Operation on pixel-order data

```
IppStatus ippiBGRToYCbCr422_8u_C3C2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiBGRToYCbCr422_8u_AC4C2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Case 2: Conversion from pixel-order to planar data

```
IppStatus ippiBGRToYCbCr422_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

```
IppStatus ippiBGRToYCbCr422_8u_AC4P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

Include Files

`ippcc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the source mage ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`pDst`	Pointer to the ROI in the destination pixel-order image. An array of pointers to ROI in each planes of the destination planar image.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination pixel-order image. An array of distances in bytes for each plane of the destination planar image.
`roiSize`	Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts a three- or four-channel gamma-corrected `B'G'R'` image `pSrc` to the two-channel or three-planes Y'Cb'Cr' image `pDst` according to the same formulas as the function ippiRGBToYCbCr does. The difference is that `ippiBGRToYCbCr422` uses the 4:2:2 sampling format (see Table "Pixel-Order Image Formats" and Table "Planar Image Formats" for more details).

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if `pSrc` or `pDst` is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize.width` is less than 2 or `roiSize.height` is less than 1.

YCbCr422ToBGR

Converts a YCbCr image with 4:2:2 sampling to the BGR image.

Syntax

Case 1: Operation on pixel-order data

```
IppStatus ippiYCbCr422ToBGR_8u_C2C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiYCbCr422ToBGR_8u_C2C4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);
```

Case 2: Conversion from planar to pixel-order data

```
IppStatus ippiYCbCr422ToBGR_8u_P3C3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Include Files

`ippcc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the source image ROI. An array of pointers to the ROI in each separate plane of the source planar image.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image. An array of such distances in bytes for each plane of the source planar image.
`pDst`	Pointer to the destination image ROI.

dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.
aval	Constant value to create the fourth channel.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the Y'Cb'Cr' image pSrc with 4:2:2 sampling (see Table "Pixel-Order Image Formats" and Table "Planar Image Formats" for more details) to the gamma-corrected B'G'R' image pDst according to the same formulas as the function ippiYCbCrToRGB does.

The output B'G'R' values are saturated to the range [0..255].

The fourth channel is created by setting channel values to the constant value aval.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if pSrc or pDst is NULL.
ippStsSizeErr	Indicates an error condition if roiSize.width is less than 2 or roiSize.height is less than 1.

YCbCr422ToGray

Converts an interlaced 4:2:2 YCbCr or YCrCb image to gray-scale extracting luminance (Y) component.

Syntax

```
IppStatus ippiYCbCr422ToGray_8u_C2C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance, in bytes, between the starting points of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.

dstStep	Distance, in bytes, between the starting points of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts an interlaced Y'Cb'Cr' or Y'Cr'Cb' image pSrc with the 4:2:2 sampling (see Table "Pixel-Order Image Formats" and Table "Planar Image Formats" for more details) to the gray-scale image pDst extracting luminance (Y) component.

Y'Cb'Cr' image with 4:2:2 sampling is also known as YUY2 format, and Y'Cr'Cb' as YVYU.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if pSrc or pDst is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with a zero or negative value.

Example

The code example below demonstrates how to use the ippiYCbCr422ToGray_8u_C2C1R function.

```
const int WIDTH  = 2;
const int HEIGHT = 2;


Ipp8u pSrc[WIDTH * HEIGHT * 2] = {
    190,70,191,80,
    200,71,201,81,
};
Ipp8u pDst[WIDTH * HEIGHT];
int srcStep = WIDTH * 2, dstStep  = WIDTH;
IppiSize roiSize = {WIDTH, HEIGHT};
IppStatus status = ippiYCbCr422ToGray_8u_C2C1R(pSrc, srcStep, pDst, dstStep, roiSize);
if ( status == ippStsNoErr)
    printf("PASS:\n%3d %3d\n%3d %3d\n", pDst[0], pDst[1], pDst[2], pDst[3]);
else
    printf("FAIL: status = %d\n", status);
```

Result:

```
PASS:
190 191
200 201
```

RGBToCbYCr422, RGBToCbYCr422Gamma

Convert 24-bit per pixel RGB image to 16-bit per pixel CbYCr image.

Syntax

```
IppStatus ippiRGBToCbYCr422_8u_C3C2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiRGBToCbYCr422Gamma_8u_C3C2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Include Files

`ippcc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image.
`roiSize`	Size of the source and destination ROI in pixels.

Description

These functions operate with ROI (see Regions of Interest in Intel IPP).

The function `ippiRGBToCbYCr422` converts the gamma-corrected `R'G'B'` image `pSrc` to the Cb'Y'Cr' image `pDst` according to the same formulas as the function ippiRGBToYCbCr does.

The function `ippiRGBToCbYCr422Gamma` performs gamma-correction of the source `RGB` image `pSrc` according to the same formula as the function ippiGammaFwd does, and then converts it to the Cb'Y'Cr' image `pDst` according to the same formulas as the function `ippiRGBToYCbCr` does.

The functions `ippiRGBToCbYCr422` and `ippiRGBToCbYCr422Gamma` use 4:2:2 sampling format for the converted image.

A `CbYCr` image has the following sequence of bytes: `Cb0Y0Cr0Y1`, `Cb1Y2Cr1Y3`, ….

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if `pSrc` or `pDst` is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize` has a field with a zero or negative value.

CbYCr422ToRGB

Converts 16-bit per pixel CbYCr image to 24-bit per pixel RGB image.

Syntax

```
IppStatus ippiCbYCr422ToRGB_8u_C2C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Include Files

`ippcc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image.
`roiSize`	Size of the source and destination ROI in pixels

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the Cb'Y'Cr' image `pSrc`, packed in the 4:2:2 sampling format, to the 24-bit gamma-corrected `R'G'B'` image `pDst` according to the same formulas as the function ippiYCbCrToRGB does.

A `CbYCr` image has the following sequence of bytes: `Cb0Y0Cr0Y1`, `Cb1Y2Cr1Y3`, … .

The output `R'G'B'` values are saturated to the range [0..255].

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if `pSrc` or `pDst` is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize` has a field with a zero or negative value.

BGRToCbYCr422

Converts 32-bit per pixel BGR image to 16-bit per pixel CbYCr image.

Syntax

```
IppStatus ippiBGRToCbYCr422_8u_AC4C2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Include Files

`ippcc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image.
`roiSize`	Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the four-channel gamma-corrected `B'G'R'` image `pSrc` to the two-channel Cb'Y'Cr' image `pDst` according to the same formulas as the function ippiRGBToYCbCr does. The function `ippiBGRToCbYCr422` uses 4:2:2 sampling format for the converted image. An alpha-channel information is lost.

An `CbYCr` image has the following sequence of bytes: `Cb0Y0Cr0Y1`, `Cb1Y2Cr1Y3`, … .

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if `pSrc` or `pDst` is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize` has a field with a zero or negative value.

BGRToCbYCr422_709HDTV

Converts BGR image to 16-bit per pixel CbYCr image for ITU-R BT.709 HDTV signal.

Syntax

```
IppStatus ippiBGRToCbYCr422_709HDTV_8u_C3C2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiBGRToCbYCr422_709HDTV_8u_AC4C2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Include Files

`ippcc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image.
`roiSize`	Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the three- or four-channel gamma-corrected `B'G'R'` image `pSrc` to the two-channel Cb'Y'Cr' image `pDst` for digital component video signals complied with the ITU-R BT.709 Recommendation ITU709 for high-definition TV (HDTV). The source image pixel values are in the range [16..235]. The conversion is performed according to the following formulas Jack01:

```
Y' = 0.213*R' + 0.715*G' + 0.072*B'
```

```
Cb' = -0.117*R' - 0.394*G' + 0.511*B' + 128
```

```
Cr' = 0.511*R' - 0.464*G' - 0.047*B' + 128
```

The values of `Y'` of the destination image are in the range [16..235], the values of `Cb'`, `Cr'` are in the range [16..240]. They should be saturated at the 1 and 254 levels.

The function `ippiBGRToCbYCr422_709HDTV` uses the 4:2:2 sampling format for the converted image. The alpha-channel information is lost.

A `CbYCr` image has the following sequence of bytes: `Cb0Y0Cr0Y1`, `Cb1Y2Cr1Y3`, … .

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr	Indicates an error condition if pSrc or pDst pointer is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with a zero or negative value.

CbYCr422ToBGR

Converts 16-bit per pixel CbYCr image to four channel BGR image.

Syntax

```
IppStatus ippiCbYCr422ToBGR_8u_C2C4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);
```

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.
aval	Constant value to create the fourth channel.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the Cb'Y'Cr' image pSrc, packed in 4:2:2 sampling format, to the four channel gamma-corrected B'G'R' image pDst according to the same formulas as the function ippiYCbCrToRGB does.

A CbYCr image has the following sequence of bytes: Cb0Y0Cr0Y1, Cb1Y2Cr1Y3, … .

The output B'G'R' values are saturated to the range [0..255].

The fourth channel is created by setting channel values to the constant value aval.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if pSrc or pDst is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with a zero or negative value.

CbYCr422ToBGR_709HDTV

Converts 16-bit per pixel CbYCr image to the BGR image for ITU-R BT.709 HDTV signal.

Syntax

```
IppStatus ippiCbYCr422ToBGR_709HDTV_8u_C2C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiCbYCr422ToBGR_709HDTV_8u_C2C4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);
```

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.
aval	Constant value to create the fourth channel.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the Cb'Y'Cr' image pSrc, packed in 4:2:2 sampling format, to the three- or four-channel gamma-corrected B'G'R' image pDst for digital component video signals complied with the ITU-R BT.709 Recommendation ITU709 for high-definition TV (HDTV). A source CbYCr image has the following

sequence of bytes: `Cb0Y0Cr0Y1`, `Cb1Y2Cr1Y3`, … . The values of `Y'` are in the range [16..235], the values of `Cb'`, `Cr'` are in the range [16..240]. The conversion is performed according to the following formulas Jack01:

```
R' = Y' + 1.540*(Cr' - 128)
G' = Y' - 0.459*(Cr' - 128) - 0.183*(Cb' - 128)
B' = Y' + 1.816*(Cb' - 128)
```

The destination image pixel values have a nominal range [16..235]. The resulting `R'G'B'` values should be saturated at the 0 and 255 levels.

The output `B'G'R'` values are saturated to the range [0..255].

The fourth channel is created by setting channel values to the constant value `aval`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if `pSrc` or `pDst` is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize` has a field with a zero or negative value.

RGBToYCbCr420

Converts an RGB image to the YCbCr color model; uses 4:2:0 sampling.

Syntax

```
IppStatus ippiRGBToYCbCr420_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
IppStatus ippiRGBToYCbCr420_8u_C3P2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep, IppiSize roiSize);
IppStatus ippiRGBToYCbCr420_8u_C4P2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep, IppiSize roiSize);
```

Include Files

`ippcc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.

pDst	An array of pointers to ROI in the separate planes of the destination image for three-plane image.
dstStep	Array of distances in bytes between starts of consecutive lines in each plane of the destination image for three-plane image.
pDstY	Pointer to ROI in the luminance plane of the destination image for two-plane image.
dstStep, dstYStep	Distance in bytes between starts of consecutive lines in the luminance plane for two-plane image.
pDstCbCr	Pointer to ROI in the interleaved chrominance plane of the destination image for two-plane image.
dstCbCrStep	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane for two-plane image.
roiSize	Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the gamma-corrected R'G'B' image pSrc to the Y'Cb'Cr' image according to the same formulas as the function ippiRGBToYCbCr does. The difference is that ippiRGBToYCbCr420 uses 4:2:0 sampling format for the converted image (see Table "Planar Image Formats" for more details).

roiSize.width should be multiple of 2, and roiSize.height should be multiple of 2 (for three-plane image) or 4 (for two-plane image). If not the function reduces their original values to the nearest multiples of 2 or 4 correspondingly, performs operation, and returns warning message.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if pSrc or pDst is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with value less than 2.
ippStsDoubleSize	Indicates a warning if roiSize.width is not a multiple of 2, or if roiSize.height is not a multiple of 2 (for three-plane image) or 4 (for two-plane image).

YCbCr420ToRGB, YCbCr420ToBGR

Convert a YCbCr image that has 4:2:0 sampling format to the RGB or BGR color model.

Syntax

```
IppStatus ippiYCbCr420ToRGB_8u_P3C3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst,
int dstStep, IppiSize roiSize);

IppStatus ippiYCbCr420ToRGB_8u_P2C3R(const Ipp8u* pSrcY, int srcYStep, const Ipp8u*
pSrcCbCr, int srcCbCrStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);

IppStatus ippiYCbCr420ToRGB_8u_P2C4R(const Ipp8u* pSrcY, int srcYStep, const Ipp8u*
pSrcCbCr, int srcCbCrStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);

IppStatus ippiYCbCr420ToBGR_8u_P2C3R(const Ipp8u* pSrcY, int srcYStep, const Ipp8u*
pSrcCbCr, int srcCbCrStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);

IppStatus ippiYCbCr420ToBGR_8u_P2C4R(const Ipp8u* pSrcY, int srcYStep, const Ipp8u*
pSrcCbCr, int srcCbCrStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);
```

Include Files

`ippcc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	An array of pointers to ROI in separate planes of the three-plane source image.
`srcStep`	An array of distances in bytes between starts of consecutive lines in each plane of the three-plane source image.
`pSrcY`	Pointer to ROI in the luminance plane of the two-plane source image.
`srcYStep`	Distance in bytes between starts of consecutive lines in the luminance plane of the two-plane source image.
`pSrcCbCr`	Pointer to ROI in the interleaved chrominance plane of the two-plane source image.
`srcCbCrStep`	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the two-plane source image.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image.
`roiSize`	Size of the source and destination ROI in pixels.
`aval`	Constant value to create the fourth channel.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the Y'Cb'Cr' image `pSrc` to the gamma-corrected `R'G'B'` or `B'G'R'` image `pDst` according to the same formulas as the function ippiYCbCrToRGB does. The difference is that the `ippiYCbCr420ToRGB` and `ippiYCbCr420ToBGR` functions use the input data in the 4:2:0 sampling format, in which the number of `Cb` and `Cr` samples is reduced by half in both vertical and horizontal directions (see Table "Planar Image Formats" for more details). Two-plane `Y'Cb'Cr'` image with 4:2:0 sampling is also known as NV12 format.

The value of `roiSize.width` and `roiSize.height` must be a multiple of 2. Otherwise, the function reduces original values to the nearest multiples of 2, performs operation, and returns a warning message.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if `pSrc` or `pDst` is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize` has a field with value less than 2.
`ippStsDoubleSize`	Indicates a warning if `roiSize` has a field that is not a multiple of 2.

Example

The code example below demonstrates how to use the `ippiYCbCr420ToRGB_8u_P2C4R` function.

```
#define WIDTH 4
#define HEIGHT 4


Ipp8u pSrcY[WIDTH * HEIGHT] =
{
    236,236,236,236,
    236,236,236,236,
    236,236,236,236,
    236,236,236,236
};
Ipp8u pSrcCbCr[WIDTH * HEIGHT / 2] =
{
    128,128,128,128,
    128,128,128,128
};
Ipp8u pDstRGB[(WIDTH * HEIGHT) * 4];
int srcYStep = WIDTH, srcCbCrStep = WIDTH, dstStep  = WIDTH * 4;
IppiSize roiSize = {WIDTH, HEIGHT};
Ipp8u alphaValue = 0xFF;
IppStatus status = ippiYCbCr420ToRGB_8u_P2C4R(pSrcY, srcYStep, pSrcCbCr, srcCbCrStep, pDstRGB,
dstStep, roiSize, alphaValue);
if ( status == ippStsNoErr)
    printf("\n *************  passed ****************\n");
else
    printf("\n *************  failed ****************\n");
```

RGBToYCrCb420

Converts an RGB image to the YCrCb image with 4:2:0 sampling format.

Syntax

```
IppStatus ippiRGBToYCrCb420_8u_AC4P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3],
int dstStep[3], IppiSize roiSize);
```

Include Files

`ippcc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`pDst`	Array of pointers to ROI in separate planes of the destination image.
`dstStep`	Array of distances in bytes between starts of consecutive lines in the destination image planes.
`roiSize`	Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts a four-channel gamma-corrected R'G'B' image `pSrc` to the planar Y'Cr'Cb' image `pDst` with the 4:2:0 sampling (see Table "Planar Image Formats" for more details). The conversion is performed according to the same formulas as the function ippiRGBToYCbCr does.

`roiSize.width` and `roiSize.height` should be multiples of 2. If not the function reduces their original values to the nearest multiples of 2, performs the operation, and returns a warning message.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if `pSrc` or `pDst` is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize` has a field with a zero or negative value.
`ippStsDoubleSize`	Indicates a warning if `roiSize` has a field that is not a multiple of 2.

YCrCb420ToRGB, YCrCb420ToBGR

Convert a YCrCb image with the 4:2:0 sampling to the RGB or BGR image.

Syntax

```
IppStatus ippiYCrCb420ToRGB_8u_P3C4R (const Ipp8u* pSrc[3],int srcStep[3], Ipp8u* pDst,
int dstStep, IppiSize roiSize, Ipp8u aval);

IppStatus ippiYCrCb420ToRGB_8u_P2C4R (const Ipp8u* pSrcY, int srcYStep, const Ipp8u*
pSrcCrCb, int srcCrCbStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);

IppStatus ippiYCrCb420ToRGB_8u_P2C3R (const Ipp8u* pSrcY, int srcYStep, const Ipp8u*
pSrcCrCb, int srcCrCbStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);

IppStatus ippiYCrCb420ToBGR_8u_P2C3R (const Ipp8u* pSrcY, int srcYStep, const Ipp8u*
pSrcCrCb, int srcCrCbStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);

IppStatus ippiYCrCb420ToBGR_8u_P2C4R (const Ipp8u* pSrcY, int srcYStep, const Ipp8u*
pSrcCrCb, int srcCrCbStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);

IppStatus ippiYCrCb420ToBGR_Filter_8u_P3C4R (const Ipp8u* pSrc[3], int srcStep[3],
Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);
```

Include Files

`ippcc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	An array of pointers to ROI in separate planes of the three-plane source image.
`srcStep`	An array of distances, in bytes, between the starting points of consecutive lines in each plane of the three-plane source image.
`pSrcY`	Pointer to ROI in the luminance plane of the two-plane source image.
`srcYStep`	Distance, in bytes, between the starting points of consecutive lines in the luminance plane of the two-plane source image.
`pSrcCrCb`	Pointer to ROI in the interleaved plane chrominance plane of the two-plane source image.
`srcCrCbStep`	Distance, in bytes, between the starting points of consecutive lines in the interleaved chrominance plane of the two-plane source image.
`pDst`	Pointer to the destination image ROI.

dstStep	Distance, in bytes, between the starting points of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI, in pixels.
aval	Constant value to create the fourth channel.

Description

This function operates with ROI.

This function converts the Y'Cr'Cb' image pSrc to the gamma-corrected R'G'B' or B'G'R' image pDst according to the same formulas as the ippiippiYCbCrToRGB function does. The difference is that the ippiYCrCb420ToRGB and ippiYCrCb420ToBGR functions use the source data in the 4:2:0 sampling format, in which the number of Cb and Cr samples is reduced by half in both vertical and horizontal directions (see Table "Planar Image Formats" for more details). Two-plane Y'Cr'Cb image with 4:2:0 sampling is also known as NV21 format.

The value of roiSize.width and roiSize.height must be a multiple of 2. Otherwise, the function reduces original values of roiSize.width and roiSize.height to the nearest multiples of 2, performs operation, and returns a warning.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when pSrc or pDst is NULL.
ippStsSizeErr	Indicates an error when roiSize has a field with a value less than 2.
ippStsDoubleSize	Indicates a warning if roiSize has a field that is not a multiple of 2.

Example

The code example below demonstrates how to use the ippiYCrCb420ToRGB_8u_P2C4R function.

```
static void sampleNV21ToRGBA()
{
    Ipp8u pY[4*4]=
    {
        236,236,236,236,
        236,236,236,236,
        236,236,236,236,
        236,236,236,236
    };
    Ipp8u pCbCr[4*2]=
    {
        128,128,128,128,
        128,128,128,128
    };
    Ipp8u pRGB[(4*4)*4];
    int YStep = 4, CbCrStep = 4, rgbStep  = 4*4;
    IppiSize roiSize = {4,4};
    Ipp8u alpha = 0xFF;
    IppStatus status = ippiYCrCb420ToRGB_8u_P2C4R(pY, YStep, pCbCr, CbCrStep, pRGB, rgbStep,
```

```
roiSize, alpha);
    if ( status == ippStsNoErr)
        printf("\n *************  passed ***************\n");
    else
        printf("\n *************  failed ***************\t");
}
```

See Also

Regions of Interest in Intel IPP

BGRToYCbCr420

Converts a BGR image to the YCbCr image with 4:2:0 sampling format.

Syntax

```
IppStatus ippiBGRToYCbCr420_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3],
int dstStep[3], IppiSize roiSize);
```

```
IppStatus ippiBGRToYCbCr420_8u_C3P2R(const Ipp8u* pRGB, int rgbStep, Ipp8u* pY, int
yStep, Ipp8u* pCbCr, int cbCrStep, IppiSize roiSize);
```

```
IppStatus ippiBGRToYCbCr420_8u_AC4P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3],
int dstStep[3], IppiSize roiSize);
```

```
IppStatus ippiBGRToYCbCr420_8u_AC4P2R(const Ipp8u* pRGB, int rgbStep, Ipp8u* pY, int
yStep, Ipp8u* pCbCr, int cbCrStep, IppiSize roiSize);
```

Include Files

`ippcc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`, `pRGB`	Pointer to the source image ROI.
`pY`	Pointer to the image Y plane.
`srcStep`, `rgbStep`	Distance, in bytes, between the starting points of consecutive lines in the source image.
`yStep`	Distance, in bytes, between the starting points of consecutive lines in the image Y plane.
`pDst`, `pCbCr`	An array of pointers to ROI in separate planes of the destination image.
`dstStep`, `cbCrStep`	An array of distances, in bytes, between the starting points of consecutive lines in the destination image planes.

roiSize	Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts a three- or four-channel gamma-corrected B'G'R' image to the planar Y'Cb'Cr' image according to the same formulas as the function ippiRGBToYCbCr does. The difference is that ippiBGRToYCbCr420 uses 4:2:0 sampling format (see Table "Planar Image Formats" for more details).

roiSize.width and roiSize.height should be multiples of 2. If not the function reduces their original values to the nearest multiples of 2, performs operation, and returns warning message.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if pSrc(pRGB) or pDst (pCbCr) is NULL.
ippStsSizeErr	Indicates an error condition if roiSize.width is less than 2 or roiSize.height is less than 2.
ippStsDoubleSize	Indicates a warning if roiSize has a field that is not a multiple of 2.

BGRToYCbCr420_709CSC

Converts a BGR image to the YCbCr image with 4:2:0 sampling for ITU-R BT.709 CSC signal.

Syntax

```
IppStatus ippiBGRToYCbCr420_709CSC_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u*
pDst[3], int dstStep[3], IppiSize roiSize);
```

```
IppStatus ippiBGRToYCbCr420_709CSC_8u_C3P2R(const Ipp8u* pSrc, int srcStep, Ipp8u*
pDstY, int dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep, IppiSize roiSize);
```

```
IppStatus ippiBGRToYCbCr420_709CSC_8u_AC4P3R(const Ipp8u* pSrc, int srcStep, Ipp8u*
pDst[3], int dstStep[3], IppiSize roiSize);
```

```
IppStatus ippiBGRToYCbCr420_709CSC_8u_AC4P2R(const Ipp8u* pSrc, int srcStep, Ipp8u*
pDstY, int dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep, IppiSize roiSize);
```

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst, pDstY, pDstCbCr	An array of pointers to ROI in separate planes of the destination image.
dstStep, dstYStep, dstCbCrStep	An array of distances in bytes between starts of consecutive lines in the destination image planes.
roiSize	Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts a three- or four-channel gamma-corrected B'G'R' image pSrc to the planar Y'Cb'Cr' image pDst for digital component video signals complied with the ITU-R BT.709 Recommendation ITU709 for computer systems consideration (CSC). The source image pixel values are in the range [0..255]. The conversion is performed according to the following formulas Jack01:

```
Y' = 0.183*R' + 0.614*G' + 0.062*B' + 16
Cb' = -0.101*R' - 0.338*G' + 0.439*B' + 128
Cr' = 0.439*R' - 0.399*G' - 0.040*B' + 128
```

The destination image pDst has 4:2:0 sampling format (see Table "Planar Image Formats" for more details).

The values of roiSize.width and roiSize.height should be multiples of 2. Otherwise, the function reduces their original values to the nearest multiples of 2, performs operation, and returns a warning message.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if pSrc or pDst is NULL.
ippStsSizeErr	Indicates an error condition if roiSize.width is less than 2 or roiSize.height is less than 2.
ippStsDoubleSize	Indicates a warning if roiSize has a field that is not a multiple of 2.

BGRToYCbCr420_709HDTV

Converts a BGR image to the YCbCr image with 4:2:0 sampling for ITU-R BT.709 HDTV signal.

Syntax

```
IppStatus ippiBGRToYCbCr420_709HDTV_8u_AC4P3R(const Ipp8u* pSrc, int srcStep, Ipp8u*
pDst[3], int dstStep[3], IppiSize roiSize);
```

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	An array of pointers to ROI in separate planes of the destination image.
dstStep	An array of distances in bytes between starts of consecutive lines in the destination image planes.
roiSize	Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts a four-channel gamma-corrected B'G'R' image pSrc to the planar Y'Cb'Cr' image pDst for digital component video signals complied with the ITU-R BT.709 Recommendation ITU709 for high-definition TV (HDTV). The source image pixel values are in the range [16..235]. The conversion is performed according to the following formulas Jack01:

```
Y' = 0.213*R' + 0.715*G' + 0.072*B'
```

```
Cb' = -0.117*R' - 0.394*G' + 0.511*B' + 128
```

```
Cr' = 0.511*R' - 0.464*G' - 0.047*B' + 128
```

The values of Y' of the destination image are in the range [16..235], the values of Cb', Cr' are in the range [16..240]. They should be saturated at the 1 and 254 levels.

The destination image pDst has the 4:2:0 sampling format (see Table "Planar Image Formats" for more details).

The values of roiSize.width and roiSize.height should be multiples of 2. Otherwise, the function reduces their original values to the nearest multiples of 2, performs the operation, and returns a warning message.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if pSrc or pDst is NULL.
ippStsSizeErr	Indicates an error condition if roiSize.width is less than 2 or roiSize.height is less than 2.

ippStsDoubleSize	Indicates a warning if roiSize has a field that is not a multiple of 2.

BGRToYCrCb420_709CSC

Converts a BGR image to the YCrCb image with 4:2:0 sampling for ITU-R BT.709 CSC signal.

Syntax

```
IppStatus ippiBGRToYCrCb420_709CSC_8u_AC4P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

```
IppStatus ippiBGRToYCrCb420_709CSC_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	An array of pointers to ROI in separate planes of the destination image.
dstStep	An array of distances in bytes between starts of consecutive lines in the destination image planes.
roiSize	Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts a three- or four-channel gamma-corrected B'G'R' image pSrc to the planar Y'Cr'Cb' image pDst for digital component video signals complied with the ITU-R BT.709 Recommendation ITU709 for computer systems consideration (CSC). The source image pixel values are in the range [0..255]. The conversion is performed according to the following formulas Jack01:

```
Y' = 0.183*R' + 0.614*G' + 0.062*B' + 16
```

```
Cb' = -0.101*R' - 0.338*G' + 0.439*B' + 128
```

```
Cr' = 0.439*R' - 0.399*G' - 0.040*B' + 128
```

The destination image pDst has 4:2:0 sampling format (see Table "Planar Image Formats" for more details).

The values of roiSize.width and roiSize.height should be multiples of 2. Otherwise, the function reduces their original values to the nearest multiples of 2, performs the operation, and returns a warning message.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if pSrc or pDst is NULL.
ippStsSizeErr	Indicates an error condition if roiSize.width is less than 2 or roiSize.height is less than 2.
ippStsDoubleSize	Indicates a warning if roiSize has a field that is not a multiple of 2.

YCbCr420ToBGR

Converts a YCbCr image with the 4:2:0 sampling to the BGR image.

Syntax

```
IppStatus ippiYCbCr420ToBGR_8u_P3C3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiYCbCr420ToBGR_8u_P3C4R(const Ipp8u* pSrc[3], int srcStep[3],Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);
```

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	An array of pointers to ROI in separate planes of the source image.
srcStep	An array of distances in bytes between starts of consecutive lines in the source image planes.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.
aval	Constant value to create fourth channel.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the Y'Cb'Cr' image pSrc with the 4:2:0 sampling (see Table "Planar Image Formats" for more details) to the gamma-corrected three- or four-channel B'G'R' image pDst. The conversion is performed according to the same formulas as the function ippiYCbCrToRGB does.

Fourth channel is created by setting channel values to the constant value aval.

The values of roiSize.width and roiSize.height should be multiples of 2. If not the function reduces their original values to the nearest multiples of 2, performs operation, and returns warning message.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if pSrc or pDst is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with a zero or negative value.
ippStsDoubleSize	Indicates a warning if roiSize has a field that is not a multiple of 2.

YCbCr420ToBGR_709CSC

Converts a YCbCr image with 4:2:0 sampling to the BGR image for ITU-R BT.709 CSC signal.

Syntax

```
IppStatus ippiYCbCr420ToBGR_709CSC_8u_P3C3R(const Ipp8u* pSrc[3], int srcStep[3],
Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	An array of pointers to ROI in separate planes of the source image.
srcStep	An array of distances in bytes between starts of consecutive lines in the source image planes.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts a planar Y'Cb'Cr' image `pSrc` to the three-channel gamma-corrected B'G'R' image `pDst` for digital component video signals complied with the ITU-R BT.709 Recommendation ITU709 for computer systems consideration (CSC). The conversion is performed according to the following formulas Jack01:

```
R' = 1.164*(Y' - 16) + 1.793*(Cr' - 128)
G' = 1.164*(Y' - 16) - 0.534*(Cr' - 128) - 0.213*(Cb' - 128)
B' = 1.164*(Y' - 16) + 2.115*(Cb' - 128)
```

The output `R'G'B'` values are saturated to the range [0..255]. The source image `pDst` has the 4:2:0 sampling format (see Table "Planar Image Formats" for more details).

The values of `roiSize.width` and `roiSize.height` should be multiples of 2. Otherwise, the function reduces their original values to the nearest multiples of 2, performs the operation, and returns a warning message.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if `pSrc` or `pDst` is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize` has a field with a zero or negative value.
`ippStsDoubleSize`	Indicates a warning if `roiSize` has a field that is not a multiple of 2.

YCbCr420ToBGR_709HDTV

Converts a YCbCr image with 4:2:0 sampling to the BGR image for ITU-R BT.709 HDTV signal.

Syntax

```
IppStatus ippiYCbCr420ToBGR_709HDTV_8u_P3C4R(const Ipp8u* pSrc[3], int srcStep[3],
Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);
```

Include Files

`ippcc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

pSrc	An array of pointers to ROI in separate planes of the source image.
srcStep	An array of distances in bytes between starts of consecutive lines in the source image planes.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.
aval	Constant value to create fourth channel.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts a planar Y'Cb'Cr' image pSrc to the four-channel gamma-corrected B'G'R' image pDst for digital component video signals complied with the ITU-R BT.709 Recommendation ITU709 for high-definition TV (HDTV). The values of Y' are in the range [16..235], the values of Cb', Cr' are in the range [16..240]. The conversion is performed according to the following formulas Jack01:

```
R' = Y' + 1.540*(Cr' - 128)
G' = Y' - 0.459*(Cr' - 128) - 0.183*(Cb' - 128)
B' = Y' + 1.816*(Cb' - 128)
```

The destination image pixel values have a nominal range [16..235]. The resulting R'G'B' values should be saturated at the 0 and 255 levels.

The source image pDst has the 4:2:0 sampling format (see Table "Planar Image Formats" for more details).

The values of roiSize.width and roiSize.height should be multiples of 2. Otherwise the function reduces their original values to the nearest multiples of 2, performs operation, and returns a warning message.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if pSrc or pDst is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with a zero or negative value.
ippStsDoubleSize	Indicates a warning if roiSize has a field that is not a multiple of 2.

BGRToYCrCb420

Converts a BGR image to the YCrCb image with 4:2:0 sampling format.

Syntax

```
IppStatus ippiBGRToYCrCb420_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3],
int dstStep[3], IppiSize roiSize);
```

```
IppStatus ippiBGRToYCrCb420_8u_AC4P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3],
int dstStep[3], IppiSize roiSize);
```

Include Files

`ippcc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`pDst`	An array of pointers to ROI in separate planes of the destination image.
`dstStep`	An array of distances in bytes between starts of consecutive lines in the destination image planes.
`roiSize`	Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts a three- or four-channel gamma-corrected `B'G'R'` image `pSrc` to the planar Y'Cr'Cb' image `pDst` according to the same formulas as the function ippiRGBToYCbCr does. The destination image `pDst` has the 4:2:0 sampling format and the following order of pointers: `Y - plane`, `Cr - plane`, `Cb - plane` (see Table "Planar Image Formats" for more details).

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if `pSrc` or `pDst` is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize.width` is less than 2 or `roiSize.height` is less than 2.

BGRToYCbCr411

Converts a BGR image to the YCbCr planar image that has a 4:1:1 sampling format.

Syntax

```
IppStatus ippiBGRToYCbCr411_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3],
int dstStep[3], IppiSize roiSize);
```

```
IppStatus ippiBGRToYCbCr411_8u_AC4P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3],
int dstStep[3], IppiSize roiSize);
```

Include Files

`ippcc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`pDst`	An array of pointers to ROI in separate planes of the destination image.
`dstStep`	An array of distances in bytes between starts of consecutive lines in the destination image planes.
`roiSize`	Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts a three- or four-channel gamma-corrected B'G'R' image `pSrc` to the planar Y'Cb'Cr' image `pDst` according to the same formulas as the function ippiRGBToYCbCr does. The difference is that `ippiBGRToYCbCr411` uses the 4:1:1 sampling format (see Table "Planar Image Formats" for more details).

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if `pSrc` or `pDst` is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize.width` is less than 4 or `roiSize.height` is less than 1.

YCbCr411ToBGR

Converts a YCbCr image that has 4:1:1 sampling format to the RGB color model.

Syntax

```
IppStatus ippiYCbCr411ToBGR_8u_P3C3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiYCbCr411ToBGR_8u_P3C4R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp8u aval);
```

Include Files

`ippcc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	An array of pointers to ROI in separate planes of the source image.
`srcStep`	An array of distances in bytes between starts of consecutive lines in the source image planes.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image.
`roiSize`	Size of the source and destination ROI in pixels.
`aval`	Constant value to create fourth channel.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the planar Y'Cb'Cr' image `pSrc` to the three- or four-channel image `pDst`.To compute gamma-corrected `R'G'B'(B'G'R')` channel values the above formulas are used. The difference is that `ippiYCbCr411ToBGR` uses the input data in the 4:1:1 sampling format (see Table "Planar Image Formats" for more details). Fourth channel is created by setting channel values to the constant value `aval`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if `pSrc` or `pDst` is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize` has a field with a zero or negative value.

RGBToXYZ

Converts an RGB image to the XYZ color model.

Syntax

```
IppStatus ippiRGBToXYZ_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

`8u_C3R`	`16u_C3R`	`16s_C3R`	`32f_C3R`
`8u_AC4R`	`16u_AC4R`	`16s_AC4R`	`32f_AC4R`

Include Files

`ippcc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image.
`roiSize`	Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the `RGB` image `pSrc` to the `CIE XYZ` image `pDst` according to the following basic equations:

```
X = 0.412453*R + 0.35758 *G + 0.180423*B
Y = 0.212671*R + 0.71516 *G + 0.072169*B
Z = 0.019334*R + 0.119193*G + 0.950227*B
```

The equations above are given on the assumption that R,G, and B values are normalized to the range [0..1]. In case of the floating-point data type, the input RGB values must already be in the range [0..1]. For integer data types, normalization is done by the conversion function internally.

The computed `XYZ` values are saturated if they fall out of range [0..1].

In case of integer function flavors, these values are then scaled to the full range of the destination data type (see Table “Image Data Types and Ranges”).

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.

ippStsNullPtrErr	Indicates an error condition if pSrc or pDst is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with a zero or negative value.

XYZToRGB

Converts an XYZ image to the RGB color model.

Syntax

```
IppStatus ippiXYZToRGB_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the CIE XYZ image pSrc to the RGB image pDst according to the following basic equations:

```
R = 3.240479 * X - 1.53715 * Y - 0.498535 * Z
G = -0.969256 * X + 1.875991 * Y + 0.041556 * Z
B = 0.055648 * X - 0.204043 * Y + 1.057311 * Z
```

The equations above are given on the assumption that X, Y, and Z values are in the range [0..1]. In case of the floating-po int data type, the input XYZ values must already be in the range [0..1]. For integer data types, normalization is done by the conversion function internally.

The computed RGB values are saturated if they fall out of range [0..1].

In case of integer function flavors, these values are then scaled to the full range of the destination data type (see Table "Image Data Types and Ranges").

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if pSrc or pDst is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with a zero or negative value.

RGBToLUV, BGRToLUV

Converts an RGB or BGR image to the LUV color model.

Syntax

Case 1: RGB to LUV

```
IppStatus ippiRGBToLUV_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

Case 2: BGR to LUV

```
IppStatus ippiBGRToLUV_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C3R	32f_C3R

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the RGB or BGR image pSrc to the CIE LUV CIE LUV image pDst in two steps. First, the conversion is done into CIE XYZ format, using equations defined for the ippiRGBToXYZ function. After that, conversion to LUV image is performed in accordance with the following equations:

$L = 116. * (Y/Y_n)^{1/3.} - 16``.``$

$U = 13. * L * (u - u_n)$

$V = 13. * L * (v - v_n)$

where

$u = 4.*X / (X + 15.*Y + 3.*Z)$

$v = 9.*Y / (X + 15.*Y + 3.*Z)$

$u_n = 0``.197839``$

$v_n = 0``.468342``$

The computed values of the L component are in the range [0..100], U component in the range [-134..220], and V component in the range [-140..122].

The equations above are given on the assumption that R, G, and B values are normalized to the range [0..1]. In case of the floating-point data type, the input RGB values must already be in the range [0..1]. For integer data types, normalization is done by the conversion function internally.

In case of 8u data type, the computed L, U, and V values are quantized and converted to fit in the range [0..IPP_MAX_8U] as follows:

```
L = L * IPP_MAX_8U / 100.
U = (U + 134.) * IPP_MAX_8U / 354.
V = (V + 140.) * IPP_MAX_8U / 262.
```

In case of 16u data type, the computed L, U, and V values are quantized and converted to fit in the range [0..IPP_MAX_16U] as follows:

```
L = L * IPP_MAX_16U / 100.
U = (U + 134.) * IPP_MAX_16U / 354.
V = (V + 140.) * IPP_MAX_16U / 262.
```

In case of 16s data type, the computed L, U, and V values are quantized and converted to fit in the range [IPP_MIN_16S..IPP_MAX_16S] as follows:

```
L = L * IPP_MAX_16U / 100. + IPP_MIN_16S
U = (U + 134.) * IPP_MAX_16U / 354. + IPP_MIN_16S
V = (V + 140.) * IPP_MAX_16U / 262. + IPP_MIN_16S
```

For `32f` data type, no further conversion is done and `L`, `U`, and `V` components remain in the ranges [0..100], [-134..220], and [-140..122], respectively.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if `pSrc` or `pDst` is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize` has a field with a zero or negative value.

LUVToRGB, LUVToBGR

Converts a LUV image to the RGB or BGR color model.

Syntax

Case 1: LUV to RGB

```
IppStatus ippiLUVToRGB_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

`8u_C3R`	`16u_C3R`	`16s_C3R`	`32f_C3R`
`8u_AC4R`	`16u_AC4R`	`16s_AC4R`	`32f_AC4R`

Case 2: LUV to BGR

```
IppStatus ippiLUVToBGR_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

`8u_C3R`	`32f_C3R`

Include Files

`ippcc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the source image ROI.

srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the CIE LUV image pSrc to the RGB or BGR image pDst in two steps. First, the conversion is carried out into CIE XYZ format.

To accomplish it, LUV components are transformed back into their original range. This is done for different data types in the following way.

For 8u data type:

```
L = L * 100./ IPP_MAX_8U
U = (U * 354./ IPP_MAX_8U) - 134.
V = (V * 262./ IPP_MAX_8U) - 140.
```

For 16u data type:

```
L = L * 100./ IPP_MAX_16U
U = (U * 354./ IPP_MAX_16U) - 134.
V = (V * 262./ IPP_MAX_16U) - 140.
```

For 16s data type:

```
L = (L - IPP_MIN_16S)* 100./ IPP_MAX_16U
U = ((U - IPP_MIN_16S)* 354./ IPP_MAX_16U) - 134.
V = ((V - IPP_MIN_16S) * 262./ IPP_MAX_16U) - 140.
```

After that, conversion to XYZ format takes place as follows:

$Y = Y_n * ((L + 16.) / 116.)**3.$

```
X = -9.* Y * u / ((u - 4.)* v - u* v )
Z = (9.* Y - 15*v*Y - v*X) / 3. * v
```

where

$u = U / (13.* L) + u_n$

$v = V / (13.* L) + v_n$

and

$u_n = 4``.*x``_n / (-2.*x_n + 12``.*y``_n + 3``.)``$

$v_n = 9``.*y``_n / (-2.*x_n + 12``.*y``_n + 3``.)``$

Here $x_n = 0.312713$, $y_n = 0.329016$ are the CIE chromaticity coordinates of the D65 white point, and $Y_n = 1.0$ is the luminance of the D65 white point.

After this intermediate conversion is done, the obtained `XYZ` image is then converted to the destination `RGB` or `BGR` format using equations defined for the ippiXYZToRGB function.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if `pSrc` or `pDst` is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize` has a field with a zero or negative value.

BGRToLab, RGBToLab

Converts a BGR or RGB image to the Lab color model.

Syntax

Case 1: BGR to Lab

```
IppStatus ippiBGRToLab_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep,
IppiSize roiSize);
```

```
IppStatus ippiBGRToLab_8u16u_C3R(const Ipp8u* pSrc, int srcStep, Ipp16u* pDst, int
dstStep, IppiSize roiSize);
```

```
IppStatus ippiBGRToLab_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int
dstStep, IppiSize roiSize);
```

Case 2: RGB to Lab

```
IppStatus ippiRGBToLab_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep,
IppiSize roiSize);
```

```
IppStatus ippiRGBToLab_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int
dstStep, IppiSize roiSize);
```

```
IppStatus ippiRGBToLab_32f_P3R(const Ipp32f* pSrc[3], int srcStep[3], Ipp32f* pDst[3],
int dstStep[3], IppiSize roiSize);
```

```
IppStatus ippiRGBToLab_64f_P3R(const Ipp64f* pSrc[3], int srcStep[3], Ipp64f* pDst[3],
int dstStep[3], IppiSize roiSize);
```

Case 3: RGB to Lab with platform-aware functions

```
IppStatus ippiRGBToLab_32f_P3R_L(const Ipp32f* pSrc[3], IppSizeL srcStep[3], Ipp32f*
pDst[3], IppSizeL dstStep[3], IppiSizeL roiSize);
```

```
IppStatus ippiRGBToLab_64f_P3R_L(const Ipp64f* pSrc[3], IppSizeL srcStep[3], Ipp64f*
pDst[3], IppSizeL dstStep[3], IppiSizeL roiSize);
```

Case 4: RGB to Lab with TL functions based on the Platform Aware API

```
IppStatus ippiRGBToLab_32f_P3R_LT(const Ipp32f* pSrc[3], IppSizeL srcStep[3], Ipp32f*
pDst[3], IppSizeL dstStep[3], IppiSizeL roiSize);
```

```
IppStatus ippiRGBToLab_64f_P3R_LT(const Ipp64f* pSrc[3], IppSizeL srcStep[3], Ipp64f*
pDst[3], IppSizeL dstStep[3], IppiSizeL roiSize);
```

Case 5: RGB to Lab with TL functions based on the Classic API

```
IppStatus ippiRGBToLab_32f_P3R_T(const Ipp32f* pSrc[3], int srcStep[3], Ipp32f*
pDst[3], int dstStep[3], IppiSize roiSize);

IppStatus ippiRGBToLab_64f_P3R_T(const Ipp64f* pSrc[3], int srcStep[3], Ipp64f*
pDst[3], int dstStep[3], IppiSize roiSize);
```

Include Files

`ippcc.h`

`ippcc_l.h`

`ippcc_tl.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`, `pSrc[3]`	Pointer to the source image ROI.
`srcStep`, `srcStep[3]`	Distance in bytes between starts of consecutive lines in the source image.
`pDst`, `pDst[3]`	Pointer to the destination image ROI.
`dstStep`, `dstStep[3]`	Distance in bytes between starts of consecutive lines in the destination image.
`roiSize`	Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the `BGR` or `RGB` image `pSrc` to the CIE Lab image `pDst`, and vice versa. Conversion to `Lab` consists of two steps. First, the conversion is done into CIE XYZ format, using equations defined for the function ippiRGBToXYZ. After that, conversion to the `Lab` image is performed in accordance with the following equations:

$L = 116. * (Y/Y_n)^{1/3.} - 16$ for $Y/Y_n > 0.008856$

$L = 903.3 * (Y/Y_n)^{1/3.}$ for $Y/Y_n \le 0.008856$

$a = 500. * [f(X/X_n) - f(Y/Y_n)]$

$b = 200. * [f(Y/Y_n) - f(Z/Z_n)]$

where

$$f(t) = \begin{cases} t^{1/3}, & t > (6/29)^3 \\ \frac{1}{3}\left(\frac{29}{6}\right)^2 t + \frac{4}{29} \end{cases}$$

Here $Y_n = 1.0$, $X_n = 0.950455$, $Z_n = 1.088753$ for the D65 white point with the CIE chromaticity coordinates $x_n = 0.312713$, $y_n = 0.329016$.

The equations above are given on the assumption that initial B, G, R values are normalized to the range [0..1]. The computed values of the `L` component are in the range [0..100], `a` and `b` component values are in the range [-128..127].

These values are quantized and scaled to the 8-bit range of 0 to 255 for `8u_C3` flavors:

```
L = L * 255./100.
a = (a + 128.)
b = (a + 128.)
```

or to the 16-bit range of 0 to 65535 for `ippiBGRToLab_8u16u_C3R`:

```
L = L * 65535./100.
a = (a + 128.)* 255
b = (a + 128.)* 255
```

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if `pSrc`, `pSrc[3]`, `pDst`, or `pDst[3]` is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize` has a field with a zero or negative value.

LabToBGR, LabToRGB

Converts a Lab image to the BGR or RGB color model.

Syntax

Case 1: Lab to BGR

```
IppStatus ippiLabToBGR_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep,
IppiSize roiSize);
IppStatus ippiLabToBGR_16u8u_C3R(const Ipp16u* pSrc, int srcStep, Ipp8u* pDst, int
dstStep, IppiSize roiSize);
IppStatus ippiLabToBGR_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int
dstStep, IppiSize roiSize);
```

Case 2: Lab to RGB

```
IppStatus ippiLabToRGB_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep,
IppiSize roiSize);
IppStatus ippiLabToRGB_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int
dstStep, IppiSize roiSize);
IppStatus ippiLabToRGB_32f_P3R(const Ipp32f* pSrc[3], int srcStep[3], Ipp32f* pDst[3],
int dstStep[3], IppiSize roiSize);
IppStatus ippiLabToRGB_64f_P3R(const Ipp64f* pSrc[3], int srcStep[3], Ipp64f* pDst[3],
int dstStep[3], IppiSize roiSize);
```

Case 3: Lab to RGB with platform-aware functions

```
IppStatus ippiLabToRGB_32f_P3R_L(const Ipp32f* pSrc[3], IppSizeL srcStep[3], Ipp32f*
pDst[3], IppSizeL dstStep[3], IppiSizeL roiSize);

IppStatus ippiLabToRGB_64f_P3R_L(const Ipp64f* pSrc[3], IppSizeL srcStep[3], Ipp64f*
pDst[3], IppSizeL dstStep[3], IppiSizeL roiSize);
```

Case 4: Lab to RGB with TL functions based on the Platform Aware API

```
IppStatus ippiLabToRGB_32f_P3R_LT(const Ipp32f* pSrc[3], IppSizeL srcStep[3], Ipp32f*
pDst[3], IppSizeL dstStep[3], IppiSizeL roiSize);

IppStatus ippiLabToRGB_64f_P3R_LT(const Ipp64f* pSrc[3], IppSizeL srcStep[3], Ipp64f*
pDst[3], IppSizeL dstStep[3], IppiSizeL roiSize);
```

Case 5: Lab to RGB with TL functions based on the Classic API

```
IppStatus ippiLabToRGB_32f_P3R_T(const Ipp32f* pSrc[3], int srcStep[3], Ipp32f*
pDst[3], int dstStep[3], IppiSize roiSize);

IppStatus ippiLabToRGB_64f_P3R_T(const Ipp64f* pSrc[3], int srcStep[3], Ipp64f*
pDst[3], int dstStep[3], IppiSize roiSize);
```

Include Files

`ippcc.h`

`ippcc_l.h`

`ippcc_tl.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`, `pSrc[3]`	Pointer to the source image ROI.
`srcStep`, `srcStep[3]`	Distance in bytes between starts of consecutive lines in the source image.
`pDst`, `pDst[3]`	Pointer to the destination image ROI.
`dstStep`, `dstStep[3]`	Distance in bytes between starts of consecutive lines in the destination image.
`roiSize`	Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the CIE Lab image `pSrc` to the `BGR` or `RGB` image `pDst` in two steps. First, the conversion is carried out into CIE XYZ format.

To accomplish it, `Lab` components are transformed back into their original range. This is done for different data types in the following way.

For `8u` data type:

```
L = L * 100./255.
```

a = a - 128.

b = b - 128.

For 16u data type:

L = L * 100./65535.

a = (a/255. - 128.)

b = (b/255.) - 128.)

After that, conversion to XYZ format takes place as follows:

$Y = Y_n * P^{3.}$

$X = X_n*(P + a/500.)^{3.}$

$Z = Z_n*(P - b/200.)^{3.}$

where

P = (L +16)/116.

After this intermediate conversion is done, the obtained XYZ image is then converted to the destination BGR or RGB format using equations defined for the ippiXYZToRGB function.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if pSrc, pSrc[3], pDst, or pDst[3] is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with a zero or negative value.

RGBToYCC

Converts an RGB image to the YCC color model.

Syntax

```
IppStatus ippiRGBToYCC_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the gamma-corrected R'B'G' image pSrc to the PhotoY'C'C' image pDst according to the following basic equations:

```
Y' = 0.299*R' + 0.587*G' + 0.114*B'
C1' = -0.299*R' - 0.587*G' + 0.886*B' = B'- Y
C2' = 0.701*R' - 0.587*G' - 0.114*B' = R'- Y
```

The equations above are given on the assumption that R',G', and B' values are normalized to the range [0..1]. In case of the floating-point data type, the input R'G'B' values must already be in the range [0..1]. For integer data types, normalization is done by the conversion function internally.

The computed Y', C1', C2' values are then quantized and converted to fit in the range [0..1] as follows:

```
Y' = 1. / 1.402 * Y'
C1' = 111.4 / 255. * C1' + 156. / 255.
C2' = 135.64 /255. * C2' + 137. / 255.
```

In case of integer function flavors, these values are then scaled to the full range of the destination data type (see Table "Image Data Types and Ranges").

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if pSrc or pDst is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with a zero or negative value.

YCCToRGB

Converts a YCC image to the RGB color model.

Syntax

```
IppStatus ippiYCCToRGB_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the PhotoY'C'C' image pSrc to the R'B'G' image pDst. The function ippiYCCToRGB first restores normal luminance and chrominance data as:

```
Y' = 1.3584 * Y'
C1' = 2.2179 * (C1' - 156./255.)
C2' = 1.8215 * (C2' - 137./255.)
```

The equations above are given on the assumption that source Y, C1, and C2 values are normalized to the range [0..1]. In case of the floating-po int data type, the input YCC values must already be in the range [0..1]. For integer data types, normalization is done by the conversion function internally.

After that, YCC data are transformed into RGB format according to the following basic equations:

```
R' = Y' + C2'
G' = Y' - 0.194*C1' - 0.509*C2'
B' = Y' + C1'
```

In case of integer function flavors, the computed R'B'G' values are then scaled to the full range of the destination data type (see Table "Image Data Types and Ranges").

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if pSrc or pDst is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with a zero or negative value.

RGBToHLS

Converts an RGB image to the HLS color model.

Syntax

```
IppStatus ippiRGBToHLS_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the R'B'G' image pSrc to the HLS image pDst. For function flavors operating on the floating point data, source RGB values must be in the range [0..1].

The conversion algorithm from RGB to HLS can be represented in pseudocode as follows:

```
// Lightness:
M1 = max(R,G,B); M2 = min(R,G,B); L = (M1+M2)/2
// Saturation:
if M1 = M2 then // achromatics case
    S = 0
    H = 0
else // chromatics case
    if L <= 0.5 then
         S = (M1-M2) / (M1+M2)
    else
         S = (M1-M2) / (2-M1-M2)
// Hue:
Cr = (M1-R) / (M1-M2)
Cg = (M1-G) / (M1-M2)
Cb = (M1-B) / (M1-M2)
if R = M1 then H = Cb - Cg              //change R=M2 to R=M1
if G = M1 then H = 2 + Cr - Cb         //change G=M2 to G=M1
if B = M1 then H = 4 + Cg - Cr         //change B=M2 to B=M1
H = 60*H
if H < 0 then H = H + 360
```

For floating point function flavors, the computed H, L, S values are scaled to the range [0..1]. In case of integer function flavors, these values are scaled to the full range of the destination data type (Table “Image Data Types and Ranges”).

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if pSrc or pDst is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with a zero or negative value.

HLSToRGB

Converts an HLS image to the RGB color model.

Syntax

```
IppStatus ippiHLSToRGB_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

Include Files

ippcc.h

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image.
`roiSize`	Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the HLS image `pSrc` to the `R'B'G'` image `pDst`. For function flavors operating on the floating point data, source HLS values must be in the range [0..1]. The conversion algorithm from `HLS` to `RGB` can be represented in pseudocode as follows:

```
if L <= 0.5 then      M2 = L *(1 + S) else      M2 = L + S - L * S M1 = 2 * L - M2 if S = 0
then    R = G = B = L else      h = H + 120      if h > 360 then           h = h - 360      if
h  <  60 then           R = ( M1 + ( M2 - M1 ) * h / 60)      else if h < 180 then           R =
M2      else if h < 240 then           R = M1 + ( M2 - M1 ) * ( 240 - h ) / 60
else           R = M1      h = H      if h  <  60 then           G = ( M1 + ( M2 - M1 ) * h /
60      else if h < 180 then           G = M2      else if h < 240 then           G = M1 + ( M2
- M1 ) * ( 240 - h ) / 60      else           G  = M1      h = H - 120      if h < 0
then           h += 360      if h  <  60 then           B = ( M1 + ( M2 - M1 ) * h / 60
else if h < 180 then           B = M2      else if h < 240 then           B = M1 + ( M2 - M1 ) *
( 240 - h ) / 60      else           B = M1
```

For floating point function flavors, the computed `R',G',B'` values are scaled to the range [0..1]. In case of integer function flavors, these values are scaled to the full range of the destination data type (see Table “Image Data Types and Ranges”).

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if `pSrc` or `pDst` is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize` has a field with a zero or negative value.

BGRToHLS

Converts a BGR image to the HLS color model.

Syntax

```
IppStatus ippiBGRToHLS_8u_AC4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);

IppStatus ippiBGRToHLS_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep, IppiSize roiSize);

IppStatus ippiBGRToHLS_8u_AC4P4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[4], int dstStep, IppiSize roiSize);

IppStatus ippiBGRToHLS_8u_P3C3R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);

IppStatus ippiBGRToHLS_8u_AP4C4R(const Ipp8u* pSrc[4], int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);

IppStatus ippiBGRToHLS_8u_P3R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst[3], int dstStep, IppiSize roiSize);

IppStatus ippiBGRToHLS_8u_AP4R(const Ipp8u* pSrc[4], int srcStep, Ipp8u* pDst[4], int dstStep, IppiSize roiSize);
```

Include Files

`ippcc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the ROI in the pixel-order source image. An array of pointers to ROI in each plane in the planar source image.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`pDst`	Pointer to the ROI in the pixel-order destination image. An array of pointers to ROI in each plane in the planar destination image.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image.
`roiSize`	Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the `B'G'R'` image `pSrc` to the HLS image `pDst` according to the same formula as the function `ippiRGBToHLS` does.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if pSrc or pDst is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with a zero or negative value.

HLSToBGR

Converts an HLS image to the RGB color model.

Syntax

```
IppStatus ippiHLSToBGR_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
IppStatus ippiHLSToBGR_8u_AC4P4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[4], int dstStep, IppiSize roiSize);
IppStatus ippiHLSToBGR_8u_AP4R(const Ipp8u* pSrc[4], int srcStep, Ipp8u* pDst[4], int dstStep, IppiSize roiSize);
IppStatus ippiHLSToBGR_8u_P3R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst[3], int dstStep, IppiSize roiSize);
IppStatus ippiHLSToBGR_8u_AP4C4R(const Ipp8u* pSrc[4], int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
IppStatus ippiHLSToBGR_8u_P3C3R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Pointer to the ROI in the pixel-order source image. An array of pointers to ROI in each plane in the planar source image.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the ROI in the pixel-order destination image. An array of pointers to ROI in each plane in the planar destination image.

dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This functionconverts the HLS image pSrc to the B'G'R' image pDst according to the same formula as the function ippiHLSToRGB does.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if pSrc or pDst is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with a zero or negative value.

RGBToHSV

Converts an RGB image to the HSV color model.

Syntax

```
IppStatus ippiRGBToHSV_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C3R	16u_C3R
8u_AC4R	16u_AC4R

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.

dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the R'G'B' image pSrc to the HSV image pDst.

The conversion algorithm from RGB to HSV can be represented in pseudocode as follows:

```
//  Value: V = max(R,G,B); //  Saturation: temp = min(R,G,B); if V = 0 then // achromatics
case     S = 0//          H = 0 else // chromatics case     S = (V - temp)/V //  Hue: Cr = (V -
R) / (V - temp) Cg = (V - G) / (V - temp) Cb = (V - B) / (V - temp) if R = V then H = Cb - Cg if
G = V then H = 2 + Cr - Cb if B = V then H = 4 + Cg - Cr H = 60*H if H < 0 then H = H + 360
```

The computed H,S,V values are scaled to the full range of the destination data type (see Table "Image Data Types and Ranges").

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if pSrc or pDst is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with a zero or negative value.

HSVToRGB

Converts an HSV image to the RGB color model.

Syntax

```
IppStatus ippiHSVToRGB_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C3R	16u_C3R
8u_AC4R	16u_AC4R

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts a HSV image pSrc to the image R'G'B'pDst.

The conversion algorithm from HSV to RGB can be represented in pseudocode as follows:

```
if S = 0 then    R = G = B = V else     if H = 360 then         H = 0     else         H =
H/60      I = floor(H)      F = H - I;      M = V * ( 1 - S);      N = V * ( 1 - S * F);      K
= V * ( 1 - S * (1 - F));      if(I == 0)then{ R = V;G = K;B = M;}      if(I == 1)then{ R = N;G
= V;B = M;}      if(I == 2)then{ R = M;G = V;B = K;}      if(I == 3)then{ R = M;G = N;B =
V;}      if(I == 4)then{ R = K;G = M;B = V;}      if(I == 5)then{ R = V;G = M;B = N;}
```

The computed R',G',B' values are scaled to the full range of the destination data type (see Table "Image Data Types and Ranges").

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if pSrc or pDst is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with a zero or negative value.

RGBToYCoCg

Converts a RGB image to the YCoCg color model.

Syntax

```
IppStatus ippiRGBToYCoCg_8u_C3P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int
dstStep, IppiSize roi);
```

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Array of pointers to the destination image ROI in each plane.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roi	Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts a RGB image pSrc to the YCoCg image pDst according to the following formulas:

```
Y = ((R + 2*G + B)+ 2)/4
Co = ((R - B) + 1))/2
Cg = (( - R + 2*G - B) + 2)/4
```

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.

YCoCgToRGB

Converts a YCoCg image to the RGB image.

Syntax

```
IppStatus ippiYCoCgToRGB_8u_P3C3R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst,int dstStep, IppiSize roi);
```

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Array of pointers to the source image ROI in each plane.

srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roi	Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the YCoCg image pSrc to the RGB image pDst according to the following formulas:

```
R = Y + Co - Cg
G = Y + Cg
B = Y - Co - Cg
```

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.

BGRToYCoCg

Converts a 24-bit BGR image to the YCoCg color model.

Syntax

```
IppStatus ippiBGRToYCoCg_8u16s_C3P3R(const Ipp8u* pBGR, int bgrStep, Ipp16s* pYCC[3],
int yccStep, IppiSize roiSize);
```

```
IppStatus ippiBGRToYCoCg_8u16s_C4P3R(const Ipp8u* pBGR, int bgrStep, Ipp16s* pYCC[3],
int yccStep, IppiSize roiSize);
```

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pBGR	Pointer to the source image ROI.
bgrStep	Distance in bytes between starts of consecutive lines in the source image.

pYCC	Array of pointers to the destination image ROI in each plane.
yccStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the 24-bit BGR image pBGR to the YCoCg image pYCC according to the following formulas:

```
Y = ((R + 2*G + B)+ 2)/4
Co = ((R - B) + 1))/2
Cg = (( - R + 2*G - B) + 2)/4
```

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.

SBGRToYCoCg

Converts a 48-bit BGR image to the YCoCg color model.

Syntax

```
IppStatus ippiSBGRToYCoCg_<mod>(const Ipp16s* pBGR, int bgrStep, Ipp<dstDatatype>*
pYCC[3], int yccStep, IppiSize roiSize);
```

Supported values for mod:

16s_C3P3R	16s32s_C3P3R
16s_C4P3R	16s32s_C4P3R

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pBGR	Pointer to the source image ROI.

bgrStep	Distance in bytes between starts of consecutive lines in the source image.
pYCC	Array of pointers to the destination image ROI in each plane.
yccStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the 48-bit BGR image pBGR to the YCoCg image pYCC according to the following formulas:

```
Y = ((R + 2*G + B)+ 2)/4
Co = ((R - B) + 1))/2
Cg = (( - R + 2*G - B) + 2)/4
```

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.

YCoCgToBGR

Converts a YCoCg image to the 24-bit BGR image.

Syntax

```
IppStatus ippiYCoCgToBGR_16s8u_P3C3R(const Ipp16s* pYCC[3], int yccStep, Ipp8u*
pBGR,int bgrStep, IppiSize roiSize);
IppStatus ippiYCoCgToBGR_16s8u_P3C4R(const Ipp16s* pYCC[3], int yccStep, Ipp8u* pBGR,
int bgrStep, IppiSize roiSize, Ipp8u aval);
```

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pYCC	Array of pointers to the source image ROI in each plane.
yccStep	Distance in bytes between starts of consecutive lines in the source image.
pBGR	Pointer to the destination image ROI.
bgrStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.
aval	Constant value to create the fourth channel.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the YCoCg image pYCC to the 24-bit BGR image pBGR according to the following formulas:

```
R = Y + Co - Cg
G = Y + Cg
B = Y - Co - Cg
```

The fourth channel is created by setting channel values to the constant value aval.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.

YCoCgToSBGR

Converts a YCoCg image to the 48-bit BGR image.

Syntax

Case 1: Conversion to 3-channel image

```
IppStatus ippiYCoCgToSBGR_16s_P3C3R(const Ipp16s* pYCC[3], int yccStep, Ipp16s* pBGR,
int bgrStep, IppiSize roiSize);
IppStatus ippiYCoCgToSBGR_32s16s_P3C3R(const Ipp32s* pYCC[3], int yccStep, Ipp16s*
pBGR, int bgrStep, IppiSize roiSize);
```

Case 2: Conversion to 4-channel image

```
IppStatus ippiYCoCgToSBGR_16s_P3C4R(const Ipp16s* pYCC[3], int yccStep, Ipp16s* pBGR,
int bgrStep, IppiSize roiSize, Ipp16s aval);
IppStatus ippiYCoCgToSBGR_32s16s_P3C4R(const Ipp32s* pYCC[3], int yccStep, Ipp16s*
pBGR, int bgrStep, IppiSize roiSize, Ipp16s aval);
```

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pYCC	Array of pointers to the source image ROI in each plane.
yccStep	Distance in bytes between starts of consecutive lines in the source image.
pBGR	Pointer to the destination image ROI.
bgrStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.
aval	Constant value to create the fourth channel.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the YCoCg image pYCC to the 48-bit BGR image pBGR according to the following formulas:

```
R = Y + Co - Cg
G = Y + Cg
B = Y - Co - Cg
```

The fourth channel is created by setting channel values to the constant value aval.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.

BGRToYCoCg_Rev

Converts a 24-bit BGR image to the YCoCg-R color model.

Syntax

```
IppStatus ippiBGRToYCoCg_Rev_8u16s_C3P3R(const Ipp8u* pBGR, int bgrStep, Ipp16s*
pYCC[3], int yccStep, IppiSize roiSize);
```

```
IppStatus ippiBGRToYCoCg_Rev_8u16s_C4P3R(const Ipp8u* pBGR, int bgrStep, Ipp16s*
pYCC[3], int yccStep, IppiSize roiSize);
```

Include Files

`ippcc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pBGR`	Pointer to the source image ROI.
`bgrStep`	Distance in bytes between starts of consecutive lines in the source image.
`pYCC`	Array of pointers to the destination image ROI in each plane.
`yccStep`	Distance in bytes between starts of consecutive lines in the destination image.
`roiSize`	Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the 24-bit `BGR` image `pSrc` to the YCoCg-R image `pDst` according to the following formulas:

```
Co = R - B
t = B + (Co >> 1)
Cg = G - t
Y = t + (Cg >> 1)
```

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.

SBGRToYCoCg_Rev

Converts a 48-bit BGR image to the YCoCg-R color model.

Syntax

```
IppStatus ippiSBGRToYCoCg_Rev_<mod>(const Ipp16s* pBGR, int bgrStep, Ipp<dstDatatype>*
pYCC[3], int yccStep, IppiSize roiSize);
```

Supported values for `mod`:

`16s_C3P3R`	`16s32s_C3P3R`
`16s_C4P3R`	`16s32s_C4P3R`

Include Files

`ippcc.h`

Parameters

`pBGR`	Pointer to the source image ROI.
`bgrStep`	Distance in bytes between starts of consecutive lines in the source image.
`pYCC`	Array of pointers to the destination image ROI in each plane.
`yccStep`	Distance in bytes between starts of consecutive lines in the destination image.
`roiSize`	Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the 48-bit `BGR` image `pBGR` to the YCoCg-R image `pYCC` according to the following formulas:

```
Co = R - B
t = B + (Co >> 1)
Cg = G - t
Y = t + (Cg >> 1)
```

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.

YCoCgToBGR_Rev

Converts a YCoCg-R image to the 24-bit BGR image.

Syntax

```
IppStatus ippiYCoCgToBGR_Rev_16s8u_P3C3R(const Ipp16s* pYCC[3], int yccStep, Ipp8u* pBGR, int bgrStep, IppiSize roiSize);
IppStatus ippiYCoCgToBGR_Rev_16s8u_P3C4R(const Ipp16s* pYCC[3], int yccStep, Ipp8u* pBGR, int bgrStep, IppiSize roiSize, Ipp8u aval);
```

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pYCC	Array of pointers to the source image ROI in each plane.
yccStep	Distance in bytes between starts of consecutive lines in the source image.
pBGR	Pointer to the destination image ROI.
bgrStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.
aval	Constant value to create the fourth channel.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the YCoCg-R image pYCC to the 24-bit BGR image pBGR according to the following formulas:

```
t = Y - (Cg >> 1)
G = Cg + t
B = t - (Co >> 1)
R = B + Co
```

The fourth channel is created by setting channel values to the constant value aval.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.

YCoCgToSBGR_Rev

Converts a YCoCg-R image to the 48-bit BGR image.

Syntax

Case 1: Conversion to 3-channel image.

```
IppStatus ippiYCoCgToSBGR_Rev_16s_P3C3R(const Ipp16s* pYCC[3], int yccStep, Ipp16s*
pBGR, int bgrStep, IppiSize roiSize);

IppStatus ippiYCoCgToSBGR_Rev_32s16s_P3C3R(const Ipp32s* pYCC[3], int yccStep, Ipp16s*
pBGR, int bgrStep, IppiSize roiSize);
```

Case 2: Conversion to 4-channel image

```
IppStatus ippiYCoCgToSBGR_Rev_16s_P3C4R(const Ipp16s* pYCC[3], int yccStep, Ipp16s*
pBGR, int bgrStep, IppiSize roiSize, Ipp16s aval);

IppStatus ippiYCoCgToSBGR_Rev_32s16s_P3C4R(const Ipp32s* pYCC[3], int yccStep, Ipp16s*
pBGR, int bgrStep, IppiSize roiSize, Ipp16s aval);
```

Include Files

`ippcc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pYCC`	Array of pointers to the source image ROI in each plane.
`yccStep`	Distance in bytes between starts of consecutive lines in the source image.
`pBGR`	Pointer to the destination image ROI.
`bgrStep`	Distance in bytes between starts of consecutive lines in the destination image.
`roiSize`	Size of the source and destination ROI in pixels.
`aval`	Constant value to create the fourth channel.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the YCoCg-R image `pYCC` to the 48-bit BGR image `pBGR` according to the following formulas:

```
t = Y - (Cg >> 1)

G = Cg + t

B = t - (Co >> 1)

R = B + Co
```

The fourth channel is created by setting channel values to the constant value `aval`.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.

Color - Gray Scale Conversions

- GrayToRGB
- RGBToGray
- ColorToGray
- CFAToBGRA
- CFAToRGB
- DemosaicAHD

GrayToRGB

Coverts a gray scale image to RGB/BGR by copying luminance component to color components.

Syntax

```
IppStatus ippiGrayToRGB_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C1C3R	16u_C1C3R	32f_C1C3R

```
IppStatus ippiGrayToRGB_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize roiSize, Ipp<datatype> aval);
```

Supported values for mod:

8u_C1C4R	16u_C1C4R	32f_C1C4R

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance, in bytes, between the starting points of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.

dstStep	Distance, in bytes, between the starting points of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI, in pixels.
aval	Constant value to create the fourth channel.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts a gray scale image to an RGB/BGR image by copying luminance component to color components.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error when pSrc or pDst is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with a zero or negative value.

Example

The code example below demonstrates how to use the ippiGrayToRGB_8u_C1C4R function.

```
const int WIDTH  = 2;
const int HEIGHT = 1;


Ipp8u pSrc[WIDTH * HEIGHT] = {
    113,113,
};
Ipp8u pDst[WIDTH * HEIGHT * 4];
int srcStep = WIDTH, dstStep  = WIDTH * 4;
IppiSize roiSize = {WIDTH, HEIGHT};
IppStatus status = ippiGrayToRGB_8u_C1C4R(pSrc, srcStep, pDst, dstStep, roiSize, 0xFF);
if ( status == ippStsNoErr) {
    printf("PASS:\n(%3d %3d %3d %3d), (%3d %3d %3d %3d)\n", pDst[0], pDst[1], pDst[2], pDst[3],
pDst[4], pDst[5], pDst[6], pDst[7]);
}
else
    printf("FAIL: status = %d\n", status);
```

Result:

```
PASS:
(113 113 113 255), (113 113 113 255)
```

See Also

Regions of Interest in Intel IPP

RGBToGray

Converts an RGB image to gray scale using fixed transform coefficients.

Syntax

```
IppStatus ippiRGBToGray_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

`8u_C3C1R`	`16u_C3C1R`	`16s_C3C1R`	`32f_C3C1R`
`8u_AC4C1R`	`16u_AC4C1R`	`16s_AC4C1R`	`32f_AC4C1R`

Include Files

`ippcc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

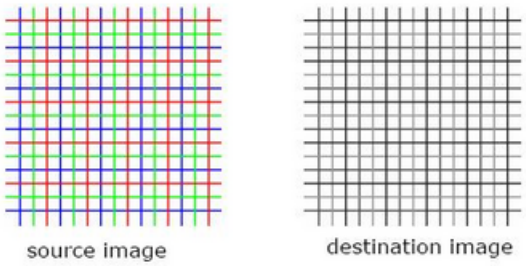
`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image.
`roiSize`	Size of the source and destination ROI in pixels.

Description

This function operates with ROI.

Conversion from RGB image to gray scale (see figure *Converting an RGB Image to Gray Scale*) uses the following basic equation to compute luma from nonlinear gamma-corrected red, green, and blue values: `Y' = 0.299 * R' + 0.587 * G' + 0.114 * B'`. Note that the transform coefficients conform to the standard for the NTSC red, green, and blue CRT phosphors.

Converting an RGB Image to Gray Scale |



Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if pSrc or pDst pointer is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with a zero or negative value.

Example

The code example below demonstrates how to use the function ippiRGBToGray_8u_C3C1R.

```
Ipp8u  src[12*3] = { 255, 0,  0, 255, 0,  0, 255, 0,  0, 255, 0,  0,
                     0, 255, 0,  0, 255, 0,  0, 255, 0,  0, 255, 0,
                     0,  0, 255, 0,  0, 255, 0,  0, 255, 0,  0, 255};
Ipp8u dst[4*3];
IppiSize srcRoi = { 4, 3 };


ippiRGBToGray_8u_C3C1R ( src, 12, dst, 4, srcRoi );
```

Result:

```
255 0   0   255 0   0   255 0   0   255 0   0
0   255 0   0   255 0   0   255 0   0   255 0        src
0   0   255 0   0   255 0   0   255 0   0   255


 76  76  76  76
149 149 149 149    dst
 29  29  29  29
```

ColorToGray

Converts an RGB image to gray scale using custom transform coefficients.

Syntax

```
IppStatus ippiColorToGray_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize roiSize, const Ipp32f coeffs[3]);
```

Supported values for mod:

8u_C3C1R	16u_C3C1R	16s_C3C1R	32f_C3C1R
8u_AC4C1R	16u_AC4C1R	16s_AC4C1R	32f_AC4C1R

```
IppStatus ippiColorToGray_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize roiSize, const Ipp64f coeffs[3]);
```

Supported values for mod:

64f_C3C1R
64f_AC4C1R

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.
coeffs	Transform coefficients.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function uses the following equation to convert an RGB image to gray scale:

Y =coeffs[0] * R +coeffs[1] * G +coeffs[2] * B,

where the coeffs array contains user-defined transform coefficients which must be non-negative and satisfy the condition

coeffs [0] + coeffs [1] + coeffs [2] ≤ 1.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if pSrc or pDst is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with a zero or negative value.

CFAToBGRA

Restores the RGB image from the gray-scale CFA image using the VNG algorithm.

Syntax

```
IppStatus ippiCFAToBGRA_VNG_8u_C1C4R(const Ipp8u* pSrc, IppiRect srcRoi, IppiSize
srcSize, int srcStep, Ipp32f scale[4], Ipp8u* pDst, int dstStep, IppiBayerGrid grid);
```

```
IppStatus ippiCFAToBGRA_VNG_16u_C1C4R(const Ipp16u* pSrc, IppiRect srcRoi, IppiSize
srcSize, int srcStep, Ipp32f scale[4], Ipp16u* pDst, int dstStep, IppiBayerGrid grid);
```

Platform-aware functions

```
IppStatus ippiCFAToBGRA_VNG_8u_C1C4R_L(const Ipp8u* pSrc, IppiRectL srcRoiL, IppiSizeL
srcSizeL, IppSizeL srcStepL, Ipp32f scale[4], Ipp8u* pDst, IppSizeL dstStepL,
IppiBayerGrid grid);
```

```
IppStatus ippiCFAToBGRA_VNG_16u_C1C4R_L(const Ipp16u* pSrc, IppiRectL srcRoiL,
IppiSizeL srcSizeL, IppSizeL srcStepL, Ipp32f scale[4], Ipp16u* pDst, IppSizeL
dstStepL, IppiBayerGrid grid);
```

Include Files

`ippcc.h`

`ippcc_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the source image.
`srcRoi`, `srcRoiL`	Region of interest in the source image (of the `IppiRect` or `IppiRectL` type).
`srcSize`, `srcSizeL`	Size of the source image.
`srcStep`, `srcStepL`	Distance, in bytes, between the starting points of consecutive lines in the source image.
`scale[4]`	Coefficients by which the resulting RGB channels are multiplied after interpolation. By default, equal to 1.0.
`pDst`	Pointer to the destination image.
`dstStep`, `dstStepL`	Distance, in bytes, between the starting points of consecutive lines in the destination image.
`grid`	Specifies the configuration of the Bayer grid in the source image. The function copies 2-pixel width border pixels from the internal neighborhood pixels. The following values are possible: `ippiBayerBGGR ippiBayerRGGB ippiBayerGBRG ippiBayerGRBG`

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function transforms the one-channel gray-scale image `pSrc` that is produced by applying the color filter array (CFA) to 24-bit three-channel RGB image using the Variable Number of Gradients (VNG) demosaicing algorithm.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if the srcSize or srcSizeL has a field that is less than 2, or if the srcRoi or srcRoiL has a field with a negative or zero value.
ippStsBadArgErr	Indicates an error condition if grid has an illegal value.

See Also

Structures and Enumerators

Structures and Enumerators for Platform-Aware Functions

CFAToRGB

Restores the RGB image from the gray-scale CFA image.

Syntax

```
IppStatus ippiCFAToRGB_8u_C1C3R(const Ipp8u* pSrc, IppiRect srcRoi, IppiSize srcSize,
int srcStep, Ipp8u* pDst, int dstStep, IppiBayerGrid grid, int interpolation);
```

```
IppStatus ippiCFAToRGB_16u_C1C3R(const Ipp16u* pSrc, IppiRect srcRoi, IppiSize srcSize,
int srcStep, Ipp16u* pDst, int dstStep, IppiBayerGrid grid, int interpolation);
```

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Pointer to the source image origin.
srcSize	Size of the source image.
srcRoi	Region of interest in the source image (of the IppiRect type).
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image.

dstStep	Distance in bytes between starts of consecutive lines in the destination image.
grid	Specifies the configuration of the Bayer grid in the source image. The following values are possible: ippiBayerBGGR ippiBayerRGGB ippiBayerGBRG ippiBayerGRBG
interpolation	Interpolation method, reserved, must be 0.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function transforms the one-channel gray-scale image pSrc that is produced by applying the color filter array (CFA) - an array of Bayer filters, to 24-bit three-channel RGB image. The order of the color component in the source image - Bayer grid - is specified by the parameter grid. Four possible values of this parameter correspond to the allowed variants of the Bayer grid (see Figure below).

Possible Configurations of the Bayer Grids |



Each element of the source image contains an intensity value for only one color component, two others are interpolated using neighbor elements. R and B values are interpolated linearly from the nearest neighbors of the same color. When interpolating R and B values on green pixel, the average values of the two nearest neighbors (above and below, or left and right) of the same colors are used. When interpolating R or B values on the blue or red pixel respectively, the average values of the four nearest blue (red) pixels cornering the red (blue) pixel are used. G values are interpolated using an adaptive interpolation Sak98 from a pair of nearest neighbors (vertical or horizontal) and taking into account the correlation in the red (or blue) component. The pair is chosen depending on the values of the difference between the red (blue) pixels in the vertical and horizontal directions. If the difference is smaller in the vertical direction - a vertical pair of green pixels is used, if it is smaller in the horizontal direction - a horizontal pair is used. If the difference is the same, all four neighbors are used.

This interpolation requires border pixels for the input pixels near the horizontal or vertical edge of the image. The function uses the mirrored border of two edge rows or columns of the input image. In this case the G values is calculated as the average of four nearest green pixels.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointer is NULL.
ippStsSizeErr	Indicates an error condition if the srcSize has a field that is less than 2, or if the roiSize has a field with negative or zero value.

ippStsBadArgErr	Indicates an error condition if grid has an illegal value.

DemosaicAHD

Restores the RGB image from the gray-scale CFA image using AHD algorithm.

Syntax

```
IppStatus ippiDemosaicAHD_8u_C1C3R(const Ipp8u* pSrc, IppiRect srcRoi, IppiSize srcSize, int srcStep, Ipp8u* pDst, int dstStep, IppiBayerGrid grid, Ipp8u* pTmp, int tmpStep);
```

```
IppStatus ippiDemosaicAHD_16u_C1C3R(const Ipp16u* pSrc, IppiRect srcRoi, IppiSize srcSize, int srcStep, Ipp16u* pDst, int dstStep, IppiBayerGrid grid, Ipp16u* pTmp, int tmpStep);
```

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Pointer to the source image.
srcRoi	Region of interest in the source image (of the IppiRect type).
srcSize	Size of the source image.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
grid	Specifies the configuration of the Bayer grid in the source image. The following values are possible (see Figure Possible Configurations of the Bayer Grids): ippiBayerBGGR ippiBayerRGGB ippiBayerGBRG ippiBayerGRBG
pTmp	Pointer to the temporary image of (srcRoi.width + 6, 30) size.
tmpStep	Distance in bytes between starts of consecutive lines in the temporary image.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function transforms the one-channel gray-scale image `pSrc` that is produced by applying the color filter array (CFA) to 24-bit three-channel RGB image using the adaptive homogeneity-directed demosaicing (AHD) algorithm Hir05. The algorithm requires the temporary image `pTmp` of size `srcRoi.width + 6`.30.

The type of the Bayer grid (see Figure Possible Configurations of the Bayer Grids) is specified by the parameter `grid`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if the `srcSize` has a field that is less than 5, or if the `roiSize` has a field with negative or zero value.
`ippStsBadArgErr`	Indicates an error condition if `grid` has an illegal value.

Format Conversion

This section describes Intel IPP functions that perform image color conversion without changing the color space. These functions convert pixel-order images to planar format and vice versa, change the number of channels or planes, alter sampling formats and sequences of samples and planes. Several functions additionally perform filtering - deinterlacing and upsampling.

Intel IPP format conversion functions are specified mainly in the YCbCr color space, but as they do not transform color model they may be used to perform described types of conversion for any other color spaces with decoupled luminance and chrominance coordinates (YUV type).

- YCbCr422
- YCbCr422ToYCrCb422
- YCbCr422ToCbYCr422
- YCbCr422ToYCbCr420
- YCbCr422To420_Interlace
- YCbCr422ToYCrCb420
- YCbCr422ToYCbCr411
- YCrCb422ToYCbCr422
- YCrCb422ToYCbCr420
- YCrCb422ToYCbCr411
- CbYCr422ToYCbCr422
- CbYCr422ToYCbCr420
- CbYCr422ToYCbCr420_Interlace
- CbYCr422ToYCrCb420
- CbYCr422ToYCbCr411
- YCbCr420
- YCbCr420ToYCbCr422
- YCbCr420ToYCbCr422_Filter
- YCbCr420To422_Interlace
- YCbCr420ToCbYCr422
- YCbCr420ToCbYCr422_Interlace
- YCbCr420ToYCrCb420

- YCbCr420ToYCrCb420_Filter
- YCbCr420ToYCbCr411
- YCrCb420ToYCbCr422
- YCrCb420ToYCbCr422_Filter
- YCrCb420ToCbYCr422
- YCrCb420ToYCbCr420
- YCrCb420ToYCbCr411
- YCbCr411
- YCbCr411ToYCbCr422
- YCbCr411ToYCrCb422
- YCbCr411ToYCbCr420, YCbCr411To420
- YCbCr411ToYCrCb420

YCbCr422

Converts 4:2:2 YCbCr image.

Syntax

```
IppStatus ippiYCbCr422_8u_C2P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

```
IppStatus ippiYCbCr422_8u_P3C2R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Include Files

`ippcc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the ROI in the pixel-order source image. Array of pointers to the ROI in each plane of the planar source image.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image. Array of distance values for the source image planes.
`pDst`	Pointer to the ROI in the pixel-order destination image. Array of pointers to the ROI in each plane of the planar destination image.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image. Array of distance values for the destination image planes.
`roiSize`	Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the 4:2:2 two-channel source image pSrc to the 4:2:2 three-plane image pDst and vice versa (see Table "Pixel-Order Image Formats" and Table "Planar Image Formats" for more details on 4:2:2 planar and pixel-order formats).

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize.width of the first plane is less than 2, or roiSize.height is less than or equal to zero.

YCbCr422ToYCrCb422

Converts 4:2:2 YCbCr image to 4:2:2 YCrCb image.

Syntax

```
IppStatus ippiYCbCr422ToYCrCb422_8u_C2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiYCbCr422ToYCrCb422_8u_P3C2R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Pointer to the ROI in the pixel-order source image. Array of pointers to the ROI in each plane of the planar source image.
srcStep	Distance in bytes between starts of consecutive lines in the source image. Array of distance values for the source image planes.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the ROI in pixels, its width should be multiple of 2.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts 4:2:2 `YCbCr` source image `pSrc` to the 4:2:2 `YCrCb` two-channel image `pDst` that has the following sequence of samples: `Y0`, `Cr0`, `Y1`, `Cb0`, `Y2`, `Cr1`, `Y3`, `Cb1`, … (see Table "Pixel-Order Image Formats"). The source image can be either two-channel or three-plane (see Table "Pixel-Order Image Formats" and Table "Planar Image Formats").

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize.width` is less than 2.

YCbCr422ToCbYCr422

Converts 4:2:2 YCbCr image to 4:2:2 CbYCr image.

Syntax

```
IppStatus ippiYCbCr422ToCbYCr422_8u_C2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiYCbCr422ToCbYCr422_8u_P3C2R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Include Files

`ippcc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the ROI in the pixel-order source image. Array of pointers to the ROI in each plane of the planar source image.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image. Array of distance values for the source image planes.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image.

roiSize	Size of the source and destination ROI in pixels; its width should be multiple of 2.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts 4:2:2 YCbCr source image pSrc to the 4:2:2 CbYCr two-channel image pDst that has the following sequence of samples: Cb0, Y0, Cr0, Y1, Cb1, Y2, Cr1, Y3, Cb3, … (see Table "Pixel-Order Image Formats"). The source image can be either two-channel or three-plane (see Table "Pixel-Order Image Formats" and Table "Planar Image Formats").

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize.width of the first plane is less than 2, or roiSize.height is less than or equal to zero.

YCbCr422ToYCbCr420

Converts YCbCr image from 4:2:2 sampling format to 4:2:0 format.

Syntax

Case 1: Operation on planar data

```
IppStatus ippiYCbCr422ToYCbCr420_8u_P3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u*
pDst[3], int dstStep[3], IppiSize roiSize);
```

```
IppStatus ippiYCbCr422ToYCbCr420_8u_P3P2R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u*
pDstY, int dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep, IppiSize roiSize);
```

Case 2: Conversion from pixel-order to planar data

```
IppStatus ippiYCbCr422ToYCbCr420_8u_C2P3R(const Ipp8u* pSrc, int srcStep, Ipp8u*
pDst[3], int dstStep[3], IppiSize roiSize);
```

```
IppStatus ippiYCbCr422ToYCbCr420_8u_C2P2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDstY,
int dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep, IppiSize roiSize);
```

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Pointer to the ROI in the pixel-order source image. Array of pointers to the ROI in each plane of the planar source image.
srcStep	Distance in bytes between starts of consecutive lines in the source image. Array of distance values for the source image planes.
pDst	Array of pointers to the ROI in each plane for a three-plane destination image.
dstStep	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane destination image.
pDstY	Pointer to the ROI in the luminance plane for a two-plane destination image.
dstYStep	Distance in bytes between starts of consecutive lines in the luminance plane of a destination image.
pDstCbCr	Pointer to the ROI in the interleaved chrominance plane for a two-plane destination image.
dstCbCrStep	Distance in bytes between starts of consecutive lines in the chrominance plane of a destination image.
roiSize	Size of the ROI in pixels, height and width should be multiple of 2.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the 4:2:2 image pSrc to the 4:2:0 image. The source image can be two-channel or three-plane , destination image always is planar with two or three planes (see Table "Pixel-Order Image Formats" and Table "Planar Image Formats"). Two-plane image contains luminance samples Y0, Y1, Y2, .. in the first plane pDstY, and interleaved chrominance samples Cb0, Cr0, Cb1, Cr1, … in the second plane pDstCbCr.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if any field of the roiSize is less than 2.

Example

The code example below shows how to use the function `ippiYCbCr422ToYCbCr420_8u_C2P3R`.

```
{
  Ipp8u*   ImageI420[3];
  int      stepI420[3];
  Ipp8u*   ImageYUY2;
  int      stepYUY2;
  IppiSize roiSize = { 1024, 768};
  ImageI420[0] = ippiMalloc_8u_C1( roiSize.width, roiSize.height, &(stepI420[0]));
  ImageI420[1] = ippiMalloc_8u_C1( roiSize.width, roiSize.height, &(stepI420[1]));
  ImageI420[2] = ippiMalloc_8u_C1( roiSize.width, roiSize.height, &(stepI420[2]));
  ImageYUY2    = ippiMalloc_8u_C2( roiSize.width, roiSize.height, &stepYUY2 );
  ippiYCbCr422ToYCbCr420_8u_C2P3R( ImageYUY2, stepYUY2, ImageI420, stepI420, roiSize);


  ippiFree(ImageI420[0]);
  ippiFree(ImageI420[1]);
  ippiFree(ImageI420[2]);
  ippiFree(ImageYUY2);
}
```

YCbCr422To420_Interlace

Converts interlaced YCbCr image from 4:2:2 sampling format to 4:2:0 format.

Syntax

```
IppStatus ippiYCbCr422To420_Interlace_8u_P3R(const Ipp8u* pSrc[3], int srcStep[3],
Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

Include Files

`ippcc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Array of pointers to the ROI in each plane for source image.
`srcStep`	Array of distances in bytes between starts of consecutive lines in each plane for the source image.
`pDst`	Array of pointers to the ROI in each plane for destination image.
`dstStep`	Array of distances in bytes between starts of consecutive lines in each plane fore destination image.

roiSize	Size of the source and destination ROI in pixels, its width must be multiple of 2, and height must be multiple of 4.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the interlaced 4:2:2 image pSrc to the 4:2:0 image pDst (see Table "Planar Image Formats").

The conversion is performed in accordance with the following formulas:

$Y1_{dest} = Y1_{src};$

$Cb0(Cr0)_{dest} = (3*Cb0(Cr0)_{src} + Cb2``(Cr2)``_{src} + 2``)/4;``$

$Cb1(Cr1)_{dest} = (Cb1(Cr1)_{src} + 3 * Cb3``(Cr3)``_{src} + 2``)/4;``$

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize.width is less than 2, or roiSize.height is less than 4.
ippStsDoubleSize	Indicates a warning if roiSize.width is not multiple of 2, or roiSize.height is not multiple of 4.

YCbCr422ToYCrCb420

Converts 4:2:2 YCbCr image to 4:2:0 YCrCb image.

Syntax

```
IppStatus ippiYCbCr422ToYCrCb420_8u_C2P3R(const Ipp8u* pSrc, int srcStep, Ipp8u*
pDst[3], int dstStep[3], IppiSize roiSize);
```

```
IppStatus ippiYCbCr422ToYCrCb420_8u_C2P2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDstY,
int dstYStep, Ipp8u* pDstCrCb, int dstUVStep, IppiSize roiSize);
```

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Array of pointers to the ROI in each plane of the destination image.
pDstY	Pointer to the destination image Y plane.
dstStep	Array of distances, in bytes, between the starting points of consecutive lines in the destination image planes.
dstYStep	Array of distances, in bytes, between the starting points of consecutive lines in the destination image Y plane.
dstUVStep	Array of distances, in bytes, between the starting points of consecutive lines in the destination image UV plane.
roiSize	Size of the source and destination ROI in pixels, height and width should be multiple of 2.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the 4:2:2 two-channel image pSrc that has the following sequence of samples: Y0, Cb0, Y1, Cr0, Y2, Cb1, Y3, Cr1, … to the 4:2:0 three-plane image pDst with the following order of pointers: Y - plane, Cr - plane, Cb - plane (see Table "Pixel-Order Image Formats" and Table "Planar Image Formats").

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if any field of the roiSize is less than 2.

YCbCr422ToYCbCr411

Converts YCbCr image from 4:2:2 sampling format to 4:1:1 format.

Syntax

Case 1: Operation on planar data

```
IppStatus ippiYCbCr422ToYCbCr411_8u_P3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u*
pDst[3], int dstStep[3], IppiSize roiSize);
```

```
IppStatus ippiYCbCr422ToYCbCr411_8u_P3P2R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u*
pDstY, int dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep, IppiSize roiSize);
```

Case 2: Conversion from pixel-order to planar data

```
IppStatus ippiYCbCr422ToYCbCr411_8u_C2P3R(const Ipp8u* pSrc, int srcStep, Ipp8u*
pDst[3], int dstStep[3], IppiSize roiSize);
```

```
IppStatus ippiYCbCr422ToYCbCr411_8u_C2P2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDstY,
int dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep, IppiSize roiSize);
```

Include Files

`ippcc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the ROI in the pixel-order source image. Array of pointers to the ROI in each plane of the planar source image.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image. Array of distance values for the source image planes.
`pDst`	Array of pointers to the ROI in each plane for a three-plane destination image.
`dstStep`	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane destination image.
`pDstY`	Pointer to the ROI in the luminance plane for a two-plane destination image.
`dstYStep`	Distance in bytes between starts of consecutive lines in the luminance plane of a destination image.
`pDstCbCr`	Pointer to the ROI in the interleaved chrominance plane for a two-plane destination image.
`dstCbCrStep`	Distance in bytes between starts of consecutive lines in the chrominance plane of a destination image.
`roiSize`	Size of the ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the 4:2:2 image pSrc to the 4:1:1 image. The source image can be two-channel or three-plane (see Table "Pixel-Order Image Formats" for more details), destination image always is planar with two or three planes (see Table "Planar Image Formats" for more details). The two-plane image contains luminance samples Y0, Y1, Y2, .. in the first plane pDstY, and interleaved chrominance samples Cb0, Cr0, Cb1, Cr1, … in the second plane pDstCbCr.

The value of the fields of the roiSize have certain limitations:

- its width should be multiple of 4 and cannot be less than 4 for operation on two-channel images;
- its width should be multiple of 4 and cannot be less than 4, and its height should be multiple of 2 and can not be less than 2 for three-plane to two-plane image conversion;
- both height and width should be multiple of 2 and cannot be less than 2 for operation on three-plane images.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if corresponding fileds of the roiSize is less than specified above values.

YCrCb422ToYCbCr422

Converts 4:2:2 YCrCb image to 4:2:2 YCbCr image.

Syntax

```
IppStatus ippiYCrCb422ToYCbCr422_8u_C2P3R(const Ipp8u* pSrc, int srcStep, Ipp8u*
pDst[3], int dstStep[3], IppiSize roiSize);
```

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Array of pointers to the ROI in each plane of the destination image.
dstStep	Array of distances in bytes between starts of consecutive lines in the destination image planes.

roiSize	Size of the ROI in pixels, its width should be multiple of 2.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the 4:2:2 YCrCb two-channel image pSrc (see Table "Pixel-Order Image Formats") to the 4:2:2 YCbCr three-plane image pDst (see Table "Planar Image Formats").

Return Values

ppStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize.width is less than 2.

YCrCb422ToYCbCr420

Converts 4:2:2 YCrCb image to 4:2:0 YCbCr image.

Syntax

```
IppStatus ippiYCrCb422ToYCbCr420_8u_C2P3R(const Ipp8u* pSrc, int srcStep, Ipp8u*
pDst[3], int dstStep[3], IppiSize roiSize);
```

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Array of pointers to the ROI in each plane of the destination image.
dstStep	Array of distances in bytes between starts of consecutive lines in the destination image planes.
roiSize	Size of the ROI in pixels, height and width should be multiple of 2.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This functions converts the 4:2:2 YCrCb two-channel image pSrc (see Table "Pixel-Order Image Formats") to the 4:2:0 YCbCr three-plane image pDst (see Table "Planar Image Formats").

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if any field of the roiSize is less than 2.

YCrCb422ToYCbCr411

Converts 4:2:2 YCrCb image to 4:1:1 YCbCr image.

Syntax

```
IppStatus ippiYCrCb422ToYCbCr411_8u_C2P3R(const Ipp8u* pSrc, int srcStep, Ipp8u*
pDst[3], int dstStep[3], IppiSize roiSize);
```

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Array of pointers to the ROI in each plane of the destination image.
dstStep	Array of distances in bytes between starts of consecutive lines in the destination image planes.
roiSize	Size of the ROI in pixels, its width should be multiple of 4.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the 4:2:2 YCrCb two-channel image pSrc (see Table "Pixel-Order Image Formats") to the 4:1:1 YCbCr three-plane image pDst (see Table "Planar Image Formats").

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize.width is less than 4.

CbYCr422ToYCbCr422

Converts 4:2:2 CbYCr image to 4:2:2 YCbCr image.

Syntax

```
IppStatus ippiCbYCr422ToYCbCr422_8u_C2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
int dstStep, IppiSize roiSize);
```

```
IppStatus ippiCbYCr422ToYCbCr422_8u_C2P3R(const Ipp8u* pSrc, int srcStep, Ipp8u*
pDst[3], int dstStep[3], IppiSize roiSize);
```

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the ROI in the pixel-order destination image. Array of pointers to the ROI in each plane of the planar destination image.
dstStep	Distance in bytes between starts of consecutive lines in the destination image. Array of distance values for the destination image planes.
roiSize	Size of the ROI in pixels, its width should be multiple of 2.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the 4:2:2 CbYCr two-channel image pSrc to the 4:2:2 YCbCr two-channel or three-plane image pDst (see Table "Pixel-Order Image Formats" and Table "Planar Image Formats"). The source image has the following sequence of samples: Cb0, Y0, Cr0, Y1, Cb1, Y2, Cr1, Y3, Cb2, … . Two-channel destination image has different sequence of samples: Y0, Cb0, Y1, Cr0, Y2, Cb1, Y3, Cr1, Y4, ….

Return Values

ppStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize.width is less than 2.

CbYCr422ToYCbCr420

Converts 4:2:2 CbYCr image to 4:2:0 YCbCr image.

Syntax

```
IppStatus ippiCbYCr422ToYCbCr420_8u_C2P3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

```
IppStatus ippiCbYCr422ToYCbCr420_8u_C2P2R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDstY, int dstYStep,Ipp8u* pDstCbCr, int dstCbCrStep, IppiSize roiSize);
```

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Array of pointers to the ROI in each plane for a three-plane destination image.
dstStep	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane destination image.
pDstY	Pointer to the ROI in the luminance plane for a two-plane destination image.

dstYStep	Distance in bytes between starts of consecutive lines in the luminance plane of a destination image.
pDstCbCr	Pointer to the ROI in the interleaved chrominance plane for a two-plane destination image.
dstCbCrStep	Distance in bytes between starts of consecutive lines in the chrominance plane of a destination image.
roiSize	Size of the source and destination ROI in pixels, height and width should be multiple of 2.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the 4:2:2 CbYCr two-channel image pSrc to the 4:2:0 YCbCr two- or three-plane image pDst (see Table "Pixel-Order Image Formats" and Table "Planar Image Formats"). The source image has the following sequence of samples: Cb0, Y0, Cr0, Y1, Cb1, Y2, Cr1, Y3, Cb2, … . Three-plane destination image has the following order of pointers: Y – plane, Cb – plane, Cr – plane. Two-plane destination image contains luminance samples Y0, Y1, Y2, … in the first plane pDstY, and interleaved chrominance samples Cb0, Cr0, Cb1, Cr1, … in the second plane pDstCbCr.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if any field of the roiSize is less than 2.

CbYCr422ToYCbCr420_Interlace

Converts interlaced 4:2:2 CbYCr image to 4:2:0 YCbCr image.

Syntax

```
IppStatus ippiCbYCr422ToYCbCr420_Interlace_8u_C2P3R(const Ipp8u* pSrc, int srcStep,
Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Array of pointers to the ROI in each plane for destination image.
dstStep	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane destination image.
roiSize	Size of the source and destination ROI in pixels, its width must be multiple of 2, and height must be multiple of 4.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts interlaced 4:2:2 CbYCr two-channel image pSrc to the 4:2:0 YCbCr three-plane image pDst (see Table "Pixel-Order Image Formats" and Table "Planar Image Formats"). The source image has the following sequence of samples: Cb0, Y0, Cr0, Y1, Cb1, Y2, Cr1, Y3, Cb2, … . Three-plane destination image has the following order of pointers: Y – plane, Cb – plane, Cr – plane.

The conversion is performed in accordance with the following formulas:

$Y1_{dest} = Y1_{src};$

$Cb0(Cr0)_{dest} = (3*Cb0(Cr0)_{src} + Cb2``(Cr2)``_{src} + 2``)/4;``$

$Cb1(Cr1)_{dest} = (Cb1(Cr1)_{src} + 3 * Cb3``(Cr3)``_{src} + 2``)/4;``$

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize.width is less than 2, or roiSize.height is less than 4.
ippStsDoubleSize	Indicates a warning if roiSize.width is not multiple of 2, or roiSize.height is not multiple of 4.

CbYCr422ToYCrCb420

Converts 4:2:2 CbYCr image to 4:2:0 YCrCb image.

Syntax

```
IppStatus ippiCbYCr422ToYCrCb420_8u_C2P3R(const Ipp8u* pSrc, int srcStep, Ipp8u*
pDst[3], int dstStep[3], IppiSize roiSize);
```

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Array of pointers to the ROI in each plane of the destination image.
dstStep	Array of distances in bytes between starts of consecutive lines in the destination image planes.
roiSize	Size of the source and destination ROI in pixels, height and width should be multiple of 2.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the 4:2:2 CbYCr two-channel image pSrc to the 4:2:0 YCrCb three-plane image pDst. The source image has the following sequence of samples: Cb0, Y0, Cr0, Y1, Cb1, Y2, Cr1, Y3, Cb2, … . The destination image has the following order of pointers: Y – plane, Cr – plane, Cb – plane (see Table "Pixel-Order Image Formats" and Table "Planar Image Formats").

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if any field of the roiSize is less than 2.

CbYCr422ToYCbCr411

Converts 4:2:2 CbYCr image to 4:1:1 YCbCr image.

Syntax

```
IppStatus ippiCbYCr422ToYCbCr411_8u_C2P3R(const Ipp8u* pSrc, int srcStep, Ipp8u*
pDst[3], int dstStep[3], IppiSize roiSize);
```

Include Files

ippcc.h

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`pDst`	Array of pointers to the ROI in each plane of the destination image.
`dstStep`	Array of distances in bytes between starts of consecutive lines in the destination image planes.
`roiSize`	Size of the ROI in pixels, its width should be multiple of 4.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the 4:2:2 `CbYCr` two-channel image `pSrc` to the 4:1:1 `YCbCr` three-plane image `pDst`. The source image has the following sequence of samples: `Cb0`, `Y0`, `Cr0`, `Y1`, `Cb1`, `Y2`, `Cr1`, `Y3`, `Cb2`, … . The destination image has the following order of pointers: `Y – plane`, `Cb – plane`, `Cr – plane` (see Table "Pixel-Order Image Formats" and Table "Planar Image Formats").

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize.width` is less than 4.

YCbCr420

Converts 4:2:0 YCbCr image.

Syntax

```
IppStatus ippiYCbCr420_8u_P3P2R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep, IppiSize roiSize);
```

```
IppStatus ippiYCbCr420_8u_P2P3R(const Ipp8u* pSrcY, int srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

Include Files

`ippcc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Array of pointers to the ROI in each plane for a three-plane source image.
`srcStep`	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane source image.
`pSrcY`	Pointer to the ROI in the luminance plane for a two-plane source image.
`srcYStep`	Distance in bytes between starts of consecutive lines in the luminance plane of a source image.
`pSrcCbCr`	Pointer to the ROI in the interleaved chrominance plane for a two-plane source image.
`srcCbCrStep`	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.
`pDst`	Array of pointers to the ROI in each plane for a three-plane destination image.
`dstStep`	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane destination image.
`pDstY`	Pointer to the ROI in the luminance plane for a two-plane destination image.
`dstYStep`	Distance in bytes between starts of consecutive lines in the luminance plane of a destination image.
`pDstCbCr`	Pointer to the ROI in the interleaved chrominance plane for a two-plane destination image.
`dstCbCrStep`	Distance in bytes between starts of consecutive lines in the chrominance plane of a destination image.
`roiSize`	Size of the source and destination ROI in pixels, height and width should be multiple of 2.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the 4:2:0 three-plane (see Table "Planar Image Formats") source image `pSrc` to the 4:2:0 two-plane image and vice versa. Two-plane image contains luminance samples `Y0`, `Y1`, `Y2`, .. in the first plane, and interleaved chrominance samples `Cb0`, `Cr0`, `Cb1`, `Cr1`,… in the second plane.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if any field of the roiSize is less than 2.

YCbCr420ToYCbCr422

Converts YCbCr image from 4:2:0 sampling format to 4:2:2 format.

Syntax

```
IppStatus ippiYCbCr420ToYCbCr422_8u_P3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

```
IppStatus ippiYCbCr420ToYCbCr422_8u_P2P3R(const Ipp8u* pSrcY, int srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

```
IppStatus ippiYCbCr420ToYCbCr422_8u_P2C2R(const Ipp8u* pSrcY, int srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Array of pointers to the ROI in each plane for a three-plane source image.
srcStep	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane source image.
pSrcY	Pointer to the ROI in the luminance plane for a two-plane source image.
srcYStep	Distance in bytes between starts of consecutive lines in the luminance plane of a source image.
pSrcCbCr	Pointer to the ROI in the interleaved chrominance plane for a two-plane source image.
srcCbCrStep	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.

pDst	Pointer to the ROI in the pixel-order destination image. Array of pointers to the ROI in each plane of the planar destination image.
dstStep	Distance in bytes between starts of consecutive lines in the destination image. Array of distance values for the destination image planes.
roiSize	Size of the ROI in pixels, height and width should be multiple of 2.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the 4:2:0 planar source image pSrc to the 4:2:2 image pDst. The source image can be two- or three-plane image (see Table "Planar Image Formats"). The first plane of the two-plane source image pSrcY contains luminance samples Y0, Y1, Y2, …, the second plane pSrcCbCr contains interleaved chrominance samples Cb0, Cr0, Cb1, Cr1, …. . The destination image pDst can be three-plane or two-channel (see Table "Pixel-Order Image Formats" and Table "Planar Image Formats").

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if any field of the roiSize is less than 2.

YCbCr420ToYCbCr422_Filter

Convert 4:2:0 image to 4:2:2 image with additional filtering.

Syntax

```
IppStatus ippiYCbCr420ToYCbCr422_Filter_8u_P3R(const Ipp8u* pSrc[3], int srcStep[3],
Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

```
IppStatus ippiYCbCr420ToYCbCr422_Filter_8u_P2P3R(const Ipp8u* pSrcY, int srcYStep,
const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst[3], int dstStep[3], IppiSize
roiSize);
```

```
IppStatus ippiYCbCr420ToYCbCr422_Filter_8u_P2C2R(const Ipp8u* pSrcY, int srcYStep,
const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, int
layout);
```

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Array of pointers to the ROI in each plane for a three-plane source image.
srcStep	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane source image.
pSrcY	Pointer to the ROI in the luminance plane for a two-plane source image.
srcYStep	Distance in bytes between starts of consecutive lines in the luminance plane of a source image.
pSrcCbCr	Pointer to the ROI in the interleaved chrominance plane for a two-plane source image.
srcCbCrStep	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.
pDst	Pointer to the ROI in the pixel-order destination image. Array of pointers to the ROI in each plane of the planar destination image.
dstStep	Distance in bytes between starts of consecutive lines in the destination image. Array of distance values for the destination image planes.
roiSize	Size of the ROI in pixels.
layout	Slice layout. Possible values: • IPP_UPPER • Upper (first) slice • IPP_CENTER • Middle slices • IPP_LOWER • Lowermost (last) slice • IPP_LOWER && IPP_UPPER && IPP_CENTER • Image is not sliced

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the 4:2:0 planar source image pSrc to the 4:2:2 image pDst and performs additional filtering. The source image can be two- or three-plane image (see Table "Planar Image Formats"). The first plane of the two-plane source image pSrcY contains luminance samples Y0, Y1, Y2, …, the second plane pSrcCbCr contains interleaved chrominance samples Cb0, Cr0, Cb1, Cr1, …. . The destination image pDst can be three-plane or two-channel (see Table "Pixel-Order Image Formats" and Table "Planar Image Formats").

The function flavors `ippiYCbCr420ToYCbCr422_Filter_8u_P3R` and `ippiYCbCr420ToYCbCr422_Filter_8u_P2P3R` additionally perform the vertical upsampling using a Catmull-Rom interpolation (cubic convolution interpolation). In this case `roiSize.width` should be multiple of 2, and `roiSize.height` should be multiple of 8.

The function `ippiYCbCr420ToYCbCr422_Filter_8u_P2C2R` additionally performs deinterlace filtering. Commonly it is used to process images that are divided into slices. In this case slice `layout` should be specified, since the function processes the first (upper), last (lowermost), and intermediate (middle) slices differently. The height of slices should be a multiple of 16.

NOTE
CAUTION
The image slices should be processed exactly in the following order: the first slice, intermediate slices, the last slice.

The function may be applied to a not-sliced image as well. In this case `roiSize.width` and `roiSize.height` should be multiple of 2.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize.width` has wrong value.

YCbCr420To422_Interlace

Converts interlaced YCbCr image from 4:2:0 sampling format to 4:2:2 format.

Syntax

```
IppStatus ippiYCbCr420To422_Interlace_8u_P3R(const Ipp8u* pSrc[3], int srcStep[3],
Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

Include Files

`ippcc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Array of pointers to the ROI in each plane for source image.

srcStep	Array of distances in bytes between starts of consecutive lines in each plane for the source image.
pDst	Array of pointers to the ROI in each plane for destination image.
dstStep	Array of distances in bytes between starts of consecutive lines in each plane for the destination image.
roiSize	Size of the source and destination ROI in pixels, its width must be multiple of 2, and height must be multiple of 4.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the interlaced planar 4:2:0 source image pSrc to the 4:2:2 image pDst. Three-plane image has the following order of pointers: Y - plane, Cb - plane, Cr - plane.

The conversion is performed in accordance with the following formulas:

$Yn_{dest} = Yn_{src};$

$Cb0(Cr0)_{dest} = (5*Cb0(Cr0)_{src} + 3 * Cb2``(Cr2)``_{src} + 4``)/8;``$

$Cb1(Cr1)_{dest} = (7*Cb1(Cr1)_{src} + Cb3``(Cr3)``_{src} + 4``)/8;``$

$Cb2(Cr2)_{dest} = (Cb0(Cr0)_{src} + 7 * Cb2``(Cr2)``_{src} + 4``)/8;``$

$Cb3(Cr3)_{dest} = (3*Cb1(Cr1)_{src} + 5 * Cb3``(Cr3)``_{src} + 4``)/8;``$

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize.width is less than 2, or roiSize.height is less than 4.
ippStsDoubleSize	Indicates a warning if roiSize.width is not multiple of 2, or roiSize.height is not multiple of 4.

YCbCr420ToCbYCr422

Converts 4:2:0 YCbCr image to 4:2:2 CbYCr image.

Syntax

```
IppStatus ippiYCbCr420ToCbYCr422_8u_P2C2R(const Ipp8u* pSrcY, int srcYStep, const
Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrcY	Pointer to the ROI in the luminance plane of the source image.
srcYStep	Distance in bytes between starts of consecutive lines in the luminance plane of the source image.
pSrcCbCr	Pointer to the ROI in the interleaved chrominance plane of the source image.
srcCbCrStep	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels, height and width should be multiple of 2.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the planar 4:2:0 two-plane source image to the pixel-order 4:2:2 two-channel image. The first plane of the source image pSrcY contains luminance samples Y0, Y1, Y2, .., the second plane pSrcCbCr contains interleaved chrominance samples Cb0, Cr0, Cb1, Cr1, … . The destination image pDst has the following sequence of samples: Cb0, Y0, Cr0, Y1, Cb1, Y2, Cr1, Y3, Cb2, …

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if any field of the roiSize is less than 2.

YCbCr420ToCbYCr422_Interlace

Converts interlaced 4:2:0 YCbCr image to 4:2:2 CbYCr image.

Syntax

```
IppStatus ippiYCbCr420ToCbYCr422_Interlace_8u_P3C2R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Include Files

`ippcc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Array of pointers to the ROI in each plane for source image.
`srcStep`	Array of distances in bytes between starts of consecutive lines in each plane for the source image.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image.
`roiSize`	Size of the source and destination ROI in pixels, its width must be multiple of 2, and height must be multiple of 4.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the interlaced planar 4:2:0 image to the pixel-order 4:2:2 two-channel image. Three-plane source image has the following order of pointers: `Y - plane`, `Cb - plane`, `Cr - plane`.The destination image `pDst` has the following sequence of samples: `Cb0`, `Y0`, `Cr0`, `Y1`, `Cb1`, `Y2`, `Cr1`, `Y3`, `Cb2`, …

The conversion is performed in accordance with the following formulas:

$Yn_{dest} = Yn_{src};$

$Cb0(Cr0)_{dest} = (5*Cb0(Cr0)_{src} + 3 * Cb2``(Cr2)``_{src} + 4``)/8;``$

$Cb1(Cr1)_{dest} = (7*Cb1(Cr1)_{src} + Cb3``(Cr3)``_{src} + 4``)/8;``$

$Cb2(Cr2)_{dest} = (Cb0(Cr0)_{src} + 7 * Cb2``(Cr2)``_{src} + 4``)/8;``$

$Cb3(Cr3)_{dest} = (3*Cb1(Cr1)_{src} + 5 * Cb3``(Cr3)``_{src} + 4``)/8;``$

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.

ippStsSizeErr	Indicates an error condition if roiSize.width is less than 2, or roiSize.height is less than 4.
ippStsDoubleSize	Indicates a warning if roiSize.width is not multiple of 2, or roiSize.height is not multiple of 4.

YCbCr420ToYCrCb420

Converts 4:2:0 YCbCr image to 4:2:0 YCrCb image.

Syntax

```
IppStatus ippiYCbCr420ToYCrCb420_8u_P2P3R(const Ipp8u* pSrcY, int srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrcY	Pointer to the ROI in the luminance plane of the source image.
srcYStep	Distance in bytes between starts of consecutive lines in the luminance plane of the source image.
pSrcCbCr	Pointer to the ROI in the interleaved chrominance plane of the source image.
srcCbCrStep	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.
pDst	Array of pointers to the ROI in each plane of the destination image.
dstStep	Array of distances in bytes between starts of consecutive lines in the destination image planes.
roiSize	Size of the source and destination ROI in pixels, height and width should be multiple of 2.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the 4:2:0 two-plane source image pSrc to the 4:2:0 three-plane image pDst. The first plane of the source image pSrcY contains luminance samples Y0, Y1, Y2, …, the second plane pSrcCbCr contains interleaved chrominance samples Cb0, Cr0, Cb1, Cr1, …. The destination image pDst has the following order of pointers:Y - plane, Cr - plane, Cb - plane (see Table "Planar Image Formats").

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if any field of the roiSize is less than 2.

YCbCr420ToYCrCb420_Filter

Convert 4:2:0 YCbCr image to 4:2:0 YCrCb image with deinterlace filtering.

Syntax

```
IppStatus ippiYCbCr420ToYCrCb420_Filter_8u_P2P3R(const Ipp8u* pSrcY, int srcYStep,
const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst[3], int dstStep[3], IppiSize
roiSize, int layout);
```

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrcY	Pointer to the ROI in the luminance plane of the source image.
srcYStep	Distance in bytes between starts of consecutive lines in the luminance plane of the source image.
pSrcCbCr	Pointer to the ROI in the interleaved chrominance plane of the source image.
srcCbCrStep	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.
pDst	Array of pointers to the ROI in each plane of the destination image.
dstStep	Array of distances in bytes between starts of consecutive lines in the destination image planes.

roiSize	Size of the source and destination ROI in pixels, height and width should be multiple of 2.
layout	Slice layout. Possible values: • IPP_UPPER • Upper (first) slice • IPP_CENTER • Middle slices • IPP_LOWER • Lowermost (last) slice • IPP_LOWER && IPP_UPPER && IPP_CENTER • Image is not sliced

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the 4:2:0 two-plane source image to the 4:2:0 three-plane image. The first plane of the source image pSrcY contains luminance samples Y0, Y1, Y2, …, the second plane pSrcCbCr contains interleaved chrominance samples Cb0, Cr0, Cb1, Cr1, …. The destination image pDst has the following order of pointers: Y – plane, Cr – plane, Cb – plane. The function additionally performs deinterlace filtering. Commonly it is used to process sliced images. In this case the slice layout should be specified, since the function processes the first (upper), last (lowermost), and intermediate (middle) slices differently. The height of slices should be a multiple of 16. The function may be applied to a not-sliced image as well.

NOTE
CAUTION
The image slices should be processed exactly in the following order: the first slice, intermediate slices, the last slice.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if any field of the roiSize is less than 2.

YCbCr420ToYCbCr411

Converts YCbCr image from 4:2:0 sampling format to 4:1:1 format.

Syntax

```
IppStatus ippiYCbCr420ToYCbCr411_8u_P3P2R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep, IppiSize roiSize);
```

```
IppStatus ippiYCbCr420ToYCbCr411_8u_P2P3R(const Ipp8u* pSrcY, int srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

```
IppStatus ippiYCbCr420To411_8u_P3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u*
pDst[3] , int dstStep[3], IppiSize roiSize);

IppStatus ippiYCbCr420To1620_8u_P3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u*
pDst[3] , int dstStep[3], IppiSize roiSize);

IppStatus ippiYCbCr1620To420_8u_P3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u*
pDst[3] , int dstStep[3], IppiSize roiSize);
```

Include Files

`ippcc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Array of pointers to the ROI in each plane for a three-plane source image.
`srcStep`	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane source image.
`pSrcY`	Pointer to the ROI in the luminance plane for a two-plane source image.
`srcYStep`	Distance in bytes between starts of consecutive lines in the luminance plane of a source image.
`pSrcCbCr`	Pointer to the ROI in the interleaved chrominance plane for a two-plane source image.
`srcCbCrStep`	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.
`pDst`	Array of pointers to the ROI in each plane for a three-plane destination image.
`dstStep`	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane destination image.
`pDstY`	Pointer to the ROI in the luminance plane for a two-plane destination image.
`dstYStep`	Distance in bytes between starts of consecutive lines in the luminance plane of a destination image.
`pDstCbCr`	Pointer to the ROI in the interleaved chrominance plane for a two-plane destination image.

dstCbCrStep	Distance in bytes between starts of consecutive lines in the chrominance plane of a destination image.
roiSize	Size of the ROI in pixels, its width should be multiple of 4, its height should be multiple of 2.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the 4:2:0 source image to the 4:1:1 destination image. The source two-plane image is converted to destination three-plane image and vice versa. The first plane of the two-plane image contains luminance samples Y0, Y1, Y2, …, the second plane contains interleaved chrominance samples Cb0, Cr0, Cb1, Cr1, …. The three-plane image has the following order of pointers: Y – plane, Cb – plane, Cr – plane (see Table "Planar Image Formats").

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize.width is less than 4 or roiSize.height is less than 2.

YCrCb420ToYCbCr422

Converts 4:2:0 YCrCb image to 4:2:2 YCbCr image.

Syntax

```
IppStatus ippiYCrCb420ToYCbCr422_8u_P3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

```
IppStatus ippiYCrCb420ToYCbCr422_8u_P3C2R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Array of pointers to the ROI in each plane of the source image.

srcStep	Array of distances in bytes between starts of consecutive lines in each plane of the source image.
pDst	Pointer to the ROI in the pixel-order destination image. Array of pointers to the ROI in each plane of the planar destination image.
dstStep	Distance in bytes between starts of consecutive lines in the destination image. Array of distance values for the destination image planes.
roiSize	Size of the ROI in pixels, height and width should be multiple of 2.

Description

This function converts the 4:2:0 YCrCb three-plane image pSrc to the 4:2:2 YCbCr three-plane or two-channel image pDst (see Table "Pixel-Order Image Formats" and Table "Planar Image Formats").

This function operates with ROI (see Regions of Interest in Intel IPP).

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if any field of the roiSize is less than 2.

YCrCb420ToYCbCr422_Filter

Converts 4:2:0 YCrCb image to 4:2:2 YCbCr image with additional filtering.

Syntax

```
IppStatus ippiYCrCb420ToYCbCr422_Filter_8u_P3R(const Ipp8u* pSrc[3], int srcStep[3],
Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Array of pointers to the ROI in each plane of the source image.

srcStep	Array of distances in bytes between starts of consecutive lines in each plane of the source image.
pDst	Array of pointers to the ROI in each plane of the destination image.
dstStep	Array of distances in bytes between starts of consecutive lines in each plane of the destination image.
roiSize	Size of the ROI in pixels, its width should be multiple of 2, its height should be multiple of 8.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the 4:2:0 YCrCb three-plane image pSrc to the 4:2:2 YCbCr three-plane image pDst (see Table "Planar Image Formats").

Additionally, this function performs the vertical upsampling using a Catmull-Rom interpolation (cubic convolution interpolation).

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize.width is less than 2 or roiSize.height is less than 8.

YCrCb420ToCbYCr422

Converts 4:2:0 YCrCb image to 4:2:2 CbYCr image.

Syntax

```
IppStatus ippiYCrCb420ToCbYCr422_8u_P3C2R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Array of pointers to the ROI in each plane of the source image.
srcStep	Array of distances in bytes between starts of consecutive lines in each plane of the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels, height and width should be multiple of 2.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the 4:2:0 YCrCb three-plane image pSrc (see Table "Planar Image Formats") to the 4:2:2 CbYCr two-channel image pDst with the following sequence of samples: Cb0, Y0, Cr0, Y1, Cb1, Y2, Cr1, Y3, Cb2, … (see Table "Pixel-Order Image Formats").

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if any field of the roiSize is less than 2.

YCrCb420ToYCbCr420

Converts 4:2:0 YCrCb image to 4:2:0 YCbCr image.

Syntax

```
IppStatus ippiYCrCb420ToYCbCr420_8u_P3P2R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u*
pDstY, int dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep, IppiSize roiSize);
```

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Array of pointers to the ROI in each plane of the source image.
srcStep	Array of distances in bytes between starts of consecutive lines in each plane of the source image.
pDstY	Pointer to the ROI in the luminance plane of a destination image.
dstYStep	Distance in bytes between starts of consecutive lines in the luminance plane of the destination image.
pDstCbCr	Pointer to the ROI in the interleaved chrominance plane of the destination image.
dstCbCrStep	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of a destination image.
roiSize	Size of the source and destination ROI in pixels, height and width should be multiple of 2.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the 4:2:0 YCrCb three-plane image pSrc (see Table "Planar Image Formats") to the 4:2:0 YCbCr two-plane image that contains luminance samples Y0, Y1, Y2, .. in the first plane pDstY, and interleaved chrominance samples Cb0, Cr0, Cb1, Cr1, … in the second plane pDstCbCr.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if any field of the roiSize is less than 2.

YCrCb420ToYCbCr411

Converts 4:2:0 YCrCb image to 4:1:1 YCbCr image.

Syntax

```
IppStatus ippiYCrCb420ToYCbCr411_8u_P3P2R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u*
pDstY, int dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep, IppiSize roiSize);
```

Include Files

ippcc.h

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Array of pointers to the ROI in each plane of the source image.
`srcStep`	Array of distances in bytes between starts of consecutive lines in each plane of the source image.
`pDstY`	Pointer to the ROI in the luminance plane of a destination image.
`dstYStep`	Distance in bytes between starts of consecutive lines in the luminance plane of the destination image.
`pDstCbCr`	Pointer to the ROI in the interleaved chrominance plane of the destination image.
`dstCbCrStep`	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of a destination image.
`roiSize`	Size of the ROI in pixels, its width should be multiple of 4, its height should be multiple of 2.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the 4:2:0 `YCrCb` three-plane image `pSrc` (see Table "Planar Image Formats") to the 4:1:1 `YCbCr` two-plane image that contains luminance samples `Y0`, `Y1`, `Y2`, .. in the first plane `pDstY`, and interleaved chrominance samples `Cb0`, `Cr0`, `Cb1`, `Cr1`, … in the second plane `pDstCbCr`..

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize.width` is less than 4 or `roiSize.height` is less than 2.

YCbCr411

Converts 4:1:1 YCbCr image.

Syntax

```
IppStatus ippiYCbCr411_8u_P3P2R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDstY, int dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep, IppiSize roiSize);
```

```
IppStatus ippiYCbCr411_8u_P2P3R(const Ipp8u* pSrcY, int srcYStep, const Ipp8u*
pSrcCbCr, int srcCbCrStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

Include Files

`ippcc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Array of pointers to the ROI in each plane for a three-plane source image.
`srcStep`	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane source image.
`pSrcY`	Pointer to the ROI in the luminance plane for a two-plane source image.
`srcYStep`	Distance in bytes between starts of consecutive lines in the luminance plane of a source image.
`pSrcCbCr`	Pointer to the ROI in the interleaved chrominance plane for a two-plane source image.
`srcCbCrStep`	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.
`pDst`	Array of pointers to the ROI in each plane for a three-plane destination image.
`dstStep`	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane destination image.
`pDstY`	Pointer to the ROI in the luminance plane for a two-plane destination image.
`dstYStep`	Distance in bytes between starts of consecutive lines in the luminance plane of a destination image.
`pDstCbCr`	Pointer to the ROI in the interleaved chrominance plane for a two-plane destination image.
`dstCbCrStep`	Distance in bytes between starts of consecutive lines in the chrominance plane of a destination image.
`roiSize`	Size of the ROI in pixels, its width should be multiple of 4.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the 4:1:1 three-plane (see Table "Planar Image Formats") source image `pSrc` to the 4:1:1 two-plane image and vice versa. Two-plane image contains luminance samples `Y0`, `Y1`, `Y2`, .. in the first plane, and interleaved chrominance samples `Cb0`, `Cr0`, `Cb1`, `Cr1`, … in the second plane.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize.width` is less than 4.

YCbCr411ToYCbCr422

Converts 4:1:1 YCbCr image to 4:2:2 YCbCr image.

Syntax

```
IppStatus ippiYCbCr411ToYCbCr422_8u_P3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

```
IppStatus ippiYCbCr411ToYCbCr422_8u_P3C2R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiYCbCr411ToYCbCr422_8u_P2P3R(const Ipp8u* pSrcY, int srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

```
IppStatus ippiYCbCr411ToYCbCr422_8u_P2C2R(const Ipp8u* pSrcY, int srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Include Files

`ippcc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Array of pointers to the ROI in each plane for a three-plane source image.
`srcStep`	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane source image.
`pSrcY`	Pointer to the ROI in the luminance plane for a two-plane source image.

srcYStep	Distance in bytes between starts of consecutive lines in the luminance plane of a source image.
pSrcCbCr	Pointer to the ROI in the interleaved chrominance plane for a two-plane source image.
srcCbCrStep	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.
pDst	Pointer to the ROI in the pixel-order destination image. Array of pointers to the ROI in each plane of the planar destination image.
dstStep	Distance in bytes between starts of consecutive lines in the destination image. Array of distance values for the destination image planes.
roiSize	Size of the ROI in pixels, its width should be multiple of 4.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This functionconverts 4:1:1 planar source image pSrc to the 4:2:2 image pDst. The first plane of the two-plane source image pSrcY contains luminance samples Y0, Y1, Y2, …, the second plane pSrcCbCr contains interleaved chrominance samples Cb0, Cr0, Cb1, Cr1, …. The destination image pDst can be either three-plane (see Table "Planar Image Formats") or two-channel image (see Table "Pixel-Order Image Formats").

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize.width is less than 4.

YCbCr411ToYCrCb422

Converts 4:1:1 YCbCr image to 4:2:2 YCrCb image.

Syntax

```
IppStatus ippiYCbCr411ToYCrCb422_8u_P3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

```
IppStatus ippiYCbCr411ToYCrCb422_8u_P3C2R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Include Files

ippcc.h

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Array of pointers to the ROI in each plane for a three-plane source image.
`srcStep`	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane source image.
`pDst`	Pointer to the ROI in the pixel-order destination image. Array of pointers to the ROI in each plane of the planar destination image.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image. Array of distance values for the destination image planes.
`roiSize`	Size of the ROI in pixels, its width should be multiple of 4.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the 4:1:1 three-plane image `pSrc` to the 4:2:2 two-channel or three-plane image `pDst` with different order of components. The source image has the following order of pointers: `Y - plane`, `Cb - plane`, `Cr - plane` (see Table "Planar Image Formats"). The three-plane destination image has the following order of pointers: `Y - plane`, `Cr - plane`, `Cb - plane` (see Table "Planar Image Formats"), and two-channel destination image has the following sequence of samples: `Y0`, `Cr0`, `Y1`, `Cb0`, `Y2`, `Cr1`, `Y3`, `Cb1`, … (see Table "Pixel-Order Image Formats").

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize.width` is less than 4.

YCbCr411ToYCbCr420, YCbCr411To420

Converts 4:1:1 YCbCr image to 4:2:0 YCbCr image.

Syntax

```
IppStatus ippiYCbCr411ToYCbCr420_8u_P3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u*
pDst[3], int dstStep[3], IppiSize roiSize);
```

```
IppStatus ippiYCbCr411ToYCbCr420_8u_P3P2R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u*
pDstY, int dstYStep, Ipp8u* pDstCbCr, int dstCbCrStep, IppiSize roiSize);

IppStatus ippiYCbCr411ToYCbCr420_8u_P2P3R(const Ipp8u* pSrcY, int srcYStep, const
Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);

IppStatus ippiYCbCr411To420_8u_P3R(const Ipp8u* pSrc[3], int srcStep[3], Ipp8u*
pDst[3], int dstStep[3], IppiSize roiSize);
```

Include Files

`ippcc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Array of pointers to the ROI in each plane for a three-plane source image.
`srcStep`	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane source image.
`pSrcY`	Pointer to the ROI in the luminance plane for a two-plane source image.
`srcYStep`	Distance in bytes between starts of consecutive lines in the luminance plane of a source image.
`pSrcCbCr`	Pointer to the ROI in the interleaved chrominance plane for a two-plane source image.
`srcCbCrStep`	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.
`pDst`	Array of pointers to the ROI in each plane for a three-plane destination image.
`dstStep`	Array of distances in bytes between starts of consecutive lines in each plane for a three-plane destination image.
`pDstY`	Pointer to the ROI in the luminance plane for a two-plane destination image.
`dstYStep`	Distance in bytes between starts of consecutive lines in the luminance plane of a destination image.
`pDstCbCr`	Pointer to the ROI in the interleaved chrominance plane for a two-plane destination image.

dstCbCrStep	Distance in bytes between starts of consecutive lines in the chrominance plane of a destination image.
roiSize	Size of the ROI in pixels, its width should be multiple of 4, its height should be multiple of 2.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the 4:1:1 planar source image pSrc (see Table "Planar Image Formats") to the 4:2:0 planar image pDst. Both source and destination images can be three- or two-plane. Three-plane images has the following order of pointers: Y – plane, Cb – plane, Cr – plane (see Table "Planar Image Formats"). Two-plane images contain luminance samples Y0, Y1, Y2, .. in the first plane, and interleaved chrominance samples Cb0, Cr0, Cb1, Cr1, … in the second plane.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize.width is less than 4 or roiSize.height is less than 2.

YCbCr411ToYCrCb420

Converts 4:1:1 YCbCr image to 4:2:0 YCrCb image.

Syntax

```
IppStatus ippiYCbCr411ToYCrCb420_8u_P2P3R(const Ipp8u* pSrcY, int srcYStep, const Ipp8u* pSrcCbCr, int srcCbCrStep, Ipp8u* pDst[3], int dstStep[3], IppiSize roiSize);
```

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrcY	Pointer to the ROI in the luminance plane of the source image.
srcYStep	Distance in bytes between starts of consecutive lines in the luminance plane of the source image.

pSrcCbCr	Pointer to the ROI in the interleaved chrominance plane of the source image.
srcCbCrStep	Distance in bytes between starts of consecutive lines in the interleaved chrominance plane of the source image.
pDst	Array of pointers to the ROI in each plane of the destination image.
dstStep	Array of distances in bytes between starts of consecutive lines in each plane of the destination image.
roiSize	Size of the ROI in pixels, its width should be multiple of 4, its height should be multiple of 2.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the 4:1:1 two-plane source image pSrc to the 4:2:0 three-plane image pDst with a different order of components. The first plane of the source image pSrcY contains luminance samples Y0, Y1, Y2, …, the second plane pSrcCbCr contains interleaved chrominance samples Cb0, Cr0, Cb1, Cr1, … .The destination image has the following order of pointers: Y – plane, Cr – plane, Cb – plane (see Table "Planar Image Formats"),

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize.width is less than 4 or roiSize.height is less than 2.

Color Twist

Color twist conversion functions use values of all color channels of a source pixel to compute the resultant destination channel value. The destination channel value is obtained as the result of multiplying the corresponding row of the color-twist matrix by the vector of source pixel channel values.

For example, if (r,g,b) is a source pixel, then the destination pixel values (R,G,B) are computed as follows:

$R = t_{11} * r + t_{12} * g + t_{13} * b + t_{14}$

$G = t_{21} * r + t_{22} * g + t_{23} * b + t_{24}$

$B = t_{31} * r + t_{32} * g + t_{33} * b + t_{343}$

where

$$T = \begin{bmatrix} t_{11} & t_{12} & t_{13} & t_{14} \\ t_{21} & t_{22} & t_{23} & t_{24} \\ t_{31} & t_{32} & t_{33} & t_{34} \end{bmatrix}$$

is the color twist matrix. The color twist matrix used by the Intel IPP functions is a matrix of size 3x4, or 4x4 with floating-point elements. The matrix elements are specific for each particular type of color conversion.

- ColorTwist
- ColorTwist32f

ColorTwist

Applies a color twist matrix to an image with floating-point pixel values.

Syntax

Case 1: Not-in-place operation on pixel-order data

```
IppStatus ippiColorTwist_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, const Ipp32f twist[3][4]);
```

Supported values for `mod`:

`32f_C3R`
`32f_AC4R`

```
IppStatus ippiColorTwist_32f_C4R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, const Ipp32f twist[4][4]);
```

Case 2: Not-in-place operation on planar data

```
IppStatus ippiColorTwist_32f_P3R(const Ipp32f* pSrc[3], int srcStep, Ipp32f* pDst[3], int dstStep, IppiSize roiSize, const Ipp32f twist[3][4]);
```

Case 3: In-place operation on pixel-order data

```
IppStatus ippiColorTwist_<mod>(Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize, const Ipp32f twist[3][4]);
```

Supported values for `mod`:

`32f_C3IR`
`32f_AC4IR`

Case 4: In-place operation on planar data

```
IppStatus ippiColorTwist_32f_IP3R(Ipp32f* pSrcDst[3], int srcDstStep, IppiSize roiSize, const Ipp32f twist[3][4]);
```

Include Files

`ippcc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
pSrcDst	Pointer to the source and destination image ROI for the in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
roiSize	Size of the source and destination ROI in pixels.
twist	The array containing color-twist matrix elements.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function applies the color-twist matrix to all three color channels in the source image with floating-point pixel values to obtain the resulting data in the destination image. The destination channel value is obtained as the result of multiplying the corresponding row of the color-twist matrix by the vector of source pixel channel values.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if pSrc, pDst, or pSrcDst is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with a zero or negative value.

ColorTwist32f

Applies a color twist matrix to an image with integer pixel values.

Syntax

Case 1: Not-in-place operation on pixel-order data

```
IppStatus ippiColorTwist32f_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const Ipp32f twist[3][4]);
```

Supported values for mod:

8u_C3R	16u_C3R	16s_C3R
8u_AC4R	16u_AC4R	16s_AC4R

Case 2: Not-in-place operation on planar data

```
IppStatus ippiColorTwist32f_<mod>(const Ipp<datatype>* pSrc[3], int srcStep,
Ipp<datatype>* pDst[3], int dstStep, IppiSize roiSize, const Ipp32f twist[3][4]);
```

Supported values for `mod` :

8u_P3R	16u_P3R	16s_P3R

Case 3: In-place operation on pixel-order data

```
IppStatus ippiColorTwist32f_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize
roiSize, const Ipp32f twist[3][4]);
```

Supported values for `mod` :

8u_C3IR	16u_C3IR	16s_C3IR
8u_AC4IR	16u_AC4IR	16s_AC4IR

Case 4: In-place operation on planar data

```
IppStatus ippiColorTwist32f_<mod>(Ipp<datatype>* pSrcDst[3], int srcDstStep, IppiSize
roiSize, const Ipp32f twist[3][4]);
```

Supported values for `mod` :

8u_IP3R	16u_IP3R	16s_IP3R

Include Files

`ippcc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance, in bytes, between the starting points of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance, in bytes, between the starting points of consecutive lines in the destination image.
`pSrcDst`	Pointer to the source and destination image ROI for the in-place operation.

srcDstStep	Distance, in bytes, between the starting points of consecutive lines in the source and destination image for the in-place operation.
roiSize	Size of the source and destination ROI in pixels.
twist	The array containing color-twist matrix elements.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function applies the color-twist matrix to all three color channel values in the integer source image to obtain the resulting data in the destination image. For example, the conversion from the RGB to the YCbCr format can be done as

```
Y = 0.299*R + 0.587*G + 0.114*B
```

```
Cb = - 0.16874*R - 0.33126*G + 0.5*B + 0.5
```

```
Cr = 0.5*R - 0.41869*G - 0.08131*B + 0.5
```

which can be described in terms of the following color twist matrix:

```
0.29900f     0.58700f       0.11400f   0.000f
-0.16874f    -0.33126f       0.50000f    128.0f
 0.50000f     -0.41869f   -0.08131f  128.0f
```

Color-twist matrices may also be used to perform many other color conversions.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if pSrc, pDst, or pSrcDst is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with a zero or negative value.

Color Keying

- CompColorKey
- AlphaCompColorKey

CompColorKey

Performs color keying of two images.

Syntax

Case 1: Operation on one-channel data

```
IppStatus ippiCompColorKey_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
Ipp<datatype> colorKey);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R

Case 2: Operation on multi-channel data

```
IppStatus ippiCompColorKey_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
Ipp<datatype> colorKey[3]);
```

Supported values for `mod`:

8u_C3R	16u_C3R	16s_C3R

```
IppStatus ippiCompColorKey_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize roiSize,
Ipp<datatype> colorKey[4]);
```

Supported values for `mod`:

8u_C4R	16u_C4R	16s_C4R

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc1, pSrc2`	Pointer to the source images ROI.
`src1Step, src2Step`	Distances in bytes between starts of consecutive lines in the source images.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image.
`roiSize`	Size of the source and destination ROI in pixels.
`colorKey`	Value of the key color for 1-channel images, array of color values for multi-channel images.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function replaces all areas of the source image `pSrc1` containing the specified key color `colorKey` with the corresponding pixels of the background image `pSrc2` and stores the result in the destination image `pDst`.

The Figure Applying the Function ippiCompColorKey to Sample Images shows an example of how the function `ippiCompColorKey` works.

Applying the Function ippiCompColorKey to Sample Images



Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize` has a field with a zero or negative value.
`ippStsStepErr`	Indicates an error condition if one of the step values is less than or equal to 0.

AlphaCompColorKey

Performs color keying and alpha composition of two images.

Syntax

```
IppStatus ippiAlphaCompColorKey_8u_AC4R(const Ipp8u* pSrc1, int src1Step, Ipp8u alpha1,
const Ipp8u* pSrc2, int src2Step, Ipp8u alpha2, Ipp8u* pDst, int dstStep, IppiSize
roiSize, Ipp8u colorKey[4], IppiAlphaType alphaType);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

pSrc1, pSrc2	Pointer to the source images ROI.
src1Step, src2Step	Distances in bytes between starts of consecutive lines in the source images.
alpha1, alpha2	Alpha value.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.
colorKey	Array of color values.
alphaType	The type of composition to perform (without pre-multiplying). See Table "Possible Values of alphaType Parameter" for more details.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function replaces all areas of the source image pSrc1 containing the specified key color colorKey with the corresponding pixels of the background image pSrc2 and additionally performs alpha composition (see supported Table "Types of Image Compositing Operations") in accordance with the parameter alphaType. Note the alpha channel in the pDst is not changed after color keying.

The parameter alphaType should not be set to the values intended for operations with pre-multiplying.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with a zero or negative value.
ippStsStepErr	Indicates an error condition if one of the step values is less than or equal to 0.
ippStsAlphaTypeErr	Indicates an error condition if alphaType specifies the unsupported type of composition.

Gamma Correction

- GammaFwd
- GammaInv

GammaFwd

Performs gamma-correction of the source image with RGB data.

Syntax

Case 1: Not-in-place operation on integer pixel-order data

```
IppStatus ippiGammaFwd_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

8u_C3R	16u_C3R
8u_AC4R	16u_AC4R

Case 2: Not-in-place operation on integer planar data

```
IppStatus ippiGammaFwd_<mod>(const Ipp<datatype>* pSrc[3], int srcStep, Ipp<datatype>*
pDst[3], int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

8u_P3R	16u_P3R

Case 3: Not-in-place operation on floating-point pixel-order data

```
IppStatus ippiGammaFwd_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int
dstStep, IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);
```

Supported values for `mod`:

32f_C3R
32f_AC4R

Case 4: Not-in-place operation on floating-point planar data

```
IppStatus ippiGammaFwd_32f_P3R (const Ipp32f* pSrc[3], int srcStep, Ipp32f* pDst[3],
int dstStep, IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);
```

Case 5: In-place operation on integer pixel-order data

```
IppStatus ippiGammaFwd_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for `mod`:

8u_C3IR	16u_C3IR
8u_AC4IR	16u_AC4IR

Case 6: In-place operation on integer planar data

```
IppStatus ippiGammaFwd_<mod>(Ipp<datatype>* pSrcDst[3], int srcDstStep, IppiSize
roiSize);
```

Supported values for `mod`:

8u_IP3R	16u_IP3R

Case 7: In-place operation on floating-point pixel-order data

```
IppStatus ippiGammaFwd_<mod>(Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize, Ipp32f
vMin, Ipp32f vMax);
```

Supported values for mod:

32f_C3IR
32f_AC4IR

Case 8: In-place operation on floating-point planar data

```
IppStatus ippiGammaFwd_32f_IP3R (Ipp32f* pSrcDst[3], int srcDstStep, IppiSize roiSize,
Ipp32f vMin, Ipp32f vMax);
```

Include Files

ippcc.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Pointer to the ROI in the pixel-order source image. Array of pointers to the ROI in each plane of the planar source image.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the ROI in the pixel-order destination image. Array of pointers to the ROI in each plane of the planar destination image.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
pSrcDst	Pointer to the source and destination image ROI for the in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
roiSize	Size of the source and destination ROI in pixels.
vMin, vMax	Minimum and maximum values of the input floating-point data.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function performs gamma-correction of the source image with RGB data. It uses the following basic equations to convert an RGB image to the gamma-corrected R'G'B' image:

for R, G, B < 0.018

R' = 4.5 * R

G' = 4.5 * G

$B' = 4.5 * B$

for $R, G, B \geq 0.018$

$R' = 1.099 * R^{0.45} - 0``.099``$

$G' = 1.099 * G^{0.45} - 0``.099``$

$B' = 1.099 * B^{0.45} - 0``.099``$

Note that the channel intensity values are normalized to fit in the range of [0..1]. The gamma value is equal to 1/0.45 = 2.22 in conformity with ITU709 specification.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if pSrc, pDst, or pSrcDst is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with a zero or negative value.
ippStsGammaRangeErr	Indicates an error condition if the input data bounds are incorrect, that is vMax is less than or equal to vMin.

GammaInv

Converts a gamma-corrected R'G'B' image back to the original RGB image.

Syntax

Case 1: Not-in-place operation on integer pixel-order data

```
IppStatus ippiGammaInv_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C3R	16u_C3R
8u_AC4R	16u_AC4R

Case 2: Not-in-place operation on integer planar data

```
IppStatus ippiGammaInv_<mod>(const Ipp<datatype>* pSrc[3], int srcStep, Ipp<datatype>*
pDst[3], int dstStep, IppiSize roiSize);
```

Supported values for mod:

8u_P3R	16u_P3R

Case 3: Not-in-place operation on floating-point pixel-order data

```
IppStatus ippiGammaInv_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int
dstStep, IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);
```

Supported values for mod:

32f_C3R
32f_AC4R

Case 4: Not-in-place operation on floating-point planar data

```
IppStatus ippiGammaInv_32f_P3R (const Ipp32f* pSrc[3], int srcStep, Ipp32f* pDst[3],
int dstStep, IppiSize roiSize, Ipp32f vMin, Ipp32f vMax);
```

Case 5: In-place operation on integer pixel-order data

```
IppStatus ippiGammaInv_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for `mod`:

8u_C3IR	16u_C3IR
8u_AC4IR	16u_AC4IR

Case 6: In-place operation on integer planar data

```
IppStatus ippiGammaInv_<mod>(Ipp<datatype>* pSrcDst[3], int srcDstStep, IppiSize
roiSize);
```

Supported values for `mod`:

8u_IP3R	16u_IP3R

Case 7: In-place operation on floating-point pixel-order data

```
IppStatus ippiGammaInv_<mod>(Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize, Ipp32f
vMin, Ipp32f vMax);
```

Supported values for `mod`:

32f_C3IR
32f_AC4IR

Case 8: In-place operation on floating-point planar data

```
IppStatus ippiGammaInv_32f_IP3R (Ipp32f* pSrcDst[3], int srcDstStep, IppiSize roiSize,
Ipp32f vMin, Ipp32f vMax);
```

Include Files

`ippcc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the ROI in the pixel-order source image. Array of pointers to the ROI in each plane of the planar source image.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.

pDst	Pointer to the ROI in the pixel-order destination image. Array of pointers to the ROI in each plane of the planar destination image.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
pSrcDst	Pointer to the source and destination image ROI for the in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
roiSize	Size of the source and destination ROI in pixels.
vMin, vMax	Minimum and maximum values of the input floating-point data.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts a gamma-corrected R'G'B' image back to the original RGB image. It uses the following equations:

for R', G', B' < 0.0812

R = R'/4.5

G = G'/4.5

B = B'/4.5

for R', G', B' ≥ 0.0812

$$R = \left(\frac{R' + 0.099}{1.099}\right)^{2.22}$$

$$G = \left(\frac{G' + 0.099}{1.099}\right)^{2.22}$$

$$B = \left(\frac{B' + 0.099}{1.099} \right)^{2.22}$$

Note that the channel intensity values are normalized to fit in the range of [0..1]. The gamma value is equal to 1/0.45 = 2.22 in conformity with ITU709 specification.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if `pSrc`, `pDst`, or `pSrcDst` is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize` has a field with a zero or negative value.
`ippStsGammaRangeErr`	Indicates an error condition if the input data bounds are incorrect, that is, `vMax` is less than or equal to `vMin`.

Intensity Transformation

The functions described in this section perform different types of intensity transformation including reduction of the intensity levels in each channel of the image, intensity transformation using lookup tables, and mapping high dynamic range image (HDRI) into low dynamic range image (LDRI).

- ReduceBitsGetBufferSize
- ReduceBits
- LUT_GetSize
- LUT_Init
- LUT
- LUTPalette, LUTPaletteSwap
- ToneMapLinear, ToneMapMean

ReduceBitsGetBufferSize

Computes the size of the work buffer for the `ippiReduceBits` function.

Syntax

```
IppStatus ippiReduceBitsGetBufferSize(IppChannels ippChan, IppiSize roiSize, int noise,
IppiDitherType dtype, int* pBufferSize);
```

Include Files

`ippcc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`ippChan`	Number of channels in the source images. Possible values: `ippC1`, `ippC3`, or `ippAC4`.
`roiSize`	Size, in pixels, of the source images.
`noise`	Number specifying the amount of noise added (as a percentage of the range [0..100]).
`dtype`	Type of dithering to be used. For the list of supported types, refer to the ippiReduceBits function description.
`pBufferSize`	Pointer to the computed value of the buffer size, in bytes.

Description

The function computes the size of the work buffer, in bytes, for the ippiReduceBits function and stores the result in the `pBufferSize` parameter.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error when any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error when `roiSize` is less than, or equal to zero.
`ippStsChannelErr`	Indicates an error when `ippChan` has an illegal value.
`ippStsDataTypeErr`	Indicates an error when `noise` is less than 0, or greater than 100.
`ippStsDataTypeErr`	Indicates an error when the specified dithering type is not supported.

See Also

ReduceBits Reduces the bit resolution of an image.

ReduceBits

Reduces the bit resolution of an image.

Syntax

Case 1: Operation on data of the same source and destination bit depths

```
IppStatus ippiReduceBits_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize roiSize, int noise, IppiDitherType dtype, int levels,
Ipp8u* pBuffer);
```

Supported values for `mod`:

8u_C1R	16u_C1R	16s_C1R
8u_C3R	16u_C3R	16s_C3R
8u_C4R	16u_C4R	16s_C4R

```
IppStatus ippiReduceBits_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize roiSize, int noise, IppiDitherType dtype, int levels,
Ipp8u* pBuffer);
```

Supported values for `mod`:

8u_AC4R	16u_AC4R	16s_AC4R

Case 2: Operation on data of different source and destination bit depths

```
IppStatus ippiReduceBits_8u1u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int
dstStep, int dstBitOffset, IppiSize roiSize, int noise, int seed, IppiDitherType dtype,
Ipp8u threshold, Ipp8u* pBuffer);
```

```
IppStatus ippiReduceBits_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize, int noise, IppiDitherType dtype,
int levels, Ipp8u* pBuffer);
```

Supported values for `mod`:

16u8u_C1R	16s8u_C1R	32f8u_C1R	32f16u_C1R	32f16s_C1R
16u8u_C3R	16s8u_C3R	32f8u_C3R	32f16u_C3R	32f16s_C3R
16u8u_C4R	16s8u_C4R	32f8u_C4R	32f16u_C4R	32f16s_C4R

```
IppStatus ippiReduceBits_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize, int noise, IppiDitherType dtype,
int levels, Ipp8u* pBuffer);
```

Supported values for `mod`:

16u8u_AC4R	16s8u_AC4R	32f8u_AC4R	32f16u_AC4R	32f16s_AC4R

Include Files

`ippcc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the source image ROI.

srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
dstBitOffset	Offset (in bits) in the first byte of the destination image row.
roiSize	Size of the source and destination ROI in pixels.
noise	The number specifying the amount of noise added. This parameter is set as a percentage of range [0..100].
seed	The seed value used by the pseudo-random number generation, should be set to 0.
dtype	The type of dithering to be used. The following types are supported: • ippDitherNone • No dithering is done • ippDitherStucki • The Stucki's error diffusion dithering algorithm is used • ippDitherFS • The Floid-Steinberg error diffusion dithering algorithm is used • ippDitherJJN • The Jarvice-Judice-Ninke error diffusion dithering algorithm is used • ippDitherBayer • The Bayer's threshold dithering algorithm is used
levels	The number of output levels for halftoning (dithering); can be varied in the range [2..(1<<depth)], where depth is the bit depth of the destination image.
threshold	Threshold level for Stucki's dithering for the function ippiReduceBits_8u1u_C1R.
pBuffer	Pointer to the buffer for internal calculations. To compute the size of the buffer, use the ReduceBitsGetBufferSize function.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function reduces the number of intensity levels in each channel of the source image pSrc and places the results in respective channels of the destination image pDst. Note that for floating point source data type, RGB values must be in the range [0..1].

The `levels` parameter sets the resultant number of intensity levels in each channel of the destination image.

If the `noise` value is greater than 0, some random noise is added to the threshold level used in computations. The amplitude of the noise signal is specified by the `noise` parameter set as a percentage of the destination image luminance range. For the 4x4 ordered dithering mode, the threshold value is determined by the dither matrix used, whereas for the error diffusion dithering mode the input threshold is set as half of the `range` value, where

```
range= ((1<<depth) - 1)/(levels- 1)
```

and `depth` is the bit depth of the source image.

For floating-point data type, `range = 1.0/(levels - 1)`.

8u to 1u conversion. Source image is converted to a bitonal image. The function `ippiReduceBits_8u1u_C1R` supports only one dithering algorithm - Stucki's error diffusion. The destination image has a `8u` data type, where each byte represents eight consecutive pixels of the bitonal image (1 bit per pixel). In this case, additional parameter `dstBitOffset` is required to specify the start position of the destination ROI buffer.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if `pSrc` or `pDst` is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize` has a field with a zero or negative value.
`ippStsNoiseValErr`	Indicates an error condition if `noise` has an illegal value.
`ippStsDitherTypeErr`	Indicates an error condition if the specified dithering type is not supported.
`ippStsDitherLevelsErr`	Indicates an error condition if `levels` value is out of admissible range.
`ippStsMemAllocErr`	Indicates an error condition if memory allocation fails.

See Also

ReduceBitsGetBufferSize Computes the size of the work buffer for the ippiReduceBits function.

LUT_GetSize

Computes the size of the LUT specification structure.

Syntax

```
IppStatus ippiLUT_GetSize(IppiInterpolationType interpolation, IppDataType dataType,
IppChannels channels, IppiSize roiSize, const int nLevels[], int* pSpecSize);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`interpolation`	Interpolation algorithm, possible values are: • `ippNearest` • Nearest neighbor interpolation. • `ippCubic` • Cubic interpolation. • `ippLinear` • Linear interpolation.
`dataType`	Data type of the image. Possible values are `ipp8u`, `ipp16u`, `ipp16s`, or `ipp32f`.
`channels`	Number of channels in the image. Possible values are: `ippC1`, `ippC3`, `ippC4`, or `ippAC4`.
`roiSize`	Size, in pixels, of the destination ROI.
`nLevels`	Number of levels, separate for each channel.
`pSpecSize`	Pointer to the computed size, in bytes, of the specification structure.

Description

This function computes the size of the specification structure for the ippiLUT function. The result is stored in the `pSpecSize` parameter.

For an example on how to use this function, refer to the example provided with the ippiLUT function description.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error when `roiSize` has a field with a value less than 1.
`ippStsChannelErr`	Indicates an error when `channel` has an illegal value.
`ippStsDataTypeErr`	Indicates an error when `dataType` has an illegal value.
`ippStsInterpolationErr`	Indicates an error when `interpolation` has an illegal value.

See Also

LUT MODIFIED API. Maps an image by applying intensity transformation.

LUT_Init

Initializes the LUT specification structure.

Syntax

```
IppStatus ippiLUT_Init_8u(IppiInterpolationType interpolation, IppChannels channels,
IppiSize roiSize, const Ipp32s* pValues[], const Ipp32s* pLevels[], int nLevels[],
IppiLUT_Spec* pSpec);
```

```
IppStatus ippiLUT_Init_16u(IppiInterpolationType interpolation, IppChannels channels,
IppiSize roiSize, const Ipp32s* pValues[], const Ipp32s* pLevels[], int nLevels[],
IppiLUT_Spec* pSpec);
```

```
IppStatus ippiLUT_Init_16s(IppiInterpolationType interpolation, IppChannels channels,
IppiSize roiSize, const Ipp32s* pValues[], const Ipp32s* pLevels[], int nLevels[],
IppiLUT_Spec* pSpec);
```

```
IppStatus ippiLUT_Init_32f(IppiInterpolationType interpolation, IppChannels channels,
IppiSize roiSize, const Ipp32f* pValues[], const Ipp32f* pLevels[], int nLevels[],
IppiLUT_Spec* pSpec);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`interpolation`	Interpolation algorithm, possible values are: • `ippNearest` • Nearest neighbor interpolation. • `ippCubic` • Cubic interpolation. • `ippLinear` • Linear interpolation.
`channels`	Number of channels in the image. Possible values are: `ippC1`, `ippC3`, `ippC4`, or `ippAC4`.
`roiSize`	Size, in pixels, of the destination ROI.
`pValues`	Pointer to the array with intensity values, separate for each channel.
`pLevels`	Pointer to the array with level values, separate for each channel.
`nLevels`	Number of levels, separate for each channel.
`pSpec`	Pointer to the LUT specification structure.

Description

This function initializes the specification structure for the ippiLUT function. To compute the size of the structure, use the ippiLUT_GetSize function.

Length of the `pLevels` and `pValues` arrays is defined by the `nLevels` parameter. Number of level and intensity values are `nLevels - 1`.

The `interpolation` parameter defines the mapping algorithm for the LUT function:

- **``ippNearest``**: Every source pixel `pSrc(x,y)` from the range `[pLevels[k],pLevels[k+1])` is mapped to the destination pixel `pDst(x,y)` which value is equal to `pValues[k]`.
- **``ippLinear``**: Every source pixel `pSrc(x, y)` from the range `[pLevels[k],pLevels[k+1])` is mapped to the destination pixel `pDst(x, y)` which value is computed according to the following formula:

  `pDst(x, y) = pValues[k]+(pSrc(x, y)-pLevels[k])*(pValues[k+1]-pValues[k])/(pLevels[k+1]-Levels[k])`
- **``ippCubic``**: Every source pixel `pSrc(x,y)` from the range `[pLevels[k],pLevels[k+1])` is mapped to the destination pixel `pDst(x,y)` which value is computed as

  `pDst(x,y)= A*pSrc(x,y)`^{3} + B * pSrc``(x,y)`` ^{2} + C * pSrc``(x,y)+ D.``

  The function operates on the assumption that the cubic polynomial curve passes through the following four points:

  `([pLevels[k-1],pLevels[k-1])`

  `([pLevels[k],pLevels[k])`

  `([pLevels[k+1],pLevels[k+1])`

  `([pLevels[k+2],pLevels[k+2])`

  Based on that, coefficients `A`, `B`, `C`, `D` are computed by solving the following set of linear equations:

  `A*pLevels[k-1]`^{3} + B * pLevels``[k-1]`` ^{2} + C * pLevels``[k-1] + D =pValues[k-1]``

  `A*pLevels[k]`^{3} + B * pLevels``[k]`` ^{2} + C * pLevels``[k] + D =pValues[k]``

  `A*pLevels[k+1]`^{3} + B * pLevels``[k+1]`` ^{2} + C * pLevels``[k+1] + D =pValues[k+1]``

  `A*pLevels[k+2]`^{3} + B * pLevels``[k+2]`` ^{2} + C * pLevels``[k+2] + D =pValues[k+2]``

Pixels in the `pSrc` image that are not in the range `[pLevels[0],pLevels[nLevels-1])` are copied to the `pDst` image without any transformation.

For an example on how to use this function, refer to the example provided with the ippiLUT function description.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error when `roiSize` has a field with a value less than 1.
`ippStsChannelErr`	Indicates an error when `channel` has an illegal value.
`ippStsLUTNofLevelsErr`	Indicates an error when `nLevels` is less than 2.

ippStsInterpolationErr	Indicates an error when interpolation has an illegal value.

See Also

LUT MODIFIED API. Maps an image by applying intensity transformation.

LUT_GetSize Computes the size of the LUT specification structure.

LUT

MODIFIED API. Maps an image by applying intensity transformation.

Syntax

Case 1: Not-in-place operation on one-channel integer data

```
IppStatus ippiLUT_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize roiSize, IppiLUT_Spec* pSpec);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R

Case 2: Not-in-place operation on multi-channel integer data

```
IppStatus ippiLUT_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize roiSize, IppiLUT_Spec* pSpec);
```

Supported values for mod:

8u_C3R	16u_C3R	16s_C3R
8u_AC4R	16u_AC4R	16s_AC4R

```
IppStatus ippiLUT_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize roiSize, IppiLUT_Spec* pSpec);
```

Supported values for mod:

8u_C4R	16u_C4R	16s_C4R

Case 3: Not-in-place operation on one-channel floating-point data

```
IppStatus ippiLUT_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep,
IppiSize roiSize, IppiLUT_Spec* pSpec);
```

Case 4: Not-in-place operation on multi-channel floating-point data

```
IppStatus ippiLUT_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep,
IppiSize roiSize, IppiLUT_Spec* pSpec);
```

Supported values for mod:

32f_C3R
32f_AC4R

```
IppStatus ippiLUT_32f_C4R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep,
IppiSize roiSize, IppiLUT_Spec* pSpec);
```

Case 5: In-place operation on one-channel integer data

```
IppStatus ippiLUT_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize,
IppiLUT_Spec* pSpec);
```

Supported values for `mod`:

`8u_C1IR`	`16u_C1IR`	`16s_C1IR`

Case 6: In-place operation on multi-channel integer data

```
IppStatus ippiLUT_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize,
IppiLUT_Spec* pSpec);
```

Supported values for `mod`:

`8u_C3IR`	`16u_C3IR`	`16s_C3IR`
`8u_AC4IR`	`16u_AC4IR`	`16s_AC4IR`

```
IppStatus ippiLUT_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize,
IppiLUT_Spec* pSpec);
```

Supported values for `mod`:

`8u_C4IR`	`16u_C4IR`	`16s_C4IR`

Case 7: In-place operation on one-channel floating-point data

```
IppStatus ippiLUT_32f_C1IR(Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize,
IppiLUT_Spec* pSpec);
```

Case 8: In-place operation on multi-channel floating-point data

```
IppStatus ippiLUT_<mod>(Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize,
IppiLUT_Spec* pSpec);
```

Supported values for `mod`:

`32f_C3IR`
`32f_AC4IR`

```
IppStatus ippiLUT_32f_C4IR(Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize,
IppiLUT_Spec* pSpec);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance, in bytes, between the starting points of consecutive lines in the source image.

pDst	Pointer to the destination image ROI.
dstStep	Distance, in bytes, between the starting points of consecutive lines in the destination image.
pSrcDst	Pointer to the source and destination mage ROI for the in-place operation.
srcDstStep	Distance, in bytes, between the starting points of consecutive lines in the source and destination image for the in-place operation.
roiSize	Size of the source ROI, in pixels.
pSpec	Pointer to the LUT specification structure.

Description

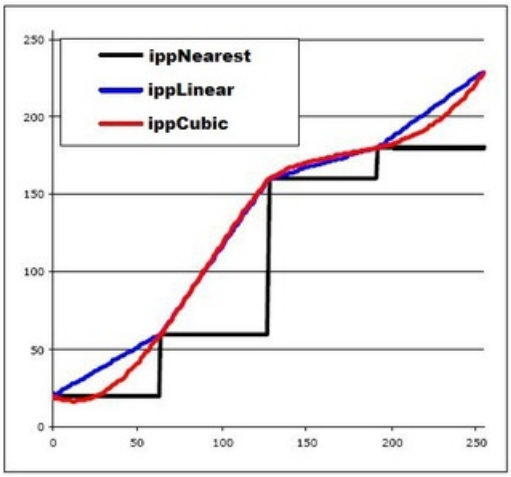
NOTE
Important
The API of this function has been modified in Intel IPP 9.0 release.

This function operates with ROI (see Regions of Interest in Intel IPP).

Before using this function, you need to compute the size of the specification structure using the LUT_GetSize function and initialize the structure using LUT_Init.

This function performs intensity transformation of the source image `pSrc` using the lookup table (LUT) specified by the arrays `pLevels`, `pValues`, and `interpolation` type specified in the LUT_Init function when `pSpec` is initialized.

The figure below shows particular curves that are used in all the `ippiLUT` function flavors for mapping. The level values are 0, 64, 128, 192, 256; the intensity values are 20, 60, 160, 180, 230.



Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when roiSize has a field with a value less than 1.
ippStsStepErr	Indicates an error when srcStep, dstStep, or srcDstStep has a zero or negative value.
ippStsBadArgErr	Indicates an error when pSpec initialization is incorrect.

Example

The code example below demonstrates how to use LUT_GetSize, LUT_Init, and ippiLUT functions.

```
#include "ippcore.h"
#include "ippi.h"
#include <iostream>
#include <iomanip>


void func_LUTLinear()
{
    IppStatus status;
    Ipp32f pSrc[8 * 8];
    int srcStep = 8 * sizeof(Ipp32f);
    IppiSize roiSize = { 8, 8 };


    Ipp32f pDst[8 * 8];
    int dstStep = 8 * sizeof(Ipp32f);
    Ipp32f pLevels[5] = { 0.0, 0.128, 0.256, 0.512, 1.0 };
    const Ipp32f* ppLevels[1] = { pLevels };
    Ipp32f pValues[5] = { 0.2, 0.4, 0.6, 0.8, 1.0 };
    const Ipp32f* ppValues[1] = { pValues };
    int nLevels[1] = { 5 };
    int specSize;
    IppiLUT_Spec* pSpec;


    status = ippiImageJaehne_32f_C1R(pSrc, srcStep, roiSize);


    std::cout << "pSrc:\n";
    for (int row = 0; row < roiSize.height; row++) {
        for (int col = 0; col < roiSize.width; col++) {
            std::cout << std::fixed << std::setprecision(2) <<pSrc[roiSize.width * row + col] <<
"\t";
        }
        std::cout << "\n";
    }
```

```
    ippiLUT_GetSize(ippLinear, ipp32f, ippC1, roiSize, nLevels, &specSize);

    pSpec = (IppiLUT_Spec*)ippMalloc(specSize);

    ippiLUT_Init_32f(ippLinear, ippC1, roiSize, ppValues, ppLevels, nLevels, pSpec);
    status = ippiLUT_32f_C1R(pSrc, srcStep, pDst, dstStep, roiSize, pSpec);

    std::cout << "pDst:\n";
    for (int row = 0; row < roiSize.height; row++) {
        for (int col = 0; col < roiSize.width; col++) {
            std::cout << pDst[roiSize.width * row + col] << "\t";
        }
        std::cout << "\n";
    }

    ippFree(pSpec);
}
```

Result:

```
pSrc:
0.00 0.26 0.65 0.82 0.82 0.65 0.26 0.00
0.26 0.82 1.00 0.98 0.98 1.00 0.82 0.26
0.65 1.00 0.89 0.74 0.74 0.89 1.00 0.65
0.82 0.98 0.74 0.55 0.55 0.74 0.98 0.82
0.82 0.98 0.74 0.55 0.55 0.74 0.98 0.82
0.65 1.00 0.89 0.74 0.74 0.89 1.00 0.65
0.26 0.82 1.00 0.98 0.98 1.00 0.82 0.26
0.00 0.26 0.65 0.82 0.82 0.65 0.26 0.00

pDst:
0.20 0.61 0.85 0.93 0.93 0.85 0.61 0.20
0.61 0.93 1.00 0.99 0.99 1.00 0.93 0.61
0.85 1.00 0.95 0.89 0.89 0.95 1.00 0.85
0.93 0.99 0.89 0.82 0.82 0.89 0.99 0.93
0.93 0.99 0.89 0.82 0.82 0.89 0.99 0.93
0.85 1.00 0.95 0.89 0.89 0.95 1.00 0.85
0.61 0.93 1.00 0.99 0.99 1.00 0.93 0.61
0.20 0.61 0.85 0.93 0.93 0.85 0.61 0.20
```

See Also

Regions of Interest in Intel IPP

LUT_GetSize Computes the size of the LUT specification structure.

LUT_Init Initializes the LUT specification structure.

LUTPalette, LUTPaletteSwap

Maps an image by applying intensity transformation in accordance with a palette table.

Syntax

Case 1: Operations on one-channel data

```
IppStatus ippiLUTPalette_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize, const Ipp<dstDatatype>* pTable,
int nBitSize);
```

Supported values for `mod`:

`8u_C1R`	`16u_C1R`
`8u32u_C1R`	`16u8u_C1R`
	`16u32u_C1R`

```
IppStatus ippiLUTPalette_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp8u* pDst,
int dstStep, IppiSize roiSize, const Ipp8u* pTable, int nBitSize);
```

Supported values for `mod`:

`8u24u_C1R`	`16u24u_C1R`

Case 2: Operations on multi-channel data

```
IppStatus ippiLUTPalette_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize roiSize, const Ipp<datatype>* const pTable[3], int
nBitSize);
```

Supported values for `mod`:

`8u_C3R`	`16u_C3R`
`8u_AC4R`	`16u_AC4R`

```
IppStatus ippiLUTPalette_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize roiSize, const Ipp<datatype>* const pTable[4], int
nBitSize);
```

Supported values for `mod`:

`8u_C4R`	`16u_C4R`

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI.

dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source ROI in pixels.
pTable	Pointer to the palette table, or an array of pointers to the palette tables for each source channel.
nBitSize	Number of significant bits in the source image.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

The function ippiLUTPalette performs intensity transformation of the source image pSrc using the palette lookup table pTable. This table is a vector with $2^{nBitSize}$ elements that contain intensity values specified by the user. The function uses nBitSize lower bits of intensity value of each source pixel as an index in the pTable and assigns the correspondent intensity value from the table to the respective pixel in the destination image pDst. The number of significant bits nBitSize should be in the range [1, 8] for functions that operate on 8u source images, and [1, 16] for functions that operate on 16u source images.

Some function flavors that operate on the 3-channel source image additionally create a 4-th channel - alpha channel - in the destination image and place it at first position. The channel values of the alpha channel can be set to the arbitrary constant value alhaValue. If this value is less than 0 or greater than the upper boundary of the data range, the channel values are not set.

The function flavor ippiLUTPaletteSwap reverses the order of channels in the destination image.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with a zero or negative value.
ippStsOutOfRangeErr	Indicates an error if nBitSize is out of the range.

Example

The code example below shows how to use the function ippiLUTPalette_8u32u_C1R.

```
#include "ippcore.h"
#include "ippi.h"
#include <iostream>


void func_LUTPalette()
{
    IppStatus status;
    Ipp8u pSrc[8 * 8];
    int srcStep = 8 * sizeof(Ipp8u);
    IppiSize roiSize = { 8, 8 };
```

```
    Ipp32u pDst[8 * 8];
    int dstStep = 8 * sizeof(Ipp32f);
    int nBitSize = 3;
    Ipp32u pTable[8] = { 1, 2, 3, 4, 5, 6, 7, 8 };


    status = ippiImageJaehne_8u_C1R(pSrc, srcStep, roiSize);


    std::cout << "pSrc:\n";
    for (int row = 0; row < roiSize.height; row++) {
        for (int col = 0; col < roiSize.width; col++) {
            std::cout << (int)pSrc[roiSize.width * row + col] << "\t";
        }
        std::cout << "\n";
    }


    status = ippiLUTPalette_8u32u_C1R(pSrc, srcStep, pDst, dstStep, roiSize, pTable, nBitSize);


    std::cout << "pDst:\n";
    for (int row = 0; row < roiSize.height; row++) {
        for (int col = 0; col < roiSize.width; col++) {
            std::cout << pDst[roiSize.width * row + col] << "\t";
        }
        std::cout << "\n";
    }
}
```

Result:

```
pSrc:
1 68 165 209 209 165 68 1
68 209 255 250 250 255 209 68
165 255 227 188 188 227 255 165
209 250 188 141 141 188 250 209
209 250 188 141 141 188 250 209
165 255 227 188 188 227 255 165
68 209 255 250 250 255 209 68
1 68 165 209 209 165 68 1




pDst:
2 5 6 2 2 6 5 2
5 2 8 3 3 8 2 5
6 8 4 5 5 4 8 6
2 3 5 6 6 5 3 2
2 3 5 6 6 5 3 2
6 8 4 5 5 4 8 6
5 2 8 3 3 8 2 5
2 5 6 2 2 6 5 2
```

ToneMapLinear, ToneMapMean

Maps an HDRI image to the LDRI image.

Syntax

```
IppStatus ippiToneMapLinear_32f8u_C1R(const Ipp32f* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiToneMapMean_32f8u_C1R(const Ipp32f* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

Include Files

`ippcc.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the ROI in the HDRI source image.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`pDst`	Pointer to the ROI in the LDRI destination image.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image.
`roiSize`	Size of the source and destination ROI in pixels.

Description

They both operate with ROI (see Regions of Interest in Intel IPP).

These functions convert the source high dynamic range image (HDRI) `pSrc` into low dynamic range image (LDRI) `pDst`. Pixel values of the source image must be positive.

The function `ippiToneMapLinear` implements the Linear Scale-Factor method converting each source pixel `pSrc[i]` in accordance with the formula:

`pSrc[i] =pSrc[i]/Lmax,Lmax= max{pSrc[i]}.`

The function `ippiToneMapMean` implements the Mean Value method converting each source pixel `pSrc[i]` in accordance with the formula:

`pSrc[i] = 0.5*pSrc[i]/Lave,Lave= average{pSrc[i]}.`

If the value of `Lmax` or `Lave` is less than 0, then the function does not perform the operation and returns the warning message.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if `pSrc` or `pDst` is `NULL`.

ippStsSizeErr	Indicates an error condition if roiSize has a field with a zero or negative value.
ippStsNoOperation	Indicates a warning if the values of Lmax or Lave are less than 0.

Image Bit Conversion

Functions described in this section convert images from one bit format to another.

- Unpack
- Pack

Unpack

Converts 12-bit image to 16-bit image by padding with 4 zero bits.

Syntax

```
IppStatus ippiUnpack_12u16u_C1R(Ipp8u* pSrc, int srcStep, Ipp16u* pDst, int dstStep,
IppiSize roiSize);
```

Include Files

ippcc.h

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Steps throught the source image.
pDst	Pointer to the destination image ROI.
dstStep	Steps throught the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts 12-bit image to 16-bit image by padding with 4 zero bits.

For 12-bit images, neighboring 2 pixels occupy 3 bytes and occupy memory space in the following order:

```
bits      7         6         5         4         3         2         1         0
Byte 0: P0[11]    P0[10]    P0[ 9]    P0[ 8]    P0[ 7]    P0[ 6]    P0[ 5]    P0[ 4]
Byte 1: P0[ 3]    P0[ 2]    P0[ 1]    P0[ 0]    P1[11]    P1[10]    P1[ 9]    P1[ 8]
Byte 2: P1[ 7]    P1[ 6]    P1[ 5]    P1[ 4]    P1[ 3]    P1[ 2]    P1[ 1]    P1[ 0]
```

If the width of the image is odd, the last pixel occupies 2 bytes.

Despite the fact that the pointer to a 12-bit image is of the Ipp8u* type, in fact, such an image occupies at least roiSize.height*roiSize.width*(12/8) bytes, satisfying the condition srcStep >= roiSize.width*(12/8).

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if one of the pointers is NULL.
ippStsStepErr	Indicates an error condition if srcStep or dstStep is less than or equal to zero.
ippStsSizeErr	Indicates an error condition if roiSize has a field with a zero or negative value.

Pack

Converts 16-bit image to 12-bit image by discarding 4 upper bits (MSB).

Syntax

```
IppStatus ippiPack_16u12u_C1R(Ipp16u* pSrc, int srcStep, Ipp8u* pDst, int dstStep,
IppiSize roiSize);
```

Include Files

ippcc.h

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Steps throught the source image.
pDst	Pointer to the destination image ROI.
dstStep	Steps throught the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts 16-bit image to 12-bit image by discarding 4 upper bits (MSB).

For 12-bit images, neighboring 2 pixels occupy 3 bytes and occupy memory space in the following order:

```
bits      7         6         5         4         3         2         1         0
Byte 0: P0[11]    P0[10]    P0[ 9]    P0[ 8]    P0[ 7]    P0[ 6]    P0[ 5]    P0[ 4]
Byte 1: P0[ 3]    P0[ 2]    P0[ 1]    P0[ 0]    P1[11]    P1[10]    P1[ 9]    P1[ 8]
Byte 2: P1[ 7]    P1[ 6]    P1[ 5]    P1[ 4]    P1[ 3]    P1[ 2]    P1[ 1]    P1[ 0]
```

If the width of the image is odd, the last pixel occupies 2 bytes.

Despite the fact that the pointer to a 12-bit image is of the Ipp8u* type, in fact, such an image occupies at least roiSize.height*roiSize.width*(12/8) bytes, satisfying the condition srcStep >= roiSize.width*(12/8).

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if one of the pointers is NULL.
ippStsStepErr	Indicates an error condition if srcStep or dstStep is less than or equal to zero.
ippStsSizeErr	Indicates an error condition if roiSize has a field with a zero or negative value.

Threshold and Compare Operations

This chapter describes the Intel® IPP image processing functions that operate on a pixel-by-pixel basis: threshold and compare functions.

- Thresholding
- Compare Operations

Thresholding

The threshold functions change pixel values depending on whether they are less or greater than the specified threshold.

The type of comparison operation used to threshold pixel values is specified by the ippCmpOp parameter; this operation can be either "greater than" or "less than" (see Structures and Enumerators for more information). For some thresholding functions the type of comparison operation is fixed.

If an input pixel value satisfies the compare condition, the corresponding output pixel is set to the fixed value that is specific for a given threshold function flavor. Otherwise, it is either not changed, or set to another fixed value, which is defined in a particular function description.

For images with multi-channel data, the compare conditions should be set separately for each channel.

- Threshold
- Threshold_GT
- Threshold_LT
- Threshold_Val
- Threshold_GTVal
- Threshold_LTVal
- Threshold_LTValGTVal
- ComputeThreshold_Otsu

Threshold

Performs thresholding of pixel values in an image buffer.

Syntax

Case 1: Not-in-place operation on one-channel data

```
IppStatus ippiThreshold_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize roiSize, Ipp<datatype> threshold, IppCmpOp ippCmpOp);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32f_C1R

Case 2: Not-in-place operation on multi-channel data

```
IppStatus ippiThreshold_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize roiSize, const Ipp<datatype> threshold[3], IppCmpOp
ippCmpOp);
```

Supported values for `mod`:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

Case 3: In-place operation on one-channel data

```
IppStatus ippiThreshold_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize,
Ipp<datatype> threshold, IppCmpOp ippCmpOp);
```

Supported values for `mod`:

8u_C1IR	16u_C1IR	16s_C1IR	32f_C1IR

Case 4: In-place operation on multi-channel data

```
IppStatus ippiThreshold_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize,
const Ipp<datatype> threshold[3], IppCmpOp ippCmpOp);
```

Supported values for `mod`:

8u_C3IR	16u_C3IR	16s_C3IR	32f_C3IR
8u_AC4IR	16u_AC4IR	16s_AC4IR	32f_AC4IR

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image.
`pSrcDst`	Pointer to the source and destination image ROI (for the in-place operation).

srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image buffer (for the in-place operation).
roiSize	Size of the source and destination ROI in pixels.
threshold	The threshold level value to use for each pixel. In case of multi-channel data, an array of threshold values for each color channel is used.
ippCmpOp	The operation specified for comparing pixel values and the threshold. Comparison for either "less than" or "greater than" can be used.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function thresholds pixels in the source image pSrc using the specified level threshold. Pixel values in the source image are compared to the threshold value according to the type of comparison operation specified in the ippCmpOp. The following values for ippCmpOp are possible:

- ippCmpLess specifies the "less than" comparison and defines the threshold value as a lower bound. Comparison is performed by the following formula:

$$pDst[n] = \begin{cases} threshold, & pSrc[n] < threshold \\ pSrc[n], & otherwise \end{cases}$$

- ippCmpGreater specifies the "greater than" comparison and defines the threshold value as an upper bound. Comparison is performed by the following formula:

$$pDst[n] = \begin{cases} threshold, & pSrc[n] > threshold \\ pSrc[n], & otherwise \end{cases}$$

If the result of comparison is true, the corresponding output pixel is set to the threshold value. Otherwise, it is set to the source pixel value.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointer is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if srcStep, dstStep, or srcDstStep has a zero or negative value

ippStsNotSupportedModeErr	Indicates an error if the comparison mode is not supported.

Example

The code example below shows how to use the ippiThreshold_8u_C1R function.

```
void func_threshold()
{
    IppiSize ROI = {5,4};
    Ipp8u src[9*4] = {1, 2, 4, 8, 16, 8, 4, 2, 1,
                      1, 2, 4, 8, 16, 8, 4, 2, 1,
                      1, 2, 4, 8, 16, 8, 4, 2, 1,
                      1, 2, 4, 8, 16, 8, 4, 2, 1};
    Ipp8u dst[9*4];
    Ipp8u threshold = 6;
    ippiThreshold_8u_C1R(src, 9, dst, 9, ROI, threshold, ippCmpGreater);


}
```

Result:

```
      dst
1  2  4  6  6  8  4  2  1
1  2  4  6  6  8  4  2  1
1  2  4  6  6  8  4  2  1
1  2  4  6  6  8  4  2  1
```

Threshold_GT

Performs thresholding of pixel values in an image, using the comparison for “greater than”.

Syntax

Case 1: Not-in-place operation on one-channel data

```
IppStatus ippiThreshold_GT_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize roiSize, Ipp<datatype> threshold);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32f_C1R

Case 2: Not-in-place operation on multi-channel data

```
IppStatus ippiThreshold_GT_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize roiSize, const Ipp<datatype> threshold[3]);
```

Supported values for mod:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

Case 3: In-place operation on one-channel data

```
IppStatus ippiThreshold_GT_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize
roiSize, Ipp<datatype> threshold);
```

Supported values for mod:

8u_C1IR	16u_C1IR	16s_C1IR	32f_C1IR

Case 4: In-place operation on multi-channel data

```
IppStatus ippiThreshold_GT_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize
roiSize, const Ipp<datatype> threshold[3]);
```

Supported values for `mod`:

8u_C3IR	16u_C3IR	16s_C3IR	32f_C3IR
8u_AC4IR	16u_AC4IR	16s_AC4IR	32f_AC4IR

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image.
`pSrcDst`	Pointer to the source and destination image ROI (for the in-place operation).
`srcDstStep`	Distance in bytes between starts of consecutive lines in the source and destination image buffer (for the in-place operation).
`roiSize`	Size of the source and destination ROI in pixels.
`threshold`	The threshold level value to use for each pixel. In case of multi-channel data, an array of threshold values for each color channel is used.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function performs thresholding of pixels in the source image `pSrc` using the specified level `threshold`. Pixel values in the source image are compared to the `threshold` value for "greater than".

If the result of the compare is true, the corresponding output pixel is set to the `threshold` value. Otherwise, it is set to the source pixel value.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointer is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if srcStep, dstStep, or srcDstStep has a zero or negative value.

Threshold_LT

Performs thresholding of pixel values in an image buffer, using the comparison for “less than”.

Syntax

Case 1: Not-in-place operation on one-channel data

```
IppStatus ippiThreshold_LT_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize roiSize, Ipp<datatype> threshold);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32f_C1R

Case 2: Not-in-place operation on multi-channel data

```
IppStatus ippiThreshold_LT_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize roiSize, const Ipp<datatype> threshold[3]);
```

Supported values for mod:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

Case 3: In-place operation on one-channel data

```
IppStatus ippiThreshold_LT_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize
roiSize, Ipp<datatype> threshold);
```

Supported values for mod:

8u_C1IR	16u_C1IR	16s_C1IR	32f_C1IR

Case 4: In-place operation on multi-channel data

```
IppStatus ippiThreshold_LT_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize
roiSize, const Ipp<datatype> threshold[3]);
```

Supported values for mod:

8u_C3IR	16u_C3IR	16s_C3IR	32f_C3IR
8u_AC4IR	16u_AC4IR	16s_AC4IR	32f_AC4IR

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
pSrcDst	Pointer to the source and destination image ROI (for the in-place operation).
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image buffer (for the in-place operation).
roiSize	Size of the source and destination ROI in pixels.
threshold	The threshold level value to use for each pixel. In case of multi-channel data, an array of threshold values for each color channel is used.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function performs thresholding of pixels in the source image pSrc using the specified level threshold. Pixel values in the source image are compared to the threshold value for "less than". If the result of the compare is true, the corresponding output pixel is set to the threshold value. Otherwise, it is set to the source pixel value.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointer is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if srcStep, dstStep, or srcDstStep has a zero or negative value.

Threshold_Val

Performs thresholding of pixel values in an image buffer. Pixels that satisfy the compare condition are set to a specified value.

Syntax

Case 1: Not-in-place operation on one-channel data

```
IppStatus ippiThreshold_Val_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, Ipp<datatype> threshold,
Ipp<datatype> value, IppCmpOp ippCmpOp);
```

Supported values for `mod`:

8u_C1R	16u_C1R	16s_C1R	32f_C1R

Case 2: Not-in-place operation on multi-channel data

```
IppStatus ippiThreshold_Val_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const Ipp<datatype> threshold[3],
const Ipp<datatype> value[3], IppCmpOp ippCmpOp);
```

Supported values for `mod`:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

Case 3: In-place operation on one-channel data

```
IppStatus ippiThreshold_Val_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize
roiSize, Ipp<datatype> threshold, Ipp<datatype> value, IppCmpOp ippCmpOp);
```

Supported values for `mod`:

8u_C1IR	16u_C1IR	16s_C1IR	32f_C1IR

Case 4: In-place operation on multi-channel data

```
IppStatus ippiThreshold_Val_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize
roiSize, const Ipp<datatype> threshold[3], const Ipp<datatype> value[3], IppCmpOp
ippCmpOp);
```

Supported values for `mod`:

8u_C3IR	16u_C3IR	16s_C3IR	32f_C3IR
8u_AC4IR	16u_AC4IR	16s_AC4IR	32f_AC4IR

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
pSrcDst	Pointer to the source and destination image ROI (for the in-place operation).
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image buffer (for the in-place operation).
roiSize	Size of the source and destination ROI in pixels.
threshold	The threshold value to use for each pixel.In case of multi-channel data, an array of 3 threshold values (one for each color channel) is used.
value	The output value to be set for each pixel that satisfies the compare condition. In case of multi-channel data, an array of 3 output values (one for each color channel) is used.
ippCmpOp	The operation to use for comparing pixel values and the threshold. Comparison for either “less than” or “greater than” can be used.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function thresholds pixels in the source image pSrc using the specified level threshold. Pixel values in the source image are compared to the threshold value using the ippCmpOp comparison operation. If the result of the compare is true, the corresponding output pixel is set to the specified value. Otherwise, it is set to the source pixel value.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointer is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if srcStep, dstStep, or srcDstStep has a zero or negative value.

Threshold_GTVal

Performs thresholding of pixel values in an image. Pixels that are greater than threshold, are set to a specified value.

Syntax

Case 1: Not-in-place operation on one-channel data

```
IppStatus ippiThreshold_GTVal_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, Ipp<datatype> threshold,
Ipp<datatype> value);
```

Supported values for `mod`:

8u_C1R	16u_C1R	16s_C1R	32f_C1R

Case 2: Not-in-place operation on multi-channel data

```
IppStatus ippiThreshold_GTVal_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const Ipp<datatype> threshold[3],
const Ipp<datatype> value[3]);
```

Supported values for `mod`:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

```
IppStatus ippiThreshold_GTVal_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const Ipp<datatype> threshold[4],
const Ipp<datatype> value[4]);
```

Supported values for `mod`:

8u_C4R	16u_C4R	16s_C4R	32f_C4R

Case 3: In-place operation on one-channel data

```
IppStatus ippiThreshold_GTVal_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize
roiSize, Ipp<datatype> threshold, Ipp<datatype> value);
```

Supported values for `mod`:

8u_C1IR	16u_C1IR	16s_C1IR	32f_C1IR

Case 4: In-place operation on multi-channel data

```
IppStatus ippiThreshold_GTVal_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize
roiSize, const Ipp<datatype> threshold[3], const Ipp<datatype> value[3]);
```

Supported values for `mod`:

8u_C3IR	16u_C3IR	16s_C3IR	32f_C3IR
8u_AC4IR	16u_AC4IR	16s_AC4IR	32f_AC4IR

```
IppStatus ippiThreshold_GTVal_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize
roiSize, const Ipp<datatype> threshold[4], const Ipp<datatype> value[4]);
```

Supported values for `mod`:

8u_C4IR	16u_C4IR	16s_C4IR	32f_C4IR

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
pSrcDst	Pointer to the source and destination image ROI (for the in-place operation).
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image buffer (for the in-place operation).
roiSize	Size of the source and destination ROI in pixels.
threshold	The threshold value to use for each pixel.In case of multi-channel data, an array of threshold values (one for each channel) is used.
value	The output value to be set for each pixel that satisfies the compare condition. In case of multi-channel data, an array of output values (one for each channel) is used.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function thresholds pixels in the source image pSrc using the specified level threshold. Pixel values in the source image are compared to the threshold value for "greater than".

If the result of the compare is true, the corresponding output pixel is set to the specified value. Otherwise, it is set to the source pixel value.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr	Indicates an error condition if one of the specified pointer is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if srcStep, dstStep, or srcDstStep has a zero or negative value.

Threshold_LTVal

Performs thresholding of pixel values in an image. Pixels that are less than threshold, are set to a specified value.

Syntax

Case 1: Not-in-place operation on one-channel data

```
IppStatus ippiThreshold_LTVal_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, Ipp<datatype> threshold,
Ipp<datatype> value);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32f_C1R

Case 2: Not-in-place operation on multi-channel data

```
IppStatus ippiThreshold_LTVal_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const Ipp<datatype> threshold[3],
const Ipp<datatype> value[3]);
```

Supported values for mod:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

```
IppStatus ippiThreshold_LTVal_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const Ipp<datatype> threshold[4],
const Ipp<datatype> value[4]);
```

Supported values for mod:

8u_C4R	16u_C4R	16s_C4R	32f_C4R

Case 3: In-place operation on one-channel data

```
IppStatus ippiThreshold_LTVal_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize
roiSize, Ipp<datatype> threshold, Ipp<datatype> value);
```

Supported values for mod:

8u_C1IR	16u_C1IR	16s_C1IR	32f_C1IR

Case 4: In-place operation on multi-channel data

```
IppStatus ippiThreshold_LTVal_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize
roiSize, const Ipp<datatype> threshold[3], const Ipp<datatype> value[3]);
```

Supported values for `mod`:

`8u_C3IR`	`16u_C3IR`	`16s_C3IR`	`32f_C3IR`
`8u_AC4IR`	`16u_AC4IR`	`16s_AC4IR`	`32f_AC4IR`

```
IppStatus ippiThreshold_LTVal_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize
roiSize, const Ipp<datatype> threshold[4], const Ipp<datatype> value[4]);
```

Supported values for `mod`:

`8u_C4IR`	`16u_C4IR`	`16s_C4IR`	`32f_C4IR`

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image.
`pSrcDst`	Pointer to the source and destination image ROI (for the in-place operation).
`srcDstStep`	Distance in bytes between starts of consecutive lines in the source and destination image buffer (for the in-place operation).
`roiSize`	Size of the source and destination ROI in pixels.
`threshold`	The threshold value to use for each pixel.In case of multi-channel data, an array of threshold values (one for each channel) is used.
`value`	The output value to be set for each pixel that satisfies the compare condition. In case of multi-channel data, an array of output values (one for each channel) is used.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function thresholds pixels in the source image `pSrc` using the specified level `threshold`. Pixel values in the source image are compared to the `threshold` value for “less than”. If the result of the compare is true, the corresponding output pixel is set to the specified `value`. Otherwise, it is set to the source pixel value.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointer is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if srcStep, dstStep, or srcDstStep has a zero or negative value.

Threshold_LTValGTVal

Performs double thresholding of pixel values in an image buffer.

Syntax

Case 1: Not-in-place operation on one-channel data

```
IppStatus ippiThreshold_LTValGTVal_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, Ipp<datatype> thresholdLT,
Ipp<datatype> valueLT, Ipp<datatype> thresholdGT, Ipp<datatype> valueGT);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32f_C1R

Case 2: Not-in-place operation on multi-channel data

```
IppStatus ippiThreshold_LTValGTVal_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, const Ipp<datatype> thresholdLT[3],
const Ipp<datatype> valueLT[3], const Ipp<datatype> thresholdGT[3], const Ipp<datatype>
valueGT[3]);
```

Supported values for mod:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

Case 3: In-place operation on one-channel data

```
IppStatus ippiThreshold_LTValGTVal_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
IppiSize roiSize, Ipp<datatype> thresholdLT, Ipp<datatype> valueLT, Ipp<datatype>
thresholdGT, Ipp<datatype> valueGT);
```

Supported values for mod:

8u_C1IR	16u_C1IR	16s_C1IR	32f_C1IR

Case 4: In-place operation on multi-channel data

```
IppStatus ippiThreshold_LTValGTVal_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep,
IppiSize roiSize, const Ipp<datatype> thresholdLT[3], const Ipp<datatype> valueLT[3],
const Ipp<datatype> thresholdGT[3], const Ipp<datatype> valueGT[3]);
```

Supported values for mod

8u_C3IR	16u_C3IR	16s_C3IR	32f_C3IR
8u_AC4IR	16u_AC4IR	16s_AC4IR	32f_AC4IR

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
pSrcDst	Pointer to the source and destination image ROI (for the in-place operation).
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image buffer (for the in-place operation).
roiSize	Size of the source and destination ROI in pixels.
thresholdLT	The lower threshold value to use for each pixel.In case of multi-channel data, an array of three lower threshold values (one for each color channel) is used.
valueLT	The lower output value to be set for each pixel that is less than thresholdLT. In case of multi-channel data, an array of 3 lower output values (one for each color channel) is used.
thresholdGT	The upper threshold value to use for each pixel.In case of multi-channel data, an array of three upper threshold values (one for each color channel) is used.
valueGT	The upper output value to be set for each pixel that exceeds thresholdGT. In case of multi-channel data, an array of three upper output values (one for each color channel) is used.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function thresholds pixels in the source image pSrc using two specified levels thresholdLT and thresholdGT. Pixel values in the source image are compared to these levels. If the pixel value is less than thresholdLT, the corresponding output pixel is set to valueLT. If the pixel value is greater than thresholdGT, the output pixel is set to valueGT. Otherwise, it is set to the source pixel value. The value of thresholdLT should be less than or equal to thresholdGT.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointer is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value.
ippStsThresholdErr	Indicates an error when thresholdLT is greater than thresholdGT.
ippStsStepErr	Indicates an error condition if srcStep, dstStep, or srcDstStep has a zero or negative value.

Example

The code example below illustrates thresholding with two levels.

```
IppStatus threshold( void ) {
   Ipp8u x[5*4];
   IppiSize roi = {5,4};
   int i;


   for( i=0; i<5*4; ++i ) x[i] = (Ipp8u)i;


   return ippiThreshold_LTValGTVal_8u_C1IR( x, 5, roi, 2,1,6,7 );
}
```

The destination image x contains:

```
01 01 02 03 04
05 06 07 07 07
07 07 07 07 07
07 07 07 07 07
```

ComputeThreshold_Otsu

Computes the value of the Otsu threshold.

Syntax

```
IppStatus ippiComputeThreshold_Otsu_8u_C1R(const Ipp8u* pSrc, int srcStep, IppiSize roiSize, Ipp8u* pThreshold);
```

Include Files

ippi.h

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`roiSize`	Size of the image ROI in pixels.
`pThreshold`	Pointer to the Otsu threshold value.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function calculates the Otsu threshold (Otsu79) for the source image `pSrc` in accordance with the following formula:

$$\min_{T}(\sigma\,(pSrc(x,y)\le T)+\sigma\,(pSrc(x,y)>T))$$

where $0\le x <$ `roiSize.width`, $0\le y <$ `roiSize.height`, and `T` is the Otsu threshold.

$$\sigma\,(pSrc(x,y)) = \sqrt{\frac{\sum\limits_{\substack{x<roiSize.width\\ y<roiSize.height\\ x,y\ge 0}} (pSrc(x,y)-mean)^2}{roiSize.height\cdot roiSize.width}}$$

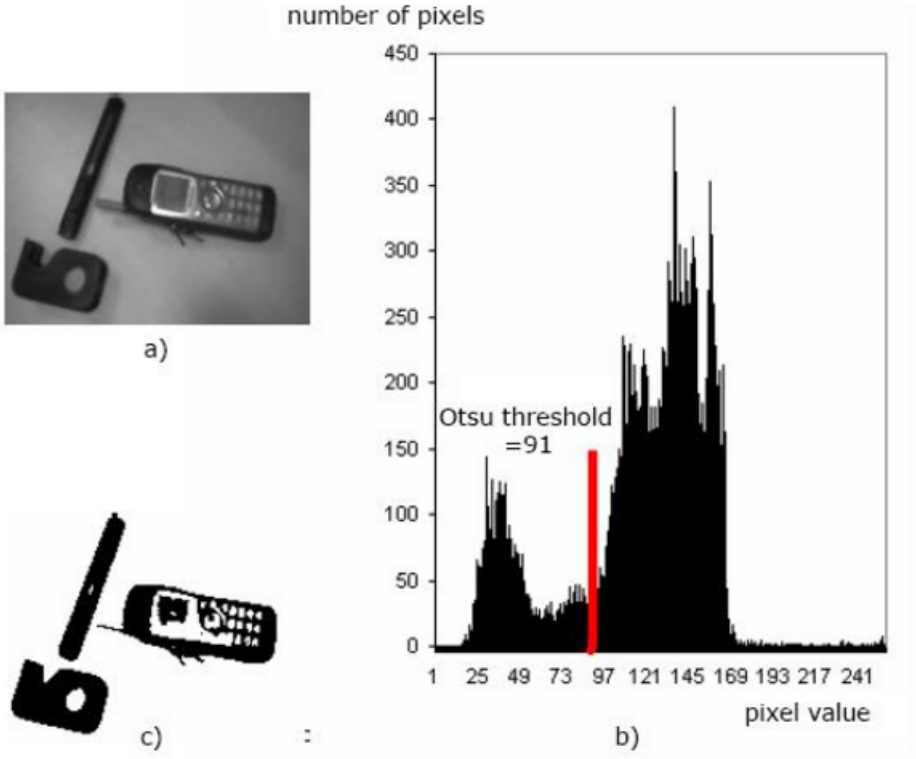
where

$$mean = \frac{\sum\limits_{\substack{x<roiSize.width\\ y<roiSize.height\\ x,y\ge 0}} pSrc(x,y)}{roiSize.height\cdot roiSize.width}$$

The computed Otsu threshold can be used in the thresholding functions described above.

For example, the following figures shows how the Otsu threshold can be used for background/foreground selection. The figure a) below shows the initial image, the figure b) shows the histogram of the initial image with the red line indicating the computed Otsu threshold. The figure c) demonstrates the resulting image obtained by applying the function `ippiThreshold_8u_C1R` with a computed Otsu threshold value to the source image.

Using Otsu Threshold for Background/Foreground Selection



Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize` has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if `srcStep` is less than or equal to 0.

Compare Operations

This section describes functions that compare images. Each compare function writes its results to a one-channel `Ipp8u` output image. The output pixel is set to a non-zero value if the corresponding input pixel(s) satisfy the compare condition; otherwise, the output pixel is set to 0. You can compare either two images, or an image and a constant value, using the following compare conditions: “greater”, “greater or equal”, “less”,

"less or equal", "equal". Compare condition is specified as a function argument of `IppCmpOp` type (see Structures and Enumerators for more information). Images containing floating-point data can also be compared for being equal within a given tolerance `eps`.

For images with multi-channel data, the compare condition for a given pixel is true only when each color channel value of that pixel satisfies this condition.

- Compare
- CompareC
- CompareEqualEps
- CompareEqualEpsC

Compare

Compares pixel values of two images using a specified compare operation.

Syntax

```
IppStatus ippiCompare_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize roiSize,
IppCmpOp ippCmpOp);
```

Supported values for `mod`:

`8u_C1R`	`16u_C1R`	`16s_C1R`	`32f_C1R`
`8u_C3R`	`16u_C3R`	`16s_C3R`	`32f_C3R`
`8u_C4R`	`16u_C4R`	`16s_C4R`	`32f_C4R`

```
IppStatus ippiCompare_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize roiSize,
IppCmpOp ippCmpOp);
```

Supported values for `mod`:

`8u_AC4R`	`16u_AC4R`	`16s_AC4R`	`32f_AC4R`

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc1`, `pSrc2`	Pointers to the source image ROIs.
`src1Step`, `src2Step`	Distances in bytes between starts of consecutive lines in the source images.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image.

roiSize	Size of the source and destination ROI in pixels.
ippCmpOp	Compare operation to be used for comparing the pixel values.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function compares the corresponding pixels of ROI in two source images pSrc1, pSrc2 using the ippCmpOp compare operation, and writes the results to a one-channel Ipp8u image pDst. If the result of the compare is true, the corresponding output pixel is set to an IPP_MAX_8U value; otherwise, it is set to 0.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointer is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if src1Step, src2Step, or dstStep has a zero or negative value.

CompareC

Compares pixel values of a source image to a given value using a specified compare operation.

Syntax

Case 1: Operation on one-channel data

```
IppStatus ippiCompareC_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>
value, Ipp8u* pDst, int dstStep, IppiSize roiSize, IppCmpOp ippCmpOp);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32f_C1R

Case 2: Operation on multi-channel data

```
IppStatus ippiCompareC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const
Ipp<datatype> value[3], Ipp8u* pDst, int dstStep, IppiSize roiSize, IppCmpOp ippCmpOp);
```

Supported values for mod:

8u_C3R	16u_C3R	16s_C3R	32f_C3R

```
IppStatus ippiCompareC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const
Ipp<datatype> value[4], Ipp8u* pDst, int dstStep, IppiSize roiSize, IppCmpOp ippCmpOp);
```

Supported values for mod:

8u_C4R	16u_C4R	16s_C4R	32f_C4R

```
IppStatus ippiCompareC_<mod>(const Ipp<datatype>* pSrc, int srcStep, const
Ipp<datatype> value[3], Ipp8u* pDst, int dstStep, IppiSize roiSize, IppCmpOp ippCmpOp);
```

Supported values for `mod`:

8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`value`	The value to compare each pixel to. In case of multi-channel data, an array of separate values (one for each channel).
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image.
`roiSize`	Size of the source and destination ROI in pixels.
`ippCmpOp`	Compare operation to be used for comparing the pixel values.

Description

This function operates with ROI (Regions of Interest in Intel IPP).

This function compares pixels of the each channel of the source image ROI `pSrc` to a given `value` specified for each channel using the `ippCmpOp` compare operation, and writes the results to a one-channel `Ipp8u` image `pDst`. If the result of the compare is true, that is, all pixels of all channels meet the specified condition, then the corresponding output pixel is set to an `IPP_MAX_8U` value; otherwise, it is set to 0.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointer is `NULL`.

ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if srcStep or dstStep has a zero or negative value.

Example

The code example below shows how to use the comparison function and create a mask image:

```
IppStatus compare ( void ) {
   Ipp8u x[5*4], y[5*4];
   IppiSize roi = {5,4};
   int i;
   for( i=0; i<5*4; ++i ) x[i] = (Ipp8u)i;
   return ippiCompareC_8u_C1R ( x, 5, 7, y, 5, roi, ippCmpGreater );
}
```

The mask image y contains:

```
00 00 00 00 00
00 00 00 FF FF
FF FF FF FF FF
FF FF FF FF FF
```

CompareEqualEps

Compares two images with floating-point data, testing whether pixel values are equal within a certain tolerance eps.

Syntax

```
IppStatus ippiCompareEqualEps_<mod>(const Ipp32f* pSrc1, int src1Step, const Ipp32f*
pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp32f eps);
```

Supported values for mod:

32f_C1R
32f_C3R
32f_C4R

```
IppStatus ippiCompareEqualEps_32f_AC4R(const Ipp32f* pSrc1, int src1Step, const Ipp32f*
pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp32f eps);
```

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc1, pSrc2	Pointers to the source image ROIs.
src1Step, src2Step	Distances in bytes between starts of consecutive lines in the source images.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.
eps	The tolerance value.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function tests if the corresponding pixels of ROI in two source images pSrc1, pSrc2 are equal within the tolerance eps, and writes the results to a one-channel Ipp8u image pDst. If the absolute value of difference of the pixel values in pSrc1 and pSrc2 is less than or equal to eps, then the corresponding pixel in pDst is set to an IPP_MAX_8U value; otherwise the pixel in pDst is set to 0. For multi-channel images, the differences for all color channel values of a pixel must be within the eps tolerance for the compare condition to be true.

This function processes images with floating-point data only.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointer is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if src1Step, src2Step, or dstStep has a zero or negative value.
ippStsEpsValErr	Indicates an error condition if eps has a negative value.

CompareEqualEpsC

Tests whether floating-point pixel values of an image are equal to a given value within a certain tolerance eps.

Syntax

Case 1: Operation on one-channel data

```
IppStatus ippiCompareEqualEpsC_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f value,
Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp32f eps);
```

Case 2: Operation on multi-channel data

```
IppStatus ippiCompareEqualEpsC_32f_C3R(const Ipp32f* pSrc, int srcStep, const Ipp32f
value[3], Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp32f eps);
```

```
IppStatus ippiCompareEqualEpsC_32f_C4R(const Ipp32f* pSrc, int srcStep, const Ipp32f
value[4], Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp32f eps);
```

```
IppStatus ippiCompareEqualEpsC_32f_AC4R(const Ipp32f* pSrc, int srcStep, const Ipp32f
value[3], Ipp8u* pDst, int dstStep, IppiSize roiSize, Ipp32f eps);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`value`	The value to compare each pixel to. In case of multi-channel data, an array of separate values (one for each channel).
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image.
`roiSize`	Size of the source and destination ROI in pixels.
`eps`	The tolerance value.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function tests if pixel values of the source image ROI `pSrc` are equal to a given constant `value` within the tolerance `eps`, and writes the results to a one-channel `Ipp8u` image `pDst`. If the absolute value of difference between the pixel value in `pSrc` and `value` is less than or equal to `eps`, then the corresponding pixel in `pDst` is set to an `IPP_MAX_8U` value; otherwise the pixel in `pDst` is set to 0. For multi-channel images, the differences between all color channel values of a pixel and the respective components of `value` must be within the tolerance `eps` for the compare condition to be true.

This function processes images with floating-point data only.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr	Indicates an error condition if one of the specified pointer is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with a zero or negative value.
ippStsStepErr	Indicates an error condition if srcStep or dstStep has a zero or negative value.
ippStsEpsValErr	Indicates an error condition if eps has a negative value.

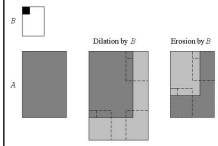
Morphological Operations

This chapter describes the Intel® IPP image processing functions that perform morphological operations on images.

Generally, the erosion and dilation smooth the boundaries of objects without significantly changing their area. Opening and closing smooth thin projections or gaps. Morphological operations use a structuring element (SE) that is a a user-defined rectangular mask, or for some functions - symmetric 3x3 mask.

In a more general sense, morphological operations involve an image A called the object of interest and a kernel element B called the structuring element. The image and structuring element could be in any number of dimensions, but the most common use is with a 2D binary image, or with a 3D gray scale image. The element B is most often a square or a circle, but it could be of any shape. Just like in convolution, B is a kernel or template with an anchor point. Figure Dilation and Erosion of A by B shows dilation and erosion of object A by B. In the figure, B is rectangular with an anchor point at upper left shown as a dark square.

Dilation and Erosion of A by B



Let B_t is the SE with pixel t in the anchor position, B is transpose of the SE.

Dilation of binary image A{A(t) = 1, t ∈ A; 0 - otherwise} by binary SE B is

$$A \oplus B = \{t : B_t \cap A \neq 0\}$$

It means that every pixel is in the set, if the intersection is not null. That is, a pixel under the anchor point of B is marked “on”, if at least one pixel of B is inside of A.

Erosion of the binary image A by the binary SE B is

$$A \ominus B = \{t : B_t \subseteq A\}$$

That is, a pixel under the anchor of B is marked “on”, if B is entirely within A.

Generalization of dilation and erosion for the gray-scale image A and the binary SE B is

$$A \oplus B = \left\{ \max_{u \in B_t \cap A} A(u) \right\}, \qquad A \Theta B = \left\{ \min_{u \in B_t \cap A} A(u) \right\}$$

Generalization of dilation and erosion for the gray-scale image A and the gray-scale SE B is

$$A \oplus B = \left\{ \max_{u \in B_t \cap A} \left(A(u) + B(u - t) \right) \right\}, \qquad A \Theta B = \left\{ \min_{u \in B_t \cap A} \left(A(u) + B(u - t) \right) \right\}$$

Opening operation of A by B is $A \circ B = (A \Theta B) \oplus B$.

Closing operation of A by B is $A \bullet B = (A \oplus B) \Theta B$.

Top-hat operation of A by B is $A - A \circ B$.

Black-hat operation of A by B is $A \bullet B - A$.

Black-hat operation of A by B is $A \oplus B - A \Theta B$.

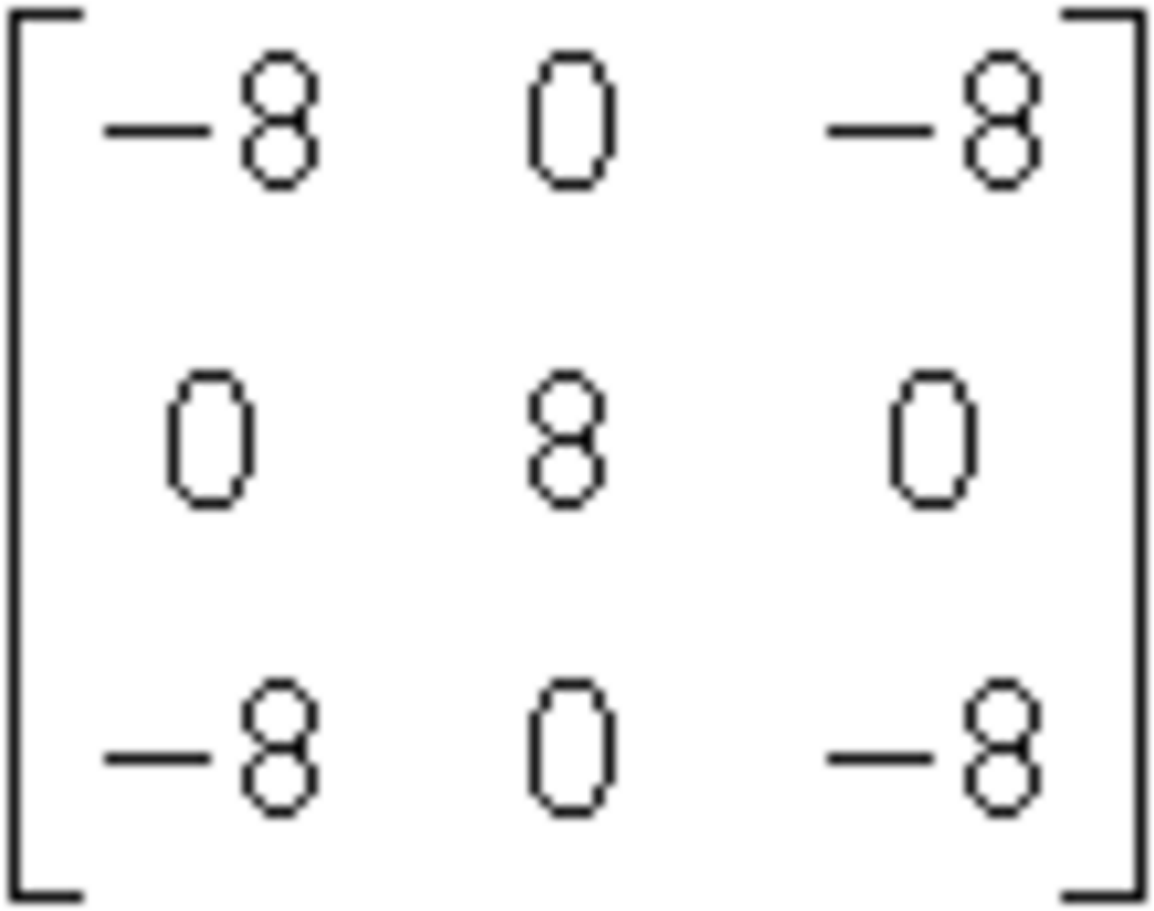
Morphological reconstruction Vincent93 $\rho_A(C)$ of an image A from the image C, $A(t) \geq C(t)\ \forall\ t$ by dilation with the mask B is an image

$$C_k : k = \min_i C_i \equiv C_{i-1},\ C_0 = C,\ C_{i+1}(t) = \min\{ (C_i \oplus B)(t),\ A(t) \}$$

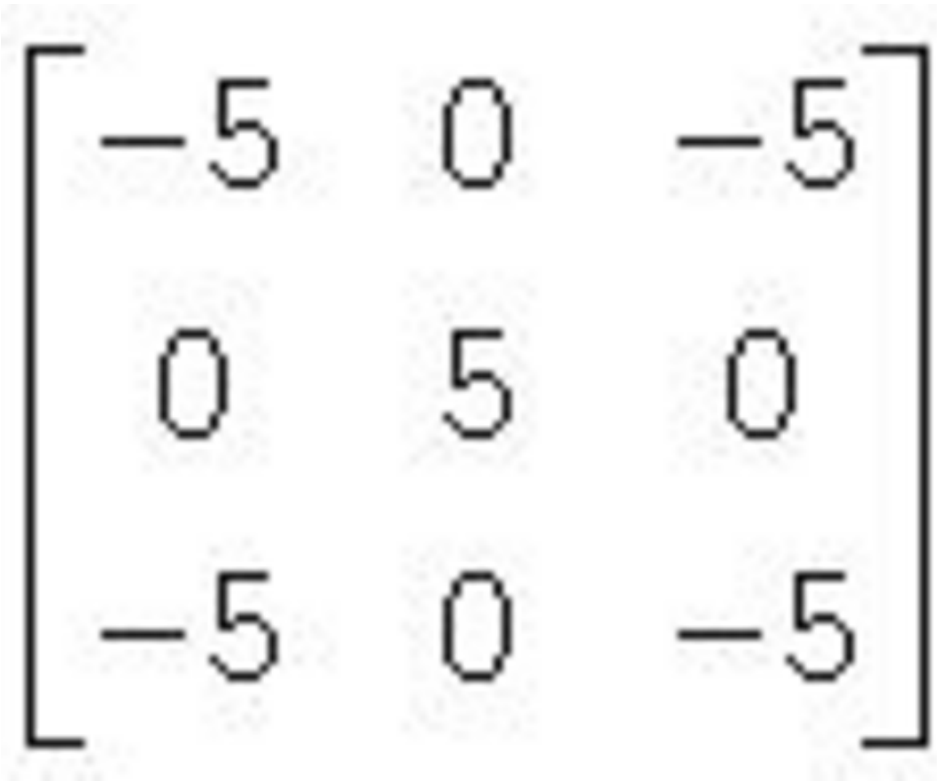
Morphological reconstruction $\rho_A(C)$ of an image A from the image C, $A(t) \leq C(t)\ \forall\ t$ by erosion with the mask B is an image

$$C_k : k = \min_i C_i \equiv C_{i-1},\ C_0 = C,\ C_{i+1}(t) = \max\{ (C_i \Theta B)(t),\ A(t) \}$$

Figure Morphological Operations Performed by Intel IPP presents the results of different morphological operations applied to the initial image. In these operations, the SE is a matrix of 3x3 size with the following values:



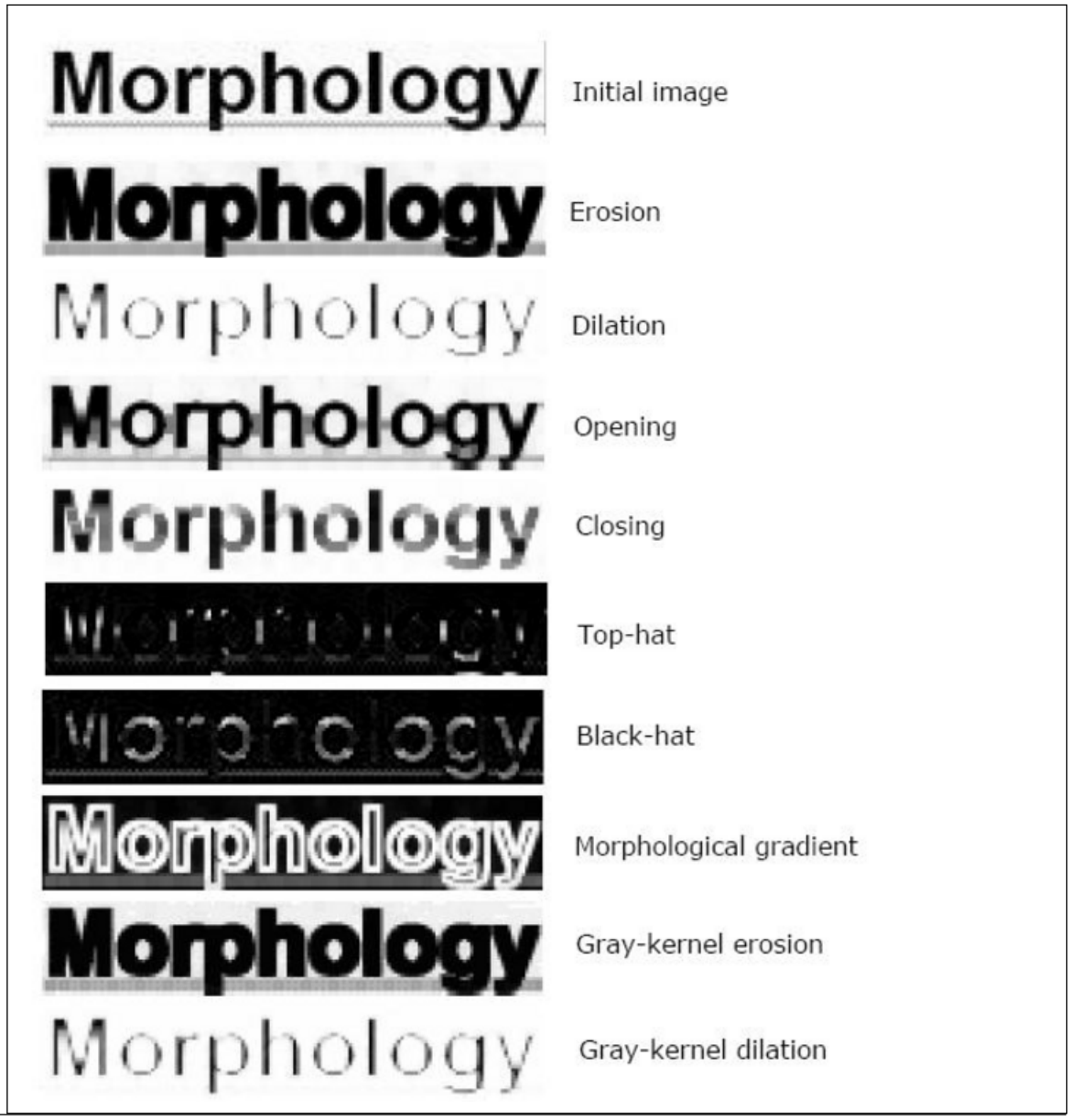
for common and advanced morphology, and



for gray morphology.

The anchor cell is in the center cell (1,1) of the matrix.

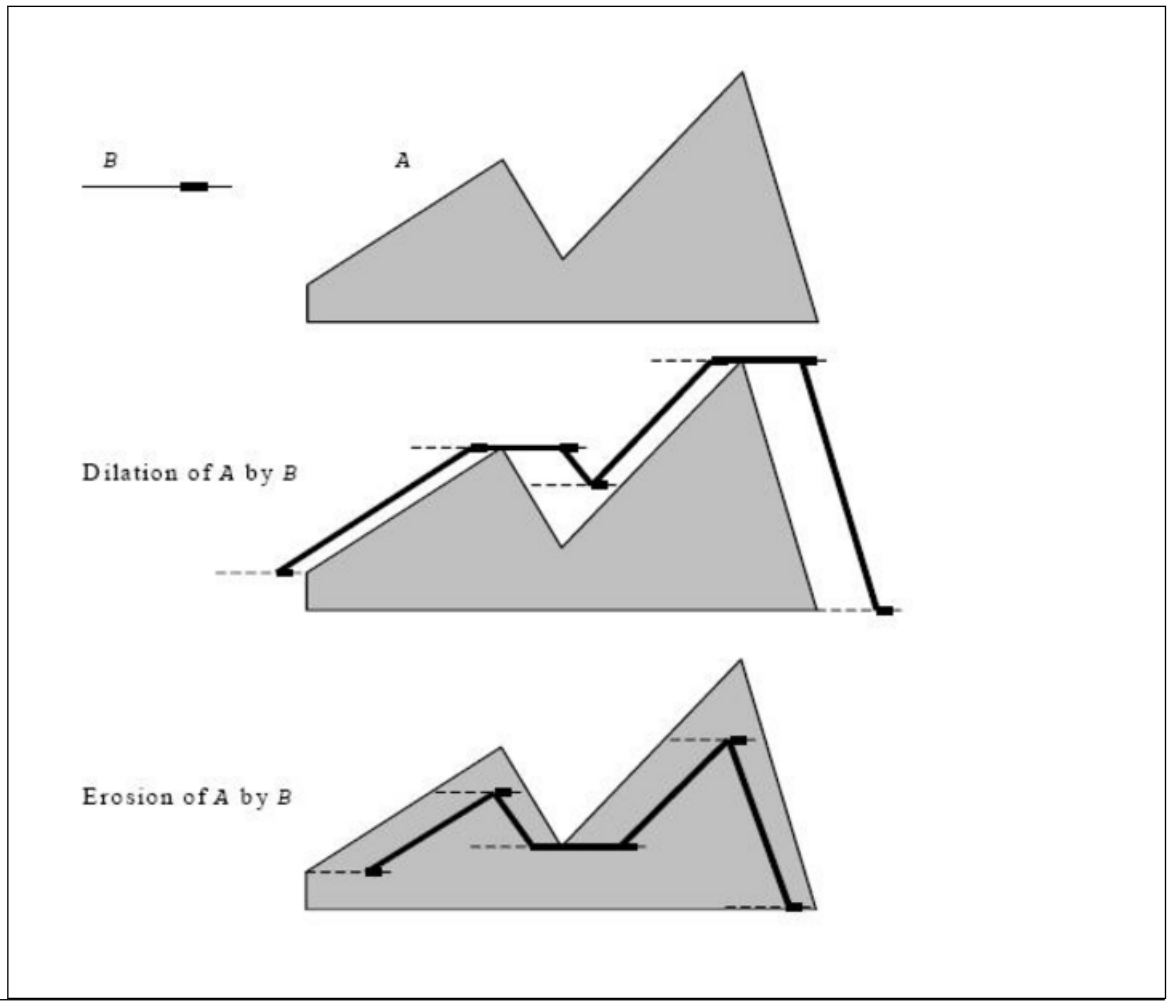
Morphological Operations Performed by Intel IPP



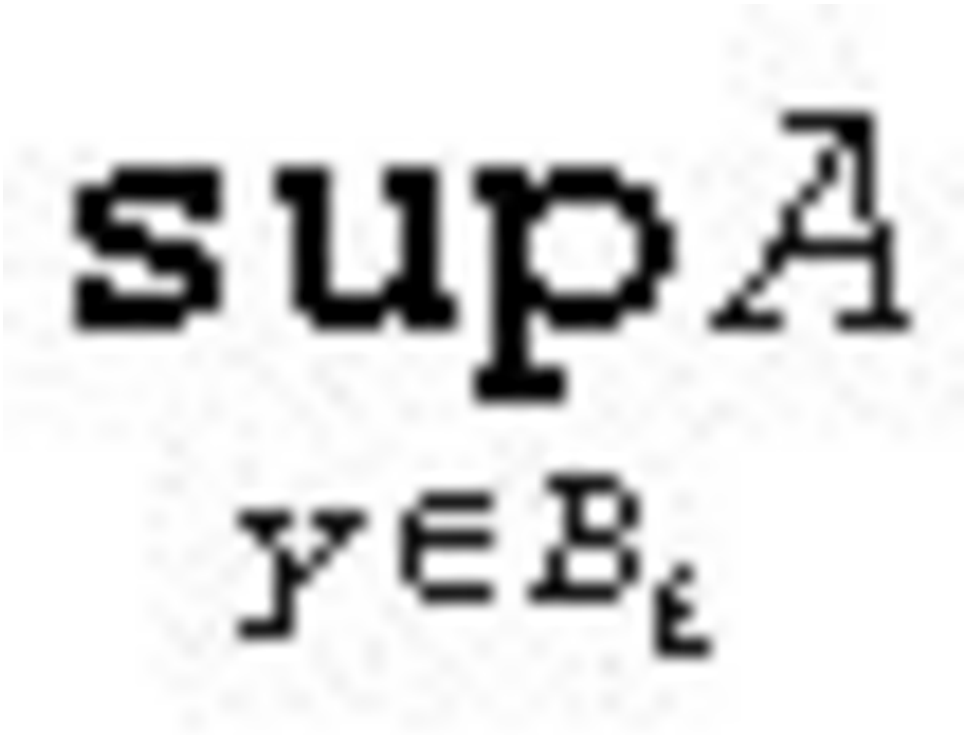
Flat Structuring Elements for Grayscale Image

Erosion and dilation can be done in 3D space, that is, with gray levels. 3D structuring elements can be used, but the simplest and the best way is to use a flat structuring element `B`. Figure 1D Cross Section of Dilation and Erosion of A by B is a 1D cross section of dilation and erosion of a grayscale image `A` by a flat structuring element `B`. In the figure, `B` has an anchor slightly to the right of the center as shown by the dark mark on `B`.

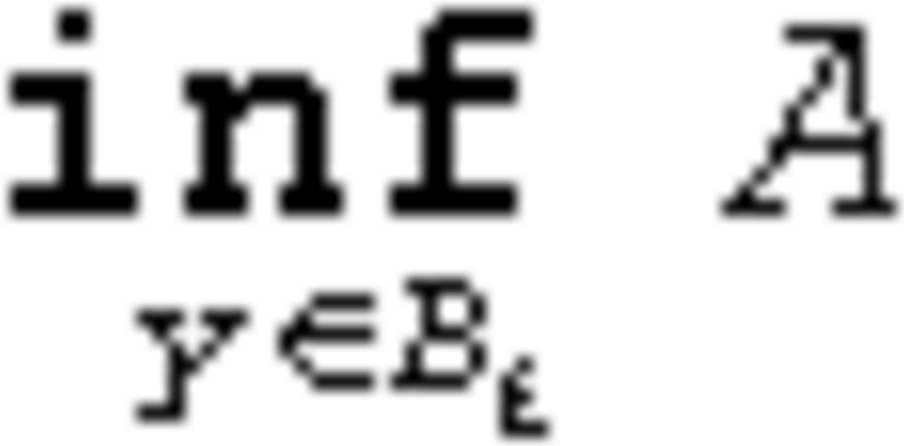
1D Cross Section of Dilation and Erosion of A by B



In Figure 1D Cross Section of Dilation and Erosion of A by B above, dilation is mathematically



and erosion is



.

- Dilate3x3
- Dilate
- DilateBorder
- DilateGetBufferSize
- DilateGetSpecSize
- DilateInit
- Erode3x3
- Erode
- ErodeBorder

- ErodeGetBufferSize
- ErodeGetSpecSize
- ErodeInit
- GrayDilateBorder
- GrayErodeBorder
- MorphAdvInit
- MorphAdvGetSize
- MorphGetBufferSize
- MorphGetSpecSize
- MorphInit
- MorphologyBorderGetSize
- MorphologyBorderInit
- MorphBlackhat
- MorphBlackhatBorder
- MorphClose
- MorphCloseBorder
- MorphGradient
- MorphGradientBorder
- MorphOpen
- MorphOpenBorder
- MorphTophat
- MorphTophatBorder
- MorphGrayInit
- MorphGrayGetSize
- MorphReconstructGetBufferSize
- MorphReconstructDilate
- MorphReconstructErode
- MorphSetMode

Dilate3x3

Performs dilation of an image using a 3x3 mask.

Syntax

```
IppStatus ippiDilate3x3_64f_C1R(const Ipp64f* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize roiSize);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.

pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function performs dilation of a rectangular ROI area inside a 2D image using a symmetric 3x3 mask.

Source and destination images can be of different sizes, but the ROI size is the same for both images. The output pixel is set to the maximum of the corresponding input pixel and its eight neighboring pixels.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if pSrc or pDst is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if srcStep or dstStep has a zero or negative value.

Dilate

Performs dilation of an image using a specified mask.

Syntax

```
IppStatus ippiDilate_8u_C1R_L(const Ipp8u* pSrc, IppSizeL srcStep, Ipp8u* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp8u
borderValue[1], const IppiMorphStateL* pMorphSpec, Ipp8u* pBuffer);
```

```
IppStatus ippiDilate_8u_C3R_L(const Ipp8u* pSrc, IppSizeL srcStep, Ipp8u* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp8u
borderValue[3], const IppiMorphStateL* pMorphSpec, Ipp8u* pBuffer);
```

```
IppStatus ippiDilate_8u_C4R_L(const Ipp8u* pSrc, IppSizeL srcStep, Ipp8u* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp8u
borderValue[4], const IppiMorphStateL* pMorphSpec, Ipp8u* pBuffer);
```

```
IppStatus ippiDilate_32f_C3R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp32f
borderValue[3], const IppiMorphStateL* pMorphSpec, Ipp8u* pBuffer);
```

```
IppStatus ippiDilate_32f_C1R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp32f
borderValue[1], const IppiMorphStateL* pMorphSpec, Ipp8u* pBuffer);
```

```
IppStatus ippiDilate_32f_C4R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp32f
borderValue[4], const IppiMorphStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatus ippiDilate_16u_C1R_L(const Ipp16u* pSrc, IppSizeL srcStep, Ipp16u* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp16u
borderValue[1], const IppiMorphStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatus ippiDilate_16s_C1R_L(const Ipp16s* pSrc, IppSizeL srcStep, Ipp16s* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp16s
borderValue[1], const IppiMorphStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatus ippiDilate_1u_C1R_L(const Ipp8u* pSrc, IppSizeL srcStep, int srcBitOffset,
Ipp8u* pDst, IppSizeL dstStep, int dstBitOffset, IppiSizeL roiSize, IppiBorderType
borderType, const Ipp8u borderValue[1], const IppiMorphStateL* pMorphSpec, Ipp8u*
pBuffer);
```

Include Files

`ippcv_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance, in bytes, between the starting points of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance, in bytes, between the starting points of consecutive lines in the destination image.
`roiSize`	Size of the source and destination ROI, in pixels.
`borderType`	Type of the border; supported values: • `ippBorderDefault` • The border is set to `ippBorderConst` with • `ippBorderRepl` • Border is replicated from the edge pixels. • `ippBorderMirror` • Border pixels are mirrored from the source image boundary • `ippBorderConst` • Values of all border pixels are set to a constant. • `ippBorderInMem` • Border is obtained from the source image pixels in
`borderValue`	Pointer to the vector of values for the constant border type.

srcBitOffset	Offset in the first byte of the source image row.
dstBitOffset	Offset in the first byte of the destination image row.
pMorphSpec	Pointer to the morphology specification structure.
pBuffer	Pointer to the external work buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function performs dilation of a rectangular ROI area inside a one-, three-, or four-channel 2D image using a specified mask pMask of size maskSize and alignment anchor.

Source and destination images can be of different sizes, but the ROI size is the same for both images. The output pixel is set to the maximum of the corresponding input pixel and its neighboring pixels that are picked out by the nonzero mask values. In the four-channel image the alpha channel is not processed.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if pSrc or pDst is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with a zero or negative value.
ippStsStepErr	Indicates an error condition if srcStep or dstStep has a zero or negative value.
ippStsNotEvenStepErr	Indicates an error condition if srcStep or dstStep has a not pixel multiple value.
ippStsBadArgErr	Indicates an error condition if borderType has an incorrect value.

DilateBorder

Performs dilation of an image.

Syntax

```
IppStatus ippiDilateBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, Ipp<datatype>
borderValue, const IppiMorphState* pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32f_C1R

```
IppStatus ippiDilateBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, const Ipp<datatype>
borderValue[3], const IppiMorphState* pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u_C3R	32f_C3R

```
IppStatus ippiDilateBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, const Ipp<datatype>
borderValue[4], const IppiMorphState* pSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

8u_C4R	32f_C4R

```
IppStatus ippiDilateBorder_1u_C1R(const Ipp8u* pSrc, int srcStep, int srcBitOffset,
Ipp8u* pDst, int dstStep, int dstBitOffset, IppiSize roiSize, IppiBorderType
borderType, Ipp8u borderValue, const IppiMorphState* pSpec, Ipp8u* pBuffer);
```

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between the starting points of consecutive lines in the source image.
`srcBitOffset`	Offset, in bits, from the first byte of the source image (for `1u_C1R` flavor).
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance, in bytes, between the starting points of consecutive lines in the destination image.
`dstBitOffset`	Offset, in bits, from the first byte of the destination image (for `1u_C1R` flavor).
`roiSize`	Size of the source and destination image ROI.
`borderType`	Type of border. Possible values are: • `ippBorderRepl` • Border is replicated from the edge pixels. • `ippBorderInMem` • Border is obtained from the source image pixels in memory.
`borderValue`	Constant value to assign to pixels of the constant border. This parameter is applicable only to the `ippBorderConst` border type.
`pSpec`	Pointer to the specification structure.

pBuffer	Pointer to the external buffer required for dilation operations.

Description

This function operates with ROI.

This function expands a rectangular ROI area inside a one-channel two-dimensional image using a mask specified in the specification structure pSpec. Before using this function, you need to initialize the structure using the MorphologyBorderInit function.

The output pixel is set to the maximum of the corresponding input pixel and its neighboring pixels that are picked out by the nonzero mask values.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when: • roiSize has a field with a zero or negative value • roiSize.width is more than the maximum ROI roiWidth passed to the initialization function
ippStsStepErr	Indicates an error when srcStep or dstStep is less than roiSize.width* < pixelSize>.
ippStsNotEvenStepErr	Indicates an error when one of the step values for 16-bit integer images is not divisible by 2.
ippStsBorderErr	Indicates an error when borderType has an illegal value.
ippStsInplaceModeNotSupportedErr	Indicates an error when the pSrc pointer is equal to the pDst pointer.

Example

The code example below demonstrates how to use the ippiMorphologyBorderGetSize_16u_C1R, ippiMorphologyBorderInit_16u_C1R, and ippiDilateBorder_16u_C1R functions to perform dilation of the source image.

```
IppStatus func_MorfDilateBorder()
{
    IppiMorphState* pSpec = NULL;
    Ipp8u* pBuffer = NULL;
    IppiSize roiSize = {5, 5};
    Ipp8u pMask[3*3] = {1, 1, 1,
                        1, 0, 1,
                        1, 1, 1};
    IppiSize maskSize = {3, 3};
    Ipp16u pSrc[5*5] = { 1, 2, 4, 1, 2,
                         5, 1, 2, 1, 2,
                         1, 2, 1, 2, 1,
```

```
                        1, 2, 1, 2, 1,
                        2, 1, 5, 1, 2};
    Ipp16u pDst[5*5];
    int srcStep = 5*sizeof(Ipp16u);
    int dstStep = 5*sizeof(Ipp16u);
    int dstSize = 5;
    IppStatus status = ippStsNoErr;
    int specSize = 0, bufferSize = 0;
    IppiBorderType borderType= ippBorderRepl;
    Ipp16u borderValue = 0;


    status = ippiMorphologyBorderGetSize_16u_C1R( roiSize, maskSize, &specSize, &bufferSize );
    if (status != ippStsNoErr) return status;


    pSpec = (IppiMorphState*)ippsMalloc_8u(specSize);
    pBuffer = (Ipp8u*)ippsMalloc_8u(bufferSize);


    status = ippiMorphologyBorderInit_16u_C1R( roiSize, pMask, maskSize, pSpec, pBuffer );
    if (status != ippStsNoErr) {
        ippsFree(pBuffer);
        ippsFree(pSpec);
        return status;
    }


    status = ippiDilateBorder_16u_C1R( pSrc, srcStep, pDst, dstStep, roiSize, borderType,
borderValue, pSpec, pBuffer);


    ippsFree(pBuffer);
    ippsFree(pSpec);
    return status;
}
```

The result is as follows:

```
pDst->
 5 5 4 4 2
 5 5 4 4 2
 5 5 2 2 2
 2 5 5 5 2
 2 5 5 5 2
```

See Also

Regions of Interest in Intel IPP

User-defined Border Types

MorphologyBorderInit Initializes the morphology specification structure for erosion or dilation operations.

DilateGetBufferSize

Computes the size of the working buffer for the `Dilate` function.

Syntax

```
IppStatus ippiDilateGetBufferSize_L(IppiSizeL roiSize, IppiSizeL maskSize, IppDataType datatype, int numChannels, IppSizeL* pBufferSize);
```

Include Files

ippcv_l.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

roiSize	Size of the source and destination image ROI, in pixels.
maskSize	Size of the structuring element.
dataType	Data type for the morphological function.
numChannels	Number of channels in the image.
pBufferSize	Pointer to the buffer size value for the morphological initialization function.

Description

This function computes the size of the working buffer required for the ippiDilate functions with the _L suffix.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when one of the pointers is NULL.
ippStsSizeErr	Width of the image, or width or height of the structuring element is less than, or equal to zero.

DilateGetSpecSize

Computes the size of the internal state or specification structure for the Dilate function.

Syntax

```
IppStatus ippiDilateGetSpecSize_L(IppiSizeL roiSize, IppiSizeL maskSize, IppSizeL* pSpecSize);
```

Include Files

ippcv_l.h

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`roiSize`	Size of the source and destination image ROI, in pixels.
`maskSize`	Size of the structuring element.
`pSpecSize`	Pointer to the size of the specification structure.

Description

This function computes the size of the specification structure required for the `ippiDilate` functions with the `_L` suffix.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error when one of the pointers is `NULL`.
`ippStsSizeErr`	Width of the image, or width or height of the structuring element is less than, or equal to zero.

DilateInit

Initializes the internal state or specification structure for the `Dilate` function.

Syntax

```
IppStatus ippiDilateInit_L(IppiSizeL roiSize, const Ipp8u* pMask, IppiSizeL maskSize,
IppiMorphStateL* pMorphSpec);
```

Include Files

`ippcv_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`roiSize`	Size of the source and destination image ROI, in pixels.
`pMask`	Pointer to the structuring element (mask).

maskSize	Size of the structuring element.
pMorphSpec	Pointer to the morphology specification structure.

Description

This function initializes the internal state or specification structure for the ippiDilate functions with the _L suffix.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when one of the pointers is NULL.
ippStsSizeErr	Width of the image, or width or height of the structuring element is less than, or equal to zero.
ippStsAnchorErr	Anchor point is outside the structuring element.

Erode3x3

Performs erosion of an image using a 3x3 mask.

Syntax

```
IppStatus ippiErode3x3_64f_C1R(const Ipp64f* pSrc, int srcStep, Ipp64f* pDst, int dstStep, IppiSize roiSize);
```

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function performs erosion of a rectangular ROI area inside a 2D image using a symmetric 3x3 mask.

Source and destination images can have different size, but the ROI size is the same for both images. The output pixel is set to the minimum of the corresponding input pixel and its 8 neighboring pixels.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if `pSrc` or `pDst` is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize` has a field with a zero or negative value.
`ippStsStepErr`	Indicates an error condition if `srcStep` or `dstStep` has a zero or negative value.

Erode

Performs erosion of an image using a specified mask.

Syntax

```
IppStatus ippiErode_8u_C1R_L(const Ipp8u* pSrc, IppSizeL srcStep, Ipp8u* pDst, IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp8u borderValue[1], const IppiMorphStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatus ippiErode_8u_C3R_L(const Ipp8u* pSrc, IppSizeL srcStep, Ipp8u* pDst, IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp8u borderValue[3], const IppiMorphStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatus ippiErode_8u_C4R_L(const Ipp8u* pSrc, IppSizeL srcStep, Ipp8u* pDst, IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp8u borderValue[4], const IppiMorphStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatus ippiErode_32f_C3R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f* pDst, IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp32f borderValue[3], const IppiMorphStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatus ippiErode_32f_C1R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f* pDst, IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp32f borderValue[1], const IppiMorphStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatus ippiErode_32f_C4R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f* pDst, IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp32f borderValue[4], const IppiMorphStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatus ippiErode_16u_C1R_L(const Ipp16u* pSrc, IppSizeL srcStep, Ipp16u* pDst, IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp16u borderValue[1], const IppiMorphStateL* pMorphSpec, Ipp8u* pBuffer);
```

```
IppStatus ippiErode_16s_C1R_L(const Ipp16s* pSrc, IppSizeL srcStep, Ipp16s* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp16s
borderValue[1], const IppiMorphStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatus ippiErode_1u_C1R_L(const Ipp8u* pSrc, IppSizeL srcStep, int srcBitOffset,
Ipp8u* pDst, IppSizeL dstStep, int dstBitOffset, IppiSizeL roiSize, IppiBorderType
borderType, const Ipp8u borderValue[1], const IppiMorphStateL* pMorphSpec, Ipp8u*
pBuffer);
```

Include Files

`ippcv_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance, in bytes, between the starting points of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance, in bytes, between the starting points of consecutive lines in the destination image.
`roiSize`	Size of the source and destination ROI, in pixels.
`borderType`	Type of the border; supported values: • `ippBorderDefault` • The border is set to `ippBorderConst` with • `ippBorderRepl` • Border is replicated from the edge pixels. • `ippBorderMirror` • Border pixels are mirrored from the source image boundary • `ippBorderConst` • Values of all border pixels are set to a constant. • `ippBorderInMem` • Border is obtained from the source image pixels in
`borderValue`	Pointer to the vector of values for the constant border type.
`srcBitOffset`	Offset in the first byte of the source image row.
`dstBitOffset`	Offset in the first byte of the destination image row.
`pMorphSpec`	Pointer to the morphology specification structure.
`pBuffer`	Pointer to the external work buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function performs erosion of a rectangular ROI area inside a one-, three-, or four-channel 2D image using a specified mask `pMask` of size `maskSize` and alignment `anchor`.

Source and destination images can be of different sizes, but the ROI size is the same for both images (`dstRoiSize`). The output pixel is set to the minimum of the corresponding input pixel and its neighboring pixels that are picked out by the non-zero mask values. In the four-channel image the alpha channel is not processed.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if `pSrc` or `pDst` is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize` has a field with a zero or negative value.
`ippStsStepErr`	Indicates an error condition if `srcStep` or `dstStep` has a zero or negative value.
`ippStsNotEvenStepErr`	Indicates an error condition if `srcStep` or `dstStep` has a not pixel multiple value.
`ippStsBadArgErr`	Indicates an error condition if `borderType` has an incorrect value.

ErodeBorder

Performs erosion of an image.

Syntax

```
IppStatus ippiErodeBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, Ipp<datatype>
borderValue, const IppiMorphState* pSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

`8u_C1R`	`16u_C1R`	`16s_C1R`	`32f_C1R`

```
IppStatus ippiErodeBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, const Ipp<datatype>
borderValue[3], const IppiMorphState* pSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

`8u_C3R`	`32f_C3R`

```
IppStatus ippiErodeBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, const Ipp<datatype>
borderValue[4], const IppiMorphState* pSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

8u_C4R	32f_C4R

```
IppStatus ippiErodeBorder_1u_C1R(const Ipp8u* pSrc, int srcStep, int srcBitOffset,
Ipp8u* pDst, int dstStep, int dstBitOffset, IppiSize roiSize, IppiBorderType
borderType, Ipp8u borderValue, const IppiMorphState* pSpec, Ipp8u* pBuffer);
```

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance, in bytes, between the starting points of consecutive lines in the source image.
srcBitOffset	Offset, in bits, from the first byte of the source image (for 1u_C1R flavor).
pDst	Pointer to the destination image ROI.
dstStep	Distance, in bytes, between the starting points of consecutive lines in the destination image.
dstBitOffset	Offset, in bits, from the first byte of the destination image (for 1u_C1R flavor).
roiSize	Size of the source and destination image ROI.
borderType	Type of border. Possible values are: • ippBorderRepl • Border is replicated from the edge pixels. • ippBorderInMem • Border is obtained from the source image pixels in memory.
borderValue	Constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.
pSpec	Pointer to the specification structure.
pBuffer	Pointer to the external buffer required for erosion operations.

Description

This function operates with ROI.

This function performs erosion of a rectangular ROI area inside a one-channel 2D image using a mask specified in the specification structure `pSpec`. Before using this function, you need to initialize the structure using the MorphologyBorderInit function.

The output pixel is set to the maximum of the corresponding input pixel and its neighboring pixels that are picked out by the nonzero mask values.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error when: • `roiSize` has a field with a zero or negative value • `roiSize.width` is more than the maximum ROI `roiWidth` passed to the initialization function • `srcBitOffset` or `dstBitOffset` is less than zero
`ippStsStepErr`	Indicates an error when `srcStep` or `dstStep` is less than `roiSize.width* < pixelSize>`.
`ippStsNotEvenStepErr`	Indicates an error when one of the step values for 16-bit integer images is not divisible by 2.
`ippStsBorderErr`	Indicates an error when `borderType` has an illegal value.
`ippStsInplaceModeNotSupportedErr`	Indicates an error when the `pSrc` pointer is equal to the `pDst` pointer.

See Also

Regions of Interest in Intel IPP

User-defined Border Types

MorphologyBorderInit Initializes the morphology specification structure for erosion or dilation operations.

ErodeGetBufferSize

Computes the size of the working buffer for the `Erode` function.

Syntax

```
IppStatus ippiErodeGetBufferSize_L(IppiSizeL roiSize, IppiSizeL maskSize, IppDataType datatype, int numChannels, IppSizeL* pBufferSize);
```

Include Files

`ippcv_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

roiSize	Size of the source and destination image ROI, in pixels.
maskSize	Size of the structuring element.
dataType	Data type for the morphological function.
numChannels	Number of channels in the image.
pBufferSize	Pointer to the buffer size value for the morphological initialization function.

Description

This function computes the size of the working buffer required for the ippiErode functions with the _L suffix.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when one of the pointers is NULL.
ippStsSizeErr	Width of the image, or width or height of the structuring element is less than, or equal to zero.

ErodeGetSpecSize

Computes the size of the internal state or specification structure for the Erode function.

Syntax

```
IppStatus ippiErodeGetSpecSize_L(IppiSizeL roiSize, IppiSizeL maskSize, IppSizeL*
pSpecSize);
```

Include Files

ippcv_l.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

roiSize	Size of the source and destination image ROI, in pixels.

maskSize	Size of the structuring element.
pSpecSize	Pointer to the size of the specification structure.

Description

This function computes the size of the specification structure required for the ippiErode functions with the _L suffix.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when one of the pointers is NULL.
ippStsSizeErr	Width of the image, or width or height of the structuring element is less than, or equal to zero.

ErodeInit

Initializes the internal state or specification structure for the Erode function.

Syntax

```
IppStatus ippiErodeInit_L(IppiSizeL roiSize, const Ipp8u* pMask, IppiSizeL maskSize, IppiMorphStateL* pMorphSpec);
```

Include Files

ippcv_l.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

roiSize	Size of the source and destination image ROI, in pixels.
pMask	Pointer to the structuring element (mask).
maskSize	Size of the structuring element.
pMorphSpec	Pointer to the morphology specification structure.

Description

This function initializes the internal state or specification structure for the ippiErode functions with the _L suffix.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when one of the pointers is NULL.
ippStsSizeErr	Width of the image, or width or height of the structuring element is less than, or equal to zero.
ippStsAnchorErr	Anchor point is outside the structuring element.

GrayDilateBorder

Performs gray-kernel dilation of an image.

Syntax

```
IppStatus ippiGrayDilateBorder_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, IppiBorderType border, IppiMorphGrayState_8u* pState);
```

```
IppStatus ippiGrayDilateBorder_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiBorderType border, IppiMorphGrayState_32f* pState);
```

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination image ROI.
border	Type of border; the possible value is ippBorderRepl, which means that a replicated border is used.
pState	Pointer to the morphology state structure.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function performs gray-kernel dilation of a rectangular ROI area inside a one-channel 2D image using a specified in the gray-kernel morphology state structure `pState` mask and the anchor cell. This structure must be initialized by MorphGrayInit beforehand.

NOTE The structure can be used to process images with ROI that does not exceed the specified maximum width and height `roiSize`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize` has a field with a zero or negative value, or if one of ROI width or height is greater than corresponding size of ROI passed to the initialization functions.
`ippStsStepErr`	Indicates an error condition if `srcStep` or `dstStep` is less than `roiSize.width * <pixelSize>`.
`ippStsNotEvenStepErr`	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images.
`ippStsBorderErr`	Indicates an error condition if `border` has an illegal value.

GrayErodeBorder

Performs gray-kernel erosion of an image.

Syntax

```
IppStatus ippiGrayErodeBorder_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, IppiBorderType border, IppiMorphGrayState_8u* pState);
```

```
IppStatus ippiGrayErodeBorder_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiBorderType border, IppiMorphGrayState_32f* pState);
```

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination image ROI.
border	Type of border; the possible value is ippBorderRepl, which means that a replicated border is used.
pState	Pointer to the morphology state structure.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function performs gray-kernel erosion of a rectangular ROI area inside a one-channel 2D image using a specified in the gray-kernel morphology state structure pState mask and the anchor cell. This structure must be initialized by MorphGrayInit beforehand.

NOTE The structure can be used to process images with ROI that does not exceed the specified maximum width and height roiSize.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with a zero or negative value, or if one of ROI width or height is greater than corresponding size of ROI passed to the initialization functions.
ippStsStepErr	Indicates an error condition if srcStep or dstStep is less than roiSize.width * <pixelSize>.
ippStsNotEvenStepErr	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images.
ippStsBorderErr	Indicates an error condition if border has an illegal value.

MorphAdvInit

Initializes the specification structure for advanced morphological operations.

Syntax

```
IppStatus ippiMorphAdvInit_<mod>(IppiSize roiSize, const Ipp8u* pMask, IppiSize maskSize, IppiMorphAdvState* pSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

`1u_C1R`	`8u_C1R`	`16u_C1R`	`16s_C1R`	`32f_C1R`
	`8u_C3R`			`32f_C3R`
	`8u_C4R`			`32f_C4R`

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSpec`	Pointer to the specification structure.
`roiSize`	Maximal size of the image ROI (in pixels) that can be processed using the allocated structure.
`pMask`	Pointer to the mask.
`maskSize`	Size of the mask in pixels.
`pBuffer`	Pointer to the external buffer for advanced morphological operations.

Description

This function operates with ROI.

This function initializes the specification structure `pSpec` in the external buffer. This structure is used by the MorphOpenBorder and MorphCloseBorder functions that perform open and close morphological operations.

All advanced morphological operations are performed on the source image pixels corresponding to non-zero values of the structuring element (mask) `pMask`.

NOTE This function required that the image ROI does not exceed the maximum width and height `roiSize` specified by the initialization functions.

For an example on how to use this function, see the code example provided with the MorphCloseBorder function description.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when maskSize has a field with a zero or negative value, or if roiWidth is less than 1.
ippStsAnchorErr	Indicates an error when anchor is outside the mask.

See Also

Regions of Interest in Intel IPP

MorphAdvGetSize Computes the size of the specification structure for advanced morphological operations.

MorphOpenBorder Opens an image.

MorphCloseBorder Closes an image.

MorphTophatBorder Performs top-hat operation on an image.

MorphBlackhatBorder Performs black-hat operation on an image.

MorphGradientBorder Calculates morphological gradient of an image.

MorphAdvGetSize

Computes the size of the specification structure for advanced morphological operations.

Syntax

Case 2: Computing the size of morphology specification structure

```
IppStatus ippiMorphAdvGetSize_<mod>(IppiSize roiSize, IppiSize maskSize, int*
pSpecSize, int* pBufferSize);
```

Supported values for mod:

1u_C1R	8u_C1R	16u_C1R	16s_C1R	32f_C1R
	8u_C3R			32f_C3R
	8u_C4R			32f_C4R

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

roiSize	Maximal size of the image ROI (in pixels) that can be processed using the allocated structure.
maskSize	Size of the mask, in pixels.
pSpecSize	Pointer to the size of the morphology specification structure.
pBufferSize	Pointer to the size of the external buffer required for advanced morphological operations.

Description

This function operates with ROI.

This function computes the size of the specification structure and the size of the buffer required for advanced morphological operations. Call this function before using the ippiMorphAdvInit function.

For an example on how to use this function, see the code example provided with the ippiMorphCloseBorder function description.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when maskSize or roiSize has a field with a zero or negative value.

See Also

Regions of Interest in Intel IPP

MorphAdvInit Initializes the specification structure for advanced morphological operations.

MorphCloseBorder Closes an image.

MorphGetBufferSize

Computes the size of the working buffer for advanced morphological operations.

Syntax

```
IppStatus ippiMorphGetBufferSize_L (IppiSizeL roiSize, IppiSizeL maskSize, IppDataType dataType, int numChannels, IppSizeL* pBufferSize);
```

Include Files

ippcv_l.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

roiSize	Maximum size of the image ROI, in pixels.
maskSize	Size of the structuring element.
dataType	Data type for the processing function.
numChannels	Number of channels in the image.
pBufferSize	Pointer to the buffer size value for the initialization function.

Description

This function computes the size of the working buffer required for advanced morphological operations.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when one of the pointers is NULL.
ippStsSizeErr	Width of the image, or width or height of the structuring element is less than, or equal to zero.

MorphGetSpecSize

Computes the size of the internal state or specification structure for advanced morphological operations.

Syntax

```
ippiMorphGetSpecSize_L(IppiSizeL roiSize, IppiSizeL maskSize, IppDataType dataType, int numChannels, IppSizeL* pSpecSize);
```

Include Files

ippcv_l.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

roiSize	Maximum size of the image ROI, in pixels.
maskSize	Size of the structuring element.
dataType	Data type for the processing function.
numChannels	Number of channels in the image.

pSpecSize	Pointer to the size of the specification structure.

Description

This function computes the size of the specification structure for advanced morphological operations.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when one of the pointers is NULL.
ippStsSizeErr	Width of the image, or width or height of the structuring element is less than, or equal to zero.

MorphInit

Initializes the internal state or specification structure for advanced morphological operations.

Syntax

```
IppStatus ippiMorphInit_L( IppiSizeL roiSize, const Ipp8u* pMask, IppiSizeL maskSize,
IppDataType dataType, int numChannels, IppiMorphAdvStateL* pMorphSpec);
```

Include Files

ippcv_l.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

roiSize	Maximum size of the image ROI, in pixels.
pMask	Pointer to the structuring element (mask).
maskSize	Size of the structuring element.
dataType	Data type for the processing function.
numChannels	Number of channels in the image.
pMorphSpec	Pointer to the advanced morphology specification structure.

Description

This function initializes the internal state or specification structure for advanced morphological operations.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when one of the pointers is NULL.
ippStsSizeErr	Width of the image, or width or height of the structuring element is less than, or equal to zero.
ippStsAnchorErr	Anchor point is outside the structuring element.

MorphologyBorderGetSize

Computes the size of the morphology specification structure.

Syntax

```
IppStatus ippiMorphologyBorderGetSize_<mod>(IppiSize roiSize, IppiSize maskSize, int* pSpecSize, int* pBufferSize);
```

Supported values for mod:

1u_C1R	8u_C1R	16u_C1R	16s_C1R	32f_C1R
	8u_C3R			32f_C3R
	8u_C4R			32f_C4R

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

roiSize	Size of the image ROI, in pixels.
maskSize	Size of the mask, in pixels.
pSpecSize	Pointer to the size of the morphology specification structure.
pBufferSize	Pointer to the size of the buffer required for dilation or erosion operations.

Description

This function operates with ROI.

This function computes the size of the morphology specification structure pMorphSpec and the size of the buffer required for dilation and erosion operations. Call this function before using the ippiMorphologyBorderInit function.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when maskSize has a field with a zero or negative value, or if width or height of roiSize is less than 1.

See Also

Regions of Interest in Intel IPP

MorphologyBorderInit Initializes the morphology specification structure for erosion or dilation operations.

MorphologyBorderInit

Initializes the morphology specification structure for erosion or dilation operations.

Syntax

```
IppStatus ippiMorphologyBorderInit_<mod>(IppiSize roiSize, const Ipp8u* pMask, IppiSize maskSize, IppiMorphState* pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

1u_C1R	8u_C1R	16u_C1R	16s_C1R	32f_C1R
	8u_C3R			32f_C3R
	8u_C4R			32f_C4R

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

roiSize	Size of the image ROI, in pixels.
pMask	Pointer to the mask.
maskSize	Size of the mask, in pixels.
pSpec	Pointer to the specification structure.
pBuffer	Pointer to the external buffer required for dilation or erosion operations.

Description

This function operates with ROI.

This function initializes the specification structure `pSpec` in the external buffer. Before using this function, you need to compute the size of the specification structure using the MorphologyBorderGetSize function. This structure is used by the ippiDilateBorder and ippiErodeBorder functions that perform morphological operations on the source image pixels corresponding to non-zero values of the structuring element (mask) `pMask`.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error when `maskSize` has a field with a zero or negative value, or if width or height of `roiSize` is less than 1.

See Also

Regions of Interest in Intel IPP

MorphologyBorderGetSize Computes the size of the morphology specification structure.

DilateBorder Performs dilation of an image.

ErodeBorder Performs erosion of an image.

MorphBlackhat

Performs top-hat operation on an image.

Syntax

```
IppStatus ippiMorphBlackhat_16u_C1R_L(const Ipp16u* pSrc, IppSizeL srcStep, Ipp16u*
pDst, IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp16u
borderValue[1], const IppiMorphAdvStateL* pMorthSpec, Ipp8u* pBuffer);

IppStatus ippiMorphBlackhat_16s_C1R_L(const Ipp16s* pSrc, IppSizeL srcStep, Ipp16s*
pDst, IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp16s
borderValue[1], const IppiMorphAdvStateL* pMorthSpec, Ipp8u* pBuffer);

IppStatus ippiMorphBlackhat_1u_C1R_L(const Ipp8u* pSrc, IppSizeL srcStep, int
srcBitOffset, Ipp8u* pDst, IppSizeL dstStep, int dstBitOffset, IppiSizeL roiSize,
IppiBorderType borderType, const Ipp8u borderValue[1], const IppiMorphAdvStateL*
pMorthSpec, Ipp8u* pBuffer);

IppStatus ippiMorphBlackhat_8u_C1R_L(const Ipp8u* pSrc, IppSizeL srcStep, Ipp8u* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp8u
borderValue[1], const IppiMorphAdvStateL* pMorthSpec, Ipp8u* pBuffer);

IppStatus ippiMorphBlackhat_8u_C3R_L(const Ipp8u* pSrc, IppSizeL srcStep, Ipp8u* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp8u
borderValue[3], const IppiMorphAdvStateL* pMorthSpec, Ipp8u* pBuffer);

IppStatus ippiMorphBlackhat_8u_C4R_L(const Ipp8u* pSrc, IppSizeL srcStep, Ipp8u* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp8u
borderValue[4], const IppiMorphAdvStateL* pMorthSpec, Ipp8u* pBuffer);
```

```
IppStatus ippiMorphBlackhat_32f_C1R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f*
pDst, IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp32f
borderValue[1], const IppiMorphAdvStateL* pMorthSpec, Ipp8u* pBuffer);

IppStatus ippiMorphBlackhat_32f_C3R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f*
pDst, IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp32f
borderValue[3], const IppiMorphAdvStateL* pMorthSpec, Ipp8u* pBuffer);

IppStatus ippiMorphBlackhat_32f_C4R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f*
pDst, IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp32f
borderValue[4], const IppiMorphAdvStateL* pMorthSpec, Ipp8u* pBuffer);
```

Include Files

`ippcv_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance, in bytes, between the starting points of consecutive lines in the source image.
`srcBitOffset`	Offset, in bits, from the first byte of the source image row.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance, in bytes, between the starting points of consecutive lines in the destination image.
`dstBitOffset`	Offset, in bits, from the first byte of the destination image row.
`roiSize`	Size of the source and destination image ROI.
`borderType`	Type of border. Possible values are: • `ippBorderDefault` • The border is set to `ippBorderConst` with • `ippBorderRepl` • Border is replicated from the edge pixels. • `ippBorderMirror` • Border pixels are mirrored from the source image boundary • `ippBorderConst` • Values of all border pixels are set to a constant. • `ippBorderFirstStageInMem` • You can use this border type together with the

borderValue, borderValue[3], borderValue[4]	Constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.
pMorphSpec	Pointer to the morphology specification structure.
pBuffer	Pointer to the external buffer.

Description

Before using this function, you need to initialize the morphology specification structure using the ippiMorphInit function.

This function performs a black-hat operation on a rectangular ROI area inside a one-, three-, or four-channel 2D image using a specified in the advanced morphology state or specification structure mask and the anchor cell.

The result is equivalent to the subtraction of the initial source image from the closed source image.

NOTE The function can only process a ROI that does not exceed the maximum width and height roiSize specified by the initialization functions.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition when: • roiSize has a field with a zero or negative value • ROI width is more than ROI width passed to the initialization function
ippStsStepErr	Indicates an error condition when srcStep or dstStep is less than roiSize.width * <pixelSize>.
ippStsNotEvenStepErr	Indicates an error when one of the step values is not a multiple of an element size.
ippStsBadArgErr	Indicates an error when borderType has an illegal value.

See Also

Regions of Interest in Intel IPP

User-defined Border Types

MorphInit Initializes the internal state or specification structure for advanced morphological operations.

MorphBlackhatBorder

Performs black-hat operation on an image.

Syntax

```
IppStatus ippiMorphBlackhatBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType,
Ipp<datatype> borderValue, const IppiMorphAdvState* pSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

`8u_C1R`	`16u_C1R`	`16s_C1R`	`32f_C1R`

```
IppStatus ippiMorphBlackhatBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, const
Ipp<datatype> borderValue[3], const IppiMorphAdvState* pSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

`8u_C3R`	`32f_C3R`

```
IppStatus ippiMorphBlackhatBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, const
Ipp<datatype> borderValue[4], const IppiMorphAdvState* pSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

`8u_C4R`	`32f_C4R`

```
IppStatus ippiMorphBlackhatBorder_1u_C1R(const Ipp8u* pSrc, int srcStep, int
srcBitOffset, Ipp8u* pDst, int dstStep, int dstBitOffset, IppiSize roiSize,
IppiBorderType borderType, Ipp8u borderValue, const IppiMorphAdvState* pSpec, Ipp8u*
pBuffer);
```

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance, in bytes, between the starting points of consecutive lines in the source image.
`srcBitOffset`	Offset, in bits, from the first byte of the source image (for the `1u_C1R` flavor).
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance, in bytes, between the starting points of consecutive lines in the destination image.
`dstBitOffset`	Offset, in bits, from the first byte of the destination image (for the `1u_C1R` flavor).

roiSize	Size of the source and destination image ROI.
borderType	Type of border. Possible values are: • ippBorderRepl • Border is replicated from the edge pixels. • ippBorderInMem • Border is obtained from the source image pixels in memory.
borderValue, borderValue[3], borderValue[4]	Constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.
pSpec	Pointer to the specification structure.
pBuffer	Pointer to the external buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function performs black-hat operation on a rectangular ROI area inside a one-, three-, or four-channel 2D image using a specified in the advanced morphology specification structure mask and anchor cell. The structure must be initialized by MorphAdvInit beforehand.

The result is equivalent to the subtraction of the initial source image from the closed source image.

NOTE The function can process only images with ROI that does not exceed the maximum width and height roiSize specified by the initialization functions.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with a zero or negative value, or if one of ROI width or height is greater than corresponding size of ROI passed to the initialization functions.
ippStsStepErr	Indicates an error condition if srcStep or dstStep is less than roiSize.width * <pixelSize>.
ippStsNotEvenStepErr	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images.
ippStsBorderErr	Indicates an error condition if borderType has an illegal value.

See Also

MorphAdvInit Initializes the specification structure for advanced morphological operations.

MorphClose

Closes an image.

Syntax

```
IppStatus ippiMorphClose_8u_C1R_L(const Ipp8u* pSrc, IppSizeL srcStep, Ipp8u* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp8u
borderValue[1], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatus ippiMorphClose_8u_C3R_L(const Ipp8u* pSrc, IppSizeL srcStep, Ipp8u* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp8u
borderValue[3], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatus ippiMorphClose_8u_C4R_L(const Ipp8u* pSrc, IppSizeL srcStep, Ipp8u* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp8u
borderValue[4], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatus ippiMorphClose_16u_C1R_L(const Ipp16u* pSrc, IppSizeL srcStep, Ipp16u* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp16u
borderValue[1], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatus ippiMorphClose_16s_C1R_L(const Ipp16s* pSrc, IppSizeL srcStep, Ipp16s* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp16s
borderValue[1], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatus ippiMorphClose_1u_C1R_L(const Ipp8u* pSrc, IppSizeL srcStep, int
srcBitOffset, Ipp8u* pDst, IppSizeL dstStep, int dstBitOffset, IppiSizeL roiSize,
IppiBorderType borderType, const Ipp8u borderValue[1], const IppiMorphAdvStateL*
pMorphSpec, Ipp8u* pBuffer);

IppStatus ippiMorphClose_32f_C1R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp32f
borderValue[1], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatus ippiMorphClose_32f_C3R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp32f
borderValue[3], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatus ippiMorphClose_32f_C4R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp32f
borderValue[4], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer);
```

Include Files

`ippcv_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the source image ROI.

srcStep	Distance, in bytes, between the starting points of consecutive lines in the source image.
srcBitOffset	Offset, in bits, from the first byte of the source image row.
pDst	Pointer to the destination image ROI.
dstStep	Distance, in bytes, between the starting points of consecutive lines in the destination image.
dstBitOffset	Offset, in bits, from the first byte of the destination image row.
roiSize	Size of the source and destination image ROI.
borderType	Type of border. Possible values are: • ippBorderDefault • The border is set to ippBorderConst with • ippBorderRepl • Border is replicated from the edge pixels. • ippBorderMirror • Border pixels are mirrored from the source image boundary • ippBorderConst • Values of all border pixels are set to a constant. • ippBorderFirstStageInMem • You can use this border type together with the
borderValue, borderValue[3], borderValue[4]	Constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.
pMorphSpec	Pointer to the morphology specification structure.
pBuffer	Pointer to the external buffer.

Description

Before using this function, you need to initialize the morphology specification structure using the ippiMorphInit function.

This function operates with ROI.

This function performs closing of a rectangular ROI area inside a one-, three-, or four-channel 2D image using the mask specified in the pMorphSpec structure.

The result is equivalent to successive dilation of the source image by the structured element (mask) and erosion by the reverted structured element.

NOTE The function can only process a ROI that does not exceed the maximum width and height roiSize specified by the initialization functions.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition when: • roiSize has a field with a zero or negative value • ROI width is more than ROI width passed to the initialization function
ippStsStepErr	Indicates an error condition when srcStep or dstStep is less than roiSize.width * <pixelSize>.
ippStsNotEvenStepErr	Indicates an error when one of the step values is not a multiple of an element size.
ippStsBadArgErr	Indicates an error when borderType has an illegal value.

See Also

Regions of Interest in Intel IPP

User-defined Border Types

MorphInit Initializes the internal state or specification structure for advanced morphological operations.

MorphCloseBorder

Closes an image.

Syntax

```
IppStatus ippiMorphCloseBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType,
Ipp<datatype> borderValue, const IppiMorphAdvState* pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32f_C1R

```
IppStatus ippiMorphCloseBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, const
Ipp<datatype> borderValue[3], const IppiMorphAdvState* pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u_C3R	32f_C3R

```
IppStatus ippiMorphCloseBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, const
Ipp<datatype> borderValue[4], const IppiMorphAdvState* pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u_C4R	32f_C4R

```
IppStatus ippiMorphCloseBorder_1u_C1R(const Ipp8u* pSrc, int srcStep, int srcBitOffset,
Ipp8u* pDst, int dstStep, int dstBitOffset, IppiSize roiSize, IppiBorderType
borderType, Ipp8u borderValue, const IppiMorphAdvState* pSpec, Ipp8u* pBuffer);
```

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance, in bytes, between the starting points of consecutive lines in the source image.
srcBitOffset	Offset, in bits, from the first byte of the source image (for 1u_C1R flavor).
pDst	Pointer to the destination image ROI.
dstStep	Distance, in bytes, between the starting points of consecutive lines in the destination image.
dstBitOffset	Offset, in bits, from the first byte of the destination image (for 1u_C1R flavor).
roiSize	Size of the source and destination image ROI.
borderType	Type of border. Possible values are: • ippBorderRepl • Border is replicated from the edge pixels. • ippBorderInMem • Border is obtained from the source image pixels in memory.
borderValue, borderValue[3], borderValue[4]	Constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.
pSpec	Pointer to the specification structure.
pBuffer	Pointer to the external buffer required for dilation operations.

Description

Before using this function, you need to initialize the morphology specification structure using the MorphAdvInit function.

This function operates with ROI.

This function performs closing of a rectangular ROI area inside a one-, three-, or four-channel 2D image using the mask specified in the `pSpec` structure.

The result is equivalent to successive dilation of the source image by the structured element (mask) and erosion by the reverted structured element.

NOTE This function requires that the image ROI does not exceed the maximum width and height `roiSize` specified by the initialization functions.
Usage example of this function is similar to the example provided with the MorphOpenBorder function description.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition when: • `roiSize` has a field with a zero or negative value • one of the ROI width or height is more than the corresponding size of ROI passed to the initialization functions • `srcBitOffset` or `dstBitOffset` is less than zero
`ippStsStepErr`	Indicates an error condition if `srcStep` or `dstStep` is less than `roiSize.width` * <pixelSize>.
`ippStsNotEvenStepErr`	Indicates an error condition if one of the step values for 16-bit integer images is not divisible by 2.
`ippStsBorderErr`	Indicates an error condition if `borderType` has an illegal value.

Example

The code example below demonstrates how to use the `ippiMorphAdvGetSize_16u_C1R`, `ippiMorphAdvInit_16u_C1R`, and `ippiMorphCloseBorder_16u_C1R` functions.

```
IppStatus func_MorfCloseBorder()
{
    IppiMorphAdvState* pSpec = NULL;
    Ipp8u* pBuffer = NULL;
    IppiSize roiSize = {5, 5};
    Ipp8u pMask[3*3] = {1, 1, 1,
                        1, 0, 1,
                        1, 1, 1};
    IppiSize maskSize = {3, 3};
    Ipp16u pSrc[5*5] = { 1, 2, 4, 1, 2,
                         5, 1, 2, 1, 2,
                         1, 2, 1, 2, 1,
                         2, 1, 5, 1, 2};
```

```
    Ipp16u pDst[5*5];
    int srcStep = 5*sizeof(Ipp16u);
    int dstStep = 5*sizeof(Ipp16u);
    int dstSize = 5;
    IppStatus status = ippStsNoErr;
    int specSize = 0, bufferSize = 0;
    IppiBorderType borderType= ippBorderRepl;
    Ipp16u borderValue = 0;




    status = ippiMorphAdvGetSize_16u_C1R( roiSize, maskSize, &specSize, &bufferSize );
    if (status != ippStsNoErr) return status;


    pSpec = (IppiMorphAdvState*)ippsMalloc_8u(specSize);
    pBuffer = (Ipp8u*)ippsMalloc_8u(bufferSize);


    status = ippiMorphAdvInit_16u_C1R( roiSize, pMask, maskSize, pSpec, pBuffer );
    if (status != ippStsNoErr) {
        ippsFree(pBuffer);
        ippsFree(pSpec);
        return status;
    }


    status = ippiMorphCloseBorder_16u_C1R (pSrc, srcStep, pDst, dstStep, roiSize, borderType,
borderValue, pSpec, pBuffer);


    ippsFree(pBuffer);
    ippsFree(pSpec);
    return status;
}
```

The result is as follows:

```
pDst->
5 4 4 2 2
5 4 4 2 2
2 2 2 2 2
2 2 5 2 2
2 2 2 2 2
```

See Also

Regions of Interest in Intel IPP

User-defined Border Types

MorphAdvInit Initializes the specification structure for advanced morphological operations.

MorphGradient

Calculates morphological gradient of an image.

Syntax

```
IppStatus ippiMorphGradient_16u_C1R_L(const Ipp16u* pSrc, IppSizeL srcStep, Ipp16u*
pDst, IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp16u
borderValue[1], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatus ippiMorphGradient_16s_C1R_L(const Ipp16s* pSrc, IppSizeL srcStep, Ipp16s*
pDst, IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp16s
borderValue[1], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatus ippiMorphGradient_1u_C1R_L(const Ipp8u* pSrc, IppSizeL srcStep, int
srcBitOffset, Ipp8u* pDst, IppSizeL dstStep, int dstBitOffset, IppiSizeL roiSize,
IppiBorderType borderType, const Ipp8u borderValue[1], const IppiMorphAdvStateL*
pMorphSpec, Ipp8u* pBuffer);

IppStatus ippiMorphGradient_8u_C1R_L(const Ipp8u* pSrc, IppSizeL srcStep, Ipp8u* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp8u
borderValue[1], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatus ippiMorphGradient_8u_C3R_L(const Ipp8u* pSrc, IppSizeL srcStep, Ipp8u* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp8u
borderValue[3], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatus ippiMorphGradient_8u_C4R_L(const Ipp8u* pSrc, IppSizeL srcStep, Ipp8u* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp8u
borderValue[4], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatus ippiMorphGradient_32f_C1R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f*
pDst, IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp32f
borderValue[1], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatus ippiMorphGradient_32f_C3R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f*
pDst, IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp32f
borderValue[3], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer);

IppStatus ippiMorphGradient_32f_C4R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f*
pDst, IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp32f
borderValue[4], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer);
```

Include Files

`ippcv_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance, in bytes, between the starting points of consecutive lines in the source image.
`srcBitOffset`	Offset, in bits, from the first byte of the source image row.

pDst	Pointer to the destination image ROI.
dstStep	Distance, in bytes, between the starting points of consecutive lines in the destination image.
dstBitOffset	Offset, in bits, from the first byte of the destination image row.
roiSize	Size of the source and destination image ROI.
borderType	Type of border. Possible values are: • ippBorderDefault • The border is set to ippBorderConst with • ippBorderRepl • Border is replicated from the edge pixels. • ippBorderMirror • Border pixels are mirrored from the source image boundary • ippBorderConst • Values of all border pixels are set to a constant. • ippBorderFirstStageInMem • You can use this border type together with the
borderValue, borderValue[3], borderValue[4]	Constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.
pMorphSpec	Pointer to the morphology specification structure.
pBuffer	Pointer to the external buffer.

Description

Before using this function, you need to initialize the morphology specification structure using the ippiMorphInit function.

This function calculates a morphological gradient of a rectangular ROI area inside a one-, three-, or four-channel 2D image using a specified in the advanced morphology specification structure mask and anchor cell.

The result is equivalent to the subtraction of an opened source image from a closed source image.

NOTE The function can only process a ROI that does not exceed the maximum width and height roiSize specified by the initialization functions.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.

ippStsSizeErr	Indicates an error condition when: • roiSize has a field with a zero or negative value • ROI width is more than ROI width passed to the initialization function
ippStsStepErr	Indicates an error condition when srcStep or dstStep is less than roiSize.width * <pixelSize>.
ippStsNotEvenStepErr	Indicates an error when one of the step values is not a multiple of an element size.
ippStsBadArgErr	Indicates an error when borderType has an illegal value.

See Also

Regions of Interest in Intel IPP

User-defined Border Types

MorphInit Initializes the internal state or specification structure for advanced morphological operations.

MorphGradientBorder

Calculates morphological gradient of an image.

```
IppStatus ippiMorphGradientBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType,
Ipp<datatype> borderValue, const IppiMorphAdvState* pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32f_C1R

```
IppStatus ippiMorphGradientBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, const
Ipp<datatype> borderValue[3], const IppiMorphAdvState* pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u_C3R	32f_C3R

```
IppStatus ippiMorphGradientBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, const
Ipp<datatype> borderValue[4], const IppiMorphAdvState* pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u_C4R	32f_C4R

```
IppStatus ippiMorphGradientBorder_8u_C1R(const Ipp8u* pSrc, int srcStep, int
srcBitOffset, Ipp8u* pDst, int dstStep, int dstBitOffset, IppiSize roiSize,
IppiBorderType borderType, Ipp8u borderValue, const IppiMorphAdvState* pSpec, Ipp8u*
pBuffer);
```

Include Files

ippcv.h

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image.
`roiSize`	Size of the source and destination image ROI.
`borderType`	Type of border; the possible value is `ippBorderRepl`, which means that a replicated border is used.
`srcBitOffset`	Offset, in bits, from the first byte of the source image.
`dstBitOffset`	Offset, in bits, from the first byte of the destination image.
`pSpec`	Pointer to the specification structure.
`pBuffer`	Pointer to the external work buffer.
`borderValue`, `borderValue[3]`, `borderValue[4]`	Constant value to assign to pixels of the constant border. This parameter is applicable only to the `ippBorderConst` border type.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function calculates a morphological gradient of a rectangular ROI area inside a one-, three-, or four-channel 2D image using a specified in the advanced morphology specification structure mask and anchor cell. The structure must be initialized by MorphAdvInit beforehand.

The result is equivalent to the subtraction of an opened source image from a closed source image.

NOTE The function can process only images with ROI that does not exceed the maximum width and height `roiSize` specified by the initialization functions.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with a zero or negative value, or if one of ROI width or height is greater than corresponding size of ROI passed to the initialization functions.
ippStsStepErr	Indicates an error condition if srcStep or dstStep is less than roiSize.width * <pixelSize>.
ippStsNotEvenStepErr	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images.
ippStsBorderErr	Indicates an error condition if borderType has an illegal value.

See Also

MorphAdvInit Initializes the specification structure for advanced morphological operations.

MorphOpen

Opens an image.

Syntax

```
IppStatus ippiMorphOpen_16u_C1R_L(const Ipp16u* pSrc, IppSizeL srcStep, Ipp16u* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp16u
borderValue[1], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer );

IppStatus ippiMorphOpen_16s_C1R_L(const Ipp16s* pSrc, IppSizeL srcStep, Ipp16s* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp16s
borderValue[1], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer );

IppStatus ippiMorphOpen_1u_C1R_L(const Ipp8u* pSrc, IppSizeL srcStep, int srcBitOffset,
Ipp8u* pDst, IppSizeL dstStep, int dstBitOffset, IppiSizeL roiSize, IppiBorderType
borderType, const Ipp8u borderValue[1], const IppiMorphAdvStateL* pMorphSpec, Ipp8u*
pBuffer );

IppStatus ippiMorphOpen_8u_C1R_L(const Ipp8u* pSrc, IppSizeL srcStep, Ipp8u* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp8u
borderValue[1], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer );

IppStatus ippiMorphOpen_8u_C3R_L(const Ipp8u* pSrc, IppSizeL srcStep, Ipp8u* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp8u
borderValue[3], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer );

IppStatus ippiMorphOpen_8u_C4R_L(const Ipp8u* pSrc, IppSizeL srcStep, Ipp8u* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp8u
borderValue[4], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer );

IppStatus ippiMorphOpen_32f_C1R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp32f
borderValue[1], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer);
```

```
IppStatus ippiMorphOpen_32f_C3R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp32f
borderValue[3], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer );

IppStatus ippiMorphOpen_32f_C4R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp32f
borderValue[4], const IppiMorphAdvStateL* pMorphSpec, Ipp8u* pBuffer );
```

Include Files

`ippcv_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance, in bytes, between the starting points of consecutive lines in the source image.
`srcBitOffset`	Offset, in bits, from the first byte of the source image row.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance, in bytes, between the starting points of consecutive lines in the destination image.
`dstBitOffset`	Offset, in bits, from the first byte of the destination image row.
`roiSize`	Size of the source and destination image ROI.
`borderType`	Type of border. Possible values are: • `ippBorderDefault` • The border is set to `ippBorderConst` with • `ippBorderRepl` • Border is replicated from the edge pixels. • `ippBorderMirror` • Border pixels are mirrored from the source image boundary • `ippBorderConst` • Values of all border pixels are set to a constant. • `ippBorderFirstStageInMem` • You can use this border type together with the
`borderValue`, `borderValue[3]`, `borderValue[4]`	Constant value to assign to pixels of the constant border. This parameter is applicable only to the `ippBorderConst` border type.
`pMorphSpec`	Pointer to the morphology specification structure.

pBuffer	Pointer to the external buffer.

Description

Before using this function, you need to initialize the morphology specification structure using the ippiMorphInit function.

This function operates with ROI.

This function performs opening of a rectangular ROI area inside a one-, three-, or four-channel 2D image using the mask specified in the pMorphSpec structure.

The result is equivalent to successive dilation of the source image by the structured element (mask) and erosion by the reverted structured element.

NOTE The function can only process a ROI that does not exceed the maximum width and height roiSize specified by the initialization functions.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition when: • roiSize has a field with a zero or negative value • ROI width is more than ROI width passed to the initialization function
ippStsStepErr	Indicates an error condition when srcStep or dstStep is less than roiSize.width * <pixelSize>.
ippStsNotEvenStepErr	Indicates an error when one of the step values is not a multiple of an element size.
ippStsBadArgErr	Indicates an error when borderType has an illegal value.

See Also

Regions of Interest in Intel IPP

User-defined Border Types

MorphInit Initializes the internal state or specification structure for advanced morphological operations.

MorphOpenBorder

Opens an image.

Syntax

```
IppStatus ippiMorphOpenBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType,
Ipp<datatype> borderValue, const IppiMorphAdvState* pSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

`8u_C1R`	`16u_C1R`	`16s_C1R`	`32f_C1R`

```
IppStatus ippiMorphOpenBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, const
Ipp<datatype> borderValue[3], const IppiMorphAdvState* pSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

`8u_C3R`	`32f_C3R`

```
IppStatus ippiMorphOpenBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, const
Ipp<datatype> borderValue[4], const IppiMorphAdvState* pSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

`8u_C4R`	`32f_C4R`

```
IppStatus ippiMorphOpenBorder_1u_C1R(const Ipp8u* pSrc, int srcStep, int srcBitOffset,
Ipp8u* pDst, int dstStep, int dstBitOffset, IppiSize roiSize, IppiBorderType
borderType, Ipp8u borderValue, const IppiMorphAdvState* pSpec, Ipp8u* pBuffer);
```

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance, in bytes, between the starting points of consecutive lines in the source image.
`srcBitOffset`	Offset, in bits, from the first byte of the source image (for the `1u_C1R` flavor).
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance, in bytes, between the starting points of consecutive lines in the destination image.
`dstBitOffset`	Offset, in bits, from the first byte of the destination image (for the `1u_C1R` flavor).
`roiSize`	Size of the source and destination image ROI.

borderType	Type of border. Possible values are: • ippBorderRepl • Border is replicated from the edge pixels. • ippBorderInMem • Border is obtained from the source image pixels in memory.
borderValue, borderValue[3], borderValue[4]	Constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.
pSpec	Pointer to the specification structure.
pBuffer	Pointer to the external buffer required for dilation operations.

Description

Before using this function, you need to initialize the morphology specification structure by using MorphAdvInit function.

This function operates with ROI.

This function performs opening of a rectangular ROI area inside a one-, three-, or four-channel 2D image using the mask specified in the pSpec structure.

The result is equivalent to successive dilation of the source image by the structured element (mask) and erosion by the reverted structured element.

NOTE The function can only process a ROI that does not exceed the maximum width and height roiSize specified by the initialization functions.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition when: • roiSize has a field with a zero or negative value • one of the ROI width or height is more than the corresponding size of ROI passed to the initialization functions • srcBitOffset or dstBitOffset is less than zero
ippStsStepErr	Indicates an error condition if srcStep or dstStep is less than roiSize.width * <pixelSize>.
ippStsNotEvenStepErr	Indicates an error condition if one of the step values for 16-bit integer images is not divisible by 2.

ippStsBorderErr	Indicates an error condition if borderType has an illegal value.

See Also

Regions of Interest in Intel IPP

User-defined Border Types

MorphAdvInit Initializes the specification structure for advanced morphological operations.

MorphTophat

Performs top-hat operation on an image.

Syntax

```
IppStatus ippiMorphTophat_16u_C1R_L(const Ipp16u* pSrc, IppSizeL srcStep, Ipp16u* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp16u
borderValue[1], const IppiMorphAdvStateL* pMorthSpec, Ipp8u* pBuffer);

IppStatus ippiMorphTophat_16s_C1R_L(const Ipp16s* pSrc, IppSizeL srcStep, Ipp16s* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp16s
borderValue[1], const IppiMorphAdvStateL* pMorthSpec, Ipp8u* pBuffer);

IppStatus ippiMorphTophat_1u_C1R_L(const Ipp8u* pSrc, IppSizeL srcStep, int
srcBitOffset, Ipp8u* pDst, IppSizeL dstStep, int dstBitOffset, IppiSizeL roiSize,
IppiBorderType borderType, const Ipp8u borderValue[1], const IppiMorphAdvStateL*
pMorthSpec, Ipp8u* pBuffer);

IppStatus ippiMorphTophat_8u_C1R_L(const Ipp8u* pSrc, IppSizeL srcStep, Ipp8u* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp8u
borderValue[1], const IppiMorphAdvStateL* pMorthSpec, Ipp8u* pBuffer);

IppStatus ippiMorphTophat_8u_C3R_L(const Ipp8u* pSrc, IppSizeL srcStep, Ipp8u* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp8u
borderValue[3], const IppiMorphAdvStateL* pMorthSpec, Ipp8u* pBuffer);

IppStatus ippiMorphTophat_8u_C4R_L(const Ipp8u* pSrc, IppSizeL srcStep, Ipp8u* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp8u
borderValue[4], const IppiMorphAdvStateL* pMorthSpec, Ipp8u* pBuffer);

IppStatus ippiMorphTophat_32f_C1R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp32f
borderValue[1], const IppiMorphAdvStateL* pMorthSpec, Ipp8u* pBuffer);

IppStatus ippiMorphTophat_32f_C3R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp32f
borderValue[3], const IppiMorphAdvStateL* pMorthSpec, Ipp8u* pBuffer);

IppStatus ippiMorphTophat_32f_C4R_L(const Ipp32f* pSrc, IppSizeL srcStep, Ipp32f* pDst,
IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType, const Ipp32f
borderValue[4], const IppiMorphAdvStateL* pMorthSpec, Ipp8u* pBuffer);
```

Include Files

ippcv_l.h

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance, in bytes, between the starting points of consecutive lines in the source image.
`srcBitOffset`	Offset, in bits, from the first byte of the source image row.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance, in bytes, between the starting points of consecutive lines in the destination image.
`dstBitOffset`	Offset, in bits, from the first byte of the destination image row.
`roiSize`	Size of the source and destination image ROI.
`borderType`	Type of border. Possible values are: • `ippBorderDefault` • The border is set to `ippBorderConst` with • `ippBorderRepl` • Border is replicated from the edge pixels. • `ippBorderMirror` • Border pixels are mirrored from the source image boundary • `ippBorderConst` • Values of all border pixels are set to a constant. • `ippBorderFirstStageInMem` • You can use this border type together with the
`borderValue`, `borderValue[3]`, `borderValue[4]`	Constant value to assign to pixels of the constant border. This parameter is applicable only to the `ippBorderConst` border type.
`pMorphSpec`	Pointer to the morphology specification structure.
`pBuffer`	Pointer to the external buffer.

Description

Before using this function, you need to initialize the morphology specification structure using the ippiMorphInit function.

This function performs a top-hat operation on a rectangular ROI area inside a one-, three-, or four-channel 2D image using a specified in the advanced morphology state or specification structure mask and the anchor cell.

The result is equivalent to the opening the source image and following subtraction from the initial source image.

NOTE The function can only process a ROI that does not exceed the maximum width and height `roiSize` specified by the initialization functions.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition when: • `roiSize` has a field with a zero or negative value • ROI width is more than ROI width passed to the initialization function
`ippStsStepErr`	Indicates an error condition when `srcStep` or `dstStep` is less than `roiSize.width` * <pixelSize>.
`ippStsNotEvenStepErr`	Indicates an error when one of the step values is not a multiple of an element size.
`ippStsBadArgErr`	Indicates an error when `borderType` has an illegal value.

See Also

Regions of Interest in Intel IPP

User-defined Border Types

MorphInit Initializes the internal state or specification structure for advanced morphological operations.

MorphTophatBorder

Performs top-hat operation on an image.

Syntax

Case 1: Operating with morphology state structure

```
IppStatus ippiMorphTophatBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType,
Ipp<datatype> borderValue, const IppiMorphAdvState* pSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

`8u_C1R`	`32f_C1R`

```
IppStatus ippiMorphTophatBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, const
Ipp<datatype> borderValue[3], const IppiMorphAdvState* pSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

8u_C3R	32f_C3R

```
IppStatus ippiMorphTophatBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, const
Ipp<datatype> borderValue[4], const IppiMorphAdvState* pSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

8u_C4R	32f_C4R

Case 2: Operating with morphology specification structure

```
IppStatus ippiMorphTophatBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType,
Ipp<datatype> borderValue, const IppiMorphAdvState* pSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

16u_C1R	16s_C1R

```
IppStatus ippiMorphTophatBorder_1u_C1R(const Ipp8u* pSrc, int srcStep, int
srcBitOffset, Ipp8u* pDst, int dstStep, int dstBitOffset, IppiSize roiSize,
IppiBorderType borderType, Ipp8u borderValue, const IppiMorphAdvState* pSpec, Ipp8u*
pBuffer);
```

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance, in bytes, between the starting points of consecutive lines in the source image.
`srcBitOffset`	Offset, in bits, from the first byte of the source image (for the `1u_C1R` flavor).
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance, in bytes, between the starting points of consecutive lines in the destination image.
`dstBitOffset`	Offset, in bits, from the first byte of the destination image (for the `1u_C1R` flavor).
`roiSize`	Size of the source and destination image ROI.
`borderType`	Type of border. Possible values are: • `ippBorderRepl` • Border is replicated from the edge pixels.

	• ippBorderInMem • Border is obtained from the source image pixels in memory.
borderValue, borderValue[3], borderValue[4]	Constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.
pSpec	Pointer to the specification structure.
pBuffer	Pointer to the external buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function performs a top-hat operation on a rectangular ROI area inside a one-, three-, or four-channel 2D image using a specified in the advanced morphology state or specification structure mask and the anchor cell. The structure must be initialized by MorphAdvInit beforehand.

The result is equivalent to the opening the source image and following subtraction from the initial source image.

NOTE The function can process only images with ROI that does not exceed the maximum width and height roiSize specified by the initialization functions.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with a zero or negative value, or if one of ROI width or height is greater than corresponding size of ROI passed to the initialization functions.
ippStsStepErr	Indicates an error condition if srcStep or dstStep is less than roiSize.width * <pixelSize>.
ippStsNotEvenStepErr	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images.
ippStsBorderErr	Indicates an error condition if borderType has an illegal value.

See Also

MorphAdvInit Initializes the specification structure for advanced morphological operations.

MorphGrayInit

Initializes morphology state structure for gray-kernel morphology operations.

Syntax

```
IppStatus ippiMorphGrayInit_8u_C1R(IppiMorphGrayState_8u* pState, IppiSize roiSize,
const Ipp32s* pMask, IppiSize maskSize, IppiPoint anchor);
```

```
IppStatus ippiMorphGrayInit_32f_C1R(IppiMorphGrayState_32f* pState, IppiSize roiSize,
const Ipp32f* pMask, IppiSize maskSize, IppiPoint anchor);
```

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pState`	Pointer to the gray-kernel morphology state structure.
`roiSize`	Maximal size of the image ROI in pixels, that can be processed using the allocated structure.
`pMask`	Pointer to the mask.
`maskSize`	Size of the mask in pixels.
`anchor`	Coordinates of the anchor cell.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function initializes the gray-kernel morphology state structure `pState` in the external buffer. Its size should be computed by the function MorphGrayGetSize. It is used by the functions GrayDilateBorder and GrayErodeBorder that perform gray-kernel dilation and erosion of the source image pixels corresponding to the specified `pMask` of size `maskSize`. The anchor cell `anchor` is positioned in the arbitrary point in the mask and is used for positioning the mask.

NOTE
warning
The structure can be used to process images with ROI that does not exceed the specified maximum width and height `roiSize`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error when one of the specified pointers is `NULL`.

ippStsSizeErr	Indicates an error condition if maskSize or if roiSize has a field with a zero or negative value.
ippStsAnchorErr	Indicates an error if anchor is outside the mask.

MorphGrayGetSize

Computes the size of the gray-kernel morphology state structure.

Syntax

```
IppStatus ippiMorphGrayGetSize_8u_C1R(IppiSize roiSize, const Ipp32s* pMask, IppiSize maskSize, int* pSize);
```

```
IppStatus ippiMorphGrayGetSize_32f_C1R(IppiSize roiSize, const Ipp32f* pMask, IppiSize maskSize, int* pSize);
```

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

roiSize	Maximal size of the image ROI in pixels, that can be processed using the allocated structure.
pMask	Pointer to the mask.
maskSize	Size of the mask in pixels.
pSize	Pointer to the size of the advanced morphology state structure.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function computes the size of the morphology state structure pState for gray-kernel dilation and erosion. This function should be run prior to the function MorphGrayInit.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if maskSize or if roiSize has a field with a zero or negative value.

MorphReconstructGetBufferSize

Computes the size of the buffer for morphological reconstruction operation.

```
IppStatus ippiMorphReconstructGetBufferSize(IppiSize roiSize, IppDataType dataType, int numChannels, int* pBufSize);
```

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`roiSize`	Maximal size of the image ROI in pixels, that can be processed using the buffer.
`dataType`	Data type of the image.
`numChannels`	Number of channels in the image.
`pBufSize`	Pointer to the size of the work buffer (in bytes), returned by the `ippiMorphReconstructGetBufferSize` function.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function computes the size of the buffer for the morphological reconstruction of the source image. This buffer can be used by the functions MorphReconstructDilate and MorphReconstructErode.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if `pSize` is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize` has a field with a zero or negative value.

MorphReconstructDilate

Reconstructs an image by dilation.

Syntax

```
IppStatus ippiMorphReconstructDilate_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, Ipp8u* pBuffer, IppiNorm norm);
```

Supported values for `mod`:

`8u_C1IR`	`16u_C1IR`	`64f_C1IR`

```
IppStatus ippiMorphReconstructDilate_32f_C1IR(const Ipp32f* pSrc, int srcStep, Ipp32f*
pSrcDst, int srcDstStep, IppiSize roiSize, Ipp32f* pBuffer, IppiNorm norm);
```

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`pSrcDst`	Pointer to the descreased and reconstructed image ROI.
`srcDstStep`	Distance in bytes between starts of consecutive lines in the decreased and reconstructed image.
`roiSize`	Size of the source and destination image ROI.
`norm`	Type of norm to form the mask for dilation; the following values are possible: • `ippiNormInf` • Infinity norm (8-connectivity, 3x3 rectangular mask). • `ippiNormL1` • L1 norm (4-connectivity, 3x3 cross mask).
`pBuffer`	Pointer to the buffer.

Description

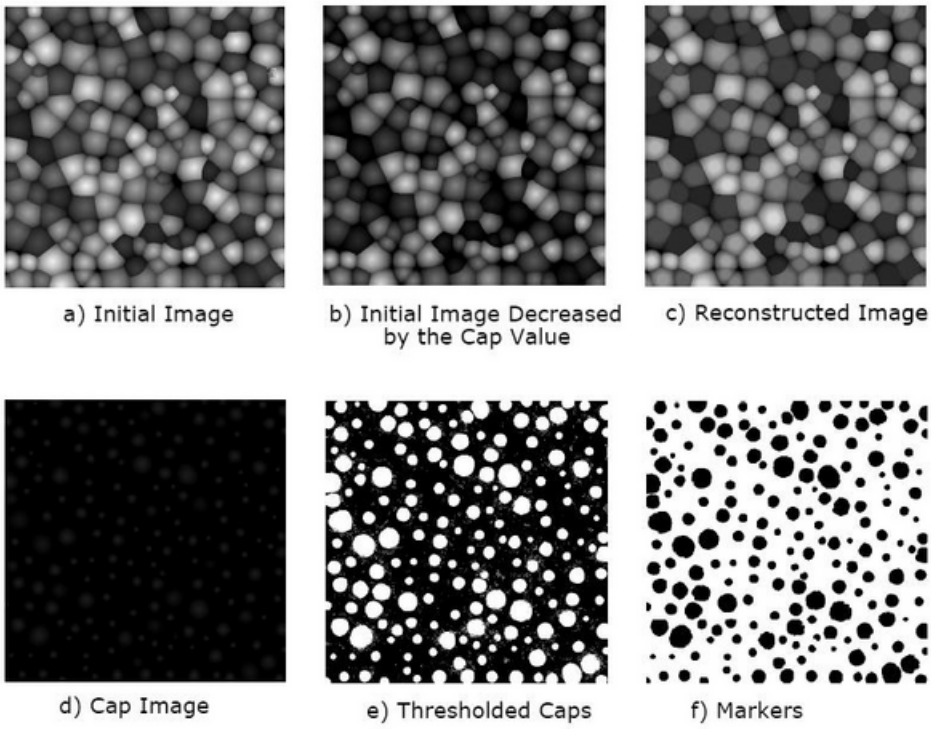
This function operates with ROI (see Regions of Interest in Intel IPP).

This function performs morphological reconstruction of the decreased source image by dilation Vincent93. The operation is performed in the working buffer whose size should be computed using the function MorphReconstructGetBufferSize beforehand.

This operation enables detection of the regional maximums that can be used as markers for successive watershed segmentation.

Example below shows how the morphological reconstruction can be used to build markers of objects with different brightness. Some value (cap size) is subtracted from the initial image and then the subtracted image is reconstructed to the initial one. Thresholding and opening complete the building of markers. The figure below shows the results of these operations.

Building Markers for Segmentation by the Morphological Reconstruction |



Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize` has a field with a zero or negative value.
`ippStsStepErr`	Indicates an error condition if `srcStep` or `srcDstStep` is less than `roiSize.width` * `<pixelSize>`.
`ippStsNotEvenStepErr`	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images.
`ippStsBadArgErr`	Indicates an error condition if `norm` has an illegal value.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
```

```
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 ********************************************************************************/

// Performs morphological reconstruction of pSrcDst under/above pSrc
//     ippiMorphReconstructGetBufferSize
//     ippiMorphReconstructDilate_8u_C1IR

#include <stdio.h>
#include "ipp.h"

#define WIDTH   128 /* Image width  */
#define HEIGHT  128 /* Image height */
#define KERSIZE 3   /* Kernel size  */

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define
check_sts(st)
                                  \
    if ((st) !=
ippStsNoErr)
                         \
        goto exitLine; /* Go to Exit if Intel(R) Integrated Performance Primitives (Intel(R)
IPP) function returned status different from \
                          ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main(void)
{
    IppStatus status = ippStsNoErr;
    IppiMorphAdvState *pState = NULL;
    IppiSize roiSize = {WIDTH, HEIGHT};             /* image size */
    IppiSize maskSize = {3, 3};                     /* mask size */
    Ipp8u *pSrc = NULL, *pDst = NULL, *pImg = NULL; /*  initial and working image */
    Ipp8u *pBuf = NULL, *pAdvBuf = NULL, pMask[9] = {1, 1, 1, 1, 1, 1, 1, 1, 1};
    int srcStep = 0;                            /* srcStep, in bytes, through the source image */
    int size = 0, specSize = 0, bufferSize = 0; /*  working buffers size */
    int cap = 100;                              /* cap value */

    pSrc = ippiMalloc_8u_C1(roiSize.width, roiSize.height, &srcStep);

    pDst = ippiMalloc_8u_C1(roiSize.width, roiSize.height, &srcStep);

    pImg = ippiMalloc_8u_C1(roiSize.width, roiSize.height, &srcStep);

    check_sts(status = ippiMorphReconstructGetBufferSize(roiSize, ipp8u, 1, &size));

    pBuf = ippsMalloc_8u(size);
```

```
    check_sts(status = ippiMorphAdvGetSize_8u_C1R(roiSize, maskSize, &specSize, &bufferSize));

    pState = (IppiMorphAdvState *)ippsMalloc_8u(specSize);

    pAdvBuf = (Ipp8u *)ippsMalloc_8u(bufferSize);

    check_sts(status = ippiMorphAdvInit_8u_C1R(roiSize, pMask, maskSize, pState, pAdvBuf));

    check_sts(status = ippiCopy_8u_C1R(pSrc, srcStep, pDst, srcStep, roiSize));
    check_sts(status = ippiCopy_8u_C1R(pSrc, srcStep, pImg, srcStep, roiSize));
    /* subtract cap size */
    check_sts(status = ippiSubC_8u_C1IRSfs(cap, pDst, srcStep, roiSize, 0));
    /* reconstruct image */
    check_sts(status = ippiMorphReconstructDilate_8u_C1IR(pSrc, srcStep, pDst, srcStep, roiSize,
pBuf, (IppiNorm)ippiNormL1));
    /* get caps */
    check_sts(status = ippiSub_8u_C1IRSfs(pDst, srcStep, pImg, srcStep, roiSize, 0));
    /* caps to white */
    check_sts(status = ippiThreshold_GTVal_8u_C1IR(pImg, srcStep, roiSize, 0, 255));
    /* delete noise */
    check_sts(status = ippiMorphOpenBorder_8u_C1R(pImg, srcStep, pDst, srcStep, roiSize,
ippBorderRepl, 0, pState, pAdvBuf));
    /* revert to get markers */
    check_sts(status = ippiXorC_8u_C1IR(0xff, pDst, srcStep, roiSize));

    EXIT_MAIN
    ippsFree(pBuf);
    ippsFree(pState);
    ippsFree(pAdvBuf);
    ippiFree(pSrc);
    ippiFree(pDst);
    ippiFree(pImg);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return status;
}
```

MorphReconstructErode

Reconstructs an image by erosion.

Syntax

`IppStatus ippiMorphReconstructErode_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize, Ipp8u* pBuf, IppiNorm norm);`

Supported values for `mod`:

8u_C1IR	16u_C1IR	64f_C1IR

`IppStatus ippiMorphReconstructErode_32f_C1IR(const Ipp32f* pSrc, int srcStep, Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize, Ipp32f* pBuf, IppiNorm norm);`

Include Files

`ippcv.h`.

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pSrcDst	Pointer to the decreased and reconstructed image ROI.
srcDstStep	Distance in bytes between starts of consecutive lines in the decreased and reconstructed image.
roiSize	Size of the source and destination image ROI.
norm	Type of norm to form the mask for dilation; the following values are possible: • ippiNormInf • Infinity norm (8-connectivity, 3x3 rectangular mask). • ippiNormL1 • L1 norm (4-connectivity, 3x3 cross mask).
pBuf	Pointer to the buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function performs morphological reconstruction of the increased source image by erosion Vincent93. The operation is performed in the working buffer whose size should be computed using the function MorphReconstructGetBufferSize beforehand.

This operation enables detection of the regional minimums that can be used as markers for successive watershed segmentation.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with a zero or negative value.
ippStsStepErr	Indicates an error condition if srcStep or srcDstStep is less than roiSize.width * <pixelSize>.
ippStsNotEvenStepErr	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images.
ippStsBadArgErr	Indicates an error condition if norm has an illegal value.

MorphSetMode

Sets the mask processing mode for advanced morphological operations.

Syntax

```
IppStatus ippiMorphSetMode(int mode, IppiMorphAdvState* pMorphSpec);
IppStatus ippiMorphSetMode_L(int mode, IppiMorphAdvStateL* pMorphSpec);
```

Include Files

`ippcv.h`

`ippcv_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`mode`	Mask processing `mode`; supported values: `IPP_MORPH_DEFAULT` Invert the mask `IPP_MORPH_MASK_NO_FLIP` Do not invert the mask; use the same mask for the first and second stage.
`pMorphSpec`	Pointer to the specification structure for advanced morphological operations.

Description

This function sets the mask processing `mode` for the second stage of an advanced morphological operation. Before using this function, initialize the specification structure using the `ippiMorphInit` function.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNotSupportedModeErr`	Indicates an error when `mode` has an invalid value.
`ippStsNullPtrErr`	Indicates an error when `pMorphSpec` is `NULL`.

See Also

MorphInit Initializes the internal state or specification structure for advanced morphological operations.

Filtering Functions

This section describes the Intel® IPP image processing functions that perform linear and non-linear filtering operations on an image.

You can use filtering in image processing operations like edge detection, blurring, noise removal, and feature detection.

Most filtering functions operate with regions of interest (ROI). The size of the source image ROI is equal to the destination image ROI size `dstRoiSize`. Most functions use different source and destination image buffers. These functions are not in-place.

See Also

Regions of Interest in Intel IPP

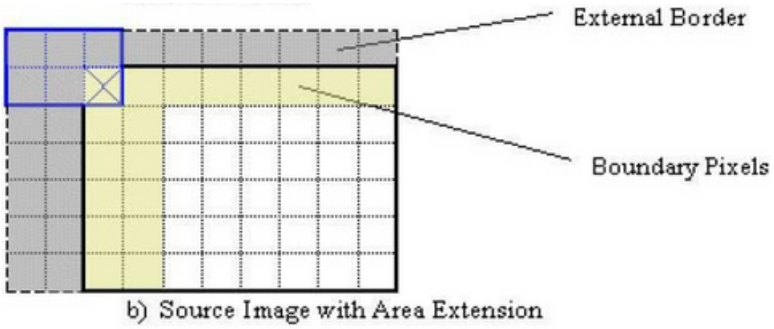
- Borders in Neighborhood Operations
- User-defined Border Types
- Filters with Borders
- Median Filters
- General Linear Filters
- Separable Filters
- Smoothing Filters
- Wiener Filters
- Convolution
- Deconvolution
- Fixed Filters
- Deinterlacing Filters

Borders in Neighborhood Operations

Filtering functions described in this section perform neighborhood operations. They operate on the assumption that for each pixel to be processed, all neighborhood pixels required for the operation are also available.

The neighborhood for each given pixel is defined by the filter kernel (or mask) size and anchor cell position. For more information about anchors and how to define the anchor cell position refer to Neighborhood Operations.

As the following figure illustrates, if the input pixel is near the horizontal or vertical edge of the image, the overlaid kernel may refer to neighborhood pixels that do not exist within the source image and are located outside the image area.

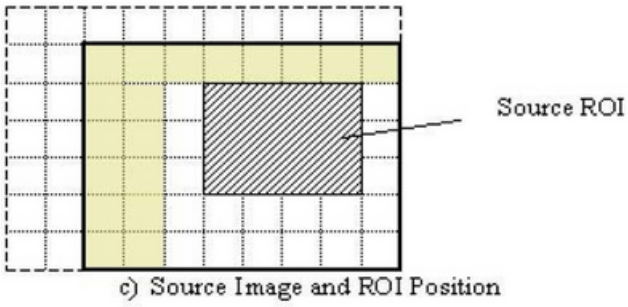


The set of all boundary source image pixels that require such non-existent pixels to complete the neighborhood operation for the given kernel and anchor is shaded yellow, while the collection of all scanned external pixels (called border pixels) is shaded gray.

If you want to apply some filtering operation to the whole source image, you must figure out what additional border pixels are required for the operation, and define these non-existent pixels. To do this, you can use the Intel IPP functions `ippiCopyConstBorder`, `ippiCopyReplicateBorder`, or `ippiCopyWrapBorder`, which fill the border of the extended image with the pixel values that you define, or you can apply your own extension method.

NOTE If the required border pixels are not defined prior to the filtering function call, you may get memory violation errors.

If you want to apply the filtering operation to the part of the source image, or ROI, then the necessity of extending the image area with border pixels depends on the ROI size and position within the image. The figure below shows that if ROI does not cover yellow (internal boundary) pixels, then no external pixels are scanned, and border extension is not required.



c) Source Image and ROI Position

If boundary pixels are part of ROI, you still need to extend some area of the source image.

To provide valid results of filtering operations, the application must check the following:

- ROI parameters passed to the filtering function have such values
- All required neighborhood pixels actually exist in the image and define the missing pixels when necessary.

See Also

Neighborhood Operations

Regions of Interest in Intel IPP

User-defined Border Types

CopyConstBorder

CopyReplicateBorder

CopyWrapBorder

User-defined Border Types

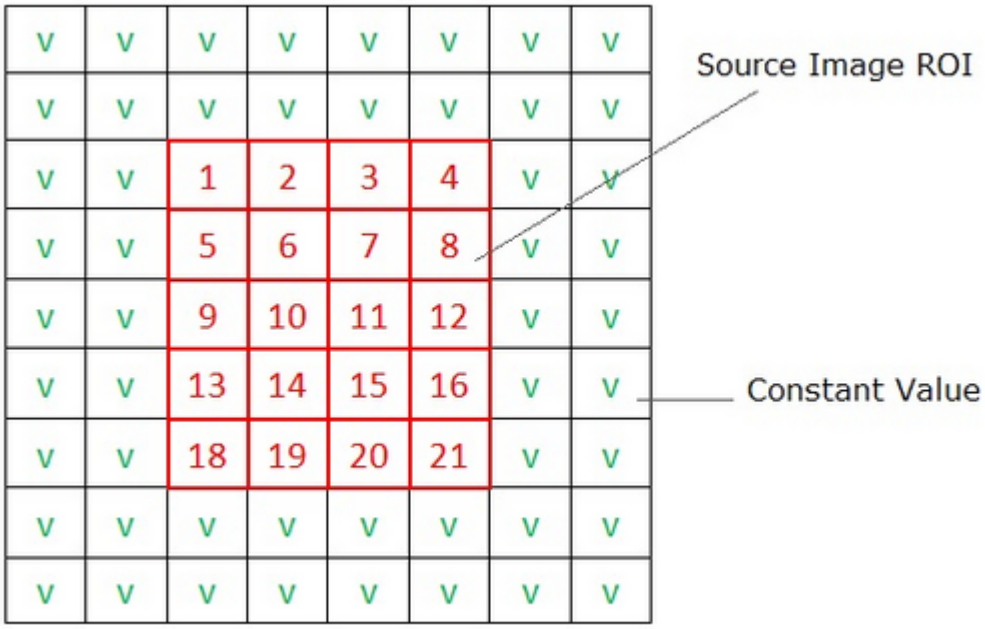
Some of the Intel® IPP image processing functions operate on user-defined border types. It means that the values of border pixels are assigned in accordance with the `borderType` (or `border`) and `borderValue` parameters.

Intel® IPP supports the following border types:

- Constant border
- Replicated border
- Mirrored border
- Mirrored border with replication
- Border in memory
- Mixed borders

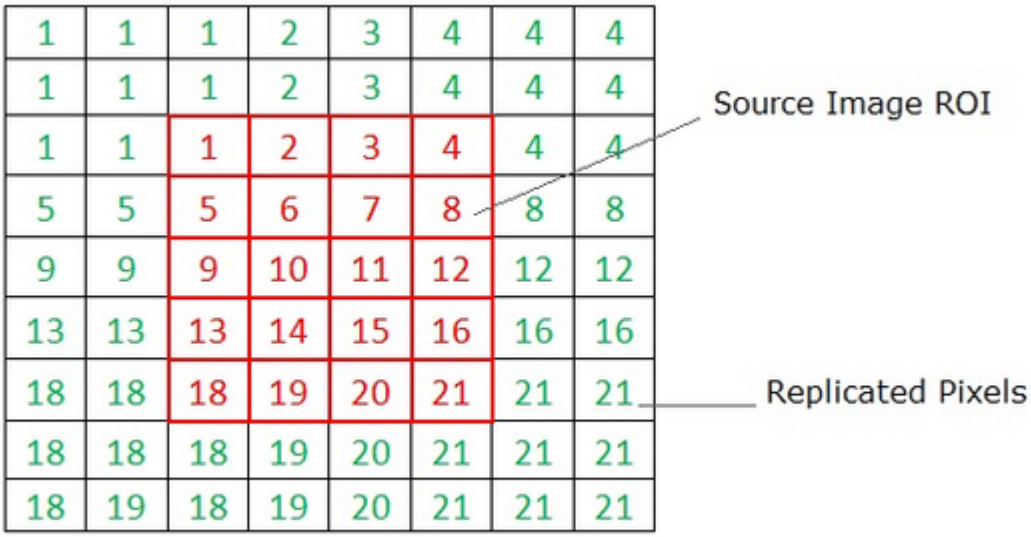
Constant Border

This type of border corresponds to the `ippBorderConst` value in the `IppiBorderType` enumerator. When using a constant border, values for all border pixels are set to the constant value that you specify in the `borderValue` parameter. In the figure below, this constant value is marked as `v`. Squares marked in red correspond to pixels copied from the source image ROI.



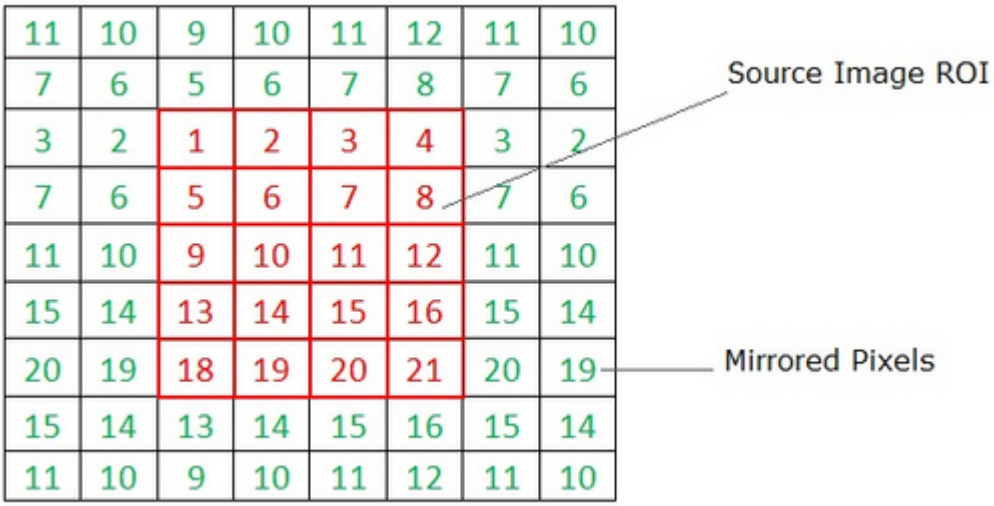
Replicated Border

This type of border corresponds to the `ippBorderRepl` value in the `IppiBorderType` enumerator. When using a replicated border, values for border pixels are obtained from the source image boundary pixels, as shown in the figure below. Squares marked in red correspond to pixels copied from the source image ROI. Squares with green values correspond to border pixels, which are replicated from the boundary pixels of the source image.



Mirrored Border

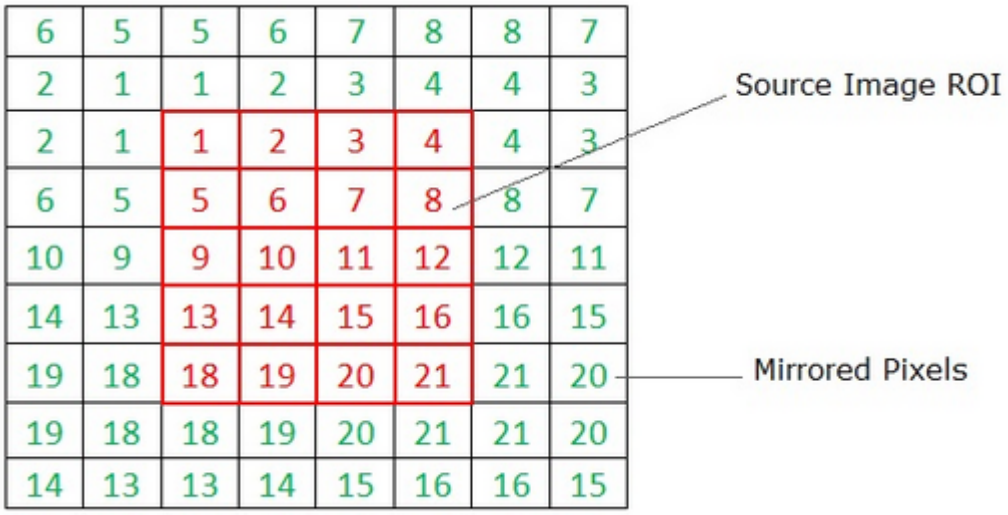
This type of border corresponds to the `ippBorderMirror` value in the `IppiBorderType` enumerator. When using a mirrored border, values for border pixels are obtained from the source image boundary pixels, as shown in the figure below. Squares marked in red correspond to pixels copied from the source image ROI. Squares with green values correspond to border pixels, which are mirrored from the source image pixels.



Mirrored Border with Replication

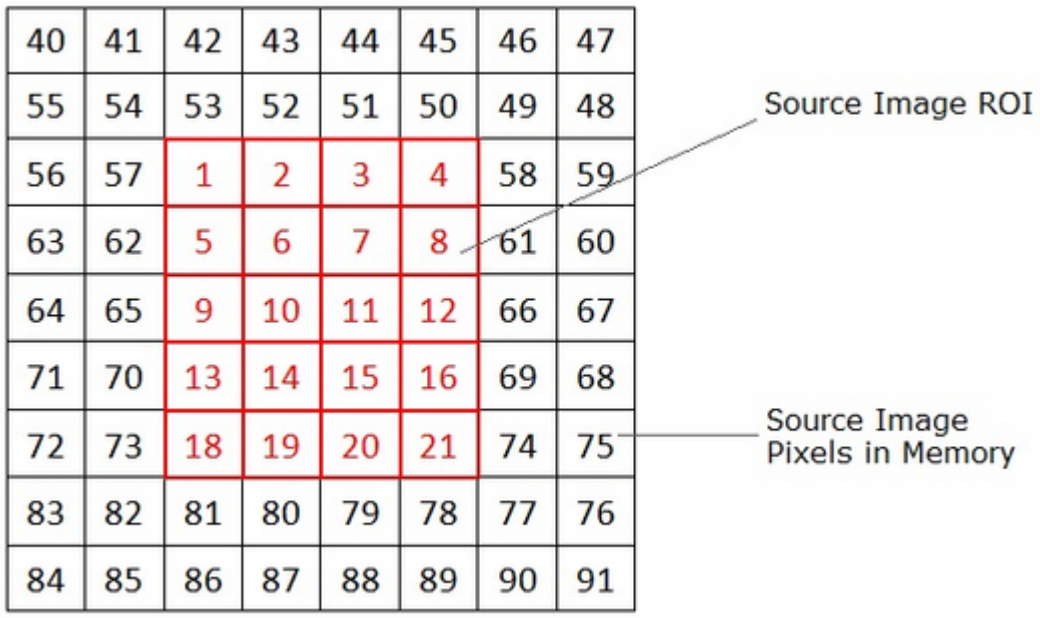
This type of border corresponds to the `ippBorderMirrorR` value in the `IppiBorderType` enumerator. When using a mirrored border with replication, values for border pixels are obtained from the source image boundary pixels, as shown in the figure below. Squares marked in red correspond to pixels copied from the

source image ROI. Squares with green values correspond to border pixels, which are mirrored from the source image pixels. The difference of this border type from the mirrored border is that the anchor cell value is replicated to the border pixels.



Border in Memory

This type of border corresponds to the `ippBorderInMem` value and its flags combinations in the `IppiBorderType` enumerator. Use this border type if the ROI does not cover internal border pixels of the source image. In this case, values for border pixels are obtained from the source image pixels in memory. In the figure below, squares marked in red correspond to pixels copied from the source image ROI. Squares with black values correspond to source image pixels in memory.



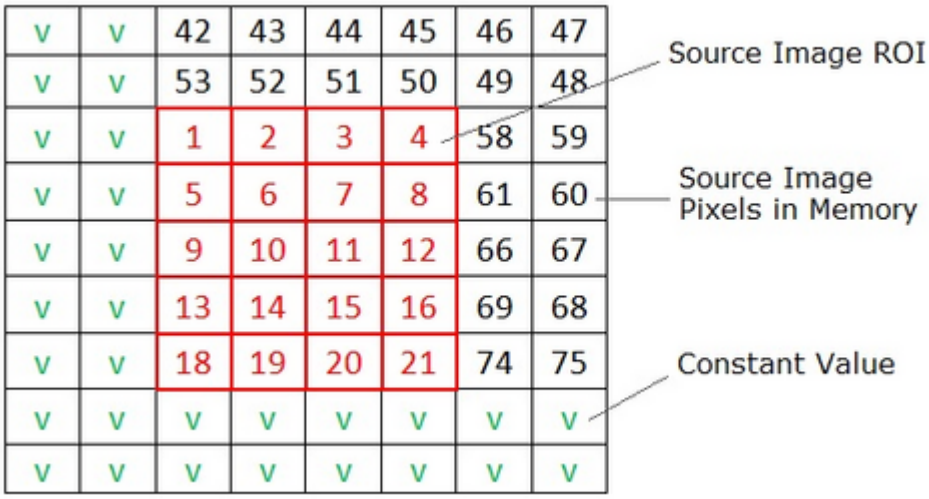
Several Intel IPP filters operate in two or more stages. For example, the `ippiMorphOpenBorder` function performs filtering by applying the `Erode` and `Dilate` filters sequentially. You should note the following when setting borders for multistage filters:

- If you set the `ippBorderInMem` value or its flags combinations, the function tries to access pixels outside of image borders to get border pixels for each filtering stage. For example, the `ippiMorphOpenBorder` function uses two stages and with 5x5 mask will access `floor(5/2)*2=4` pixels in each direction across the current ROI.
- If you set `ippBorderFirstStageInMem`, the function tries to access `floor(5/2)=2` pixels outside of the image borders to get pixels for the first stage of filtering. The second filter will use one of the following border types to reconstruct image borders: `ippBorderRepl`, `ippBorderConst`, `ippBorderMirror`, or `ippBorderMirrorR`. To specify the border type for the second and next stages, use the bitwise `OR` operation between one of the listed above border types and `ippBorderFirstStageInMem`.

Mixed Borders

You can use mixed borders by using a bitwise OR operation between one of the `ippBorderRepl`, `ippBorderConst`, `ippBorderMirror`, or `ippBorderMirrorR` types and any of the following border types: `ippBorderInMemTop`, `ippBorderInMemBottom`, `ippBorderInMemLeft`, `ippBorderInMemRight`, or `ippBorderFirstStageInMem`. In this case, values for border pixels are obtained from the source image pixels in memory in the direction specified by the flag.

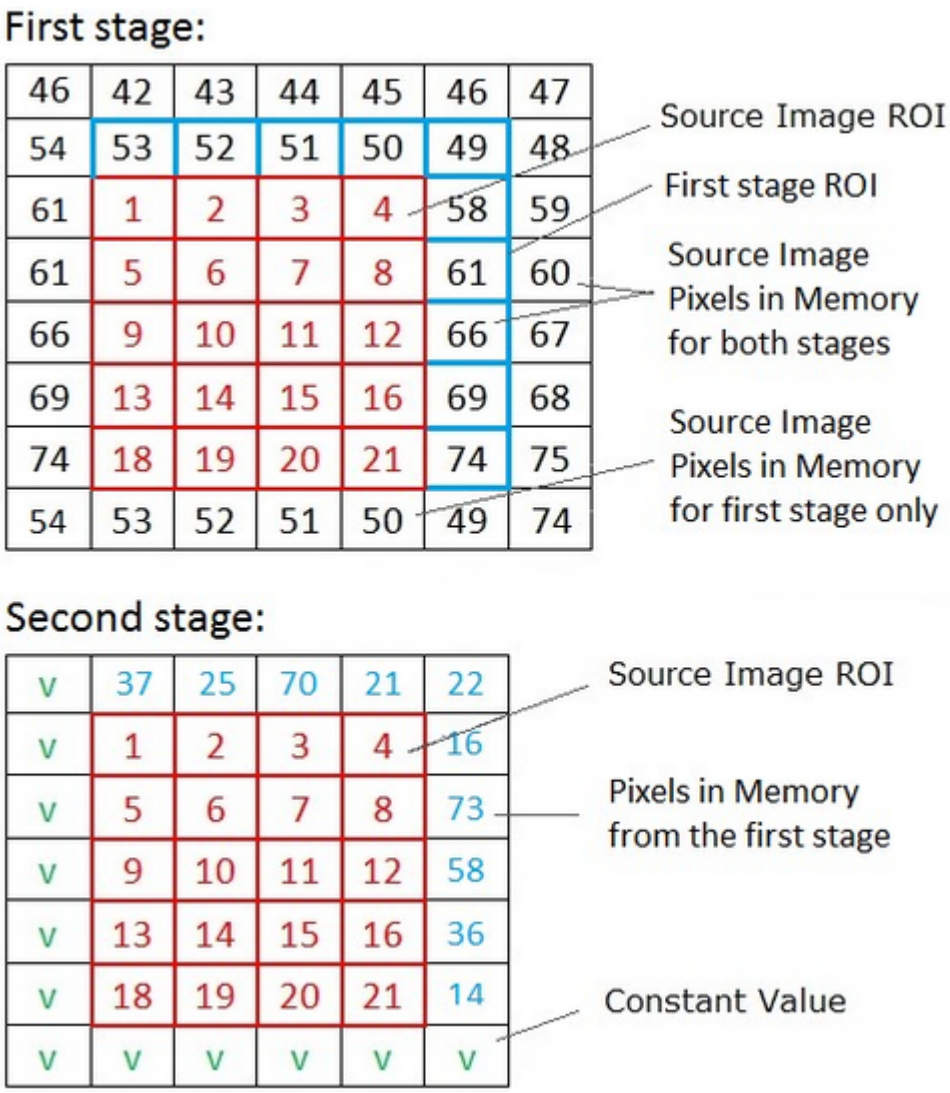
The figure below demonstrates the use of the `ippBorderConst` with the `ippBorderInMemTop` and `ippBorderInMemRight` borders. Squares marked in red correspond to pixels copied from the source image, that is the source image ROI. As you can see from the figure, top and right border pixels are obtained from the source image pixels in memory, while the rest of the border pixels are set to the constant value `V`.



NOTE The combination of `ippBorderInMem` and its flags always has priority over any other border flags or types. For example, if you specify `ippBorderFirstStageInMem|ippBorderRepl|ippBorderInMemLeft`, the left border will use `InMem` mode for each stage and other borders will use `InMem` for the first stage and replication for remaining stages.

The figure below demonstrates the use of the `ippBorderConst` with the `ippBorderInMemTop`, `ippBorderInMemRight`, and `ippBorderFirstStageInMem` flags for two-stage filtering with one pixel border for both stages.

- **First stage:** squares marked in red correspond to pixels copied from the source image, which is the source image ROI, and squares marked in blue correspond to ROI assigned to the first stage filter. As you can see from the figure, the first stage enlarges ROI for top and right sides to consume more memory and provide valid pixels for the second stage memory border.
- **Second stage:** red squares and blue pixels correspond to resulting pixels from the first stage filter. Blue pixels lie outside of the ROI providing border values for the second stage in top and right directions. Left and bottom border pixels use the constant value `v` in accordance with the border flags combination.



See Also

Regions of Interest in Intel IPP

Borders in Neighborhood Operations

Filters with Borders

This section describes Intel® IPP filtering functions that automatically create a required border and define appropriate pixel values.

See Also

Regions of Interest in Intel IPP

- FilterBilateral
- FilterBilateralGetBufferSize
- FilterBilateralInit
- FilterBilateralBorderGetBufferSize
- FilterBilateralBorderInit
- FilterBilateralBorder
- FilterBoxBorderGetBufferSize
- FilterBoxBorder
- FilterBox
- FilterGaussianBorder
- SumWindow
- SumWindowGetBufferSize
- SumWindowRow
- SumWindowColumn
- FilterMaxBorderGetBufferSize, FilterMinBorderGetBufferSize
- FilterMaxBorder, FilterMinBorder
- DecimateFilterRow, DecimateFilterColumn

FilterBilateral

Performs bilateral filtering of an image.

Syntax

Case 1: Operation on pixel-order data

```
IppStatus ippiFilterBilateral_<mod>(const Ipp<srcdatatype>* pSrc, int srcStep,
Ipp<dstdatatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiBorderType borderType,
const Ipp<datatype> pBorderValue[1], const IppiFilterBilateralSpec* pSpec, Ipp8u*
pBuffer);
```

Supported values for `mod`:

`8u_C1R`	`32f_C1R`	`64f_C1R`

```
IppStatus ippiFilterBilateral_<mod>(const Ipp<srcdatatype>* pSrc, int srcStep,
Ipp<dstdatatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiBorderType borderType,
const Ipp<datatype> pBorderValue[3], const IppiFilterBilateralSpec* pSpec, Ipp8u*
pBuffer);
```

Supported values for `mod`:

`8u_C3R`	`32f_C3R`	`64f_C3R`

Case 2: Operation on planar data

```
IppStatus ippiFilterBilateral_<mod>(const Ipp<srcdatatype>* pSrc[3], int srcStep[3],
Ipp<dstdatatype>* pDst[3], int dstStep[3], IppiSize dstRoiSize, IppiBorderType
borderType, const Ipp<datatype> pBorderValue[3], const IppiFilterBilateralSpec* pSpec,
Ipp8u* pBuffer);
```

Supported values for `mod`:

8u_P3R	32f_P3R	64f_P3R

Case 3: Operation on pixel-order data with platform-aware functions

```
IppStatus ippiFilterBilateral_<mod>(const Ipp<srcdatatype>* pSrc, IppSizeL srcStep,
Ipp<dstdatatype>* pDst, IppSizeL dstStep, IppiSizeL dstRoiSize, IppiBorderType
borderType, const Ipp<datatype> pBorderValue[1], const IppiFilterBilateralSpec* pSpec,
Ipp8u* pBuffer);
```

Supported values for `mod`:

8u_C1R_L	32f_C1R_L	64f_C1R_L

```
IppStatus ippiFilterBilateral_<mod>(const Ipp<srcdatatype>* pSrc, IppSizeL srcStep,
Ipp<dstdatatype>* pDst, IppSizeL dstStep, IppiSizeL dstRoiSize, IppiBorderType
borderType, const Ipp<datatype> pBorderValue[3], const IppiFilterBilateralSpec* pSpec,
Ipp8u* pBuffer);
```

Supported values for `mod`:

8u_C3R_L	32f_C3R_L	64f_C3R_L

Case 4: Operation on planar data with platform-aware functions

```
IppStatus ippiFilterBilateral_<mod>(const Ipp<srcdatatype>* pSrc[3], IppSizeL
srcStep[3], Ipp<dstdatatype>* pDst[3], IppSizeL dstStep[3], IppiSizeL dstRoiSize,
IppiBorderType borderType, const Ipp<datatype> pBorderValue[3], const
IppiFilterBilateralSpec* pSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

8u_P3R_L	32f_P3R_L	64f_P3R_L

Case 5: Operation on pixel-order data with Threading Layer (TL) functions based on the Platform Aware API

```
IppStatus ippiFilterBilateral_<mod>(const Ipp<srcdatatype>* pSrc, IppSizeL srcStep,
Ipp<dstdatatype>* pDst, IppSizeL dstStep, IppiSizeL dstRoiSize, IppiBorderType
borderType, const Ipp<datatype> pBorderValue[1], const IppiFilterBilateralSpec_LT*
pSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

8u_C1R_LT	32f_C1R_LT	64f_C1R_LT

```
IppStatus ippiFilterBilateral_<mod>(const Ipp<srcdatatype>* pSrc, IppSizeL srcStep,
Ipp<dstdatatype>* pDst, IppSizeL dstStep, IppiSizeL dstRoiSize, IppiBorderType
borderType, const Ipp<datatype> pBorderValue[3], const IppiFilterBilateralSpec_LT*
pSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

8u_C3R_LT	32f_C3R_LT	64f_C3R_LT

Case 6: Operation on planar data with Threading Layer (TL) functions based on the Platform Aware API

```
IppStatus ippiFilterBilateral_<mod>(const Ipp<srcdatatype>* pSrc[3], IppSizeL
srcStep[3], Ipp<dstdatatype>* pDst[3], IppSizeL dstStep[3], IppiSizeL dstRoiSize,
IppiBorderType borderType, const Ipp<datatype> pBorderValue[3], const
IppiFilterBilateralSpec_LT* pSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

8u_P3R_LT	32f_P3R_LT	64f_P3R_LT

Case 7: Operation on pixel-order data with Threading Layer (TL) functions based on the Classic API

```
IppStatus ippiFilterBilateral_<mod>(const Ipp<srcdatatype>* pSrc, int srcStep,
Ipp<dstdatatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiBorderType borderType,
const Ipp<datatype> pBorderValue[1], const IppiFilterBilateralSpec_T* pSpec, Ipp8u*
pBuffer);
```

Supported values for `mod`:

8u_C1R_T	32f_C1R_T	64f_C1R_T

```
IppStatus ippiFilterBilateral_<mod>(const Ipp<srcdatatype>* pSrc, int srcStep,
Ipp<dstdatatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiBorderType borderType,
const Ipp<datatype> pBorderValue[3], const IppiFilterBilateralSpec_T* pSpec, Ipp8u*
pBuffer);
```

Supported values for `mod`:

8u_C3R_T	32f_C3R_T	64f_C3R_T

Case 8: Operation on planar data with Threading Layer (TL) functions based on the Classic API

```
IppStatus ippiFilterBilateral_<mod>(const Ipp<srcdatatype>* pSrc[3], int srcStep[3],
Ipp<dstdatatype>* pDst[3], int dstStep[3], IppiSize dstRoiSize, IppiBorderType
borderType, const Ipp<datatype> pBorderValue[3], const IppiFilterBilateralSpec_T*
pSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

8u_P3R_T	32f_P3R_T	64f_P3R_T

Include Files

`ippi.h`

`ippi_l.h`

`ippi_tl.h`

Domain Dependencies

Flavors declared in `ippi.h`:

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance, in bytes, between the starting points of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance, in bytes, between the starting points of consecutive lines in the destination image.
dstRoiSize	Size of the source and destination ROI in pixels.
borderType	Type of border. Possible values are: • ippBorderConst • Values of all border pixels are set to constant. • ippBorderRepl • Border is replicated from the edge pixels. • ippBorderInMem • Border is obtained from the source image pixels in memory.
pBorderValue	Constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.
pSpec	Pointer to the bilateral context structure.
pBuffer	Pointer to the work buffer.

Description

This function applies the bilateral filter with a square kernel to the source image. The linear dimension of the kernel is defined in the initialization function FilterBilateralInit. The bilateral context structure contains the parameters of filtering.

Before using the ippiFilterBilateral function, compute the size of the bilateral context structure and the external buffer using the FilterBilateralGetBufferSize function and initialize the structure using the FilterBilateralInit function.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pSrc, pDst, pSpec, or pBuffer is NULL.
ippStsSizeErr	Indicates an error when dstRoiSize has a field with a zero or negative value.
ippStsContextMatchErr	Indicates an error when the pSpec structure does not match the function.
ippStsNotEvenStepErr	Indicates an error when one of the step values is not divisible by 4 for floating-point images.

ippStsBorderErr	Indicates an error when borderType has an illegal value.

See Also

Regions of Interest in Intel IPP

User-defined Border Types

FilterBilateralGetBufferSize Computes the size of the bilateral context structure and the size of the work buffer for bilateral filtering with user-defined borders.

FilterBilateralInit Initializes the bilateral context structure.

FilterBilateralGetBufferSize

Computes the size of the bilateral context structure and the size of the work buffer for bilateral filtering with user-defined borders.

Syntax

```
IppStatus ippiFilterBilateralGetBufferSize(IppiFilterBilateralType filter, IppiSize
dstRoiSize, int kernelWidthHeight, IppDataType dataType, int numChannels,
IppiDistanceMethodType distMethodType, int* pSpecSize, int* pBufferSize);
```

Platform-aware function

```
IppStatus ippiFilterBilateralGetBufferSize_L(IppiFilterBilateralType filter, IppiSizeL
dstRoiSize, IppSizeL kernelWidthHeight, IppDataType dataType, int numChannels,
IppiDistanceMethodType distMethodType, IppSizeL* pSpecSize, IppSizeL* pBufferSize);
```

Threading Layer (TL) function based on the Platform Aware API

```
IppStatus ippiFilterBilateralGetBufferSize_LT(IppiFilterBilateralType filter, IppiSizeL
dstRoiSize, IppSizeL kernelWidthHeight, IppDataType dataType, int numChannels,
IppiDistanceMethodType distMethodType, IppSizeL* pSpecSize, IppSizeL* pBufferSize);
```

Threading Layer (TL) function based on the Classic API

```
IppStatus ippiFilterBilateralGetBufferSize_T(IppiFilterBilateralType filter, IppiSize
dstRoiSize, int kernelWidthHeight, IppDataType dataType, int numChannels,
IppiDistanceMethodType distMethodType, int* pSpecSize, int* pBufferSize);
```

Include Files

ippi.h

ippi_l.h

ippi_tl.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

filter	Type of the bilateral filter. Possible value is ippiFilterBilateralGauss – Gaussian bilateral filter.
dstRoiSize	Size of the destination ROI in pixels.
kernelWidthHeight	Linear dimension of the square kernel. The value 1 corresponds to the distance between two adjacent pixels.
dataType	Data type of the source and destination images. Possible values are ipp8u, ipp32f, and ipp64f.
numChannels	Number of channels in the images.
distMethodType	Method of defining the differences in intensity between pixels. Depending on the number of channels in the image, possible value are: • numChannels value • Possible distMethodType values • 1 • ippDistNormL1 • 3 • ippDistNormL1, ippDistNormL2
pSpecSize	Pointer to the computed size of the specification structure.
pBufferSize	Pointer to the computed size of the external buffer.

Description

This function computes the size of the bilateral context structure and external work buffer for the FilterBilateral function. The results are stored in pSpecSize and pBufferSize.

Use the computed pBufferSize and pSpecSize values to allocate the memory using the ippMalloc or ippsMalloc functions. The allocated memory can be freed only by the ippFree or ippsFree functions, respectively. For more information about the memory allocation functions, refer to the "Support Functions" section of the *Intel IPP Developer Reference, vol. 1*.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error when pSpecSize or pBufferSize is NULL.
ippStsSizeErr	Indicates an error when dstRoiSize has a field with a zero or negative value.
ippStsMaskSizeErr	Indicates an error when kernelWidthHeight is less than, or equal to zero.

ippStsNotSupportedModeErr	Indicates an error when the filter or distMethodType value is not supported.
ippStsDataTypeErr	Indicates an error when dataType has an illegal value.
ippStsNumChannelsErr	Indicates an error when numChannels has an illegal value.

See Also

FIlterBilateral Performs bilateral filtering of an image.

FilterBilateralInit Initializes the bilateral context structure.

FilterBilateralInit

Initializes the bilateral context structure.

Syntax

```
IppStatus ippiFilterBilateralInit(IppiFilterBilateralType filter, IppiSize dstRoiSize,
int kernelWidthHeight, IppDataType dataType, int numChannels, IppiDistanceMethodType
distMethod, Ipp64f valSquareSigma, Ipp64f posSquareSigma, IppiFilterBilateralSpec*
pSpec);
```

Platform-aware function

```
IppStatus ippiFilterBilateralInit_L(IppiFilterBilateralType filter, IppiSizeL
dstRoiSize, IppSizeL kernelWidthHeight, IppDataType dataType, int numChannels,
IppiDistanceMethodType distMethod, Ipp64f valSquareSigma, Ipp64f posSquareSigma,
IppiFilterBilateralSpec* pSpec);
```

Threading Layer (TL) function based on the Platform Aware API

```
IppStatus ippiFilterBilateralInit_LT(IppiFilterBilateralType filter, IppiSizeL
dstRoiSize, IppSizeL kernelWidthHeight, IppDataType dataType, int numChannels,
IppiDistanceMethodType distMethod, Ipp64f valSquareSigma, Ipp64f posSquareSigma,
IppiFilterBilateralSpec_LT* pSpec);
```

Threading Layer (TL) function based on the Classic API

```
IppStatus ippiFilterBilateralInit_T(IppiFilterBilateralType filter, IppiSize
dstRoiSize, int kernelWidthHeight, IppDataType dataType, int numChannels,
IppiDistanceMethodType distMethod, Ipp64f valSquareSigma, Ipp64f posSquareSigma,
IppiFilterBilateralSpec_T* pSpec);
```

Include Files

ippi.h

ippi_l.h

ippi_tl.h

Domain Dependencies

Flavors declared in ippi.h:

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

filter	Type of the bilateral filter. Possible value is ippiFilterBilateralGauss – Gaussian bilateral filter.
dstRoiSize	Size of the destination ROI in pixels.
kernelWidthHeight	Linear dimension of the square kernel. The value 1 corresponds to the distance between two adjacent pixels.
dataType	Data type of the source and destination images. Possible values are ipp8u, ipp32f, and ipp64f.
numChannels	Number of channels in the images.
distMethod	Method of defining the differences in intensity between pixels. Depending on the number of channels in the image, possible value are: • numChannels value • Possible distMethod values • 1 • ippDistNormL1 • 3 • ippDistNormL1, ippDistNormL2
valSquareSigma	Square of the range parameter, which controls smoothing based on the differences in intensity between pixels.
posSquareSigma	Square of the spatial parameter, which controls smoothing based on the geometric distance between pixels.
pSpec	Pointer to the bilateral context structure.

Description

This function initializes the bilateral context structure pSpecSize for bilateral filtering. Before using this function, compute the size of the context structure using the FilterBilateralGetBufferSize function.

The kernelWidthHeight parameter specifies the linear dimension of the square filter kernel. The value 1 corresponds to the distance between the centers of two adjacent pixels.

Coefficients of the bilateral filter kernel depend on their positions in the kernel and on the intensity value of the source image pixels lying in the kernel.

The value of the output pixel d is computed by the following formula:

$$d = \frac{\sum_{i,j} w1_{ij} * w2_{ij} * v_{ij}}{\sum_{i,j} w1_{ij} * w2_{ij}}$$

For all indices `i` and `j` that fit within the square kernel

where

- v_{ij} is the value (or channel values) of a pixel in the kernel with coordinates `i` and `j`
- `w` 1 $_{ij}$ = `Fun(valSquareSigma,` Intensity Distance(v_{ij}, v_{00}))

  The `distMethodType` parameter specifies the method of defining the differences in intensity between pixels. FilterBilateral functions support two methods: `ippDistNormL1`, which defines the difference in intensity as the L1-norm, and `ippDistNormL2`, which defines the difference in intensity as the L2-norm.
- `w` 2 $_{ij}$ = `Fun(posSquareSigma,` Geometric Distance(v_{ij}, v_{00})= `sqrt(i * i + j * j))`

`Fun(S, I)= exp(-I * I / 2*S)`

where

- `S` is `valSquareSigma` or `posSquareSigma`
- `I` is the difference between pixel values or position

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error when `pSpec` is `NULL`.
`ippStsSizeErr`	Indicates an error when `dstRoiSize` has a field with a zero or negative value.
`ippStsMaskSizeErr`	Indicates an error when `kernelWidthHeight` is less than, or equal to zero.
`ippStsNotSupportedModeErr`	Indicates an error when the `filter` or `distMethod` value is not supported.
`ippStsBadArgErr`	Indicates an error when `valSquareSigma` or `posSquareSigma` is less than, or equal to zero.
`ippStsNumChannelsErr`	Indicates an error when `numChannels` has an illegal value.

See Also

Structures and Enumerators for Platform-Aware Functions

FilterBilateralGetBufferSize Computes the size of the bilateral context structure and the size of the work buffer for bilateral filtering with user-defined borders.

FIlterBilateral Performs bilateral filtering of an image.

FilterBilateralBorderGetBufferSize

Computes the size of the bilateral context structure and the size of the work buffer for bilateral filtering with user-defined borders.

Syntax

Processing images of 32-bit sizes

```
IppStatus ippiFilterBilateralBorderGetBufferSize(IppiFilterBilateralType filter,
IppiSize dstRoiSize, int radius, IppDataType dataType, int numChannels,
IppiDistanceMethodType distMethodType, int* pSpecSize, int* pBufferSize);
```

Platform-aware function

```
IppStatus ippiFilterBilateralBorderGetBufferSize_L(IppiFilterBilateralType filter,
IppiSizeL dstRoiSize, int radius, IppDataType dataType, int numChannels,
IppiDistanceMethodType distMethodType, IppSizeL* pSpecSize, IppSizeL* pBufferSize);
```

Threading layer function

```
IppStatus ippiFilterBilateralBorderGetBufferSize_LT(IppiFilterBilateralType filter,
IppiSizeL dstRoiSize, int radius, IppDataType dataType, int numChannels,
IppiDistanceMethodType distMethodType, IppSizeL* pSpecSize, IppSizeL* pBufferSize);
```

Include Files

`ippi.h`

Flavors with the `_LT` suffix: `ippi_tl.h`

Flavors with the `_L` suffix: `ippi_l.h`

Domain Dependencies

Flavors declared in `ippi.h`:

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Flavors declared in `ippi_tl.h`:

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`, `ippcore_tl.lib`, `ippi_tl.lib`

Parameters

`filter`	Type of the bilateral filter. Possible value is `ippiFilterBilateralGauss` - Gaussian bilateral filter.
`dstRoiSize`	Size of the destination ROI in pixels.
`radius`	Radius of the round kernel. The radius value equal to 1 corresponds to distance between the closest pixels.
`dataType`	Data type of the source and destination images. Possible values are `ipp8u` and `ipp32f`.
`numChannels`	Number of channels in the images.
`distMethodType`	Method of defining intensive distance between pixels. Possible value is `ippDistNormL1`.

pSpecSize	Pointer to the computed size of the specification structure.
pBufferSize	Pointer to the computed size of the external buffer.

Description

This function computes the size of the bilateral context structure and external work buffer for the FilterBilateralBorder function. The results are stored in pSpecSize and pBufferSize.

Use the computed pBufferSize and pSpecSize values to allocate the memory using the ippMalloc or ippsMalloc functions. The allocated memory can be freed only by the ippFree or ippsFree functions, respectively. For more information about the memory allocation functions, refer to the "Support Functions" section of the *Intel IPP Developer Reference, vol. 1*.

For an example on how to use this function, refer to the example provided with the FilterBilateralBorder function description.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error when pSpecSize or pBufferSize is NULL.
ippStsSizeErr	Indicates an error when dstRoiSize has a field with a zero or negative value.
ippStsMaskSizeErr	Indicates an error when radius is less than, or equal to zero.
ippStsNotSupportedModeErr	Indicates an error when the filter or distMethodType value is not supported.
ippStsDataTypeErr	Indicates an error when dataType has an illegal value.
ippStsNumChannelsErr	Indicates an error when numChannels has an illegal value.

See Also

FilterBilateralBorder Performs bilateral filtering of an image.

FilterBilateralBorderInit

Initializes the bilateral context structure.

Syntax

```
IppStatus ippiFilterBilateralBorderInit(IppiFilterBilateralType filter, IppiSize dstRoiSize, int radius, IppDataType dataType, int numChannels, IppiDistanceMethodType distMethod, Ipp32f valSquareSigma, Ipp32f posSquareSigma, IppiFilterBilateralSpec* pSpec);
```

Platform-aware function

```
IppStatus ippiFilterBilateralBorderInit_L(IppiFilterBilateralType filter, IppiSizeL
dstRoiSize, int radius, IppDataType dataType, int numChannels, IppiDistanceMethodType
distMethod, Ipp32f valSquareSigma, Ipp32f posSquareSigma, IppiFilterBilateralSpec*
pSpec);
```

Threading layer function

```
IppStatus ippiFilterBilateralBorderInit_LT(IppiFilterBilateralType filter, IppiSizeL
dstRoiSize, int radius, IppDataType dataType, int numChannels, IppiDistanceMethodType
distMethod, Ipp32f valSquareSigma, Ipp32f posSquareSigma, IppiFilterBilateralSpec_LT*
pSpec);
```

Include Files

`ippi.h`

Flavors with the `_LT` suffix: `ippi_tl.h`

Flavors with the `_L` suffix: `ippi_l.h`

Domain Dependencies

Flavors declared in `ippi.h`:

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Flavors declared in `ippi_tl.h`:

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`, `ippcore_tl.lib`, `ippi_tl.lib`

Parameters

`filter`	Type of the bilateral filter. Possible value is `ippiFilterBilateralGauss` – Gaussian bilateral filter.
`dstRoiSize`	Size of the destination ROI in pixels.
`radius`	Radius of the round kernel. The radius value equal to 1 corresponds to distance between the closest pixels.
`dataType`	Data type of the source and destination images. Possible values are `ipp8u` and `ipp32f`.
`numChannels`	Number of channels in the images.
`distMethod`	Method of defining intensive distance between pixels. Possible value is `ippDistNormL1`.
`valSquareSigma`	Square of the sigma for intensive distance between pixels.
`posSquareSigma`	Square of the sigma for geometric distance between pixels.
`pSpec`	Pointer to the bilateral context structure.

Description

This function initializes the bilateral context structure `pSpecSize` for bilateral filtering. Before using this function, compute the size of the context structure using the FilterBilateralBorderGetBufferSize function.

The `radius` parameter specifies the radius of the round filter kernel. The radius value equal to 1 corresponds to the distance between centers of the closest pixels.

Coefficients of the bilateral filter kernel depend on their positions in the kernel and on the intensity value of the source image pixels lying in the kernel.

The value of the output pixel `d` is computed by the following formula:

$$d = \frac{\sum_{i,j} w1_{ij} * w2_{ij} * v_{ij}}{\sum_{i,j} w1_{ij} * w2_{ij}}$$

For all `i` and `j` that `i * i + j * j <= radius * radius` (for central pixel of the kernel `i=0,j = 0`)

where

- v_{ij} is the value (or channel values) of a pixel in the kernel with coordinates `i` and `j`
- `w` 1 $_{ij}$ = `Fun(valSquareSigma,` Intensity Distance(v_{ij}, v_{00}))

  The `distMethodType` parameter specifies the method of defining the intensive distance between pixels. Currently the only supported method is `ippDistNormL1`, which defines the intensive distance as norma L1.
- `w` 2 $_{ij}$ = `Fun(posSquareSigma,` Geometric Distance(v_{ij}, v_{00})= `sqrt(i * i + j * j))`

`Fun(S, I)= exp(-I * I / 2*S)`

where

- `S` is `valSquareSigma` or `posSquareSigma`
- `I` is the difference between pixel values or position

For an example on how to use this function, refer to the example provided with the FilterBilateralBorder function description.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error when `pSpec` is `NULL`.
`ippStsSizeErr`	Indicates an error when `dstRoiSize` has a field with a zero or negative value.
`ippStsMaskSizeErr`	Indicates an error when `radius` is less than, or equal to zero.
`ippStsNotSupportedModeErr`	Indicates an error when the `filter` or `distMethod` value is not supported.
`ippStsBadArgErr`	Indicates an error when `valSquareSigma` or `posSquareSigma` is less than, or equal to zero.

ippStsNumChannelsErr	Indicates an error when numChannels has an illegal value.

See Also

FilterBilateralBorderGetBufferSize Computes the size of the bilateral context structure and the size of the work buffer for bilateral filtering with user-defined borders.

FilterBilateralBorder Performs bilateral filtering of an image.

FilterBilateralBorder

Performs bilateral filtering of an image.

Syntax

Processing images of 32-bit sizes

```
IppStatus ippiFilterBilateralBorder_<mod>(const Ipp<srcdatatype>* pSrc, int srcStep,
Ipp<dstdatatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiBorderType borderType,
Ipp<datatype>* pBorderValue, IppiFilterBilateralSpec* pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u_C1R	8u_C3R	32f_C1R	32f_C3R

Platform-aware functions

```
IppStatus ippiFilterBilateralBorder_<mod>(const Ipp<srcdatatype>* pSrc, IppSizeL
srcStep, Ipp<dstdatatype>* pDst, IppSizeL dstStep, IppiSizeL dstRoiSize, IppiBorderType
borderType, Ipp<datatype>* pBorderValue, const IppiFilterBilateralSpec* pSpec, Ipp8u*
pBuffer);
```

Supported values for mod:

8u_C1R_L	8u_C3R_L

Threading layer functions

```
IppStatus ippiFilterBilateralBorder_<mod>(const Ipp<srcdatatype>* pSrc, IppSizeL
srcStep, Ipp<dstdatatype>* pDst, IppSizeL dstStep, IppiSize dstRoiSize, IppiBorderType
borderType, Ipp<datatype>* pBorderValue, IppiFilterBilateralSpec_LT* pSpec, Ipp8u*
pBuffer);
```

Supported values for mod:

8u_C1R_L	8u_C3R_L

Include Files

ippi.h

Flavors with the _LT suffix: ippi_tl.h

Flavors with the _L suffix: ippi_l.h

Domain Dependencies

Flavors declared in ippi.h:

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Flavors declared in ippi_tl.h:

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib, ippcore_tl.lib, ippi_tl.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance, in bytes, between the starting points of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance, in bytes, between the starting points of consecutive lines in the destination image.
dstRoiSize	Size of the source and destination ROI in pixels.
borderType	Type of border. Possible values are: • ippBorderConst • Values of all border pixels are set to constant. • ippBorderRepl • Border is replicated from the edge pixels. • ippBorderInMem • Border is obtained from the source image pixels in memory.
pBorderValue	Constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.
pSpec	Pointer to the bilateral context structure.
pBuffer	Pointer to the work buffer.

Description

This function applies the bilateral filter with the round kernel to the source image. The radius of the kernel is defined in the corresponding initialization function FilterBilateralBorderInit. The bilateral context structure contains the parameters of filtering.

Before using the ippiFilterBilateralBorder function, compute the size of the bilateral context structure and the external buffer using the FilterBilateralBorderGetBufferSize function and initialize the structure using the FilterBilateralBorderInit function.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pSrc, pDst, pSpec, or pBuffer is NULL.
ippStsSizeErr	Indicates an error when dstRoiSize has a field with a zero or negative value.

ippStsContextMatchErr	Indicates an error when the pSpec structure does not match the function.
ippStsNotEvenStepErr	Indicates an error when one of the step values is not divisible by 4 for floating-point images.
ippStsBorderErr	Indicates an error when borderType has an illegal value.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

// A simple example of performing Bilateral filtering of an image implemented with Intel(R)
Integrated Performance Primitives (Intel(R) IPP)
// functions:
//     ippiFilterBilateralBorderGetBufferSize
//     ippiFilterBilateralBorderInit
//     ippiFilterBilateralBorder_8u_C1R

#include <stdio.h>
#include "ipp.h"

#define WIDTH  128 /* image width  */
#define HEIGHT 64  /* image height */

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define check_sts(st)        \
    if ((st) != ippStsNoErr) \
        goto exitLine; /* Go to Exit if Intel(R) IPP function returned status different from
ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main(void)
{
    IppStatus status = ippStsNoErr;
    Ipp8u *pSrc = NULL, *pDst = NULL;   /* Pointers to source/destination images */
    int srcStep = 0, dstStep = 0;       /* Steps, in bytes, through the source/destination
images */
    IppiSize roiSize = {WIDTH, HEIGHT}; /* Size of source/destination ROI in pixels */
```

```
    Ipp8u *pBuffer = NULL;              /* Pointer to the work buffer */
    IppiFilterBilateralSpec *pSpec = NULL;
    int iTmpBufSize = 0, iSpecSize = 0; /* Common work buffer size */
    IppiBorderType borderType = (IppiBorderType)(ippBorderRepl | ippBorderInMemTop |
ippBorderInMemRight);
    Ipp8u borderValue = 0;
    int radius = 1; /* Radius of circular neighborhood what defines pixels for calculation. */
    int numChannels = 1;
    Ipp32f valSqrSigma = 400.0f, posSqrSigma = 100.0f; /* pixel intensity/position */

    pSrc = ippiMalloc_8u_C1(roiSize.width, roiSize.height, &srcStep);
    pDst = ippiMalloc_8u_C1(roiSize.width, roiSize.height, &dstStep);

    check_sts(status = ippiFilterBilateralBorderGetBufferSize(ippiFilterBilateralGauss, roiSize,
radius, ipp8u, numChannels, ippDistNormL1,
                                                              &iSpecSize, &iTmpBufSize));

    pSpec = (IppiFilterBilateralSpec *)ippsMalloc_8u(iSpecSize);

    pBuffer = ippsMalloc_8u(iTmpBufSize);

    check_sts(status =
                  ippiFilterBilateralBorderInit(ippiFilterBilateralGauss, roiSize, radius,
ipp8u, 1, ippDistNormL1, valSqrSigma, posSqrSigma, pSpec));

    check_sts(status = ippiFilterBilateralBorder_8u_C1R(pSrc + srcStep, srcStep, pDst, dstStep,
roiSize, borderType, &borderValue, pSpec, pBuffer));

    EXIT_MAIN
    ippsFree(pBuffer);
    ippsFree(pSpec);
    ippiFree(pSrc);
    ippiFree(pDst);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

See Also

Regions of Interest in Intel IPP

User-defined Border Types

FilterBilateralBorderGetBufferSize Computes the size of the bilateral context structure and the size of the work buffer for bilateral filtering with user-defined borders.

FilterBilateralBorderInit Initializes the bilateral context structure.

FilterBoxBorderGetBufferSize

Computes the size of the external buffer for the `FilterBoxBorder` function.

Syntax

```
IppStatus ippiFilterBoxBorderGetBufferSize (IppiSize roiSize, IppiSize maskSize,
IppDataType dataType, int numChannels, int* pBufferSize);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`roiSize`	Maximum size of the destination image ROI.
`maskSize`	Size of the filter mask, in pixels.
`dataType`	Data type of the image. Possible values are `ipp8u`, `ipp16u`, `ipp16s`, or `ipp32f`.
`numChannels`	Number of channels in the image. Possible values are 1, 3, or 4.
`pBufferSize`	Pointer to the size of the external work buffer.

Description

The `ippiFilterBoxBorderGetBufferSize` function computes the size, in bytes, of the external work buffer needed for the `ippiFilterBoxBorder` function. The result is stored in the `pBufferSize` parameter.

For an example on how to use this function, see the code example provided with the `ippiFilterBoxBorder` function description.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsSizeErr`	Indicates an error when `roiSize` is negative, or equal to zero.
`ippStsMaskSizeErr`	Indicates an error when `mask` has an illegal value.
`ippStsDataTypeErr`	Indicates an error when `dataType` has an illegal value.
`ippStsNumChannelsError`	Indicates an error when `numChannels` has an illegal value.

See Also

FilterBoxBorder Blurs an image using a simple box filter.

FilterBoxBorder

Blurs an image using a simple box filter.

Syntax

```
IppStatus ippiFilterBoxBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiSize maskSize, IppiBorderType
border, const Ipp<datatype>* borderValue, Ipp8u* pBuffer);
```

Supported values for `mod`:

8u_C1R	16u_C1R	16s_C1R	32f_C1R

```
IppStatus ippiFilterBoxBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiSize maskSize, IppiBorderType
border, const Ipp<datatype> borderValue[3], Ipp8u* pBuffer);
```

Supported values for `mod`:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
	16u_C4R	16s_C4R	
	16u_AC4R	16s_AC4R	

```
IppStatus ippiFilterBoxBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiSize maskSize, IppiBorderType
border, const Ipp<datatype> borderValue[4], Ipp8u* pBuffer);
```

Supported values for `mod`:

8u_C4R	32f_C4R
8u_AC4R	32f_AC4R

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image.
`srcStep`	Distance, in bytes, between the starting points of consecutive lines in the source image.
`pDst`	Pointer to the destination image.
`dstStep`	Distance, in bytes, between the starting points of consecutive lines in the destination image.
`roiSize`	Size of the destination ROI in pixels.
`maskSize`	Size of the mask in pixels.
`border`	Type of border. Possible values are: • `ippBorderConst` • Values of all border pixels are set to constant. • `ippBorderRepl` • Border is replicated from the edge pixels. • `ippBorderInMem`

	• Border is obtained from the source image pixels in memory. • ippBorderMirror • Border pixels are mirrored from the source image boundary pixels.
borderValue	Constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.
pBuffer	Pointer to the work buffer.

Description

Before using this function, you need to compute the size of the work buffer pBuffer using the ippiFilterBoxBorderGetBufferSize function.

This function operates with ROI.

This function sets each pixel in the destination image as the average of all pixels of the source image in the rectangular neighborhood of size maskSize with the anchor cell at that pixel. This has the effect of smoothing or blurring the input image. To ensure valid operation when image boundary pixels are processed, the application must correctly define additional border pixels. If pSrc is equal to pDst, ippiFilterBoxBorder operates as an in-place function.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pSrc or pDst is NULL.
ippStsSizeErr	Indicates an error if roiSize has a field with zero or negative value.
ippStsMaskSizeErr	Indicates an error if mask has an illegal value.
ippStsBorderErr	Indicates an error when border has an illegal value.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

//   A simple example of performing Bluring an image using a simple box filter
```

```
// implemented with Intel(R) Integrated Performance Primitives (Intel(R) IPP) functions:
//     ippiFilterBoxBorderGetBufferSize
//     ippiFilterBoxBorder_8u_C1R

#include <stdio.h>
#include "ipp.h"

#define WIDTH  128 /* image width  */
#define HEIGHT 64  /* image height */

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define check_sts(st)        \
    if ((st) != ippStsNoErr) \
        goto exitLine; /* Go to Exit if Intel(R) IPP function returned status different from
ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main(void)
{
    IppStatus status = ippStsNoErr;
    Ipp8u *pSrc = NULL, *pDst = NULL;   /* Pointers to source/destination images */
    int srcStep = 0, dstStep = 0;       /* Steps, in bytes, through the source/destination
images */
    IppiSize roiSize = {WIDTH, HEIGHT}; /* Size of source/destination ROI in pixels */
    Ipp8u *pBuffer = NULL;              /* Pointer to the work buffer */
    int iTmpBufSize = 0;                /* Common work buffer size */
    IppiBorderType borderType = ippBorderRepl;
    Ipp8u borderValue = 254;
    IppiSize maskSize = {5, 5};
    int numChannels = 1;

    pSrc = ippiMalloc_8u_C1(roiSize.width, roiSize.height, &srcStep);
    pDst = ippiMalloc_8u_C1(roiSize.width, roiSize.height, &dstStep);

    check_sts(status = ippiFilterBoxBorderGetBufferSize(roiSize, maskSize, ipp8u, numChannels,
&iTmpBufSize));

    pBuffer = ippsMalloc_8u(iTmpBufSize);

    check_sts(status = ippiFilterBoxBorder_8u_C1R(pSrc, srcStep, pDst, dstStep, roiSize,
maskSize, borderType, &borderValue, pBuffer));

    EXIT_MAIN
    ippsFree(pBuffer);
    ippiFree(pSrc);
    ippiFree(pDst);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

FilterBox

Blurs an image using a simple box filter.

Syntax

```
IppStatus ippiFilterBox_64f_C1R(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiPoint anchor);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image.
`dstRoiSize`	Size of the destination ROI in pixels.
`maskSize`	Size of the mask in pixels.
`anchor`	Anchor cell specifying the mask alignment with respect to the position of the input pixel.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function sets each pixel in the destination image as the average of all the input image pixels in the rectangular neighborhood of size `maskSize` with the anchor cell at that pixel. This has the effect of smoothing or blurring on the input image. To ensure valid operation when image boundary pixels are processed, the application must correctly define additional border pixels (see Borders in Neighborhood Operations).

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error if `pSrc`, `pDst`, or `pSrcDst` is `NULL`.
`ippStsSizeErr`	Indicates an error if `dstRoiSize` has a field with zero or negative value.
`ippStsStepErr`	Indicates an error if `srcStep` or `dstStep` has a zero or negative value.

ippStsMaskSizeErr	Indicates an error if maskSize has a field with zero or negative value.
ippStsAnchorErr	Indicates an error if anchor is outside the mask size.
ippStsMemAllocErr	Indicates a memory allocation error.

FilterGaussianBorder

Performs Gaussian filtering of an image with user-defined borders.

Syntax

```
IppStatus ippiFilterGaussianBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, Ipp<datatype> borderValue,
IppFilterGaussianSpec* pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32f_C1R

```
IppStatus ippiFilterGaussianBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, Ipp<datatype> borderValue[3],
IppFilterGaussianSpec* pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u_C3R	16u_C3R	16s_C3R	32f_C3R

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance, in bytes, between the starting points of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance, in bytes, between the starting points of consecutive lines in the destination image.
roiSize	Size of the source and destination image ROI in pixels.

borderValue	Constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.
pSpec	Pointer to the Gaussian specification structure.
pBuffer	Pointer to the work buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function applies the Gaussian filter to the source image ROI pSrc. The kernel of the Gaussian filter is the matrix of size kernelSize x kernelSize with the standard deviation sigma. The values of the Gaussian kernel elements are computed by the FilterGaussianInit function. Elements of the kernel are normalized. The anchor cell is the center of the kernel.

Before using the ippiFilterGaussianBorder function, compute the size of the Gaussian specification structure and the external buffer using the FilterGaussianGetBufferSize function and initialize the structure using the FilterGaussianInit function.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when roiSize has a field with a zero or negative value.
ippStsStepErr	Indicates an error when srcStep or dstStep is less than roiSize.width *<pixelSize>.
ippStsNotEvenStepErr	Indicates an error when one of the step values is not divisible by sizeof(Ipp<dataType>).
ippStsBorderErr	Indicates an error when borderType has an illegal value.
ippStsBadArgErr	Indicates an error when kernelSize is even, or less than 3.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
```

```
 * those that are expressly stated in the License.
 *****************************************************************************/

//   A simple example of performing a filtering an image using a general integer rectangular
kernel
// implemented with Intel(R) Integrated Performance Primitives (Intel(R) IPP) functions:
//     ippiImageJaehne_32f_C1R
//     ippiFilterGaussianInit
//     ippiFilterGaussianGetBufferSize
//     ippiFilterGaussianBorder_32f_C1R

#include <stdio.h>
#include "ipp.h"

#define WIDTH  128 /* image width */
#define HEIGHT 64  /* image height */

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define check_sts(st)        \
    if ((st) != ippStsNoErr) \
        goto exitLine; /* Go to Exit if Intel(R) IPP function returned status different from
ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main(void)
{
    IppStatus status = ippStsNoErr;
    Ipp32f *pSrc = NULL, *pDst = NULL;  /* Pointers to source/destination images */
    int srcStep = 0, dstStep = 0;       /* Steps, in bytes, through the source/destination
images */
    IppiSize roiSize = {WIDTH, HEIGHT}; /* Size of source/destination ROI in pixels */
    Ipp32u kernelSize = 3;
    Ipp32f sigma = 0.35f;
    Ipp8u *pBuffer = NULL;               /* Pointer to the work buffer */
    IppFilterGaussianSpec *pSpec = NULL; /* context structure */
    int iTmpBufSize = 0, iSpecSize = 0;  /* Common work buffer size */
    IppiBorderType borderType = ippBorderRepl;
    Ipp8u borderValue = 0;
    int numChannels = 1;

    pSrc = ippiMalloc_32f_C1(roiSize.width, roiSize.height, &srcStep);
    pDst = ippiMalloc_32f_C1(roiSize.width, roiSize.height, &dstStep);

    check_sts(status = ippiImageJaehne_32f_C1R(pSrc, srcStep, roiSize)); /* fill source image */

    check_sts(status = ippiFilterGaussianGetBufferSize(roiSize, kernelSize, ipp32f, numChannels,
&iSpecSize, &iTmpBufSize));

    pSpec = (IppFilterGaussianSpec *)ippsMalloc_8u(iSpecSize);
    pBuffer = ippsMalloc_8u(iTmpBufSize);

    check_sts(status = ippiFilterGaussianInit(roiSize, kernelSize, sigma, borderType, ipp32f,
numChannels, pSpec, pBuffer));
```

```
    check_sts(status = ippiFilterGaussianBorder_32f_C1R(pSrc, srcStep, pDst, dstStep, roiSize,
borderValue, pSpec, pBuffer));

    EXIT_MAIN
    ippsFree(pBuffer);
    ippsFree(pSpec);
    ippiFree(pSrc);
    ippiFree(pDst);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

See Also

Regions of Interest in Intel IPP

User-defined Border Types

FilterGaussianGetBufferSize Computes the size of the Gaussian specification structure and the size of the external work buffer for Gaussian filtering with user-defined borders.

FilterGaussianInit Initializes the Gaussian context structure.

SumWindow

Sums pixel values in a rectangular area applied to an image.

Syntax

```
IppStatus ippiSumWindow_8u32s_C1R(const Ipp8u* pSrc, int srcStep, Ipp32s* pDst, int
dstStep, IppiSize roiSize, IppiSize maskSize, IppiBorderType BorderType, const Ipp8u*
borderValue, Ipp8u* pBuffer);

IppStatus ippiSumWindow_8u32s_C3R(const Ipp8u* pSrc, int srcStep, Ipp32s* pDst, int
dstStep, IppiSize roiSize, IppiSize maskSize, IppiBorderType BorderType, const Ipp8u*
borderValue, Ipp8u* pBuffer);

IppStatus ippiSumWindow_8u32s_C4R(const Ipp8u* pSrc, int srcStep, Ipp32s* pDst, int
dstStep, IppiSize roiSize, IppiSize maskSize, IppiBorderType BorderType, const Ipp8u*
borderValue, Ipp8u* pBuffer);

IppStatus ippiSumWindow_8u32s_AC4R(const Ipp8u* pSrc, int srcStep, Ipp32s* pDst, int
dstStep, IppiSize roiSize, IppiSize maskSize, IppiBorderType BorderType, const Ipp8u*
borderValue, Ipp8u* pBuffer);

IppStatus ippiSumWindow_16s32f_C1R(const Ipp16s* pSrc, int srcStep, Ipp32f* pDst, int
dstStep, IppiSize roiSize, IppiSize maskSize, IppiBorderType BorderType, const Ipp16s*
borderValue, Ipp8u* pBuffer);

IppStatus ippiSumWindow_16s32f_C3R(const Ipp16s* pSrc, int srcStep, Ipp32f* pDst, int
dstStep, IppiSize roiSize, IppiSize maskSize, IppiBorderType BorderType, const Ipp16s*
borderValue, Ipp8u* pBuffer);

IppStatus ippiSumWindow_16s32f_C4R(const Ipp16s* pSrc, int srcStep, Ipp32f* pDst, int
dstStep, IppiSize roiSize, IppiSize maskSize, IppiBorderType BorderType, const Ipp16s*
borderValue, Ipp8u* pBuffer);

IppStatus ippiSumWindow_16s32f_AC4R(const Ipp16s* pSrc, int srcStep, Ipp32f* pDst, int
dstStep, IppiSize roiSize, IppiSize maskSize, IppiBorderType BorderType, const Ipp16s*
borderValue, Ipp8u* pBuffer);
```

```
IppStatus ippiSumWindow_16u32f_C1R(const Ipp16u* pSrc, int srcStep, Ipp32f* pDst, int
dstStep, IppiSize roiSize, IppiSize maskSize, IppiBorderType BorderType, const Ipp16u*
borderValue, Ipp8u* pBuffer);

IppStatus ippiSumWindow_16u32f_C3R(const Ipp16u* pSrc, int srcStep, Ipp32f* pDst, int
dstStep, IppiSize roiSize, IppiSize maskSize, IppiBorderType BorderType, const Ipp16u*
borderValue, Ipp8u* pBuffer);

IppStatus ippiSumWindow_16u32f_C4R(const Ipp16u* pSrc, int srcStep, Ipp32f* pDst, int
dstStep, IppiSize roiSize, IppiSize maskSize, IppiBorderType BorderType, const Ipp16u*
borderValue, Ipp8u* pBuffer);

IppStatus ippiSumWindow_16u32f_AC4R(const Ipp16u* pSrc, int srcStep, Ipp32f* pDst, int
dstStep, IppiSize roiSize, IppiSize maskSize, IppiBorderType BorderType, const Ipp16u*
borderValue, Ipp8u* pBuffer);

IppStatus ippiSumWindow_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int
dstStep, IppiSize roiSize, IppiSize maskSize, IppiBorderType BorderType, const Ipp32f*
borderValue, Ipp8u* pBuffer);

IppStatus ippiSumWindow_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int
dstStep, IppiSize roiSize, IppiSize maskSize, IppiBorderType BorderType, const Ipp32f*
borderValue, Ipp8u* pBuffer);

IppStatus ippiSumWindow_32f_C4R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int
dstStep, IppiSize roiSize, IppiSize maskSize, IppiBorderType BorderType, const Ipp32f*
borderValue, Ipp8u* pBuffer);

IppStatus ippiSumWindow_32f_AC4R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int
dstStep, IppiSize roiSize, IppiSize maskSize, IppiBorderType BorderType, const Ipp32f*
borderValue, Ipp8u* pBuffer);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between the starting points of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance in bytes between the starting points of consecutive lines in the destination image.
`roiSize`	Size of the destination ROI in pixels.
`maskSize`	Size of the mask in pixels.
`BorderType`	Type of border. Possible values are:

	• ippBorderConst • Values of all border pixels are set to constant. • ippBoderRepl • Border is replicated from the edge pixels. • ippBorderInMem • Border is obtained from the source image pixels in memory. • ippBorderMirror • Border pixels are mirrored from the source image boundary pixels.
borderValue	Constant value to assign to border pixels. This parameter is applicable only to the ippBorderConst border type.
pBuffer	Pointer to the work buffer.

Description

Before using this function, you need to compute the size of the work buffer pBuffer using the SumWindowGetBufferSize function.

This function operates with ROI (see Regions of Interest in Intel IPP).

This function sets each pixel in the destination image ROI pDst to the sum of all the source image pixels in the rectangular neighborhood of size maskSize with the anchor cell at the corresponding pixel in the source image ROI pSrc. To ensure valid operation while processing the image boundary pixels, the application must correctly define additional border pixels (see Borders in Neighborhood Operations).

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error if pSrc or pDst is NULL.
ippStsSizeErr	Indicates an error if roiSize has a field with a zero or negative value.
ippStsMaskSizeErr	Indicates an error if maskSize has a field with a zero or negative value.
ippStsBorderErr	Indicates an error if BorderType has an illegal value.

SumWindowGetBufferSize

Computes the size of the external buffer for the SumWindow function.

Syntax

```
IppStatus ippiSumWindowGetBufferSize(IppiSize roiSize, IppiSize maskSize, IppDataType dataType, int numChannels, int* pBufferSize);
```

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

roiSize	Maximum size of the destination ROI in pixels.
maskSize	Size of the filter mask in pixels.
dataType	Data type of the image. Possible values are: ipp8u, ipp16s, ipp16u, and ipp32f
numChannels	Number of channels in the image. Possible values are 1, 3, and 4.
pBufferSize	Pointer to the size, in bytes, of the external work buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function computes the size, in bytes, of the external work buffer for the ippiSumWindow function. The result is stored in the pBufferSize parameter.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsSizeErr	Indicates an error if roiSize has a field with a zero or negative value.
ippStsMaskSizeErr	Indicates an error if maskSize has an illegal value.
ippStsDataTypeErr	Indicates an error if dataType has an illegal value.
ippStsNumChannelsError	Indicates an error if numChannels has an illegal value.

SumWindowRow

Sums pixel values in the row mask applied to the image.

Syntax

```
IppStatus ippiSumWindowRow_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp32f*
pDst, int dstStep, IppiSize dstRoiSize, int maskSize, int anchor);
```

Supported values for mod:

8u32f_C1R	16u32f_C1R	16s32f_C1R
8u32f_C3R	16u32f_C3R	16s32f_C3R
8u32f_C4R	16u32f_C4R	16s32f_C4R

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
dstRoiSize	Size of the destination ROI in pixels.
maskSize	Size of the horizontal row mask in pixels.
anchor	Anchor cell specifying the row mask alignment with respect to the position of the input pixel.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function sets each pixel in the destination image ROI pDst as the sum of all the source image pixels in the horizontal row mask of size maskSize with the anchor cell anchor at the corresponding pixel in the source image ROI pSrc. To ensure valid operation when image boundary pixels are processed, the application must correctly define additional border pixels (see Borders in Neighborhood Operations).

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error if pSrc, pDst is NULL.
ippStsSizeErr	Indicates an error if dstRoiSize has a field with a zero or negative value.
ippStsMaskSizeErr	Indicates an error if maskSize has a field with a zero or negative value.

ippStsAnchorErr	Indicates an error if anchor is outside the mask size.
ippStsMemAllocErr	Indicates a memory allocation error.

SumWindowColumn

Sums pixel values in the column mask applied to the image.

Syntax

```
IppStatus ippiSumWindowColumn_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, int maskSize, int anchor);
```

Supported values for mod:

8u32f_C1R	16u32f_C1R	16s32f_C1R
8u32f_C3R	16u32f_C3R	16s32f_C3R
8u32f_C4R	16u32f_C4R	16s32f_C4R

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
dstRoiSize	Size of the destination ROI in pixels.
maskSize	Size of the vertical column mask in pixels.
anchor	Anchor cell specifying the column mask alignment with respect to the position of the input pixel.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function sets each pixel in the destination image ROI `pDst` as the sum of all the source image pixels in the vertical column mask of size `maskSize` with the anchor cell `anchor` at the corresponding pixel in the source image ROI `pSrc`. To ensure valid operation when image boundary pixels are processed, the application must correctly define additional border pixels (see Borders in Neighborhood Operations).

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error if `pSrc`, `pDst` is `NULL`.
`ippStsSizeErr`	Indicates an error if `dstRoiSize` has a field with a zero or negative value.
`ippStsMaskSizeErr`	Indicates an error if `maskSize` has a field with a zero or negative value.
`ippStsAnchorErr`	Indicates an error if `anchor` is outside the mask size.
`ippStsMemAllocErr`	Indicates a memory allocation error.

FilterMaxBorderGetBufferSize, FilterMinBorderGetBufferSize

Compute the size of the work buffer for the maximum/minimum filter.

Syntax

```
IppStatus ippiFilterMaxBorderGetBufferSize(IppiSize dstRoiSize, IppiSize maskSize,
IppDataType dataType, int numChannels, int* pBufferSize);
```

```
IppStatus ippiFilterMinBorderGetBufferSize(IppiSize dstRoiSize, IppiSize maskSize,
IppDataType dataType, int numChannels, int* pBufferSize);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`dstRoiSize`	Size of the destination ROI, in pixels.
`maskSize`	Size of the filter kernel.
`dataType`	Data type of the source and destination images.
`numChannels`	Number of channels in the image. Possible values are 1, 3, or 4.
`pBufferSize`	Pointer to the size, in bytes, of the external buffer.

Description

The ippiFilterMaxBorderGetBufferSize and ippiFilterMinBorderGetBufferSize functions compute the size, in bytes, of the external work buffer for the ippiFilterMaxBorder and ippiFilterMinBorder functions, respectively. The result is stored in the pBufferSize parameter.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error when pBufferSize is NULL.
ippStsSizeErr	Indicates an error when dstRoiSize has a field with a zero or negative value.
ippStsMaskSizeErr	Indicates an error when maskSize is less than, or e qual to zero.
ippStsDataTypeErr	Indicates an error when dataType has an illegal value.
ippStsNumChannelErr	Indicates an error when numChannels has an illegal value.

See Also

FilterMaxBorder, FilterMinBorder Filter an image using the maximum/minimum filter.

FilterMaxBorder, FilterMinBorder

Filter an image using the maximum/minimum filter.

Syntax

Case 1: Operating on one-channel data

```
IppStatus ippiFilterMaxBorder_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, Ipp8u borderValue, Ipp8u* pBuffer);

IppStatus ippiFilterMaxBorder_16s_C1R(const Ipp16s* pSrc, int srcStep, Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, Ipp16s borderValue, Ipp8u* pBuffer);

IppStatus ippiFilterMaxBorder_16u_C1R(const Ipp16u* pSrc, int srcStep, Ipp16u* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, Ipp16u borderValue, Ipp8u* pBuffer);

IppStatus ippiFilterMaxBorder_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, Ipp32f borderValue, Ipp8u* pBuffer);

IppStatus ippiFilterMinBorder_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, Ipp8u borderValue, Ipp8u* pBuffer);

IppStatus ippiFilterMinBorder_16s_C1R(const Ipp16s* pSrc, int srcStep, Ipp16s* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, Ipp16s borderValue, Ipp8u* pBuffer);
```

```
IppStatus ippiFilterMinBorder_16u_C1R(const Ipp16u* pSrc, int srcStep, Ipp16u* pDst,
int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, Ipp16u
borderValue, Ipp8u* pBuffer);

IppStatus ippiFilterMinBorder_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, Ipp32f
borderValue, Ipp8u* pBuffer);
```

Case 2: Operating on multi-channel data

```
IppStatus ippiFilterMaxBorder_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int
dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, const Ipp8u
pBorderValue[3], Ipp8u* pBuffer);

IppStatus ippiFilterMaxBorder_16s_C3R(const Ipp16s* pSrc, int srcStep, Ipp16s* pDst,
int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, const
Ipp16s pBorderValue[3], Ipp8u* pBuffer);

IppStatus ippiFilterMaxBorder_16u_C3R(const Ipp16u* pSrc, int srcStep, Ipp16u* pDst,
int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, const
Ipp16u pBorderValue[3], Ipp8u* pBuffer);

IppStatus ippiFilterMaxBorder_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, const
Ipp32f pBorderValue[3], Ipp8u* pBuffer);

IppStatus ippiFilterMaxBorder_8u_AC4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int
dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, const Ipp8u
pBorderValue[3], Ipp8u* pBuffer);

IppStatus ippiFilterMaxBorder_16s_AC4R(const Ipp16s* pSrc, int srcStep, Ipp16s* pDst,
int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, const
Ipp16s pBorderValue[3], Ipp8u* pBuffer);

IppStatus ippiFilterMaxBorder_16u_AC4R(const Ipp16u* pSrc, int srcStep, Ipp16u* pDst,
int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, const
Ipp16u pBorderValue[3], Ipp8u* pBuffer);

IppStatus ippiFilterMaxBorder_32f_AC4R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, const
Ipp32f pBorderValue[3], Ipp8u* pBuffer);

IppStatus ippiFilterMaxBorder_8u_C4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int
dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, const Ipp8u
pBorderValue[4], Ipp8u* pBuffer);

IppStatus ippiFilterMaxBorder_16s_C4R(const Ipp16s* pSrc, int srcStep, Ipp16s* pDst,
int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, const
Ipp16s pBorderValue[4], Ipp8u* pBuffer);

IppStatus ippiFilterMaxBorder_16u_C4R(const Ipp16u* pSrc, int srcStep, Ipp16u* pDst,
int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, const
Ipp16u pBorderValue[4], Ipp8u* pBuffer);

IppStatus ippiFilterMaxBorder_32f_C4R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, const
Ipp32f pBorderValue[4], Ipp8u* pBuffer);
```

```
IppStatus ippiFilterMinBorder_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int
dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, const Ipp8u
pBorderValue[3], Ipp8u* pBuffer);

IppStatus ippiFilterMinBorder_16s_C3R(const Ipp16s* pSrc, int srcStep, Ipp16s* pDst,
int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, const
Ipp16s pBorderValue[3], Ipp8u* pBuffer);

IppStatus ippiFilterMinBorder_16u_C3R(const Ipp16u* pSrc, int srcStep, Ipp16u* pDst,
int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, const
Ipp16u pBorderValue[3], Ipp8u* pBuffer);

IppStatus ippiFilterMinBorder_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, const
Ipp32f pBorderValue[3], Ipp8u* pBuffer);

IppStatus ippiFilterMinBorder_8u_AC4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int
dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, const Ipp8u
pBorderValue[3], Ipp8u* pBuffer);

IppStatus ippiFilterMinBorder_16s_AC4R(const Ipp16s* pSrc, int srcStep, Ipp16s* pDst,
int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, const
Ipp16s pBorderValue[3], Ipp8u* pBuffer);

IppStatus ippiFilterMinBorder_16u_AC4R(const Ipp16u* pSrc, int srcStep, Ipp16u* pDst,
int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, const
Ipp16u pBorderValue[3], Ipp8u* pBuffer);

IppStatus ippiFilterMinBorder_32f_AC4R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, const
Ipp32f pBorderValue[3], Ipp8u* pBuffer);

IppStatus ippiFilterMinBorder_8u_C4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int
dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, const Ipp8u
pBorderValue[4], Ipp8u* pBuffer);

IppStatus ippiFilterMinBorder_16s_C4R(const Ipp16s* pSrc, int srcStep, Ipp16s* pDst,
int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, const
Ipp16s pBorderValue[4], Ipp8u* pBuffer);

IppStatus ippiFilterMinBorder_16u_C4R(const Ipp16u* pSrc, int srcStep, Ipp16u* pDst,
int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, const
Ipp16u pBorderValue[4], Ipp8u* pBuffer);

IppStatus ippiFilterMinBorder_32f_C4R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiBorderType borderType, const
Ipp32f pBorderValue[4], Ipp8u* pBuffer);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance, in bytes, between the starting points of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance, in bytes, between the starting points of consecutive lines in the destination image.
dstRoiSize	Size of the source and destination ROI, in pixels.
maskSize	Size of the filter kernel.
borderType	Type of border. Possible values are: • ippBorderConst • Values of all border pixels are set to a constant. • ippBorderRepl • Border is replicated from the edge pixels. • ippBorderInMem • Border is obtained from the source image pixels in memory.
borderValue	Constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.
pBorderValue[3], pBorderValue[4]	Pointer to constant values to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.
pBuffer	Pointer to the work buffer.

Description

Before using the ippiFilterMaxBorder and ippiFilterMinBorder functions, you need to compute the size of the work buffer pBuffer using the ippiFilterMaxBorderGetBufferSize or ippiFilterMinBorderGetBufferSize functions, respectively.

These functions operate with ROI (see Regions of Interest in Intel IPP).

The ippiFilterMaxBorder and ippiFilterMinBorder functions apply the maximum/minimum filters, respectively, to the source image ROI. The size of the source image ROI is equal to the destination image ROI size dstRoiSize. The values of border pixels are assigned in accordance with the borderType and borderValue parameters. The kernel size of the filter is arbitrary and depends on the mask value.

The anchor cell is the center cell of the kernel, highlighted in red. The anchor cell is defined as:

x=(maskSize.width – 1)/2

y=(maskSize.height – 1)/2

where

(x, y) are cell coordinates.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when dstRoiSize is negative, or equal to zero.
ippStsBorderErr	Indicates an error when mask is less than, or equal to zero.
ippStsBorderErr	Indicates an error when borderType has an illegal value.

See Also

Borders in Neighborhood Operations

Regions of Interest in Intel IPP

User-defined Border Types

FilterMaxBorderGetBufferSize, FilterMinBorderGetBufferSize Compute the size of the work buffer for the maximum/minimum filter.

DecimateFilterRow, DecimateFilterColumn

Decimates an image by rows or by columns.

Syntax

```
IppStatus ippiDecimateFilterRow_8u_C1R(const Ipp8u* pSrc, int srcStep, IppiSize srcRoiSize, Ipp8u* pDst, int dstStep, IppiFraction fraction);
```

```
IppStatus ippiDecimateFilterColumn_8u_C1R(const Ipp8u* pSrc, int srcStep, IppiSize srcRoiSize, Ipp8u* pDst, int dstStep, IppiFraction fraction);
```

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
srcRoiSize	Size of the source image ROI in pixels.
pDst	Pointer to the destination image.

dstStep	Distance in bytes between starts of consecutive lines in the destination image.
fraction	Specifies how the decimating is performed. Possible values: • ippPolyphase_1_2, • ippPolyphase_3_5, • ippPolyphase_2_3, • ippPolyphase_7_10, • ippPolyphase_3_4.

Description

These functions operate with ROI (see Regions of Interest in Intel IPP).

Functions DecimateFilterRow and DecimateFilterColumn perform decimating of the source image by rows or by columns respectively. These functions use the set of special internal polyphase filters. The parameter fraction specifies how the decimating is performed, for example, if the parameter is set to ippPolyphase_3_5, then each 5 pixels in the row (or column) of the source image give 3 pixels to the destination image, if the parameter is set to ippPolyphase_1_2, then each two pixels in the row (or column) of the source image give 1 pixel to the destination image, an so on.

To ensure valid operation, the application must correctly define additional border pixels (see Borders in Neighborhood Operations). For all fraction values the width of the border is four columns/rows all around the source image ROI.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error if roiSize has a field with a zero or negative value.
ippStsStepErr	Indicates an error if srcStep or dstStep is less than or equal to zero.
ippStsDecimateFractionErr	Indicates an error if fraction has an illegal value.

Median Filters

The median filter functions perform non-linear filtering of a source image data.

These functions use either an arbitrary rectangular mask, or the following predefined masks of the IppiMaskSize type to filter an image:

ippMskSize3x1	Horizontal mask of length 3
ippMskSize5x1	Horizontal mask of length 5
ippMskSize1x3	Vertical mask of length 3
ippMskSize3x3	Square mask of size 3

ippMskSize1x5	Vertical mask of length 5
ippMskSize5x5	Square mask of size 5

The size of the neighborhood and coordinates of the anchor cell in the neighborhood depend on the `mask` mean value. Table Median Filter Mask, Neighborhood, and Anchor Cell lists the mask types with the corresponding neighborhood sizes and anchor cell coordinates. Mask size in mask names is indicated in (`XY`) order. The anchor cell is specified by its coordinates `anchor.x` and `anchor.y` in the coordinate system associated with the upper left corner of the mask.

Median Filter Mask, Neighborhood, and Anchor Cell

Mask	Neighborhood Size Columns	Neighborhood Size Rows	Anchor Cell
ippMskSize3x1	3	1	[1, 0]
ippMskSize5x1	5	1	[2, 0]
ippMskSize1x3	1	3	[0, 1]
ippMskSize3x3	3	3	[1, 1]
ippMskSize1x5	1	5	[0, 2]
ippMskSize5x5	5	5	[2, 2]

Median filters have the effect of removing the isolated intensity spikes and can be used to reduce noise in an image.

For details on algorithms used in Intel IPP for median filtering, see [APMF].

- FilterMedianBorderGetBufferSize
- FilterMedianBorder
- FilterMedianGetBufferSize
- FilterMedian
- FilterMedianCross
- FilterMedianWeightedCenter3x3
- FilterMedianColor

FilterMedianBorderGetBufferSize

Computes the size of the work buffer for the `FilterMedianBorder` function.

Syntax

```
IppStatus ippiFilterMedianBorderGetBufferSize (IppiSize dstRoiSize, IppiSize maskSize,
IppDataType dataType, int numChannels, int* pBufferSize);
```

```
IppStatus ippiFilterMedianBorderGetBufferSize_T (IppiSize dstRoiSize, IppiSize
maskSize, IppDataType dataType, int numChannels, int* pBufferSize);
```

Include Files

`ippi.h`

Parameters

dstRoiSize	Size of the destination ROI, in pixels.
maskSize	Size of the filter mask, in pixels.

dataType	Data type of the source and destination images. Possible values are ipp8u, ipp16u, ipp16s, or ipp32f.
numChannels	Number of channels in the image. Possible values are 1, 3, or 4.
pBufferSize	Pointer to the computed size of the external work buffer, in bytes.

Description

The ippiFilterMedianBorderGetBufferSize function computes the size, in bytes, of the external work buffer needed for the ippiFilterMedianBorder function. The result is stored in the pBufferSize parameter.

For an example on how to use this function, see the code example provided with the ippiFilterMedianBorder function description.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pBufferSize is NULL.
ippStsSizeErr	Indicates an error when one of the dstRoiSize fields has a negative or zero value.
ippStsMaskSizeErr	Indicates an error when mask has a field with a negative, zero, or even value.
ippStsDataTypeErr	Indicates an error when dataType has an illegal value.
ippStsNumChannelsError	Indicates an error when numChannels has an illegal value.

See Also

FilterMedianBorder Performs median filtering of an image.

FilterMedianBorder

Performs median filtering of an image.

Syntax

Case 1: Operating on one-channel data

```
IppStatus ippiFilterMedianBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize,
IppiBorderType borderType, Ipp<datatype> borderValue, Ipp8u* pBuffer);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32f_C1R

Case 2: Operating on multi-channel data

```
IppStatus ippiFilterMedianBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize,
IppiBorderType borderType, const Ipp<datatype> pBorderValue[3], Ipp8u* pBuffer);
```

Supported values for `mod`:

8u_C3R	16u_C3R	16s_C3R
8u_AC4R	16u_AC4R	16s_AC4R

```
IppStatus ippiFilterMedianBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize,
IppiBorderType borderType, const Ipp<datatype> pBorderValue[4], Ipp8u* pBuffer);
```

Supported values for `mod`:

8u_C4R	16u_C4R	16s_C4R

Case 3: Operating on one-channel data with Threading Layer (TL) functions based on the Classic API

```
IppStatus ippiFilterMedianBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize,
IppiBorderType borderType, Ipp<datatype> borderValue, Ipp8u* pBuffer);
```

Supported values for `mod`:

8u_C1R_T

Case 4: Operating on multi-channel data with Threading Layer (TL) functions based on the Classic API

```
IppStatus ippiFilterMedianBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize,
IppiBorderType borderType, const Ipp<datatype> pBorderValue[3], Ipp8u* pBuffer);
```

Supported values for `mod`:

8u_C3R_T
8u_AC4R_T

```
IppStatus ippiFilterMedianBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize,
IppiBorderType borderType, const Ipp<datatype> pBorderValue[4], Ipp8u* pBuffer);
```

Supported values for `mod`:

8u_C4R_T

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

pSrc	Pointer to the source image.
srcStep	Distance, in bytes, between the starting points of consecutive lines in the source image.
pDst	Pointer to the destination image.
dstStep	Distance, in bytes, between the starting points of consecutive lines in the destination image.
dstRoiSize	Size of the destination ROI, in pixels.
maskSize	Size of the filter mask, in pixels.
borderType	Type of border. Possible values are: • ippBorderConst • Values of all border pixels are set to constant. • ippBorderRepl • Border is replicated from the edge pixels. • ippBorderInMem • Border is obtained from the source image pixels in memory.
borderValue	Constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.
pBorderValue[3], pBorderValue[4]	Pointer to the constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.
pBuffer	Pointer to the work buffer.

Description

This function operates with ROI.

Before using this function, you need to compute the size of the work buffer pBuffer using the ippiFilterMedianBorderGetBufferSize function.

The ippiFilterMedianBorder function applies a median filter to an image ROI. The anchor cell is the center of the filter kernel. The size of the source image ROI is equal to the size of the destination image ROI dstRoiSize.

This function sets each pixel in the destination buffer as the median value of all source pixels values from the neighborhood of the processed pixel.

This function removes noise and does not cut out signal brightness drops, as an averaging filter does.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pSrc, pDst, pBuffer is NULL.

ippStsSizeErr	Indicates an error if dstRoiSize has a field with a zero or negative value.
ippStsMaskSizeErr	Indicates an error if maskSize has a field with a zero, negative, or even value.
ippStsNotEvenStepErr	Indicates an error if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
ippStsBorderErr	Indicates an error when borderType has an illegal value.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

//   A simple example of performing Bluring an image using a median filter
// implemented with Intel(R) Integrated Performance Primitives (Intel(R) IPP) functions:
//     ippiFilterMedianBorderGetBufferSize
//     ippiFilterMedianBorder_8u_C1R

#include <stdio.h>
#include "ipp.h"

#define WIDTH  16 /* image width  */
#define HEIGHT 16 /* image height */

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define check_sts(st)        \
    if ((st) != ippStsNoErr) \
        goto exitLine; /* Go to Exit if Intel(R) IPP function returned status different from
ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main(void)
{
    IppStatus status = ippStsNoErr;
    Ipp8u *pSrc = NULL, *pDst = NULL;               /* Pointers to source/destination images */
    int srcStep = 0, dstStep = 0;                   /* Steps, in bytes, through the source/
```

```
destination images */
    IppiSize srcImageSize = {WIDTH + 1, HEIGHT + 1}; /* size of source image in pixels */
    IppiSize roiSize = {WIDTH, HEIGHT};              /* Size of source/destination ROI in pixels
*/
    Ipp8u *pBuffer = NULL;                           /* Pointer to the work buffer */
    int iTmpBufSize = 0;                             /* Common work buffer size */
    IppiBorderType borderType = (IppiBorderType)(ippBorderRepl | ippBorderInMemTop |
ippBorderInMemRight);
    Ipp8u borderValue = 254;
    IppiSize maskSize = {3, 3};
    int numChannels = 1;

    /* memory allocation */
    pSrc = ippiMalloc_8u_C1(srcImageSize.width, srcImageSize.height, &srcStep);
    pDst = ippiMalloc_8u_C1(roiSize.width, roiSize.height, &dstStep);

    check_sts(status = ippiFilterMedianBorderGetBufferSize(roiSize, maskSize, ipp8u,
numChannels, &iTmpBufSize));

    pBuffer = ippsMalloc_8u(iTmpBufSize);

    check_sts(status = ippiFilterMedianBorder_8u_C1R(pSrc + srcStep, srcStep, pDst, dstStep,
roiSize, maskSize, borderType, borderValue, pBuffer));

    EXIT_MAIN
    ippsFree(pBuffer);
    ippiFree(pSrc);
    ippiFree(pDst);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

See Also

Regions of Interest in Intel IPP

User-defined Border Types

FilterMedianBorderGetBufferSize Computes the size of the work buffer for the FilterMedianBorder function.

FilterMedianGetBufferSize

Computes the size of the external buffer for `ippiFilterMedian` function.

Syntax

```
IppStatus ippiFilterMedianGetBufferSize_32f(IppiSize dstRoiSize, IppiSize maskSize,
Ipp32u nChannels, Ipp32u* pBufferSize);
```

```
IppStatus ippiFilterMedianGetBufferSize_64f(IppiSize dstRoiSize, IppiSize maskSize,
Ipp32u nChannels, Ipp32u* pBufferSize);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

dstRoiSize	Size of the destination ROI in pixels.
maskSize	Size of the mask in pixels.
nChannels	Number of channels in the image.
pBufferSize	Pointer to the computed value of the external buffer size.

Description

This function computes the size in bytes of an external memory buffer that is required for the ippiFilterMedian function, and stores the result in the pBufferSize.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error if the pBufferSize pointer is NULL.
ippStsSizeErr	Indicates an error if one of the fields of dstRoiSize has a zero or negative value.
ippStsMaskSizeErr	Indicates an error if one of the fields of maskSize has a value less than or equal to 1.
ippStsNumChannelsErr	Indicates an error if nChannels is not equal to 1, 3, or 4.

FilterMedian

Filters an image using a median filter.

Syntax

```
IppStatus ippiFilterMedian_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiPoint anchor, Ipp8u* pBuffer);
```

Supported values for mod:

32f_C3R
32f_C4R
64f_C1R

Include Files

ippi.h

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image.
`dstRoiSize`	Size of the source and destination ROI in pixels.
`maskSize`	Size of the mask in pixels.
`anchor`	Anchor cell specifying the mask alignment with respect to the position of the input pixel.
`pBuffer`	Pointer to the external work buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function sets each pixel in the output buffer as the median value of all the input pixel values taken in the neighborhood of the processed pixel. To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see Borders in Neighborhood Operations). The anchor cell is specified by its coordinates `anchor.x` and `anchor.y` in the coordinate system associated with the top left corner of the kernel. The size of the source image ROI is equal to the size of the destination image ROI `dstRoiSize`.

Some flavors of the function require the external buffer `pBuffer`. Prior to using this functions, compute the size of the external buffer by using the function FilterMedianGetBufferSize.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error if `pSrc` or `pDst` is `NULL`.
`ippStsSizeErr`	Indicates an error if `dstRoiSize` has a field with a zero or negative value.
`ippStsStepErr`	Indicates an error if `srcStep` or `dstStep` has a zero or negative value.
`ippStsMaskSizeErr`	Indicates an error if `maskSize` has a field with zero, negative, or even value.
`ippStsAnchorErr`	Indicates an error if `anchor` is outside the mask size.

FilterMedianCross

Filters an image using a cross median filter.

Syntax

```
IppStatus ippiFilterMedianCross_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
```

Supported values for `mod`:

`8u_C1R`	`16u_C1R`	`16s_C1R`
`8u_C3R`	`16u_C3R`	`16s_C3R`
`8u_AC4R`	`16u_AC4R`	`16s_AC4R`

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image.
`dstRoiSize`	Size of the source and destination ROI in pixels.
`mask`	Predefined mask of the `IppiMaskSize` type.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function sets each pixel in the output buffer as the median value of all the input pixel values taken in the neighborhood of the processed pixel. The neighborhood is determined by the square mask of the predefined size, which can be either `ippMskSize3x3` or `ippMskSize5x5` (see Table "Median Filter Mask, Neighborhood, and Anchor Cell"). The function operates on the assumption that the pixels outside the source image ROI exist along the distance equal to half of the mask size. It means that the application program should provide appropriate values for the `pSrc` and `dstRoiSize` arguments, or define additional border pixels (see Borders in Neighborhood Operations). The size of the source image ROI is equal to the size of the destination image ROI `dstRoiSize`.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error if pSrc or pDst is NULL.
ippStsSizeErr	Indicates an error if dstRoiSize has a field with a zero or negative value.
ippStsStepErr	Indicates an error if srcStep or dstStep has a zero or negative value.
ippStsMaskSizeErr	Indicates an error if mask has an illegal value.

FilterMedianWeightedCenter3x3

Filters an image using a median filter with a weighted center pixel.

Syntax

```
IppStatus ippiFilterMedianWeightedCenter3x3_8u_C1R(const Ipp8u* pSrc, int srcStep,
Ipp8u* pDst, int dstStep, IppiSize dstRoiSize, int weight);
```

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
dstRoiSize	Size of the source and destination ROI in pixels.
weight	Weight of the pixel, must be an odd number.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function sets each pixel in the destination image as the median value of all the input pixel values taken in the neighborhood of the processed pixel. The neighborhood is determined by the fixed square mask of the 3x3 size with the anchor cell as the center cell of the mask. The parameter weight specifies the weight of

the processed pixel, that is how many times its value is included into calculations. The value of this parameter should be odd. If it is even, the function changes its value to the nearest less odd number and returns the warning message.

The function operates on the assumption that the pixels outside of the source image ROI exist along the distance equal to half of the mask size. It means that the application program should provide appropriate values for the `pSrc` and `dstRoiSize` arguments, or define additional border pixels (see Borders in Neighborhood Operations). The size of the source image ROI is equal to the size of the destination image ROI `dstRoiSize`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error if `pSrc` or `pDst` is `NULL`.
`ippStsSizeErr`	Indicates an error if `dstRoiSize` has a field with a zero or negative value.
`ippStsStepErr`	Indicates an error if `srcStep` or `dstStep` has a zero or negative value.
`ippStsWeightErr`	Indicates an error if `weight` is less than or equal to 0.
`ippStsEvenMedianWeight`	Indicates a warning if `weight` has an even value.

FilterMedianColor

Filters an image using a color median filter.

Syntax

```
IppStatus ippiFilterMedianColor_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask);
```

Supported values for `mod`:

`8u_C3R`	`16s_C3R`	`32f_C3R`
`8u_AC4R`	`16s_AC4R`	`32f_AC4R`

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.

srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
dstRoiSize	Size of the source and destination ROI in pixels.
mask	Predefined mask of the IppiMaskSize type.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

When applied to a color image, the previously described median filtering functions process color planes of an image separately, and as a result any correlation between color components is lost. If you want to preserve this information, use the ippiFilterMedianColor function instead. For each input pixel, this function computes differences between red ®, green (G), and blue (B) color components of pixels in the mask neighborhood and the input pixel. The distance between the input pixel i and the neighborhood pixel j is formed as the sum of absolute values:

```
abs (R(i)-R(j)) + abs (G(i)-G(j)) + abs (B(i)-B(j))
```

After scanning the entire neighborhood, the function sets the output value for pixel i as the value of the neighborhood pixel with the smallest distance to i.

The function ippiFilterMedianColor supports square masks of size either ippMskSize3x3 or ippMskSize5x5 and processes color images only. To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see Borders in Neighborhood Operations).

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error if pSrc or pDst is NULL.
ippStsSizeErr	Indicates an error if dstRoiSize has a field with a zero or negative value.
ippStsStepErr	Indicates an error if srcStep or dstStep has a zero or negative value.
ippStsMaskSizeErr	Indicates an error if mask has an illegal value.

General Linear Filters

These functions use a general rectangular kernel to filter an image. The kernel is a matrix of signed integers or single-precision real values. For each input pixel, the kernel is placed on the image in such a way that the fixed anchor cell within the kernel coincides with the input pixel. The anchor cell is usually a geometric center of the kernel, but can be skewed with respect to the geometric center.

A pointer to an array of kernel values is passed to filtering functions. These values are read in row-major order starting from the top left corner. This array must exactly have `kernelSize.width *kernelSize.height` entries. The anchor cell is specified by its coordinates `anchor.x` and `anchor.y` in the coordinate system associated with the lower right corner of the kernel.

The output value is computed as a sum of neighbor pixels values, with kernel matrix elements used as weight factors. Summation formulas implement a convolution operation, which means that kernel coefficients are used in direct order.

NOTE In Intel IPP 8.2 and lower versions, kernel coefficients are used in inverse order.

Optionally, the output pixel values may be scaled. To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see Borders in Neighborhood Operations).

- FilterBorderGetSize
- FilterBorderInit
- FilterBorder
- FilterBorderSetMode
- FilterGetBufSize
- Filter

FilterBorderGetSize

Computes the size of the filter specification structure and the size of the work buffer.

Syntax

```
IppStatus ippiFilterBorderGetSize (IppiSize kernelSize, IppiSize dstRoiSize,
IppDataType dataType, IppDataType kernelType, int numChannels, int* pSpecSize, int*
pBufferSize);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`kernelSize`	Size of the rectangular kernel in pixels.
`dstRoiSize`	Maximal size of the destination image ROI (in pixels).
`dataType`	Data type of the source image. Possible values are `ipp8u`, `ipp16u`, `ipp16s`, `ipp32f`, or `ipp64f`.
`kernelType`	Data type of the filter kernel. Possible values are `ipp16s`, `ipp32f`, or `ipp64f`.
`numChannels`	Number of channels in the image. Possible values are 1, 3, or 4.

pSpecSize	Pointer to the size of the filter specification structure.
pBufferSize	Pointer to the size of the work buffer required for filtering.

Description

This function operates with ROI.

This function computes the size of the filter specification structure pSpec and the size of the buffer required for filtering operations. Call this function before using the ippiFilterBorderInit function.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when kernelSize has a field with a zero or negative value, or if dstRoiSize is less than 1.
ippStsDataTypeErr	Indicates an error when dataType or kernelType has an illegal value.
ippStsChannelErr	Indicates an error when numChannels has an illegal value.

See Also

Regions of Interest in Intel IPP

FilterBorderInit Initializes the filter specification structure.

FilterBorder Filters an image using a rectangular filter.

FilterBorderInit

Initializes the filter specification structure.

Syntax

```
IppStatus ippiFilterBorderInit_16s(const Ipp16s* pKernel, IppiSize kernelSize, int divisor, IppDataType dataType, int numChannels, IppRoundMode roundMode, IppiFilterBorderSpec* pSpec);
```

```
IppStatus ippiFilterBorderInit_32f(const Ipp32f* pKernel, IppiSize kernelSize, IppDataType dataType, int numChannels, IppRoundMode roundMode, IppiFilterBorderSpec* pSpec);
```

```
IppStatus ippiFilterBorderInit_64f(const Ipp64f* pKernel, IppiSize kernelSize, IppDataType dataType, int numChannels, IppRoundMode roundMode, IppiFilterBorderSpec* pSpec);
```

Include Files

ippi.h

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pKernel`	Pointer to the kernel values.
`kernelSize`	Size of the rectangular kernel in pixels.
`divisor`	Integer value by which the computed result is divided.
`dataType`	Data type of the source image. Possible values are `ipp8u`, `ipp16u`, `ipp16s`, `ipp32f`, or `ipp64f`.
`numChannels`	Number of channels in the image. Possible values are 1, 3, or 4.
`roundMode`	Rounding mode, possible values: • `ippRndZero` • Floating-point values are truncated to zero. This mode is not • `ippRndNear` • Floating-point values are rounded to the nearest even integer when • `ippRndFinancial` • Floating-point values are rounded down to the nearest integer when • `ippRndHintAccurate` • The result of calculations is accurate. This mode is supported only
`pSpec`	Pointer to the filter specification structure.

Description

This function initializes the filter specification structure `pSpec` in the external buffer. Before using this function, you need to compute the size of the specification structure using the `FilterBorderGetSize` function. This structure is used by the `FilterBorder` function that performs filtering operations on the source image pixels.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error when `kernelSize` has a field with a zero or negative value.
`ippStsDataTypeErr`	Indicates an error when `dataType` has an illegal value.

ippStsRoundModeNotSupportedErr	Indicates an error when the specified rounding mode is not supported.
ippStsAccurateModeNotSupported	Indicates a warning when the ippRndHintAccurate mode is not supported.

See Also

FilterBorderGetSize Computes the size of the filter specification structure and the size of the work buffer.

FilterBorder Filters an image using a rectangular filter.

FilterBorder

Filters an image using a rectangular filter.

Syntax

Case 1: Operation on one-channel data

```
IppStatus ippiFilterBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize dstRoiSize, IppiBorderType border, const Ipp<datatype>
borderValue[1], const IppiFilterBorderSpec* pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u_C1R
16u_C1R
16s_C1R
32f_C1R
64f_C1R

Case 2: Operation on multi-channel data

```
IppStatus ippiFilterBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize dstRoiSize, IppiBorderType border, const Ipp<datatype>
borderValue[3], const IppiFilterBorderSpec* pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u_C3R
16u_C3R
16s_C3R
32f_C3R

```
IppStatus ippiFilterBorder_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize dstRoiSize, IppiBorderType border, const Ipp<datatype>
borderValue[4], const IppiFilterBorderSpec* pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u_C4R

16u_C4R
16s_C4R
32f_C4R

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance, in bytes, between the starting points of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance, in bytes, between starting points of consecutive lines in the destination image.
dstRoiSize	Size of the source and destination image ROI in pixels.
border	Type of border. Possible values are: • ippBorderConst • Values of all border pixels are set to constant. • ippBorderRepl • Border is replicated from the edge pixels. • ippBorderInMem • Border is obtained from the source image pixels in memory.
borderValue	Constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.
pSpec	Pointer to the filter specification structure.
pBuffer	Pointer to the work buffer for filtering operations.

Description

Before using this function, you need to initialize the filter specification structure using the ippiFilterBorderInit function.

This function operates with ROI.

This function performs filtering of a rectangular ROI inside a two-dimensional image using a specified structure pSpec. Type of the image border is defined by the value of the border parameter.

To change the function behavior (add offset to the result or set the rounding mode), use ippiFilterBorderSetMode after the ippiFilterBorderInit function.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when dstRoiSize has a field with a zero or negative value, or if dstRoiSize.width is more than the maximum ROI roiWidth passed to the initialization function.
ippStsStepErr	Indicates an error when the srcStep value is less than, or equal to zero.
ippStsBorderErr	Indicates an error when border has an illegal value.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

//   A simple example of performing filtering an image using a general integer rectangular kernel
// implemented with Intel(R) Integrated Performance Primitives (Intel(R) IPP) functions:
//     ippiFilterBorderGetSize
//     ippiFilterBorderInit_16s
//     ippiFilterBorder_8u_C1R

#include <stdio.h>
#include "ipp.h"

#define WIDTH  128 /* image width */
#define HEIGHT 64  /* image height */
const Ipp16s kernel[3 * 3] = {0, 1, 0, 1, 0, 1, 0, 1, 0};

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define check_sts(st)        \
    if ((st) != ippStsNoErr) \
        goto exitLine; /* Go to Exit if Intel(R) IPP function returned status different from
```

```
ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main(void)
{
    IppStatus status = ippStsNoErr;
    Ipp8u *pSrc = NULL, *pDst = NULL;   /* Pointers to source/destination images */
    int srcStep = 0, dstStep = 0;       /* Steps, in bytes, through the source/destination
images */
    IppiSize roiSize = {WIDTH, HEIGHT}; /* Size of source/destination ROI in pixels */
    IppiSize kernelSize = {3, 3};
    Ipp8u *pBuffer = NULL;              /* Pointer to the work buffer */
    IppiFilterBorderSpec *pSpec = NULL; /* context structure */
    int iTmpBufSize = 0, iSpecSize = 0; /* Common work buffer size */
    IppiBorderType borderType = ippBorderRepl;
    Ipp8u borderValue = 0;
    int numChannels = 1;

    pSrc = ippiMalloc_8u_C1(roiSize.width, roiSize.height, &srcStep);
    pDst = ippiMalloc_8u_C1(roiSize.width, roiSize.height, &dstStep);

    check_sts(status = ippiFilterBorderGetSize(kernelSize, roiSize, ipp8u, ipp16s, numChannels,
&iSpecSize, &iTmpBufSize));

    pSpec = (IppiFilterBorderSpec *)ippsMalloc_8u(iSpecSize);
    pBuffer = ippsMalloc_8u(iTmpBufSize);

    check_sts(status = ippiFilterBorderInit_16s(kernel, kernelSize, 4, ipp8u, numChannels,
ippRndNear, pSpec));

    check_sts(status = ippiFilterBorder_8u_C1R(pSrc, srcStep, pDst, dstStep, roiSize,
borderType, &borderValue, pSpec, pBuffer));

    EXIT_MAIN
    ippsFree(pBuffer);
    ippsFree(pSpec);
    ippiFree(pSrc);
    ippiFree(pDst);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

See Also

Regions of Interest in Intel IPP

User-defined Border Types

FilterBorderInit Initializes the filter specification structure.

FilterBorderSetMode Adds the offset value after filtering operation for ipp8u and ipp16u data types , and sets the rounding mode.

FilterBorderSetMode

Adds the offset value after filtering operation for `ipp8u` and `ipp16u` data types , and sets the rounding mode.

Syntax

```
IppStatus ippiFilterBorderSetMode(IppHintAlgorithm hint, int offset,
IppiFilterBorderSpec* pSpec);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`hint`	Suggests using specific code for rounding. Supported values: • `ippAlgHintNone`, `ippAlgHintFast` • Default modes. The function performs rounding in accordance • `ippAlgHintAccurate` • All output pixels are exact; accuracy takes precedence over
`offset`	Constant that is added to the final signed result before converting it to unsigned for `ipp8u` and `ipp16u` data types.
`pSpec`	Pointer to the initialized filter specification structure.

Description

This function adds the `offset` value after filtering operation for `ipp8u` and `ipp16u` data types with the ippiFilterBorder function:

```
pDst=(summ(src[i]*kern[i]))+offset
```

You can also use this function to set the rounding mode for the filtering result.

The `8u_C1R`, `8u_C3R`, `8u_C4R`, and `16s_C1RFilterBorder` function flavors initialized with the `ipp16s` coefficients support `ippAlgHintNone` and `ippAlgHintAccurate` rounding modes.

Use this function after the ippiFilterBorderInit function and before calling ippiFilterBorder.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when one of the specified pointers is `NULL`.
`ippStsNotSupportedModeErr`	The offset value is not supported (for `ipp16s` and `ipp32f` data types).

ippStsAccurateModeNotSupported	The accurate mode is not supported for some data types. The result of rounding may be not exact.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

//   A simple example of performing filtering an image using a general integer rectangular kernel
// implemented with Intel(R) Integrated Performance Primitives (Intel(R) IPP) functions:
//     ippiFilterBorderSetMode
//
//

#include <stdio.h>
#include "ipp.h"

#define WIDTH  128 /* image width */
#define HEIGHT 64  /* image height */
const Ipp16s kernel[3 * 3] = {0, 1, 0, 1, 0, 1, 0, 1, 0};

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define check_sts(st)        \
    if ((st) != ippStsNoErr) \
        goto exitLine; /* Go to Exit if Intel(R) IPP function returned status different from
ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main(void)
{
    IppStatus status = ippStsNoErr;
    Ipp8u *pSrc = NULL, *pDst = NULL;   /* Pointers to source/destination images */
    int srcStep = 0, dstStep = 0;       /* Steps, in bytes, through the source/destination
images */
    IppiSize roiSize = {WIDTH, HEIGHT}; /* Size of source/destination ROI in pixels */
    IppiSize kernelSize = {3, 3};
    Ipp8u *pBuffer = NULL;              /* Pointer to the work buffer */
    IppiFilterBorderSpec *pSpec = NULL; /* context structure */
    int iTmpBufSize = 0, iSpecSize = 0; /* Common work buffer size */
    IppiBorderType borderType = ippBorderRepl;
    Ipp8u borderValue = 0;
```

```
    int numChannels = 1;

    pSrc = ippiMalloc_8u_C1(roiSize.width, roiSize.height, &srcStep);
    pDst = ippiMalloc_8u_C1(roiSize.width, roiSize.height, &dstStep);

    check_sts(status = ippiFilterBorderGetSize(kernelSize, roiSize, ipp8u, ipp16s, numChannels,
&iSpecSize, &iTmpBufSize));

    pSpec = (IppiFilterBorderSpec *)ippsMalloc_8u(iSpecSize);
    pBuffer = ippsMalloc_8u(iTmpBufSize);

    check_sts(status = ippiFilterBorderInit_16s(kernel, kernelSize, 4, ipp8u, numChannels,
ippRndNear, pSpec));
    check_sts(status = ippiFilterBorderSetMode(ippAlgHintNone, 128, pSpec));
    check_sts(status = ippiFilterBorder_8u_C1R(pSrc, srcStep, pDst, dstStep, roiSize,
borderType, &borderValue, pSpec, pBuffer));

    EXIT_MAIN
    ippsFree(pBuffer);
    ippsFree(pSpec);
    ippiFree(pSrc);
    ippiFree(pDst);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

See Also

FilterBorder Filters an image using a rectangular filter.

FilterBorderInit Initializes the filter specification structure.

FilterGetBufSize

Computes the size of the work buffer.

Syntax

```
IppStatus ippiFilterGetBufSize_64f_C1R(IppiSize kernelSize, int roiWidth, int* pSize);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`kernelSize`	Size of the rectangular kernel in pixels.
`roiWidth`	Width of the image ROI in pixels.
`pSize`	Pointer to the size of the work buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function computes the size of the work buffer `pSize` that is required for the function ippiFilter (flavor that operates on data of the `Ipp64f` type).

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error if `pSize` is `NULL`.
`ippStsSizeErr`	Indicates an error if `kernelSize` has a field with a zero or negative value, or `roiWidth` is less than or equal to zero.

Filter

Filters an image using a general rectangular kernel.

Syntax

```
IppStatus ippiFilter_64f_C1R(const Ipp64f* pSrc, int srcStep, Ipp64f* pDst, int dstStep, IppiSize dstRoiSize, const Ipp64f* pKernel, IppiSize kernelSize, IppiPoint anchor, Ipp8u* pBuffer);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance, in bytes, between the starting points of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance, in bytes, between the starting points of consecutive lines in the destination image.
`dstRoiSize`	Size of the source and destination ROI in pixels.
`pKernel`	Pointer to the kernel values.
`kernelSize`	Size of the rectangular kernel in pixels.

anchor	Anchor cell specifying the rectangular kernel alignment with respect to the position of the input pixel.
pBuffer	Pointer to the work buffer.

Description

This function operates with ROI.

The ippiFilter function uses the general rectangular kernel of kernelSize size to filter an image ROI. This function sums the products of the kernel coefficients pKernel and pixel values taken over the source pixel neighborhood defined by kernelSize and anchor. The anchor cell is specified by its coordinates anchor.x and anchor.y in the coordinate system associated with the bottom right corner of the kernel.

Kernel coefficients are used in inverse order. The sum is written to the destination pixel.

To ensure valid operation when image boundary pixels are processed, the application should correctly define additional border pixels (see Borders in Neighborhood Operations).

This function requires a temporary work buffer. Before using this function flavor, you need to compute the buffer size using the ippiFilterGetBufSize function.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error when pSrc, pDst, or pKernel is NULL.
ippStsSizeErr	Indicates an error when dstRoiSize or kernelSize has a field with a zero or negative value.
ippStsDivisorErr	Indicates an error when the divisor has a zero value.
ippStsStepErr	Indicates an error when:

See Also

Regions of Interest in Intel IPP

Borders in Neighborhood Operations

FilterGetBufSize Computes the size of the work buffer.

Separable Filters

Separable filters use a spatial kernel consisting of a single column (as in the FilterColumn function) or a single row (as in the FilterRow function) to filter the source image.

- FilterRowBorderPipelineGetBufferSize, FilterRowBorderPipelineGetBufferSize_Low
- FilterRowBorderPipeline, FilterRowBorderPipeline_Low
- FilterColumnPipelineGetBufferSize, FilterColumnPipelineGetBufferSize_Low
- FilterColumnPipeline, FilterColumnPipeline_Low
- FilterSeparable
- FilterSeparableGetBufferSize

- FilterSeparableGetSpecSize
- FilterSeparableInit

FilterRowBorderPipelineGetBufferSize, FilterRowBorderPipelineGetBufferSize_Low

Compute the size of working buffer for the strow filter.

Syntax

```
IppStatus ippiFilterRowBorderPipelineGetBufferSize_<mod>(IppiSize roiSize, int kernelSize, int* pBufferSize);
```

Supported values for `mod`:

`8u16s_C1R`	`16s_C1R`	`16u_C1R`	`32f_C1R`
`8u16s_C3R`	`16s_C3R`	`16u_C3R`	`32f_C3R`

```
IppStatus ippiFilterRowBorderPipelineGetBufferSize_Low_<mod>(IppiSize roiSize, int kernelSize, int* pBufferSize);
```

Supported values for `mod`:

`16s_C1R`
`16s_C3R`

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`roiSize`	Maximum size of the source and destination image ROI.
`kernelSize`	Size of the kernel in pixels.
`pBufferSize`	Pointer to the computed size of the buffer.

Description

These functions operate with ROI (see Regions of Interest in Intel IPP).

These functions compute the size of the working buffer required for the functions `ippiFilterRowBorderPipeline` and `ippiFilterRowBorderPipeline_Low` respectively. The buffer with the length `pBufferSize[0]` can be used to filter images with width equal to or less than `roiSize`.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if the pointer pBufferSize is NULL.
ippStsSizeErr	Indicates an error condition if maskSize has a field with a zero or negative value, or if roiWidth is less than 1.

FilterRowBorderPipeline, FilterRowBorderPipeline_Low

Apply the filter with border to image rows.

Syntax

Case 1: Operation on one-channel integer data

```
IppStatus ippiFilterRowBorderPipeline_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>** ppDst, IppiSize roiSize, const Ipp<dstDatatype>* pKernel, int
kernelSize, int xAnchor, IppiBorderType borderType, Ipp<srcDatatype> borderValue, int
divisor, Ipp8u* pBuffer);
```

Supported values for mod:

8u16s_C1R	16s_C1R	16u_C1R

```
IppStatus ippiFilterRowBorderPipeline_Low_16s_C1R(const Ipp16s* pSrc, int srcStep,
Ipp16s** ppDst, IppiSize roiSize, const Ipp16s* pKernel, int kernelSize, int xAnchor,
IppiBorderType borderType, Ipp16s borderValue, int divisor, Ipp8u* pBuffer);
```

Case 2: Operation on one-channel floating point data

```
IppStatus ippiFilterRowBorderPipeline_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f**
ppDst, IppiSize roiSize, const Ipp32f* pKernel, int kernelSize, int xAnchor,
IppiBorderType borderType, Ipp32f borderValue, Ipp8u* pBuffer);
```

Case 3: Operation on three-channel integer data

```
IppStatus ippiFilterRowBorderPipeline_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>** ppDst, IppiSize roiSize, const Ipp<dstDatatype>* pKernel, int
kernelSize, int xAnchor, IppiBorderType borderType, Ipp<srcDatatype> borderValue[3],
int divisor, Ipp8u* pBuffer);
```

Supported values for mod:

8u16s_C3R	16s_C3R	16u_C3R

```
IppStatus ippiFilterRowBorderPipeline_Low_16s_C3R(const Ipp16s* pSrc, int srcStep,
Ipp16s** ppDst, IppiSize roiSize, const Ipp16s* pKernel, int kernelSize, int xAnchor,
IppiBorderType borderType, Ipp16s borderValue[3], int divisor, Ipp8u* pBuffer);
```

Case 4: Operation on three-channel floating point data

```
IppStatus ippiFilterRowBorderPipeline_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f**
ppDst, IppiSize roiSize, const Ipp32f* pKernel, int kernelSize, int xAnchor,
IppiBorderType borderType, Ipp32f borderValue[3], Ipp8u* pBuffer);
```

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
ppDst	Double pointer to the destination inage ROI.
roiSize	Size of the source and destination ROI in pixels.
pKernel	Pointer to the row kernel values.
kernelSize	Size of the kernel in pixels.
xAnchor	Anchor value specifying the kernel row alignment with respect to the positionof the input pixel.
borderType	Type of border (see Borders in Neighborhood Operations); following values are possible: • ippBorderZero • Values of all border pixel are set to zero. • ippBorderConst • Values of all border pixels are set to constant. • ippBorderRepl • Replicated border is used. • ippBorderWrap • Wrapped border is used • ippBorderMirror • Mirrored border is used • ippBorderMirrorR • Mirrored border with replication is used
borderValue	The constant value (constant vector in case of three-channel data) to assign to the pixels in the constant border (not applicable for other border's type).
divisor	Value by which the computed result is divided (for operations on integer data only).
pBuffer	Pointer to the working buffer.

Description

These functions operate with ROI (see Regions Regions of Interest in Intel IPP).

The function ippiFilterRowBorderPipeline_Low performs calculation exclusively with the 16s-data, and the input data must be in the range ensuring that the overflow does not occur during calculation and the result can be represented by a 32-bit integer number.

These functions apply the horizontal row filter of the separable convolution kernel to the source image pSrc. The filter coefficients are placed in the reversed order. For integer data:

$$ppDst[i][j] = \frac{1}{divisor} \cdot \sum_{k=0}^{kernelSize-1} pSrc[i, j+k-xAnchor] \cdot pKernel[k]$$

and for floating point data:

$$ppDst[i][j] = \sum_{k=0}^{kernelSize-1} pSrc[i, j+k-xAnchor] \cdot pKernel[k]$$

Here j = 0, ...roiSize.width- 1,i = 0,...roiSize.height- 1. The values of pixels of the source image that lies outside of the image ROI (that is, if for pixel pSrc[i,l] l ∉ [0,roiSize.width-1] are set in accordance with the specified parameters borderType and borderValue.

This function can be used to organize the separable convolution as a step of the image processing pipeline.

The functions requires the external buffer pBuffer, its size should be previously computed by the functions ippiFilterRowBorderPipelineGetBufferSize and ippiFilterRowBorderPipelineGetBufferSize_Low respectively

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with a zero or negative value.
ippStsStepErr	Indicates an error condition if srcStep or dstStep is less than roiSize.width* <pixelSize>
ippStsNotEvenStepErr	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
ippStsAnchorErr	Indicates an error condition if xAnchor has a wrong value.
ippStsBorderErr	Indicates an error condition if borderType has a wrong value.
ippStsBadArgErr	Indicates an error condition if divisor is equal to 0.

FilterColumnPipelineGetBufferSize, FilterColumnPipelineGetBufferSize_Low

Compute the size of working buffer for the column filter.

Syntax

```
IppStatus ippiFilterColumnPipelineGetBufferSize_<mod>(IppiSize roiSize, int kernelSize, int* pBufferSize);
```

Supported values for `mod`:

`16s_C1R`	`16u_C1R`	`16s8u_C1R`	`16s8s_C1R`	`32f_C1R`
`16s_C3R`	`16u_C3R`	`16s8u_C3R`	`16s8s_C3R`	`32f_C3R`

```
IppStatus ippiFilterColumnPipelineGetBufferSize_Low_<mod>(IppiSize roiSize, int kernelSize, int* pBufferSize);
```

Supported values for `mod`:

`16s_C1R`
`16s_C3R`

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`roiSize`	Maximum size of the source and destination image ROI.
`kernelSize`	Size of the kernel in pixels.
`pBufferSize`	Pointer to the computed size of the buffer.

Description

These functions operate with ROI (see Regions of Interest in Intel IPP).

These functions compute the size of the working buffer required for the functions `ippiFilterColumnPipeline` and `ippiFilterColumnPipeline_Low` respectively. The buffer with the length `pBufferSize[0]` can be used to filter images with width equal to or less than `roiSize`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if the pointer `pBufferSize` is `NULL`.

ippStsSizeErr	Indicates an error condition if maskSize has a field with a zero or negative value, or if roiWidth is less than 1.

FilterColumnPipeline, FilterColumnPipeline_Low

Apply the filter to image columns.

Syntax

Case 1: Operation on integer data

```
IppStatus ippiFilterColumnPipeline_<mod>(const Ipp<srcDatatype>** ppSrc,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize, const Ipp<srcDatatype>* pKernel,
int kernelSize, int divisor, Ipp8u* pBuffer);
```

Supported values for mod:

16s_C1R	16s8u_C1R	16s8s_C1R	16u_C1R
16s_C3R	16s8u_C3R	16s8s_C3R	16u_C3R

```
IppStatus ippiFilterColumnPipeline_Low_16s_C1R(const Ipp16s** ppSrc, Ipp16s* pDst, int
dstStep, IppiSize roiSize, const Ipp16s* pKernel, int kernelSize, int divisor, Ipp8u*
pBuffer);
```

```
IppStatus ippiFilterColumnPipeline_Low_16s_C3R(const Ipp16s** ppSrc, Ipp16s* pDst, int
dstStep, IppiSize roiSize, const Ipp16s* pKernel, int kernelSize, int divisor, Ipp8u*
pBuffer);
```

Case 2: Operation on floating-point data

```
IppStatus ippiFilterColumnPipeline_<mod>(const Ipp<datatype>** ppSrc, Ipp<datatype>*
pDst, int dstStep, IppiSize roiSize, const Ipp<datatype>* pKernel, int kernelSize,
Ipp8u* pBuffer);
```

Supported values for mod:

32f_C1R
32f_C3R

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

ppSrc	Double pointer to the source image ROI.
pDst	Pointer to the destination image ROI.

dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the destination ROI in pixels.
pKernel	Pointer to the strow kernel values.
kernelSize	Size of the kernel in pixels.
divisor	Value by which the computed result is divided (for operations on integer data only).
pBuffer	Pointer to the working buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

The function ippiFilterColumnPipeline_Low performs calculation exclusively with the 16s-data, and the input data must be in the range ensuring that the overflow does not occur during calculation and the result can be represented by a 32-bit integer number.

These functions apply the column filter of the separable convolution kernel to the source image pSrc. The filter coefficients are placed in the reversed order. For integer data:

$$pDst[i,j] = \frac{1}{divisor} \cdot \sum_{k=0}^{kernelSize-1} pSrc[i+k,j] \cdot pKernel[k]$$

and for floating point data:

$$pDst[i,j] = \sum_{k=0}^{kernelSize-1} pSrc[i+k,j] \cdot pKernel[k]$$

Here j = 0, ...roiSize.width-1,i=0,...roiSize.height-1.

The size of the source image is

(roiSize.height+kernelSize- 1) *roiSize.width.

The functions requires the external buffer pBuffer, its size should be previously computed by the functions ippiFilterColumnPipelineGetBufferSize and ippiFilterColumnPipelineGetBufferSize_Low respectively.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with a zero or negative value.

ippStsStepErr	Indicates an error condition if srcStep or dstStep is less than roiSize.width * <pixelSize>
ippStsNotEvenStepErr	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
ippStsBadArgErr	Indicates an error condition if divisor is equal to 0.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

//   An example of performing The code example shows how these functions can be used to organize
the separable convolution as a step of
// image processing pipeline.implemented with Intel(R) Integrated Performance Primitives
(Intel(R) IPP) functions :
//     ippiFilterRowBorderPipelineGetBufferSize_Low_16s_C1R
//     ippiFilterColumnPipeline_Low_16s_C1R
//     ippiFilterColumnPipelineGetBufferSize_Low_16s_C1R
//     ippiFilterRowBorderPipeline_Low_16s_C1R

#include <stdio.h>
#include "ipp.h"

#define WIDTH  128 /* image width  */
#define HEIGHT 64  /* image height */

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define check_sts(st)        \
    if ((st) != ippStsNoErr) \
        goto exitLine; /* Go to Exit if Intel(R) IPP function returned status different from
ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main(void)
{
    IppStatus status = ippStsNoErr;
    const Ipp16s **pGet = NULL;
```

```
    Ipp16s *src = NULL, *dst = NULL;
    int kernelSize = 3;
    int xAnchor = kernelSize >> 1;                      /* The anchor value, (0 <= xAnchor <
kernelSize) */
    Ipp16s pKerX[3] = {1, 2, 1}, pKerY[3] = {1, 0, 1}; /* The pointer to the kernels */
    Ipp16s *pSrc = NULL, *pDst = NULL;                  /* Pointers to source/destination images
*/
    int srcStep = 0, dstStep = 0;                       /* Steps, in bytes, through the source/
destination images */
    IppiSize roiSize = {WIDTH, HEIGHT};                 /* Size of source/destination ROI in
pixels */
    IppiSize roi = {WIDTH, 1};                          /* Size of destination ROI in pixels */
    int divisor = 1;                                    /* The value to divide output pixels by */
    Ipp8u borderValue = 0;
    Ipp8u *pBufRow = NULL, *pBufCol = NULL; /* Pointer to the work buffer */
    int sizeRow = 0, sizeCol = 0;           /* Common work buffer size */
    int todo = roiSize.height, bufLen;
    int mStep = (roiSize.width + 7) & (~7);
    int sStep = 0, dStep = 0;
    bufLen = mStep * 3 * sizeof(Ipp16s) + 4 * sizeof(Ipp16s *);
    pGet = (const Ipp16s **)ippsMalloc_8u(bufLen);
    pSrc = ippiMalloc_16s_C1(roiSize.width, roiSize.height, &srcStep);
    pDst = ippiMalloc_16s_C1(roiSize.width, roiSize.height, &dstStep);

    dst = pDst;
    src = pSrc;
    sStep = srcStep >> 1, dStep = dstStep >> 1;
    pGet[0] = pGet[1] = (Ipp16s *)(pGet + 4);
    pGet[2] = pGet[1] + mStep;
    pGet[3] = pGet[2] + mStep;

    check_sts(status = ippiFilterRowBorderPipelineGetBufferSize_Low_16s_C1R(roiSize, 3,
&sizeRow));

    check_sts(status = ippiFilterColumnPipelineGetBufferSize_Low_16s_C1R(roiSize, 3, &sizeCol));

    pBufRow = ippsMalloc_8u(sizeRow);
    pBufCol = ippsMalloc_8u(sizeCol);

    check_sts(status = ippiFilterRowBorderPipeline_Low_16s_C1R(pSrc, srcStep, (Ipp16s **)pGet,
roi, pKerX, kernelSize, xAnchor, ippBorderRepl,
                                                           borderValue, divisor, pBufRow));

    todo--;
    if (todo == 0) {
        pGet[2] = pGet[0];
    } else {
        pGet[2] = pGet[0] + mStep;
        pGet[3] = pGet[2] + mStep;
        for (; todo > 0; src += sStep, dst += dStep, todo--) {
            check_sts(status = ippiFilterRowBorderPipeline_Low_16s_C1R(src, srcStep, (Ipp16s **)
(pGet + 2), roi, pKerX, kernelSize, xAnchor,
                                                                ippBorderRepl,
borderValue, divisor, pBufRow));

            check_sts(status = ippiFilterColumnPipeline_Low_16s_C1R(pGet, dst, dstStep, roi,
pKerY, kernelSize, divisor, pBufCol));
```

```
            pGet[0] = pGet[1];
            pGet[1] = pGet[2];
            pGet[2] = pGet[3];
            pGet[3] = pGet[0];
        }
    }
    check_sts(status = ippiFilterColumnPipeline_Low_16s_C1R(pGet, dst, dstStep, roi, pKerY,
kernelSize, divisor, pBufCol));

    EXIT_MAIN
    ippiFree(pSrc);
    ippsFree(pBufRow);
    ippsFree(pBufCol);
    ippiFree(pDst);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

FilterSeparable

Apply the filter to an image.

Syntax

```
IppStatus ippiFilterSeparable_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType,
Ipp<datatype> borderValue, const IppiFilterSeparableSpec* pSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

8u_C1R	16s_C1R	16u_C1R	32f_C1R
8u_C3R	16s_C3R	16u_C3R	32f_C3R
8u_C4R	16s_C4R	16u_C4R	32f_C4R
8u_C1R_T	16s_C1R_T	16u_C1R_T	32f_c1R_T
8u_C3R_T	16s_C3R_T	16u_C3R_T	32f_C3R_T
8u_C4R_T	16s_C4R_T	16u_C4R_T	32f_C4R_T

NOTE `Src` and `Dst` have the same data types for these modes.

```
IppStatus ippiFilterSeparable_<mod>(const Ipp<srcdatatype>* pSrc, int srcStep,
Ipp<dstdatatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType,
Ipp<srcdatatype> borderValue, const IppiFilterSeparableSpec* pSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

8u16s_C1R
8u16s_C3R
8u16s_C4R

8u16s_C1R_T
8u16s_C3R_T
8u16s_C4R_T

NOTE `Src` and `Dst` have different data types for these modes.

Platform-aware functions

```
IppStatus ippiFilterSeparable_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep,
Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType,
Ipp<datatype> borderValue, const IppiFilterSeparableSpec* pSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

8u_C1R_L	16s_C1R_L	16u_C1R_L	32f_C1R_L
8u_C3R_L	16s_C3R_L	16u_C3R_L	32f_C3R_L
8u_C4R_L	16s_C4R_L	16u_C4R_L	32f_C4R_L
8u_C1R_LT	16s_C1R_LT	16u_C1R_LT	32f_C1R_LT
8u_C3R_LT	16s_C3R_LT	16u_C3R_LT	32f_C3R_LT
8u_C4R_LT	16s_C4R_LT	16u_C4R_LT	32f_C4R_LT

NOTE `Src` and `Dst` have the same data types for these modes.

```
IppStatus ippiFilterSeparable_<mod>(const Ipp<srcdatatype>* pSrc, IppSizeL srcStep,
Ipp<dstdatatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType,
Ipp<srcdatatype> borderValue, const IppiFilterSeparableSpec* pSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

8u16s_C1R_L
8u16s_C3R_L
8u16s_C4R_L
8u16s_C1R_LT
8u16s_C3R_LT
8u16s_C4R_LT

NOTE `Src` and `Dst` have different data types for these modes.

Include Files

`ippcv.h`

ippcv_l.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecuitve lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.
borderType	Type of border. Possible values are: • ippBorderConst • Values of all border pixels are set to constant. • ippBorderRept • Replicated border is used. • ippBorderWrap • Wrapped border is used. • ippBorderMirror • Mirrored border is used. • ippBorderMirrorR • Mirrored border with replication is used.
borderValue	Constant value (constant vector in case of three- or four-channel data) to assign to the pixels in the ippBorderConst border type (not applicable for other border types).
pSpec	Pointer to the filter specification structure.
pBuffer	Pointer to the working buffer.

Description

This function operates with ROI.

Before using this function, compute the size of the external buffer pBuffer using the ippiFilterSeparableGetBufferSize function.

This function applies the horizontal row filter of the separable convolution kernel to the source image pSrc and the column filter of the separable convolution kernel to the intermediate result.

For integer data:

$$intermediate[i][j] = \frac{1}{divisor} * \sum_{k=0}^{rowKernelSize-1} (\mathrm{pSrc[i,j+k-xAnchor]} * \mathrm{pRowKernel[k]})$$

$$pDst[i,j] = offset + \frac{1}{divisor} * \sum_{k=0}^{columnKernelSize-1} (\text{intermediate}[i+k, j] * \text{pColumnKernel}[k])$$

and for floating point data:

$$intermediate[i][j] = \sum_{k=0}^{rowKernelSize-1} (\text{pSrc}[i, j+k-\text{xAnchor}] * \text{pRowKernel}[k])$$

$$pDst[i,j] = \sum_{k=0}^{columnKernelSize-1} (\text{intermediate}[i+k, j] * \text{pColumnKernel}[k])$$

Here `j= 0, ...roiSize.width- 1,i= 0,...roiSize.height- 1`. The values of pixels of the source image that lies outside of the image ROI (that is, if for pixel `pSrc[i,l] l ∉ [0,roiSize.width-1]` are set in accordance with the specified parameters `borderType` and `borderValue`. `xAnchor= (rowKernelSize- 1) / 2`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error condition.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize` has a field with a zero or negative value.
`ippStsStepErr`	Indicates an error condition if the `srcStep` or `dstStep` value is less than `roiSize.width` * `<pixelSize>`.
`ippStsNotEvenStepErr`	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.

See Also

Regions of Interest in Intel IPP

User-defined Border Types

FilterSeparableInit Initializes the filter specification structure.

FilterSeparableGetBufferSize Computes the size of the work buffer.

FilterSeparableGetBufferSize

Computes the size of the work buffer.

Syntax

```
IppStatus ippiFilterSeparableGetBufferSize(IppiSize roiSize, IppiSize kernelSize,
IppDataType dataType, IppDataType kernelType, int numChannels, int* pBufferSize);
```

```
IppStatus ippiFilterSeparableGetBufferSize_L(IppiSizeL roiSize, IppiSize kernelSize,
IppDataType dataType, IppDataType kernelType, int numChannels, IppSizeL* pBufferSize);

IppStatus ippiFilterSeparableGetBufferSize_T(IppiSize roiSize, IppiSize kernelSize,
IppDataType dataType, IppDataType kernelType, int numChannels, int* pBufferSize);

IppStatus ippiFilterSeparableGetBufferSize_LT(IppiSizeL roiSize, IppiSize kernelSize,
IppDataType dataType, IppDataType kernelType, int numChannels, IppSizeL* pBufferSize);
```

Include Files

`ippcv.h`

`ippcv_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`roiSize`	Size of the image ROI in pixels.
`kernelSize`	Size of the rectangular kernel in pixels.
`dataType`	Data type of the source image. Possible values are `Ipp8u`, `Ipp16s`, `Ipp16u`, `Ipp32f`.
`kernelType`	Data type of the filter kernel. Possible values are `Ipp16s` and `Ipp32f`.
`numChannels`	Number of channels in the image. Possible values are 1, 3, and 4.
`pBufferSize`	Pointer to the size of the work buffer required for filtering.

Description

This function computes the size of the buffer required for filtering operations. Call this function before using the ippiFilterSeparable function.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `kernelSize` has a field with a zero or negative value.
`ippStsDataTypeErr`	Indicates an error condition if `dataType` has an illegal value.

ippStsChannelErr	Indicates an error condition if numChannels has an illegal value.

See Also

FilterSeparable Apply the filter to an image.

FilterSeparableGetSpecSize

Computes the size of the filter specification structure.

Syntax

```
IppStatus ippiFilterSeparableGetSpecSize(IppiSize kernelSize, IppDataType dataType, int numChannels, int* pSpecSize);
IppStatus ippiFilterSeparableGetSpecSize_L(IppiSize kernelSize, IppDataType dataType, int numChannels, int* pSpecSize);
IppStatus ippiFilterSeparableGetSpecSize_T(IppiSize kernelSize,IppDataType dataType, int numChannels, int* pSpecSize);
IppStatus ippiFilterSeparableGetSpecSize_LT(IppiSize kernelSize,IppDataType dataType, int numChannels, int* pSpecSize);
```

Include Files

ippcv.h

ippcv_l.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

kernelSize	Size of the rectangular kernel in pixels.
dataType	Data type of the source image. Possible values are Ipp16s, Ipp32f.
numChannels	Number of channels in the image. Possible values are 1, 3, and 4.
pSpecSize	Pointer to the size of the filter specification structure.

Description

This function computes the size of the filter specification structure pSpec. Call this function before using the ippiFilterSeparableInit function.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if kernelSize has a field with a zero or negative value.
ippStsDataTypeErr	Indicates an error condition if dataType has an illegal value.
ippStsChannelErr	Indicates an error condition if numChannels has an illegal value.

See Also

FilterSeparableInit Initializes the filter specification structure.

FilterSeparableInit

Initializes the filter specification structure.

Syntax

```
IppStatus ippiFilterSeparableInit_16s(const Ipp16s* pRowKernel, const Ipp16s*
pColumnKernel, IppiSize kernelSize, int divisor, int scaleFactor, IppDataType dataType,
int numChannels, IppiFilterSeparableSpec* pSpec);
```

```
IppStatus ippiFilterSeparableInit_32f(const Ipp32f* pRowKernel, const Ipp32f*
pColumnKernel, IppiSize kernelSize, IppDataType dataType, int numChannels,
IppiFilterSeparableSpec* pSpec);
```

Platform-aware functions

```
IppStatus ippiFilterSeparableInit_16s_L(const Ipp16s* pRowKernel, const Ipp16s*
pColumnKernel, IppiSize kernelSize, int divisor, int scaleFactor, IppDataType dataType,
int numChannels, IppiFilterSeparableSpec* pSpec);
```

```
IppStatus ippiFilterSeparableInit_32f_L(const Ipp32f* pRowKernel, const Ipp32f*
pColumnKernel, IppiSize kernelSize, IppDataType dataType, int numChannels,
IppiFilterSeparableSpec* pSpec);
```

```
IppStatus ippiFilterSeparableInit_16s_T(const Ipp16s* pRowKernel, const Ipp16s*
pColumnKernel, IppiSize kernelSize, int divisor, int scaleFactor, IppDataType dataType,
int numChannels, IppiFilterSeparableSpec* pSpec);
```

```
IppStatus ippiFilterSeparableInit_32f_T(const Ipp32f* pRowKernel, const Ipp32f*
pColumnKernel, IppiSize kernelSize, IppDataType dataType, int numChannels,
IppiFilterSeparableSpec* pSpec);
```

```
IppStatus ippiFilterSeparableInit_16s_LT(const Ipp16s* pRowKernel, const Ipp16s*
pColumnKernel, IppiSize kernelSize, int divisor, int scaleFactor, IppDataType dataType,
int numChannels, IppiFilterSeparableSpec* pSpec);
```

```
IppStatus ippiFilterSeparableInit_32f_LT(const Ipp32f* pRowKernel, const Ipp32f*
pColumnKernel, IppiSize kernelSize, IppDataType dataType, int numChannels,
IppiFilterSeparableSpec* pSpec);
```

Include Files

ippcv.h

ippcv_l.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pRowKernel, pColumnKernel	Pointer to the row and column kernel values.
kernelSize	Size of the rectangular kernel in pixels.
divisor	Integer value by which the computed result is divided.
scaleFactor	Integer value by which the computed result is divided.
dataType	Data type of the source image. Possible values are Ipp8u, Ipp16s, Ipp16u, Ipp32f.
numChannels	Number of channels in the image. Possible values are 1, 3, and 4.
pSpec	Pointer to the filter specification structure.

Description

This function initializes the filter specification structure pSpec in the external buffer. Before using this function, you need to compute the size of the specification structure using the FilterSeparableGetSpecSize function. This structure is used by the FilterSeparable function that performs filtering operations on the source image pixels.

NOTE Not all ippiFilterSeparable_* functions are compatible with ippiFilterSeparableInit_*. For more information about compatibility, see the table below.

ippiFilterSeparableInit_*	Compatible ippiFilterSeparable_* Functions
ippiFilterSeparableInit_16s	8u_C1R, 8u_C3R, 8u_C4R, 16s_C1R, 16s_C3R, 16s_C4R, 8u16s_C1R, 8u16s_C3R, 8u16s_C4R,
ippiFilterSeparableInit_32f	16u_C1R, 16u_C3R, 16u_C4R, 32f_C1R, 32f_C3R, 32f_C4R
ippiFilterSeparableInit_16s_L|16s_LT	8u_C1R_L, 8u_C3R_L, 8u_C4R_L, 16s_C1R_L, 16s_C3R_L, 16s_C4R_L, 8u16s_C1R_L, 8u16s_C3R_L, 8u16s_C4R_L,
ippiFilterSeparableInit_32f_L|32f_LT	16u_C1R_L, 16u_C3R_L, 16u_C4R_L, 32f_C1R_L, 32f_C3R_L, 32f_C4R_L

ippiFilterSeparableInit_16s_T	8u_C1R_T, 8u_C3R_T, 8u_C4R_T, 16s_C1R_T, 16s_C3R_T, 16s_C4R_T, 8u16s_C1R_T, 8u16s_C3R_T, 8u16s_C4R_T
ippiFilterSeparableInit_32f_T	16u_C1R_T, 16u_C3R_T, 16u_C4R_T, 32f_C1R_T, 32f_C3R_T, 32f_C4R_T

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if kernelSize has a field with a zero or negative value.
ippStsDataTypeErr	Indicates an error condition if dataType has an illegal value.
ippStsChannelErr	Indicates an error condition if numChannels has an illegal value.
ippStsDivisorErr	Indicates an error condition if divisor has a zero value.

See Also

FilterSeparable Apply the filter to an image.

FilterSeparableGetSpecSize Computes the size of the filter specification structure.

Smoothing Filters

Smoothing filters use edge-preserving real-time Iterative Least Squares algorithm for image processing. This algorithm is described in the article "Real-time Image Smoothing via Iterative Least Squares. Wei Liu, Pingping Zhang, Xiaolin Huang, Jie Yang, Chunhua Shen, Ian Reid". Link to the publication: https://arxiv.org/abs/2003.07504.

- FilterILSInit
- FilterILSGetBufferSize
- FilterILS

FilterILSInit

Initializes the filter specification structure.

Syntax

```
IppStatus ippiFilterILSInit (IppiFilterILSType filter, IppiSize dstRoiSize, IppDataType dataType, int numChannels, Ipp64f lambda, Ipp64f eps, Ipp64f pow, Ipp64f gamma, IppiFilterILSSpec* pSpec, Ipp8u* pBufInit);
```

Include Files

ippi.h

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`filter`	Filter type: `Norm` or `Welsch`.
`dataType`	Data type flavors.
`dstRoiSize`	Size of the source and destination ROI in pixels.
`numChannels`	Number of channels.
`pSpecSize`	Pointer to the size (in bytes) of the specification structure.
`pBufInitSize`	Pointer to the size (in bytes) of the init temp buffer.
`pBufferSize`	Pointer to the size (in bytes) of the work buffer.
`lambda`	Lambda for `Norm` and `Welsch`.
`eps`	Eps for `Norm`.
`pow`	P for `Norm`.
`gamma`	Gamma for `Welsch`.
`pSpec`	Pointer to the specification structure.
`pBufInit`	Pointer to the buffer for init function.

Description

This function initializes the filter specification structure `pSpec` in the work buffer.

Before using this function, you need to compute the size of the specification structure using the `ippiFilterILSGetBufferSize` function. This structure is used by the `ippiFilterILS` function that performs edge-preserving image smoothing via Iterative Least Squares algorithm.

The `ippiFilterILSInit` function takes the following parameters: `lambda`, `eps`, `p`, `gamma` with values that are described in the article “Real-time Image Smoothing via Iterative LeastSquares. Wei Liu, Pingping Zhang,Xiaolin Huang, Jie Yang, Chunhua Shen, Ian Reid”. Link to the publication: https://arxiv.org/abs/2003.07504.

Use the `ippiFilterILSGetBufferSize` function to allocate the work buffer `pBuffer` and `pBufInit`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsStepErr`	Indicates an error condition if one of the steps has a zero or negative value.

ippStsSizeErr	Indicates an error condition if one of the fields of dstRoiSize has a zero or negative value.

FilterILSGetBufferSize

Computes the size of the work buffer.

Syntax

```
IppStatus ippiFilterILSGetBufferSize (IppiFilterILSType filter, IppiSize dstRoiSize, IppDataType dataType, int numChannels, int* pSpecSize, int* pBufInitSize, int* pBufferSize);
```

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

filter	Filter type: Norm or Welsch.
dataType	Data type flavors.
dstRoiSize	Size of the source and destination ROI in pixels.
numChannels	Number of channels.
pSpecSize	Pointer to the size (in bytes) of the specification structure.
pBufInitSize	Pointer to the size (in bytes) of the init temp buffer.
pBufferSize	Pointer to the size (in bytes) of the work buffer.

Description

This function computes the size of the buffers required for filtering operations. Call this function before using the ippiFilterILSInit and ippiFilterILS functions.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if any of the specified pointers is NULL.
ippStsStepErr	Indicates an error condition if one of the steps has a zero or negative value.

ippStsSizeErr	Indicates an error condition if one of the fields of dstRoiSize has a zero or negative value.

FilterILS

Filters an image using smoothing algorithm.

Syntax

```
IppStatus ippiFilterILS_<mod>(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep,
IppiSize dstRoiSize, int nIter, IppiFilterILSSpec* pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u_C1R
8u_C3R

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Step through the source image.
pDst	Pointer to the destination image ROI.
dstStep	Step in destination image.
dstRoiSize	Size of the source and destination ROI in pixels.
nIter	Number of iterations.
pSpec	Pointer to the internal data structure.
pBuffer	Pointer to the external work buffer.

Description

This function implements edge-preserving image smoothing via Iterative Least Squares algorithm.

This algorithm is described in the article “Real-time Image Smoothing via Iterative LeastSquares. Wei Liu, Pingping Zhang,Xiaolin Huang, Jie Yang, Chunhua Shen, Ian Reid”. Link to the publication: https://arxiv.org/abs/2003.07504.

The function ippiFilterILS uses the internal data structure pSpec and the work buffer pBuffer. Therefore, the work buffer pBuffer must be allocated and the internal data structure pSpec must be initialized before the function call.

Use the ippiFilterILSGetBufferSize function to allocate the work buffer pBuffer.

Use the ippiFilterILSInit function to initialize the internal data structure pSpec.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if any of the specified pointers is NULL.
ippStsStepErr	Indicates an error condition if one of the steps has a zero or negative value.
ippStsSizeErr	Indicates an error condition if one of the fields of dstRoiSize has a zero or negative value.

Wiener Filters

Intel IPP functions described in this section perform adaptive noise-removal filtering of an image using Wiener filter Lim90. The adaptive filter is more selective than a comparable linear filter in preserving edges and other high frequency parts of an image. Wiener filters are commonly used in image processing applications to remove additive noise from degraded images, to restore a blurry image, and in similar operations.

These functions use a pixel-wise adaptive Wiener method based on statistics estimated from a local neighborhood (mask) of arbitrary size for each pixel.

- FilterWienerGetBufferSize
- FilterWiener

FilterWienerGetBufferSize

Computes the size of the external buffer for ippiFilterWiener function.

Syntax

```
IppStatus ippiFilterWienerGetBufferSize(IppiSize dstRoiSize, IppiSize maskSize, int channels, int* pBufferSize);
```

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

dstRoiSize	Size of the destination ROI in pixels.
maskSize	Size of the mask in pixels.
channels	Number of channels in the image.

pBufferSize	Pointer to the computed value of the external buffer size.

Description

This function computes the size in bytes of an external memory buffer that is required for the function ippiFilterWiener, and stores the result in the pBufferSize.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if the pBufferSize pointer is NULL.
ippStsSizeErr	Indicates an error condition if one of the fields of dstRoiSize has a zero or negative value.
ippStsMaskSizeErr	Indicates an error condition if one of the fields of maskSize has a value less than or equal to 1.
ippStsNumChannelsErr	Indicates an error condition if channels is not 1, 3 or 4.

FilterWiener

Filters an image using the Wiener algorithm.

Syntax

Case 1: Operation on one-channel images

```
IppStatus ippiFilterWiener_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiPoint anchor, Ipp32f
noise[1], Ipp8u* pBuffer);
```

Supported values for mod:

8u_C1R	16s_C1R	32f_C1R

Case 2: Operation on multi-channel images

```
IppStatus ippiFilterWiener_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiPoint anchor, Ipp32f
noise[3], Ipp8u* pBuffer);
```

Supported values for mod:

8u_C3R	16s_C3R	32f_C3R
8u_AC4R	16s_AC4R	32f_AC4R

```
IppStatus ippiFilterWiener_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize dstRoiSize, IppiSize maskSize, IppiPoint anchor, Ipp32f
noise[4], Ipp8u* pBuffer);
```

Supported values for mod:

8u_C4R	16s_C4R	32f_C4R

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
dstRoiSize	Size of the source and destination ROI in pixels.
maskSize	Size of the mask in pixels.
anchor	Anchor cell specifying the mask alignment with respect to the position of the input pixel.
noise	Noise level value or array of the noise level values in case of multi-channel image. This value must be in the range [0,1].
pBuffer	Pointer to the external work buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP). This function performs adaptive filtering of the image degraded by constant power additive noise. For each pixel of the input image pSrc, the function estimates the local image mean μ and variance σ in the rectangular neighborhood (mask) of size maskSize with the anchor cell anchor centered on the pixel. The anchor cell is specified by its coordinates anchor.x and anchor.y in the coordinate system associated with the bottom right corner of the mask.

The following formulas are used in computations:

$$\mu_{i,j} = \frac{1}{HW} \cdot \sum_{m=0}^{H-1} \sum_{n=0}^{W-1} x_{m,n}$$

$$\sigma_{i,j}^{2} = \frac{1}{HW} \cdot \sum_{m=0}^{H-1} \sum_{n=0}^{W-1} x_{m,n}^{2} - \mu_{i,j}^{2}$$

Here $\mu_{i,j}$ and $\sigma_{i,j}$ stand for local mean and variance for pixel $X_{i,j}$, respectively, and `H`, `W` are the vertical and horizontal sizes of the mask, respectively.

The corresponding value for the output pixel $Y_{i,j}$ is computed as:

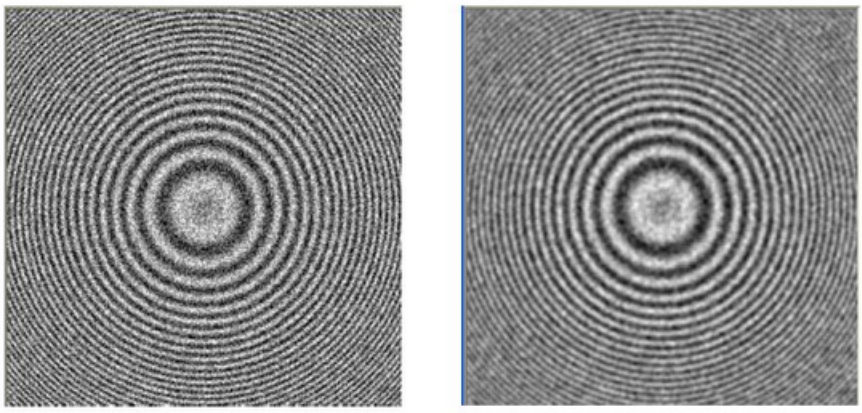
$$Y_{i,j} = \mu_{i,j} + \frac{\sigma^2_{i,j} - v^2}{\sigma^2} \cdot [X_{i,j} - \mu_{i,j}]$$

and stored in the `pDst`. Here v^2 is the noise variance, specified for each channel by the noise level parameter `noise`. If this parameter is not defined (`noise = 0`), then the function estimates the noise level by averaging through the image of all local variances $\sigma_{i,j}$, and stores the corresponding values in the `noise` for further use.

The function `ippiFilterWiener` uses the external work buffer `pBuffer`, which must be allocated before the function call. To determine the required buffer size, the function ippiFilterWienerGetBufferSize can be used.

Figure "Applying the function ippiFilterWiener" illustrates the result of using `ippiFilterWiener_32f_C1R` function.

Applying the function ippiFilterWiener



Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if one of the fields of `dstRoiSize` has a zero or negative value.
`ippStsMaskSizeErr`	Indicates an error condition if one of the fields of `maskSize` has a value less than or equal to 1.
`ippStsNoiseRangeErr`	Indicates an error condition if one of the `noise` values is less than 0 or greater than 1.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

//   A simple example of performing two-dimensional adaptive noise-removal filtering of an image
using Wiener filter
// implemented with Intel(R) Integrated Performance Primitives (Intel(R) IPP) functions:
//     ippiFilterWienerGetBufferSize
//     ippiFilterWiener_32f_C1R

#include <stdio.h>
#include "ipp.h"

#define WIDTH  128 /* image width */
#define HEIGHT 64  /* image height */

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define check_sts(st)        \
    if ((st) != ippStsNoErr) \
        goto exitLine; /* Go to Exit if Intel(R) IPP function returned status different from
ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main(void)
{
    IppStatus status = ippStsNoErr;
    Ipp32f *pSrc = NULL, *pDst = NULL;  /* Pointers to source/destination images */
    Ipp32f *pSrcW = NULL;               /* Pointers to source ROI image */
    int srcStep, dstStep;               /* Steps, in bytes, through the source/destination
images */
    IppiSize roiSize = {WIDTH, HEIGHT}; /* Size of source/destination in pixels */
    IppiSize maskSize = {3, 3};
    IppiSize dstRoiSize = {0}; /* Size of destination ROI in pixels */
    IppiPoint anchor = {1, 1}; /* Anchor cell specifying the mask alignment\
                               with respect to the position of the input pixel */
    Ipp32f noise[1] = {0.0};   /* Noise level value */
    Ipp8u *pBuffer = NULL;     /* Pointer to the work buffer */
    int iTmpBufSize = 0;       /* Common work buffer size */
    int numChannels = 1;

    dstRoiSize.width = WIDTH - maskSize.width;
```

```
    dstRoiSize.height = HEIGHT - maskSize.height;

    /* memory allocation */
    pSrc = ippiMalloc_32f_C1(roiSize.width, roiSize.height, &srcStep);
    pDst = ippiMalloc_32f_C1(roiSize.width, roiSize.height, &dstStep);

    check_sts(status = ippiImageJaehne_32f_C1R(pSrc, srcStep, roiSize)); /* fill source image */

    check_sts(status = ippiFilterWienerGetBufferSize(dstRoiSize, maskSize, numChannels,
&iTmpBufSize));

    check_sts(status = ippiSet_32f_C1R(1.f, pDst, dstStep, roiSize));

    pBuffer = ippsMalloc_8u(iTmpBufSize);
    pSrcW = (Ipp32f *)((Ipp8u *)(pSrc + anchor.x) + anchor.y * srcStep);

    check_sts(status = ippiFilterWiener_32f_C1R(pSrcW, srcStep, pDst, dstStep, dstRoiSize,
maskSize, anchor, noise, pBuffer));

    EXIT_MAIN
    ippsFree(pBuffer);
    ippiFree(pSrc);
    ippiFree(pDst);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

Convolution

Intel IPP functions described in this section perform two-dimensional finite linear convolution operation between two source images and write the result into the destination image. Convolution is used to perform many common image processing operations including sharpening, blurring, noise reduction, embossing, and edge enhancement. For convenience, any digital image `f` is represented here as a matrix with M_f columns and N_f rows that contains pixel values `f[i,j]`, $0 \le i < M_f$, $0 \le j < N_f$.

- ConvGetBufferSize
- Conv

ConvGetBufferSize

Computes the size of the work buffer for the `ippiConv` function.

Syntax

```
IppStatus ippiConvGetBufferSize (IppiSize src1Size, IppiSize src2Size, IppDataType
dataType, int numChannels, IppEnum algType, int* pBufferSize);
```

Include Files

`ippi.h`

Parameters

`src1Size`, `src2Size`	Size, in pixels, of the source images.

dataType	Data type for convolution. Possible values are ipp32f, ipp16s, or ipp8u.
numChannels	Number of image channels. Possible values are 1, 3, or 4.
algType	Bit-field mask for the algorithm type definition. Possible values are the results of composition of the IppAlgType and IppiROIShape values.
pBufferSize	Pointer to the size of the work buffer.

Description

The ippiConvGetBufferSize function computes the size, in bytes, of the external work buffer needed for the function that performs two-dimensional convolution. The result is stored in the pBufferSize parameter.

Return Values

ippStsNoErr	Indicates no error.
ippStsSizeErr	Indicates an error when the src1Size or src2Size is negative, or equal to zero.
ippStsNullChannelsErr	Indicates an error when the numChannels value differs from 1, 3, or 4.
ippStsDataTypeErr	Indicates an error when the dataType value differs from the ipp32f, ipp16s, or ipp8u.
ippStsAlgTypeErr	Indicates an error when: • the result of the bitwise AND operation between algType and ippAlgMask differs from the ippAlgAuto, ippAlgDirect, or ippAlgFFT values; • the result of the bitwise AND operation between algType and ippiROIMask differs from the ippiROIFull or ippiROIValid values.
ippStsNullPtrErr	Indicates an error when the pBufferSize is NULL.

See Also

Structures and Enumerators

Conv Performs two-dimensional convolution of two images.

Conv

Performs two-dimensional convolution of two images.

Syntax

Case 1: Operating on integer data

```
IppStatus ippiConv_<mod>(const Ipp<datatype>* pSrc1, int src1Step, IppiSize src1Size,
const Ipp<datatype>* pSrc2, int src2Step, IppiSize src2Size, Ipp<datatype>* pDst, int
dstStep, int divisor, IppEnum algType, Ipp8u* pBuffer);
```

Supported values for `mod`

`8u_C1R`	`16s_C1R`
`8u_C3R`	`16s_C3R`
`8u_C4R`	`16s_C4R`

Case 2: Operating on floating-point data

```
IppStatus ippiConv_<mod>(const Ipp32f* pSrc1, int src1Step, IppiSize src1Size, const
Ipp32f* pSrc2, int src2Step, IppiSize src2Size, Ipp32f* pDst, int dstStep, IppEnum
algType, Ipp8u* pBuffer);
```

Supported values for `mod`

`32f_C1R`
`32f_C3R`
`32f_C4R`

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc1`, `pSrc2`	Pointers to the source images ROI.
`src1Step`, `src2Step`	Distance, in bytes, between the starting points of consecutive lines in the source images.
`src1Size`, `src2Size`	Size in pixels of the source images.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance, in bytes, between the starting points of consecutive lines in the destination image.
`divisor`	The integer value by which the computed result is divided (for operations on integer data only).
`algType`	Bit-field mask for the algorithm type definition. Possible values are the results of composition of the `IppAlgType` and `IppiROIShape` values.
`pBuffer`	Pointer to the buffer for internal calculations.

Description

Before using this function, you need to compute the size of the work buffer using the `ippiConvGetBufferSize` function.

The `ippiConv` function operates with ROI. The type of convolution that function performs is defined by the value of the `algType` parameter:

1. If the `ippiROIFull` flag is set, the function performs full two-dimensional finite linear convolution between two source images pointed by the `pSrc1` and `pSrc2` parameters. The resulting destination image `h[i, j]` is computed by the following formula:

$$h[i,j] = \frac{1}{divisor} \sum_{l=0}^{N_h-1} \sum_{k=0}^{M_h-1} f[k,l] \times g[i-k, j-l] \quad ,$$

where

- $M_h = M_f + M_g - 1$

  where

  - M_f is the number of rows in the first source image matrix `f`
  - M_g is the number of rows in the second source image matrix `g`
- $N_h = N_f + N_g - 1$

  where

  - N_f is the number of columns in the first source image matrix `f`
  - N_g is the number of columns in the second source image matrix `g`
- $0 \le i < M_h, 0 \le i < N_h$
- $$f[k,l] = \begin{cases} f[k,l], & 0 \le k < M_f; \quad 0 \le l < N_f \\ 0 \quad , otherwise \end{cases}$$

  $$g[i-k, j-l] = \begin{cases} g[i-k, j-l], & 0 \le i-k < M_g; \quad 0 \le j-l < N_g \\ 0 \quad , otherwise \end{cases}$$

2. If the `ippiROIValid` flag is set up, the function performs valid two-dimensional finite linear convolution between two source images pointed by the `pSrc1` and `pSrc2` parameters. The destination image `h[i, j]` obtained as a result of the function operation is computed by the following formula:

$$h[i,j] = \frac{1}{divisor} \sum_{l=0}^{N_g-1} \sum_{k=0}^{M_g-1} f[i+k, j+l] \times g[M_g-k-1, N_g-l-1]$$

where

- $M_h = |M_f - M_g| + 1$

  where

  - M_f is the number of rows in the first source image matrix `f`
  - M_g is the number of rows in the second source image matrix `g`
- $N_h = |N_f - N_g| + 1$

  where

- N_f is the number of columns in the first source image matrix f
- N_g is the number of columns in the second source image matrix g

- $0 \le i < M_h$, $0 \le i < N_h$

This case assumes that $M_f \ge M_g$ and $N_f \ge N_g$. In case when $M_f < M_g$ and $N_f < N_g$, the subscript index g in this equation must be replaced with the index f. For any other combination of source image sizes, the function performs no operation.

NOTE The above formula provides the same result as in the case with the ippiROIFull flag, but produces only the part of the convolution image that is computed without zero-padded values.

Function flavors that accept input data of the Ipp32f type use the same summation formula, but without scaling of the result (divisor = 1 is assumed).

The following examples illustrate the function operation. For the source images f, g of size 3 x 5 represented as

$$f = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \quad g = f$$

with g = f:

- for the ippiROIFull case, the resulting convolution image h is of size 5 x 9 and contains the following data:

$$h = \begin{bmatrix} 1 & 2 & 3 & 2 & 1 \\ 2 & 2 & 2 & 0 & 0 \\ 3 & 4 & 6 & 4 & 2 \\ 2 & 2 & 4 & 2 & 2 \\ 3 & 6 & 11 & 6 & 3 \\ 2 & 2 & 4 & 2 & 2 \\ 2 & 4 & 6 & 4 & 3 \\ 0 & 0 & 2 & 2 & 2 \\ 1 & 2 & 3 & 2 & 1 \end{bmatrix}$$

- for the ippiROIValid case, the resulting convolution image h is of size 1 x 1 and contains the following data:

  h = [11]

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.

ippStsSizeErr	Indicates an error when src1Size or src2Size has a zero or negative value.
ippStsStepErr	Indicates an error when src1Step, src2Step, or dstStep has a zero or negative value.
ippStsDivisorErr	Indicates an error when divisor has a zero value.
ippStsAlgTypeErr	Indicates an error when:

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

// A simple example of performing full or valid 2-D convolution of two images using a general
integer rectangular kernel
// implemented with Intel(R) Integrated Performance Primitives (Intel(R) IPP) functions:
//     ippiConvGetBufferSize
//     ippiConv_16s_C1R

#include <stdio.h>
#include "ipp.h"

#define WIDTH_DST  128 /* destination image width  */
#define HEIGHT_DST 64  /* destination image height */
#define WIDTH_SRC  16  /* source image width  */
#define HEIGHT_SRC 16  /* source image height */

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define check_sts(st)        \
    if ((st) != ippStsNoErr) \
        goto exitLine; /* Go to Exit if Intel(R) IPP function returned status different from
ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main(void)
{
    IppStatus status = ippStsNoErr;
    Ipp16s *pSrc1 = NULL, *pSrc2 = NULL, *pDst = NULL;   /* Pointers to source/destination
images */
```

```
    int srcStep1 = 0, srcStep2 = 0, dstStep = 0;         /* Steps, in bytes, through the source/
destination images */
    IppiSize dstSize = {WIDTH_DST, HEIGHT_DST};          /* Size of destination ROI in pixels */
    IppiSize src1Size = {WIDTH_SRC, HEIGHT_SRC};         /* Size of destination ROI in pixels */
    IppiSize src2Size = {WIDTH_SRC - 1, HEIGHT_SRC - 1}; /* Size of destination ROI in pixels */
    int divisor = 2;                                     /* The integer value by which the
computed result is divided */
    Ipp8u *pBuffer = NULL;                               /* Pointer to the work buffer */
    int iTmpBufSize = 0;                                 /* Common work buffer size */
    int numChannels = 1;
    IppEnum funCfgFull = (IppEnum)(ippAlgAuto | ippiROIFull | ippiNormNone);

    pSrc2 = ippiMalloc_16s_C1(src2Size.width, src2Size.height, &srcStep2);
    pSrc1 = ippiMalloc_16s_C1(src1Size.width, src1Size.height, &srcStep1);
    pDst = ippiMalloc_16s_C1(dstSize.width, dstSize.height, &dstStep);

    check_sts(status = ippiConvGetBufferSize(src1Size, src2Size, ipp16s, numChannels,
funCfgFull, &iTmpBufSize));

    pBuffer = ippsMalloc_8u(iTmpBufSize);

    check_sts(status = ippiConv_16s_C1R(pSrc1, srcStep1, src1Size, pSrc2, srcStep2, src2Size,
pDst, dstStep, divisor, funCfgFull, pBuffer));

    EXIT_MAIN
    ippsFree(pBuffer);
    ippiFree(pSrc1);
    ippiFree(pSrc2);
    ippiFree(pDst);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

See Also

Regions of Interest in Intel IPP

Structures and Enumerators

ConvGetBufferSize Computes the size of the work buffer for the ippiConv function.

Deconvolution

Functions described in this section perform image deconvolution. They can be used for restoring the degraded image, in particular image that was obtained by applying the convolution operation with known kernel. The Intel IPP functions implement two methods: the Fourier deconvolution (noniterative method) [see for example, Puetter05], and the Richardson-Lucy method (iterative method) Ric72. Border pixels of a source image are restored before deconvolution.

- DeconvFFTGetSize
- DeconvFFTInit
- DeconvFFT
- DeconvLRGetSize
- DeconvLRInit
- DeconvLR

DeconvFFTGetSize

Computes the size of the state structure for deconvolution with the fast Fourier transform (FFT).

Syntax

```
IppStatus ippiDeconvFFTGetSize_32f(int nChannels, int kernelSize, int FFTorder, int* pSize);
```

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

nChannels	Number of channels in the image. Possible values are 1 or 3.
kernelSize	Size of the kernel.
FFTorder	Order of the created FFT state structure.
pSize	Pointer to the size of the IppiDeconvFFTState_32f_C1R or IppiDeconvFFTState_32f_C3R structure, in bytes.

Description

This function computes the fast Fourier transform (FFT) deconvolution state structure size that is required to initialize the structure with the ippiDeconvFFTInit function. This structure is used by the ippiDeconvFFT function, which performs deconvolution of the source image using FFT.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.
ippStsNumChannelsErr	Indicates an error when nChannels has an illegal value.
ippStsSizeErr	Indicates an error when kernelSize is less than, or equal to 0; or if kernelSize is greater than $2^{FFTorder}$.

See Also

DeconvFFTInit Initializes the FFT deconvolution state structure.

DeconvFFT Performs FFT deconvolution of an image.

DeconvFFTInit

Initializes the FFT deconvolution state structure.

Syntax

```
IppStatus ippiDeconvFFTInit_32f_C1R(IppiDeconvFFTState_32f_C1R* pDeconvFFTState, const Ipp32f* pKernel, int kernelSize, int FFTorder, Ipp32f threshold);
```

```
IppStatus ippiDeconvFFTInit_32f_C3R(IppiDeconvFFTState_32f_C3R* pDeconvFFTState, const Ipp32f* pKernel, int kernelSize, int FFTorder, Ipp32f threshold);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pDeconvFFTState`	Pointer to the FFT deconvolution state structure.
`pKernel`	Pointer to the kernel array.
`kernelSize`	Size of the kernel.
`FFTorder`	Order of the created FFT state structure.
`threshold`	Value of the threshold level (to exclude dividing by zero).

Description

This function initializes the FFT deconvolution state structure that is used by the ippiDeconvFFT function to perform deconvolution of the source image using FFT. Before using the `ippiDeconvFFTInit` function, compute the size of the structure using the ippiDeconvFFTGetSize function.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error when any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error when `kernelSize` is less than, or equal to 0; or if `kernelSize` is greater than $2^{FFTorder}$.
`ippStsBadArgErr`	Indicates an error when `threshold` is less than, or equal to 0.

See Also

DeconvFFTGetSize Computes the size of the state structure for deconvolution with the fast Fourier transform (FFT).

DeconvFFT Performs FFT deconvolution of an image.

DeconvFFT

Performs FFT deconvolution of an image.

Syntax

```
IppStatus ippiDeconvFFT_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiDeconvFFTState_32f_C1R* pDeconvFFTState);
```

```
IppStatus ippiDeconvFFT_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiDeconvFFTState_32f_C3R* pDeconvFFTState);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image.
`roiSize`	Size of the source and destination image ROI.
`pDeconvFFTState`	Pointer to the FFT deconvolution state structure.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function performs deconvolution of the source image `pSrc` using FFT with parameters specified in the FFT deconvolution state structure `pDeconvFFTState` and stores results to the destination image `pDst`. The FFT deconvolution state structure must be initialized by calling the function DeconvFFTInit beforehand.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.

DeconvLRGetSize

Computes the size of the state structure for Lucy-Richardson (LR) deconvolution.

Syntax

```
IppStatus ippiDeconvLRGetSize_32f(int numChannels, int kernelSize, IppiSize maxRoi,
int* pSize);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`numChannels`	Number of channels in the image. Possible values are 1 or 3.
`kernelSize`	Size of the kernel.
`maxRoi`	Maximum size of the image ROI, in pixels.
`pSize`	Pointer to the size of the `IppiDeconvLRState_32f_C1R` or `IppiDeconvLRState_32f_C3R` structure, in bytes.

Description

This function computes the Lucy-Richardson (LR) deconvolution state structure size that is required to initialize the structure with the ippiDeconvLRInit function. This structure is used by the ippiDeconvLR function, which performs LR deconvolution of the source image.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error when any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error when:

See Also

DeconvLRInit Initializes the LR deconvolution state structure.

DeconvLR Performs LR deconvolution of an image.

DeconvLRInit

Initializes the LR deconvolution state structure.

Syntax

```
IppStatus ippiDeconvLRInit_32f_C1R(IppiDeconvLR_32f_C1R* pDeconvLR, const Ipp32f*
pKernel, int kernelSize, IppiSize maxRoi, Ipp32f threshold);
```

```
IppStatus ippiDeconvLRInit_32f_C3R(IppiDeconvLR_32f_C3R* pDeconvLR, const Ipp32f*
pKernel, int kernelSize, IppiSize maxRoi, Ipp32f threshold);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pDeconvLR`	Pointer to the LR deconvolution state structure.
`pKernel`	Pointer to the kernel array.
`kernelSize`	Size of the kernel.
`maxRoi`	Maximum size of the image ROI, in pixels.
`threshold`	Value of the threshold level (to exclude dividing by zero).

Description

This function initializes the LR deconvolution state structure that is used by the ippiDeconvLR function to perform LR deconvolution of the source image. Before using the `ippiDeconvLRInit` function, compute the size of the structure using the ippiDeconvLRGetSize function.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error when any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error when: • `kernelSize` is less than, or equal to 0 • `kernelSize` is greater than `maxRoi.height` or `maxRoi.width` • `maxRoi.height` or `maxRoi.width` is less than, or equal to zero
`ippStsBadArgErr`	Indicates an error when `threshold` is less than, or equal to 0.

See Also

DeconvLRGetSize Computes the size of the state structure for Lucy-Richardson (LR) deconvolution.

DeconvLR Performs LR deconvolution of an image.

DeconvLR

Performs LR deconvolution of an image.

Syntax

```
IppStatus ippiDeconvLR_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, int numIter, IppiDeconvLR_32f_C1R* pDeconvLR);
```

```
IppStatus ippiDeconvLR_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, int numIter, IppiDeconvLR_32f_C3R* pDeconvLR);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image.
`roiSize`	Size of the source and destination image ROI.
`numIter`	Number of algorithm iterations.
`pDeconvLR`	Pointer to the LR deconvolution state structure.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function performs deconvolution of the source image `pSrc` using the Lucy-Richardson algorithm with parameters specified in the state structure `pDeconvLR` and stores results to the destination image `pDst`. The Lucy-Richardson deconvolution state structure must be initialized by calling the function DeconvLRInit beforehand.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.

ippStsSizeErr	Indicates an error condition if roi.width or roi.height is less than or equal to 0, or if roi.width is greater than (maxRoi.width – kernelSize), or roi.height is greater than (maxRoi.height – kernelSize).
ippStsStepErr	Indicates an error condition if srcStep or dstStep is less than roiSize.width*<pixelSize>.
ippStsNotEvenStepErr	Indicates an error condition if steps for floating-point images are not divisible by 4.
ippStsBadArgErr	Indicates an error condition if numIter is less than or equal to 0.

Fixed Filters

The fixed filter functions perform linear filtering of a source image using one of the predefined convolution kernels.

NOTE Kernels of fixed filters are in inverse order.

The supported fixed filters and their respective kernel sizes are listed in the following table:

Fixed Filter Type	Kernel Size
Horizontal Prewitt operator	3x3
Vertical Prewitt operator	3x3
Horizontal Scharr operator	3x3
Vertical Scharr operator	3x3
Horizontal Sobel operator	3x3 or 5x5
Vertical Sobel operator	3x3 or 5x5
Second derivative horizontal Sobel operator	3x3 or 5x5
Second derivative vertical Sobel operator	3x3 or 5x5
Second cross derivative Sobel operator	3x3 or 5x5
Horizontal Roberts operator	3x3
Vertical Roberts operator	3x3
Laplacian highpass filter	3x3 or 5x5
Gaussian lowpass filter	3x3 or 5x5
Highpass filter	3x3 or 5x5
Lowpass filter	3x3 or 5x5
Sharpening filter	3x3

Using fixed filter functions with predefined kernels is more efficient as it eliminates the need to create the convolution kernel in your application program.

NOTE The anchor cell is the center cell of the kernel for all fixed filters.

- FilterGaussianGetBufferSize
- FilterGaussianGetSpecSize
- FilterGaussianInit
- FilterGaussian
- FilterHipassBorderGetBufferSize
- FilterHipassBorder
- FilterLaplaceBorderGetBufferSize
- FilterLaplaceBorder
- FilterLaplacianGetBufferSize
- FilterLaplacianBorder
- FilterLowpassGetBufferSize
- FilterLowpassBorder
- FilterPrewittHorizBorderGetBufferSize
- FilterPrewittHorizBorder
- FilterPrewittVertBorderGetBufferSize
- FilterPrewittVertBorder
- FilterRobertsUpBorderGetBufferSize
- FilterRobertsUpBorder
- FilterRobertsDownBorderGetBufferSize
- FilterRobertsDownBorder
- FilterScharrHorizMaskBorderGetBufferSize
- FilterScharrHorizMaskBorder
- FilterScharrVertMaskBorderGetBufferSize
- FilterScharrVertMaskBorder
- FilterSharpenBorderGetBufferSize
- FilterSharpenBorder
- FilterSobelGetBufferSize
- FilterSobel
- FilterSobelHorizBorderGetBufferSize
- FilterSobelHorizBorder
- FilterSobelHorizSecondBorderGetBufferSize
- FilterSobelHorizSecondBorder
- FilterSobelVertBorderGetBufferSize
- FilterSobelVertBorder
- FilterSobelVertSecondBorderGetBufferSize
- FilterSobelNegVertBorderGetBufferSize
- FilterSobelNegVertBorder
- FilterSobelVertSecondBorder
- FilterSobelCrossGetBufferSize
- FilterSobelCrossBorder
- GenSobelKernel

FilterGaussianGetBufferSize

Computes the size of the Gaussian specification structure and the size of the external work buffer for Gaussian filtering with user-defined borders.

Syntax

```
IppStatus ippiFilterGaussianGetBufferSize(IppiSize maxRoiSize, Ipp32u kernelSize,
IppDataType dataType, int numChannels, int* pSpecSize, int* pBufferSize);
```

Platform-aware functions

```
IppStatus ippiFilterGaussianGetBufferSize_L(IppiSizeL maxRoiSize, int kernelSize,
IppDataType dataType, IppiBorderType borderType, int numChannels, IppSizeL*
pBufferSize);
```

Include Files

`ippcv.h`

Flavors with the `_L` suffix: `ippcv_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`maxRoiSize`	Maximal size of the image ROI in pixels.
`kernelSize`	Size of the Gaussian filter kernel. The value must be odd and greater, or equal to 3.
`dataType`	Data type of the source and destination images.
`borderType`	One of border supported types.
`numChannels`	Number of channels in the images. Possible values are 1 and 3.
`pSpecSize`	Pointer to the computed size (in bytes) of the Gaussian specification structure.
`pBufferSize`	Pointer to the computed size (in bytes) of the external buffer.

Description

This function computes the size of the Gaussian context structure and external work buffer for the FilterGaussianBorder and FilterGaussian functions, or the external work buffer size for the platform-aware function FilterGaussian. The results are stored in `pSpecSize` and `pBufferSize`. The buffer with the length `pBufferSize[0]` can be used to filter an image with the width and height less than, or equal to the corresponding fields of `maxRoiSize`, and/or kernel size that is less than, or equal to `kernelSize`.

The platform-aware function `FilterGaussianGetBufferSize_L` computes only the size of the external work buffer for Gaussian filtering with user-defined borders. To compute the size of the Gaussian specification structure, please use the platform-aware function FilterGaussianGetSpecSize_L.

Use the computed `pBufferSize` and `pSpecSize` values to allocate the memory using the `ippMalloc` or `ippsMalloc` functions. The allocated memory can be freed only by the `ippFree` or `ippsFree` functions, respectively. For more information about the memory allocation functions, refer to the Support Functions section.

For an example on how to use this function, refer to the example provided with the FilterGaussianBorder function description.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when maxRoiSize has a field with a zero or negative value.
ippStsBadArgErr	Indicates an error when kernelSize is even, or less than 3.
ippStsDataTypeErr	Indicates an error when dataType has an illegal value.
ippStsNumChannelErr	Indicates an error when numChannels has an illegal value.

See Also

FilterGaussianBorder Performs Gaussian filtering of an image with user-defined borders.

FilterGaussianGetSpecSize

Computes the size of the Gaussian specification structure.

Syntax

```
IppStatus ippiFilterGaussianGetSpecSize(int kernelSize, IppDataType dataType, int numChannels, int* pSpecSize, int* pInitBufferSize);
```

```
IppStatus ippiFilterGaussianGetSpecSize_L(int kernelSize, IppDataType dataType, int numChannels, IppSizeL* pSpecSize, IppSizeL* pInitBufferSize);
```

Include Files

ippcv.h

ippcv_l.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

kernelSize	Size of the Gaussian filter kernel. The value must be odd and greater, or equal to 3.
dataType	Data type of the source and destination images.
numChannels	Number of channels in the images. Possible values are 1 and 3.

pSpecSize	Pointer to the computed size (in bytes) of the Gaussian specification structure.
pInitBufferSize	Pointer to the computed size (in bytes) of the external buffer.

Description

This function computes the size of the Gaussian context structure and the size of the initialization external work buffer for the platform-aware functions FilterGaussianInit and FilterGaussian. The results are stored in pSpecSize and pInitBufferSize. The buffer with the length pInitBufferSize[0] can be used to initialize the specification structure for the Gaussian filter.

Use the computed pInitBufferSize and pSpecSize values to allocate the memory using the ippMalloc or ippsMalloc functions. The allocated memory can be freed only by the ippFree or ippsFree functions, respectively. For more information about the memory allocation functions, refer to the Support Functions section.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsBadArgErr	Indicates an error when kernelSize is even, or less than 3.
ippStsDataTypeErr	Indicates an error when dataType has an illegal value.
ippStsNumChannelErr	Indicates an error when numChannels has an illegal value.

See Also

FilterGaussian Performs Gaussian filtering of an image with user-defined borders.

FilterGaussianInit

Initializes the Gaussian context structure.

Syntax

```
IppStatus ippiFilterGaussianInit(IppiSize roiSize, Ipp32u kernelSize, Ipp32f sigma,
IppiBorderType borderType, IppDataType dataType, int numChannels,
IppFilterGaussianSpec* pSpec, Ipp8u* pBuffer);
```

Platform-aware functions

```
IppStatus ippiFilterGaussianInit_L(IppiSizeL roiSize, int kernelSize, Ipp32f sigma,
IppiBorderType borderType, IppDataType dataType, int numChannels,
IppFilterGaussianSpec* pSpec, Ipp8u* pInitBuffer);
```

Include Files

ippcv.h

Flavors with the _L suffix: ippcv_l.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

roiSize	Size of the image ROI in pixels.
kernelSize	Size of the Gaussian filter kernel. The value must be odd and greater, or equal to 3.
sigma	Standard deviation of the Gaussian kernel.
borderType	Type of border. Possible values are: • ippBorderConst • Values of all border pixels are set to constant. • ippBorderRepl • Border is replicated from the edge pixels. • ippBorderInMem • Border is obtained from the image pixels in memory. • ippBorderMirror • Border pixels are mirrored from the source image boundary pixels.
dataType	Data type of the source and destination images.
numChannels	Number of channels in the images. Possible values are 1 and 3.
pSpec	Pointer to the Gaussian specification structure.
pBuffer, pInitBuffer	Pointer to the external buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function initializes the Gaussian specification structure pSpec for the FilterGaussianBorder or the FilterGaussian function. Before using this function, compute the size of the specification structure and the external buffer using the FilterGaussianGetBufferSize function and FilterGaussianGetSpecSize for the platform-aware function.

The roiSize and kernelSize values must be less than, or equal to the corresponding values specified in the FilterGaussianGetBufferSize function.

The kernel of the Gaussian filter is the matrix of size kernelSize x kernelSize with the standard deviation sigma. The values of the Gaussian kernel elements are calculated by the formula below and then normalized:

$$G(i,j) = \exp\left\{-\frac{(K/2-i)^2+(K/2-j)^2}{2\sigma^2}\right\}, \quad i,j = 0,\ldots K$$

The anchor cell is the center of the kernel.

For an example on how to use this function, refer to the example provided with the FilterGaussianBorder function description.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when roiSize has a field with a zero or negative value.
ippStsBadArgErr	Indicates an error when kernelSize is even, or less than 3; or sigma is less than, or equal to zero.
ippStsDataTypeErr	Indicates an error when dataType has an illegal value.
ippStsChannelErr	Indicates an error when numChannels has an illegal value.
ippStsBorderErr	Indicates an error when borderType has an illegal value.

See Also

FilterGaussianGetBufferSize Computes the size of the Gaussian specification structure and the size of the external work buffer for Gaussian filtering with user-defined borders.

FilterGaussianBorder Performs Gaussian filtering of an image with user-defined borders.

FilterGaussian

Performs Gaussian filtering of an image with user-defined borders.

Syntax

Case 1: Operating on one-channel data

```
IppStatus ippiFilterGaussian_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, const
Ipp<datatype> borderValue[1], IppFilterGaussianSpec* pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32f_C1R

Case 2: Operating on multi-channel data

```
IppStatus ippiFilterGaussian_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiBorderType borderType, const
Ipp<datatype> borderValue[3], IppFilterGaussianSpec* pSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

`8u_C3R`	`16u_C3R`	`16s_C3R`	`32f_C3R`

Case 3: Operating on one-channel data with platform-aware functions

```
IppStatus ippiFilterGaussian_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep,
Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType,
const Ipp<datatype> borderValue[1], IppFilterGaussianSpec* pSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

`8u_C1R_L`	`16u_C1R_L`	`16s_C1R_L`	`32f_C1R_L`

Case 4: Operating on multi-channel data with platform-aware functions

```
IppStatus ippiFilterGaussian_<mod>(const Ipp<datatype>* pSrc, IppSizeL srcStep,
Ipp<datatype>* pDst, IppSizeL dstStep, IppiSizeL roiSize, IppiBorderType borderType,
const Ipp<datatype> borderValue[3], IppFilterGaussianSpec* pSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

`8u_C3R_L`	`16u_C3R_L`	`16s_C3R_L`	`32f_C3R_L`

Include Files

`ippcv.h`

`ippcv_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance, in bytes, between the starting points of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance, in bytes, between the starting points of consecutive lines in the destination image.
`roiSize`	Size of the source and destination image ROI, in pixels.
`borderType`	One of the supported border types.
`borderValue`	Constant value to assign to pixels of the constant border. This parameter is applicable only to the `ippBorderConst` border type.

pSpec	Pointer to the Gaussian specification structure.
pBuffer	Pointer to the work buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

These functions apply the Gaussian filter to the source image ROI pSrc. The kernel of the Gaussian filter is the matrix of size kernelSize x kernelSize with the standard deviation sigma. The values of the Gaussian kernel elements are computed by the FilterGaussianInit function. Elements of the kernel are normalized. The anchor cell is the center of the kernel.

Before using the ippiFilterGaussian function, compute the size of the Gaussian specification structure and the external buffer, and initialize the structure using the FilterGaussianInit function. For platform-aware functions, use FilterGaussianGetSpecSize for the Gaussian specification structure size calculation and FilterGaussianGetBufferSize for the external buffer size calculation. For non-platform-aware functions, use FilterGaussianGetBufferSize for both Gaussian specification structure and external buffer size calculation. For an example on how to use this function, refer to the example provided with the FilterGaussianBorder function description.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when roiSize has a field with a zero or negative value.
ippStsStepErr	Indicates an error when srcStep or dstStep is less than roiSize.width *<pixelSize>.
ippStsNotEvenStepErr	Indicates an error when one of the step values is not divisible by sizeof(Ipp<dataType>).
ippStsBorderErr	Indicates an error when borderType has an illegal value.
ippStsBadArgErr	Indicates an error when kernelSize is even, or less than 3.

Example

```
/*******************************************************************************
 * Copyright 2022 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
```

```
 * those that are expressly stated in the License.
 *****************************************************************************/

//   A simple example of performing a filtering an image using a general integer rectangular
kernel
// implemented with Intel(R) Integrated Performance Primitives (Intel(R) IPP) functions:
//     ippiImageJaehne_32f_C1R
//     ippiFilterGaussianGetSpecSize_L
//     ippiFilterGaussianInit_L
//     ippiFilterGaussianGetBufferSize_L
//     ippiFilterGaussian_32f_C1R_L

#include <stdio.h>
#include "ipp.h"

#define WIDTH  128 /* image width */
#define HEIGHT 64  /* image height */

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define check_sts(st)        \
    if ((st) != ippStsNoErr) \
        goto exitLine; /* Go to Exit if Intel(R) IPP function returned status different from
ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main(void)
{
    IppStatus status = ippStsNoErr;
    Ipp32f *pSrc = NULL, *pDst = NULL;    /* Pointers to source/destination images */
    IppSizeL srcStep = 0, dstStep = 0;    /* Steps, in bytes, through the source/destination
images */
    IppiSizeL roiSizeL = {WIDTH, HEIGHT}; /* Size of source/destination ROI in pixels */
    IppiSize roiSize = {WIDTH, HEIGHT};   /* Size of source/destination ROI in pixels - for
ImageJaehne only */
    int kernelSize = 3;
    Ipp32f sigma = 0.35f;
    Ipp8u *pBuffer = NULL;                                  /* Pointer to the work buffer */
    Ipp8u *pInitBuf = NULL;                                 /* Pointer to the Init buffer */
    IppFilterGaussianSpec *pSpec = NULL;                    /* context structure */
    IppSizeL iTmpBufSize = 0, iSpecSize = 0, iInitBufSize = 0; /* Common work buffer size */
    IppiBorderType borderType = ippBorderRepl;
    Ipp32f borderValue = 0;
    int numChannels = 1;

    pSrc = ippiMalloc_32f_C1_L(roiSize.width, roiSize.height, &srcStep);
    pDst = ippiMalloc_32f_C1_L(roiSize.width, roiSize.height, &dstStep);

    check_sts(status = ippiImageJaehne_32f_C1R(pSrc, (int)srcStep, roiSize)); /* fill source
image */

    check_sts(status = ippiFilterGaussianGetSpecSize_L(kernelSize, ipp32f, numChannels,
&iSpecSize, &iInitBufSize));

    pSpec = (IppFilterGaussianSpec *)ippsMalloc_8u_L(iSpecSize);
```

```
    pInitBuf = ippsMalloc_8u_L(iInitBufSize);

    status = ippiFilterGaussianInit_L(roiSizeL, kernelSize, sigma, borderType, ipp32f,
numChannels, pSpec, pInitBuf);
    ippsFree(pInitBuf);
    check_sts(status);

    check_sts(status = ippiFilterGaussianGetBufferSize_L(roiSizeL, kernelSize, ipp32f,
borderType, numChannels, &iTmpBufSize));

    pBuffer = ippsMalloc_8u_L(iTmpBufSize);

    check_sts(status = ippiFilterGaussian_32f_C1R_L(pSrc, srcStep, pDst, dstStep, roiSizeL,
borderType, &borderValue, pSpec, pBuffer));

    EXIT_MAIN
    ippsFree(pBuffer);
    ippsFree(pSpec);
    ippiFree(pSrc);
    ippiFree(pDst);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

See Also

Regions of Interest in Intel IPP

User-defined Border Types

FilterGaussianGetSpecSize Computes the size of the Gaussian specification structure.

FilterGaussianGetBufferSize Computes the size of the Gaussian specification structure and the size of the external work buffer for Gaussian filtering with user-defined borders.

FilterGaussianInit Initializes the Gaussian context structure.

FilterHipassBorderGetBufferSize

Computes the size of the work buffer for high-pass filtering with the `ippiFilterHipassBorder` function.

Syntax

```
IppStatus ippiFilterHipassBorderGetBufferSize(IppiSize dstRoiSize, IppiMaskSize mask,
IppDataType srcDataType, IppDataType dstDataType, int numChannels, int* pBufferSize);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`dstRoiSize`	Size of the destination ROI, in pixels.

mask	Predefined mask of IppiMaskSize type. Possible values are: ippMskSize3x3 or ippMskSize5x5.
srcDataType	Data type of the source image.
dstDataType	Data type of the destination image.
numChannels	Number of channels in the image. Possible values are 1, 3, or 4.
pBufferSize	Pointer to the size, in bytes, of the external buffer.

Description

This function computes the size, in bytes, of the external work buffer for the ippiFilterHipassBorder function. The result is stored in the pBufferSize parameter.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error when pBufferSize is NULL.
ippStsSizeErr	Indicates an error when dstRoiSize has a field with a zero or negative value.
ippStsMaskSizeErr	Indicates an error when mask has an illegal value.
ippStsDataTypeErr	Indicates an error when srcDataType or dstDataType has an illegal value.
ippStsNumChannelErr	Indicates an error when numChannels has an illegal value.

See Also

FilterHipassBorder Filters an image using a high-pass filter.

FilterHipassBorder

Filters an image using a high-pass filter.

Syntax

Case 1: Operating on one-channel data

```
IppStatus ippiFilterHipassBorder_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, Ipp8u
borderValue, Ipp8u* pBuffer);
```

```
IppStatus ippiFilterHipassBorder_16s_C1R(const Ipp16s* pSrc, int srcStep, Ipp16s* pDst,
int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, Ipp16s
borderValue, Ipp8u* pBuffer);
```

```
IppStatus ippiFilterHipassBorder_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, Ipp32f
borderValue, Ipp8u* pBuffer);
```

Case 2: Operating on multi-channel data

```
IppStatus ippiFilterHipassBorder_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, const
Ipp8u pBorderValue[3], Ipp8u* pBuffer);

IppStatus ippiFilterHipassBorder_16s_C3R(const Ipp16s* pSrc, int srcStep, Ipp16s* pDst,
int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, const
Ipp16s pBorderValue[3], Ipp8u* pBuffer);

IppStatus ippiFilterHipassBorder_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, const
Ipp32f pBorderValue[3], Ipp8u* pBuffer);

IppStatus ippiFilterHipassBorder_8u_AC4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, const
Ipp8u pBorderValue[3], Ipp8u* pBuffer);

IppStatus ippiFilterHipassBorder_16s_AC4R(const Ipp16s* pSrc, int srcStep, Ipp16s*
pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType,
const Ipp16s pBorderValue[3], Ipp8u* pBuffer);

IppStatus ippiFilterHipassBorder_32f_AC4R(const Ipp32f* pSrc, int srcStep, Ipp32f*
pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType,
const Ipp32f pBorderValue[3], Ipp8u* pBuffer);

IppStatus ippiFilterHipassBorder_8u_C4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, const
Ipp8u pBorderValue[4], Ipp8u* pBuffer);

IppStatus ippiFilterHipassBorder_16s_C4R(const Ipp16s* pSrc, int srcStep, Ipp16s* pDst,
int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, const
Ipp16s pBorderValue[4], Ipp8u* pBuffer);

IppStatus ippiFilterHipassBorder_32f_C4R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst,
int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, const
Ipp32f pBorderValue[4], Ipp8u* pBuffer);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance, in bytes, between the starting points of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance, in bytes, between the starting points of consecutive lines in the destination image.

dstRoiSize	Size of the source and destination ROI, in pixels.
mask	Predefined mask of IppiMaskSize. Possible values are ippMskSize3x3 or ippMskSize5x5.
borderType	Type of border. Possible values are: • ippBorderConst • Values of all border pixels are set to constant. • ippBorderRepl • Border is replicated from the edge pixels. • ippBorderInMem • Border is obtained from the source image pixels in memory.
borderValue	Constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.
pBorderValue[3], pBorderValue[4]	Pointer to constant values to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.
pBuffer	Pointer to the work buffer.

Description

Before using this function, you need to compute the size of the work buffer pBuffer using the ippiFilterHipassBorderGetBufferSize function.

This function operates with ROI (see Regions of Interest in Intel IPP).

This function applies a high-pass filter to the pSrc source image ROI. The size of the source image ROI is equal to the destination image ROI size dstRoiSize. The values of border pixels are assigned in accordance with the borderType and borderValue parameters. The kernel of the filter is a matrix of 3x3 or 5x5 size depending on the mask value. The kernels have the following values:

```
-1  -1   -1  -1  -1

-1  -1  -1               -1  -1   -1  -1  -1

-1   8  -1       or      -1  -1  24  -1  -1

-1  -1  -1               -1  -1   -1  -1  -1

-1  -1   -1  -1  -1
```

The anchor cell is the center cell of the kernel, highlighted in red.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when dstRoiSize is negative, or equal to zero.

ippStsBorderErr	Indicates an error when mask has an illegal value.
ippStsBorderErr	Indicates an error when borderType has an illegal value.

See Also

Borders in Neighborhood Operations

Regions of Interest in Intel IPP

User-defined Border Types

FilterHipassBorderGetBufferSize Computes the size of the work buffer for high-pass filtering with the ippiFilterHipassBorder function.

FilterLaplaceBorderGetBufferSize

Computes the size of the work buffer for Laplace filtering.

Syntax

```
IppStatus ippiFilterLaplaceBorderGetBufferSize(IppiSize dstRoiSize, IppiMaskSize mask,
IppDataType srcDataType, IppDataType dstDataType, int numChannels, int* pBufferSize);
```

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

dstRoiSize	Size of the destination ROI, in pixels.
mask	Predefined mask of IppiMaskSize type. Possible values are: ippMskSize3x3 or ippMskSize5x5.
srcDataType	Data type of the source image.
dstDataType	Data type of the destination image.
numChannels	Number of channels in the image. Possible values are 1, 3, or 4.
pBufferSize	Pointer to the size, in bytes, of the external buffer.

Description

This function computes the size, in bytes, of the external work buffer for the ippiFilterLaplaceBorder function. The result is stored in the pBufferSize parameter.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error when pBufferSize is NULL.
ippStsSizeErr	Indicates an error when dstRoiSize has a field with a zero or negative value.
ippStsMaskSizeErr	Indicates an error when mask has an illegal value.
ippStsDataTypeErr	Indicates an error when srcDataType or dstDataType has an illegal value.
ippStsNumChannelErr	Indicates an error when numChannels has an illegal value.

See Also

FilterLaplaceBorder Filters an image using a Laplace filter.

FilterLaplaceBorder

Filters an image using a Laplace filter.

Syntax

Case 1: Operating on one-channel data

```
IppStatus ippiFilterLaplaceBorder_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, Ipp8u
borderValue, Ipp8u* pBuffer);
```

```
IppStatus ippiFilterLaplaceBorder_16s_C1R(const Ipp16s* pSrc, int srcStep, Ipp16s*
pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType,
Ipp16s borderValue, Ipp8u* pBuffer);
```

```
IppStatus ippiFilterLaplaceBorder_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f*
pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType,
Ipp32f borderValue, Ipp8u* pBuffer);
```

Case 2: Operating on multi-channel data

```
IppStatus ippiFilterLaplaceBorder_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, const
Ipp8u pBorderValue[3], Ipp8u* pBuffer);
```

```
IppStatus ippiFilterLaplaceBorder_16s_C3R(const Ipp16s* pSrc, int srcStep, Ipp16s*
pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType,
const Ipp16s pBorderValue[3], Ipp8u* pBuffer);
```

```
IppStatus ippiFilterLaplaceBorder_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f*
pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType,
const Ipp32f pBorderValue[3], Ipp8u* pBuffer);
```

```
IppStatus ippiFilterLaplaceBorder_8u_AC4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, const
Ipp8u pBorderValue[3], Ipp8u* pBuffer);
```

```
IppStatus ippiFilterLaplaceBorder_16s_AC4R(const Ipp16s* pSrc, int srcStep, Ipp16s*
pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType,
const Ipp16s pBorderValue[3], Ipp8u* pBuffer);

IppStatus ippiFilterLaplaceBorder_32f_AC4R(const Ipp32f* pSrc, int srcStep, Ipp32f*
pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType,
const Ipp32f pBorderValue[3], Ipp8u* pBuffer);

IppStatus ippiFilterLaplaceBorder_8u_C4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, const
Ipp8u pBorderValue[4], Ipp8u* pBuffer);

IppStatus ippiFilterLaplaceBorder_16s_C4R(const Ipp16s* pSrc, int srcStep, Ipp16s*
pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType,
const Ipp16s pBorderValue[4], Ipp8u* pBuffer);

IppStatus ippiFilterLaplaceBorder_32f_C4R(const Ipp32f* pSrc, int srcStep, Ipp32f*
pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType,
const Ipp32f pBorderValue[4], Ipp8u* pBuffer);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance, in bytes, between the starting points of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance, in bytes, between the starting points of consecutive lines in the destination image.
`dstRoiSize`	Size of the source and destination ROI, in pixels.
`mask`	Predefined mask of `IppiMaskSize`. Possible values are `ippMskSize3x3` or `ippMskSize5x5`.
`borderType`	Type of border. Possible values are: • `ippBorderConst` • Values of all border pixels are set to constant. • `ippBorderRepl` • Border is replicated from the edge pixels. • `ippBorderInMem` • Border is obtained from the source image pixels in memory.

borderValue	Constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.
pBorderValue[3], pBorderValue[4]	Pointer to constant values to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.
pBuffer	Pointer to the work buffer.

Description

Before using this function, you need to compute the size of the work buffer pBuffer using the ippiFilterLaplaceBorderGetBufferSize function.

This function operates with ROI (see Regions of Interest in Intel IPP).

This function applies a Laplace filter to the pSrc source image ROI. The size of the source image ROI is equal to the destination image ROI size dstRoiSize. The values of border pixels are assigned in accordance with the borderType and borderValue parameters. The kernel of the filter is a matrix of 3x3 or 5x5 size depending on the mask value. The kernels have the following values:

```
-1  -3   -4  -3  -1

-1  -1  -1                -3   0    6   0  -3

-1   8  -1       or       -4   6  20   6  -4

-1  -1  -1                -3   0    6   0  -3

-1  -3   -4  -3  -1
```

The anchor cell is the center cell of the kernel, highlighted in red.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when dstRoiSize is negative, or equal to zero.
ippStsBorderErr	Indicates an error when mask has an illegal value.
ippStsBorderErr	Indicates an error when borderType has an illegal value.

See Also

Borders in Neighborhood Operations

Regions of Interest in Intel IPP

User-defined Border Types

FilterLaplaceBorder Filters an image using a Laplace filter.

FilterLaplacianGetBufferSize

Computes the size of the external buffer for the Laplace filter with border.

Syntax

```
IppStatus ippiFilterLaplacianGetBufferSize_<mod>(IppiSize roiSize, IppiMaskSize mask,
int* pBufferSize);
```

Supported values for `mod`:

`8u16s_C1R`	`32f_C1R`

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`roiSize`	Maximum size of the source and destination image ROI.
`mask`	Predefined mask of `IppiMaskSize` type.
`pBufferSize`	Pointer to the buffer size.

Description

This function computes the size of the external buffer that is required for the filter function ippiFilterLaplacianBorder. The kernel of the filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter `mask` (see Table "Types of the Fixed Filter Functions"). This buffer `pBufferSize[0]` can be used to filter an image whose width and height are equal to or less than corresponding fields of `roiSize`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if the `pBufferSize` pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize` has a field with a zero or negative value.
`ippStsMaskSizeErr`	Indicates an error condition if `mask` has a wrong value.

FilterLaplacianBorder

Applies Laplacian filter with border.

Syntax

```
IppStatus ippiFilterLaplacianBorder_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize, IppiMaskSize mask,
IppiBorderType borderType, Ipp<srcDatatype> borderValue, Ipp8u* pBuffer);
```

Supported values for `mod`:

`8u16s_C1R`	`32f_C1R`

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image.
`roiSize`	Size of the source and destination image ROI.
`mask`	Type of the filter kernel.
`borderType`	Type of border (see Borders in Neighborhood Operations); following values are possible: • `ippBorderConst` • Values of all border pixels are set to constant. • `ippBorderRepl` • Replicated border is used. • `ippBorderMirror` • Border pixels are mirrored from the source image boundary pixels.
`borderValue`	The constant value to assign to the pixels in the constant border (not applicable for other border's type).
`pBuffer`	Pointer to the working buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function applies the laplacian filter to the source image `pSrc` and stores results to the destination image of the same size `pDst`. Source image can be used as the destination image if they both have the same data type. The values of border pixels are assigned in accordance with the `borderType` and `borderValue` parameters. The kernel of this filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter `mask`. The kernels have the following values with the anchor in the center cell (red):

```
2   4    4   4   2                                
2   0   2                 4   0   -8   0   4
0  -8   0       or        4  -8  -24  -8   4
2   0   2                 4   0   -8   0   4
2   4    4   4   2
```

The function requires the working buffer `pBuffer` whose size should be computed by the function ippiFilterLaplacianGetBufferSize beforehand.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize` has a field with a zero or negative value.
`ippStsStepErr`	Indicates an error condition if `srcStep` or `dstStep` is less than `roiSize.width` * <pixelSize>
`ippStsNotEvenStepErr`	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
`ippStsBadArgErr`	Indicates an error if `borderType` or `divisor` has a wrong value.
`ippStsMaskErr`	Indicates an error condition if `mask` has a wrong value.

FilterLowpassGetBufferSize

Computes the size of the external buffer for the lowpass filter with border.

Syntax

```
IppStatus ippiFilterLowpassGetBufferSize_8u_C1R(IppiSize roiSize, IppiMaskSize mask,
int* pBufferSize);
```

```
IppStatus ippiFilterLowpassGetBufferSize_32f_C1R(IppiSize roiSize, IppiMaskSize mask,
int* pBufferSize);
```

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`roiSize`	Maximum size of the source and destination image ROI.
`mask`	Predefined mask of `IppiMaskSize` type.
`pBufferSize`	Pointer to the buffer size.

Description

This function computes the size of the external buffer that is required for the filter function ippiFilterLowpassBorder. The kernel of the filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter `mask` (see Table "Types of the Fixed Filter Functions"). This buffer `pBufferSize[0]` can be used to filter an image whose width and height are equal to or less than corresponding fields of `roiSize`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if the `pBufferSize` pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize` has a field with a zero or negative value.
`ippStsMaskSizeErr`	Indicates an error condition if `mask` has a wrong value.

FilterLowpassBorder

Applies lowpass filter with border.

Syntax

```
IppStatus ippiFilterLowpassBorder_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
int dstStep, IppiSize roiSize, IppiMaskSize mask, IppiBorderType borderType, Ipp8u
borderValue, Ipp8u* pBuffer);
```

```
IppStatus ippiFilterLowpassBorder_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f*
pDst, int dstStep, IppiSize roiSize, IppiMaskSize mask, IppiBorderType borderType,
Ipp32f borderValue, Ipp8u* pBuffer);
```

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image.
`roiSize`	Size of the source and destination image ROI.
`mask`	Type of the filter kernel.
`borderType`	Type of border (see Borders in Neighborhood Operations); the following values are possible: • `ippBorderConst` • Values of all border pixels are set to constant. • `ippBorderRepl` • Replicated border is used.
`borderValue`	The constant value to assign to the pixels in the constant border (not applicable for other border's type).
`pBuffer`	Pointer to the working buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function applies the lowpass filter (blur operation) to the source image `pSrc` and stores results to the destination image of the same size `pDst`. Source image can be used as the destination image if they both have the same data type. The values of border pixels are assigned in accordance with the `borderType` and `borderValue` parameters. The kernel of this filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter `mask`. The anchor cell is the center cell (red) of the kernel.

The 3x3 filter uses the kernel with the following values:

```
1/9  1/9  1/9
1/9  1/9  1/9
1/9  1/9  1/9
```

The 5x5 filter uses the kernel with the following values:

```
1/25  1/25  1/25  1/25  1/25
1/25  1/25  1/25  1/25  1/25
1/25  1/25  1/25  1/25  1/25
1/25  1/25  1/25  1/25  1/25
1/25  1/25  1/25  1/25  1/25
```

The function requires the working buffer `pBuffer` whose size should be computed by the function ippiFilterLowpassGetBufferSize beforehand.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize` has a field with a zero or negative value.
`ippStsStepErr`	Indicates an error condition if `srcStep` or `dstStep` is less than `roiSize.width` * <pixelSize>.
`ippStsNotEvenStepErr`	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images.
`ippStsBorderErr`	Indicates an error condition if `borderType` has a wrong value.
`ippStsMaskErr`	Indicates an error condition if `mask` has a wrong value.

FilterPrewittHorizBorderGetBufferSize

Computes the size of the work buffer for the Prewitt Horizontal filter.

Syntax

```
IppStatus ippiFilterPrewittHorizBorderGetBufferSize (IppiSize dstRoiSize, IppiMaskSize mask, IppDataType srcDataType, IppDataType dstDataType, int numChannels, int* pBufferSize);
```

Include Files

`ippi.h`

Parameters

`dstRoiSize`	Size of the destination ROI in pixels.
`mask`	Predefined mask of `IppiMaskSize` type. Possible value is `ippMskSize3x3`.
`srcDataType`	Data type of the source image.
`dstDataType`	Data type of the destination image.
`numChannels`	Number of channels in the image. Possible value is 1.
`pBufferSize`	Pointer to the size of the external work buffer.

Description

The ippiFilterPrewittHorizBorderGetBufferSize function computes the size, in bytes, of the external work buffer needed for the ippiFilterPrewittHorizBorder function. The result is stored in the pBufferSize parameter.

For an example on how to use this function, see the code example provided with the ippiFilterPrewittHorizBorder function description.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pBufferSize is NULL.
ippStsSizeErr	Indicates an error when dstRoiSize is negative, or equal to zero.
ippStsMaskSizeErr	Indicates an error when mask has an illegal value.
ippStsDataTypeErr	Indicates an error when srcDataType or dstDataType has an illegal value.
ippStsNumChannelsError	Indicates an error when numChannels has an illegal value.

See Also

FilterPrewittHorizBorder Filters and image using a horizontal Prewitt filter.

FilterPrewittHorizBorder

Filters and image using a horizontal Prewitt filter.

Syntax

```
IppStatus ippiFilterPrewittHorizBorder_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask,
IppiBorderType borderType, Ipp<srcDatatype> borderValue, Ipp8u* pBuffer);
```

Supported values for mod:

8u16s_C1R	16s_C1R	32f_C1R

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc	Pointer to the source image ROI.

srcStep	Distance, in bytes, between the starting points of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance, in bytes, between the starting points of consecutive lines in the destination image.
dstRoiSize	Size of the source and destination ROI in pixels.
mask	Predefined mask of IppiMaskSize. Possible value is ippMskSize3x3.
borderType	Type of border. Possible values are: • ippBorderConst • Values of all border pixels are set to constant. • ippBorderRepl • Border is replicated from the edge pixels. • ippBorderMirror • Mirrored border is used. • ippBorderMirrorR • Mirrored border with replication is used. • ippBorderInMem • Border is obtained from the source image pixels in memory.
borderValue	Constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.
pBuffer	Pointer to the work buffer.

Description

Before using this function, you need to compute the size of the work buffer pBuffer using the FilterPrewittHorizBorderGetBufferSize function.

This function operates with ROI.

This function applies a horizontal Prewitt filter to the pSrc source image ROI. The size of the source image ROI is equal to the destination image ROI size dstRoiSize. The values of border pixels are assigned in accordance with the borderType and borderValue parameters. The kernel of the filter is a matrix of 3x3 size with the following values:

```
1   1   1
0   0   0
-1  -1  -1
```

The anchor cell is the center cell of the kernel, highlighted in red.

This filter enhances horizontal edges of an image.

Return Values

ippStsNoErr	Indicates no error.

ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when dstRoiSize is negative, or equal to zero.
ippStsStepErr	Indicates an error when srcStep or dstStep is negative, or equal to zero.
ippStsNotEvenStepErr	Indicates an error when one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
ippStsBorderErr	Indicates an error when borderType has an illegal value.

Example

Filtering convolution kernels:

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

//   A simple example of performing linear filtering of an image using one of predefined
convolution kernels
// implemented with Intel(R) Integrated Performance Primitives (Intel(R) IPP) functions:
//     ippiFilterPrewittHorizBorderGetBufferSize
//     ippiFilterPrewittHorizBorder_8u16s_C1R
//     ippiFilterPrewittVertBorderGetBufferSize
//     ippiFilterPrewittVertBorder_8u16s_C1R
//     ippiFilterRobertsDownBorderGetBufferSize
//     ippiFilterRobertsDownBorder_8u16s_C1R
//     ippiFilterRobertsUpBorderGetBufferSize
//     ippiFilterRobertsUpBorder_8u16s_C1R
//     ippiFilterScharrHorizMaskBorderGetBufferSize
//     ippiFilterScharrHorizMaskBorder_8u16s_C1R

#include <stdio.h>
#include "ipp.h"

#define WIDTH  128 /* image width  */
#define HEIGHT 128 /* image height */

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
```

```
    exitLine: /* Label for Exit */
#define check_sts(st)        \
    if ((st) != ippStsNoErr) \
        goto exitLine; /* Go to Exit if Intel(R) IPP function returned status different from
ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main(void)
{
    IppStatus status = ippStsNoErr;
    Ipp8u *pSrc = NULL;
    Ipp16s *pDst = NULL;                              /* Pointers to source/destination images */
    int srcStep = 0, dstStep = 0;                     /* Steps, in bytes, through the source/
destination images */
    IppiSize srcImageSize = {WIDTH + 1, HEIGHT + 1}; /* size of source image in pixels */
    IppiSize roiSize = {WIDTH, HEIGHT};               /* Size of source/destination ROI in pixels
*/
    Ipp8u *pBuffer = NULL;                            /* Pointer to the work buffer */
    int iTmpBufSize = 0;                              /* Common work buffer size */
    IppiMaskSize mask = ippMskSize3x3;                /* Predefined mask of IppiMaskSize type. */
    IppDataType srcDataType = ipp8u, dstDataType = ipp16s;
    IppiBorderType borderType = (IppiBorderType)(ippBorderRepl | ippBorderInMemTop |
ippBorderInMemRight);
    Ipp8u borderValue = 254;
    int numChannels = 1;

    pSrc = ippiMalloc_8u_C1(srcImageSize.width, srcImageSize.height, &srcStep);
    pDst = ippiMalloc_16s_C1(roiSize.width, roiSize.height, &dstStep);

    /* Filters and image using a horizontal Prewitt filter */
    check_sts(status = ippiFilterPrewittHorizBorderGetBufferSize(roiSize, mask, srcDataType,
dstDataType, numChannels, &iTmpBufSize));

    pBuffer = ippsMalloc_8u(iTmpBufSize);

    check_sts(status =
                ippiFilterPrewittHorizBorder_8u16s_C1R(pSrc + srcStep, srcStep, pDst, dstStep,
roiSize, mask, borderType, borderValue, pBuffer));

    ippsFree(pBuffer);
    pBuffer = NULL;

    /* Filters and image using a vertical Prewitt filter */
    check_sts(status = ippiFilterPrewittVertBorderGetBufferSize(roiSize, mask, srcDataType,
dstDataType, numChannels, &iTmpBufSize));

    pBuffer = ippsMalloc_8u(iTmpBufSize);

    check_sts(status =
                ippiFilterPrewittVertBorder_8u16s_C1R(pSrc + srcStep, srcStep, pDst, dstStep,
roiSize, mask, borderType, borderValue, pBuffer));

    ippsFree(pBuffer);
    pBuffer = NULL;

    /* Filters an image using a horizontal Roberts kernel. */
```

```
    check_sts(status = ippiFilterRobertsDownBorderGetBufferSize(roiSize, mask, srcDataType,
dstDataType, numChannels, &iTmpBufSize));

    pBuffer = ippsMalloc_8u(iTmpBufSize);

    check_sts(status =
                ippiFilterRobertsDownBorder_8u16s_C1R(pSrc + srcStep, srcStep, pDst, dstStep,
roiSize, mask, borderType, borderValue, pBuffer));

    ippsFree(pBuffer);
    pBuffer = NULL;

    /* Filters an image using a vertical Roberts edge filter */
    check_sts(status = ippiFilterRobertsUpBorderGetBufferSize(roiSize, mask, srcDataType,
dstDataType, numChannels, &iTmpBufSize));

    pBuffer = ippsMalloc_8u(iTmpBufSize);

    check_sts(status = ippiFilterRobertsUpBorder_8u16s_C1R(pSrc + srcStep, srcStep, pDst,
dstStep, roiSize, mask, borderType, borderValue, pBuffer));

    ippsFree(pBuffer);
    pBuffer = NULL;

    /* Filters an image using a horizontal Scharr kernel */
    check_sts(status = ippiFilterScharrHorizMaskBorderGetBufferSize(roiSize, mask, srcDataType,
dstDataType, numChannels, &iTmpBufSize));

    pBuffer = ippsMalloc_8u(iTmpBufSize);

    check_sts(status =
                ippiFilterScharrHorizMaskBorder_8u16s_C1R(pSrc + srcStep, srcStep, pDst,
dstStep, roiSize, mask, borderType, borderValue, pBuffer));

    EXIT_MAIN
    ippsFree(pBuffer);
    ippiFree(pSrc);
    ippiFree(pDst);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

FilterPrewittHorizBorder:

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/
```

```
//   A simple example of performing linear filtering of an image using one of predefined
convolution kernels
// implemented with Intel(R) Integrated Performance Primitives (Intel(R) IPP) functions:
//     ippiFilterPrewittHorizBorderGetBufferSize
//     ippiFilterPrewittHorizBorder_8u16s_C1R
//     ippiFilterPrewittVertBorder_8u16s_C1R
//     ippiFilterRobertsDownBorder_8u16s_C1R
//     ippiFilterRobertsUpBorder_8u16s_C1R
//     ippiFilterScharrHorizMaskBorder_8u16s_C1R

#include <stdio.h>
#include "ipp.h"

#define WIDTH  128 /* image width  */
#define HEIGHT 128 /* image height */

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define check_sts(st)        \
    if ((st) != ippStsNoErr) \
        goto exitLine; /* Go to Exit if Intel(R) IPP function returned status different from
ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main(void)
{
    IppStatus status = ippStsNoErr;
    Ipp8u *pSrc = NULL;
    Ipp16s *pDst = NULL;                            /* Pointers to source/destination images */
    int srcStep = 0, dstStep = 0;                   /* Steps, in bytes, through the source/
destination images */
    IppiSize srcImageSize = {WIDTH + 1, HEIGHT + 1}; /* size of source image in pixels */
    IppiSize roiSize = {WIDTH, HEIGHT};             /* Size of source/destination ROI in pixels
*/
    Ipp8u *pBuffer = NULL;                          /* Pointer to the work buffer */
    int iTmpBufSize = 0;                            /* Common work buffer size */
    IppiMaskSize mask = ippMskSize3x3;              /* Predefined mask of IppiMaskSize type. */
    IppDataType srcDataType = ipp8u, dstDataType = ipp16s;
    IppiBorderType borderType = (IppiBorderType)(ippBorderRepl | ippBorderInMemTop |
ippBorderInMemRight);
    Ipp8u borderValue = 254;
    int numChannels = 1;

    /* memory allocation */
    pSrc = ippiMalloc_8u_C1(srcImageSize.width, srcImageSize.height, &srcStep);
    pDst = ippiMalloc_16s_C1(roiSize.width, roiSize.height, &dstStep);

    /* Filters and image using a horizontal Prewitt filter */
    check_sts(status = ippiFilterPrewittHorizBorderGetBufferSize(roiSize, mask, srcDataType,
dstDataType, numChannels, &iTmpBufSize));

    pBuffer = ippsMalloc_8u(iTmpBufSize);

    check_sts(status =
```

```
                ippiFilterPrewittHorizBorder_8u16s_C1R(pSrc + srcStep, srcStep, pDst, dstStep,
roiSize, mask, borderType, borderValue, pBuffer));

    ippsFree(pBuffer);
    pBuffer = NULL;

    /* Filters and image using a vertical Prewitt filter */
    check_sts(status = ippiFilterPrewittVertBorderGetBufferSize(roiSize, mask, srcDataType,
dstDataType, numChannels, &iTmpBufSize));

    pBuffer = ippsMalloc_8u(iTmpBufSize);

    check_sts(status =
                ippiFilterPrewittVertBorder_8u16s_C1R(pSrc + srcStep, srcStep, pDst, dstStep,
roiSize, mask, borderType, borderValue, pBuffer));

    ippsFree(pBuffer);
    pBuffer = NULL;

    /* Filters an image using a horizontal Roberts kernel. */
    check_sts(status = ippiFilterRobertsDownBorderGetBufferSize(roiSize, mask, srcDataType,
dstDataType, numChannels, &iTmpBufSize));

    pBuffer = ippsMalloc_8u(iTmpBufSize);

    check_sts(status =
                ippiFilterRobertsDownBorder_8u16s_C1R(pSrc + srcStep, srcStep, pDst, dstStep,
roiSize, mask, borderType, borderValue, pBuffer));

    ippsFree(pBuffer);
    pBuffer = NULL;

    /* Filters an image using a vertical Roberts edge filter */
    check_sts(status = ippiFilterRobertsUpBorderGetBufferSize(roiSize, mask, srcDataType,
dstDataType, numChannels, &iTmpBufSize));

    pBuffer = ippsMalloc_8u(iTmpBufSize);

    check_sts(status = ippiFilterRobertsUpBorder_8u16s_C1R(pSrc + srcStep, srcStep, pDst,
dstStep, roiSize, mask, borderType, borderValue, pBuffer));

    ippsFree(pBuffer);
    pBuffer = NULL;

    /* Filters an image using a horizontal Scharr kernel */
    check_sts(status = ippiFilterScharrHorizMaskBorderGetBufferSize(roiSize, mask, srcDataType,
dstDataType, numChannels, &iTmpBufSize));

    pBuffer = ippsMalloc_8u(iTmpBufSize);

    check_sts(status =
                ippiFilterScharrHorizMaskBorder_8u16s_C1R(pSrc + srcStep, srcStep, pDst,
dstStep, roiSize, mask, borderType, borderValue, pBuffer));

    EXIT_MAIN
    ippsFree(pBuffer);
    ippiFree(pSrc);
    ippiFree(pDst);
```

```
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

See Also

Borders in Neighborhood Operations

Regions of Interest in Intel IPP

User-defined Border Types

FilterPrewittHorizBorderGetBufferSize Computes the size of the work buffer for the Prewitt Horizontal filter.

FilterPrewittVertBorderGetBufferSize

Computes the size of the work buffer for the Prewitt Vertical filter.

Syntax

```
IppStatus ippiFilterPrewittVertBorderGetBufferSize (IppiSize dstRoiSize, IppiMaskSize mask, IppDataType srcDataType, IppDataType dstDataType, int numChannels, int* pBufferSize);
```

Include Files

`ippi.h`

Parameters

`dstRoiSize`	Size of the destination ROI in pixels.
`mask`	Predefined mask of `IppiMaskSize` type. Possible value is `ippMskSize3x3`.
`srcDataType`	Data type of the source image.
`dstDataType`	Data type of the destination image.
`numChannels`	Number of channels in the image. Possible value is 1.
`pBufferSize`	Pointer to the size of the external work buffer.

Description

The `ippiFilterPrewittVertBorderGetBufferSize` function computes the size, in bytes, of the external work buffer needed for the `ippiFilterPrewittVertBorder` function. The result is stored in the `pBufferSize` parameter.

For an example on how to use this function, see the code example provided with the `ippiFilterPrewittVertBorder` function description.

Return Values

`ippStsNoErr`	Indicates no error.

ippStsNullPtrErr	Indicates an error when pBufferSize is NULL.
ippStsSizeErr	Indicates an error when dstRoiSize is negative, or equal to zero.
ippStsMaskSizeErr	Indicates an error when mask has an illegal value.
ippStsDataTypeErr	Indicates an error when srcDataType or dstDataType has an illegal value.
ippStsNumChannelsError	Indicates an error when numChannels has an illegal value.

See Also

FilterPrewittVertBorder Filters and image using a vertical Prewitt kernel.

FilterPrewittVertBorder

Filters and image using a vertical Prewitt kernel.

Syntax

```
IppStatus ippiFilterPrewittVertBorder_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask,
IppiBorderType borderType, Ipp<srcDatatype> borderValue, Ipp8u* pBuffer);
```

Supported values for mod:

8u16s_C1R	16s_C1R	32f_C1R

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance, in bytes, between the starting points of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance, in bytes, between the starting points of consecutive lines in the destination image.
dstRoiSize	Size of the source and destination ROI in pixels.
mask	Predefined mask of IppiMaskSize. Possible value is ippMskSize3x3.

borderType	Type of border. Possible values are: • ippBorderConst • Values of all border pixels are set to constant. • ippBorderRepl • Border is replicated from the edge pixels. • ippBorderMirror • Mirrored border is used. • ippBorderMirrorR • Mirrored border with replication is used. • ippBorderInMem • Border is obtained from the source image pixels in memory.
borderValue	Constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.
pBuffer	Pointer to the work buffer.

Description

Before using this function, you need to compute the size of the work buffer pBuffer using the FilterPrewittVertBorderGetBufferSize function.

This function operates with ROI.

This function applies a vertical Prewitt filter to the pSrc source image ROI. The size of the source image ROI is equal to the destination image ROI size dstRoiSize. The values of border pixels are assigned in accordance with the borderType and borderValue parameters. The kernel of the filter is a matrix of 3x3 size with the following values:

```
-1   0   1
-1   0   1
-1   0   1
```

The anchor cell is the center cell of the kernel, highlighted in red.

This filter enhances vertical edges of an image.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when dstRoiSize is negative, or equal to zero.
ippStsStepErr	Indicates an error when srcStep or dstStep is negative, or equal to zero.
ippStsNotEvenStepErr	Indicates an error when one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.

ippStsBorderErr	Indicates an error when borderType has an illegal value.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

//   A simple example of performing linear filtering of an image using one of predefined
convolution kernels
// implemented with Intel(R) Integrated Performance Primitives (Intel(R) IPP) functions:
//     ippiFilterPrewittHorizBorderGetBufferSize
//     ippiFilterPrewittHorizBorder_8u16s_C1R
//     ippiFilterPrewittVertBorderGetBufferSize
//     ippiFilterPrewittVertBorder_8u16s_C1R
//     ippiFilterRobertsDownBorderGetBufferSize
//     ippiFilterRobertsDownBorder_8u16s_C1R
//     ippiFilterRobertsUpBorderGetBufferSize
//     ippiFilterRobertsUpBorder_8u16s_C1R
//     ippiFilterScharrHorizMaskBorderGetBufferSize
//     ippiFilterScharrHorizMaskBorder_8u16s_C1R

#include <stdio.h>
#include "ipp.h"

#define WIDTH  128 /* image width  */
#define HEIGHT 128 /* image height */

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define check_sts(st)        \
    if ((st) != ippStsNoErr) \
        goto exitLine; /* Go to Exit if Intel(R) IPP function returned status different from
ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main(void)
{
    IppStatus status = ippStsNoErr;
    Ipp8u *pSrc = NULL;
    Ipp16s *pDst = NULL;                          /* Pointers to source/destination images */
    int srcStep = 0, dstStep = 0;                 /* Steps, in bytes, through the source/
```

```
destination images */
    IppiSize srcImageSize = {WIDTH + 1, HEIGHT + 1}; /* size of source image in pixels */
    IppiSize roiSize = {WIDTH, HEIGHT};              /* Size of source/destination ROI in pixels
*/
    Ipp8u *pBuffer = NULL;                           /* Pointer to the work buffer */
    int iTmpBufSize = 0;                             /* Common work buffer size */
    IppiMaskSize mask = ippMskSize3x3;               /* Predefined mask of IppiMaskSize type. */
    IppDataType srcDataType = ipp8u, dstDataType = ipp16s;
    IppiBorderType borderType = (IppiBorderType)(ippBorderRepl | ippBorderInMemTop |
ippBorderInMemRight);
    Ipp8u borderValue = 254;
    int numChannels = 1;

    pSrc = ippiMalloc_8u_C1(srcImageSize.width, srcImageSize.height, &srcStep);
    pDst = ippiMalloc_16s_C1(roiSize.width, roiSize.height, &dstStep);

    /* Filters and image using a horizontal Prewitt filter */
    check_sts(status = ippiFilterPrewittHorizBorderGetBufferSize(roiSize, mask, srcDataType,
dstDataType, numChannels, &iTmpBufSize));

    pBuffer = ippsMalloc_8u(iTmpBufSize);

    check_sts(status =
                  ippiFilterPrewittHorizBorder_8u16s_C1R(pSrc + srcStep, srcStep, pDst, dstStep,
roiSize, mask, borderType, borderValue, pBuffer));

    ippsFree(pBuffer);
    pBuffer = NULL;

    /* Filters and image using a vertical Prewitt filter */
    check_sts(status = ippiFilterPrewittVertBorderGetBufferSize(roiSize, mask, srcDataType,
dstDataType, numChannels, &iTmpBufSize));

    pBuffer = ippsMalloc_8u(iTmpBufSize);

    check_sts(status =
                  ippiFilterPrewittVertBorder_8u16s_C1R(pSrc + srcStep, srcStep, pDst, dstStep,
roiSize, mask, borderType, borderValue, pBuffer));

    ippsFree(pBuffer);
    pBuffer = NULL;

    /* Filters an image using a horizontal Roberts kernel. */
    check_sts(status = ippiFilterRobertsDownBorderGetBufferSize(roiSize, mask, srcDataType,
dstDataType, numChannels, &iTmpBufSize));

    pBuffer = ippsMalloc_8u(iTmpBufSize);

    check_sts(status =
                  ippiFilterRobertsDownBorder_8u16s_C1R(pSrc + srcStep, srcStep, pDst, dstStep,
roiSize, mask, borderType, borderValue, pBuffer));

    ippsFree(pBuffer);
    pBuffer = NULL;

    /* Filters an image using a vertical Roberts edge filter */
    check_sts(status = ippiFilterRobertsUpBorderGetBufferSize(roiSize, mask, srcDataType,
dstDataType, numChannels, &iTmpBufSize));
```

```
    pBuffer = ippsMalloc_8u(iTmpBufSize);

    check_sts(status = ippiFilterRobertsUpBorder_8u16s_C1R(pSrc + srcStep, srcStep, pDst,
dstStep, roiSize, mask, borderType, borderValue, pBuffer));

    ippsFree(pBuffer);
    pBuffer = NULL;

    /* Filters an image using a horizontal Scharr kernel */
    check_sts(status = ippiFilterScharrHorizMaskBorderGetBufferSize(roiSize, mask, srcDataType,
dstDataType, numChannels, &iTmpBufSize));

    pBuffer = ippsMalloc_8u(iTmpBufSize);

    check_sts(status =
                  ippiFilterScharrHorizMaskBorder_8u16s_C1R(pSrc + srcStep, srcStep, pDst,
dstStep, roiSize, mask, borderType, borderValue, pBuffer));

    EXIT_MAIN
    ippsFree(pBuffer);
    ippiFree(pSrc);
    ippiFree(pDst);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

See Also

Borders in Neighborhood Operations

Regions of Interest in Intel IPP

User-defined Border Types

FilterPrewittVertBorderGetBufferSize Computes the size of the work buffer for the Prewitt Vertical filter.

FilterRobertsUpBorderGetBufferSize

Computes the size of the work buffer for the vertical Roberts edge filter.

Syntax

```
IppStatus ippiFilterRobertsUpBorderGetBufferSize (IppiSize dstRoiSize, IppiMaskSize
mask, IppDataType srcDataType, IppDataType dstDataType, int numChannels, int*
pBufferSize);
```

Include Files

`ippi.h`

Parameters

`dstRoiSize`	Size of the destination ROI in pixels.
`mask`	Predefined mask of `IppiMaskSize` type. Possible value is `ippMskSize3x3`.

srcDataType	Data type of the source image.
dstDataType	Data type of the destination image.
numChannels	Number of channels in the image. Possible value is 1.
pBufferSize	Pointer to the size of the external work buffer.

Description

The ippiFilterRobertsUpBorderGetBufferSize function computes the size, in bytes, of the external work buffer needed for the ippiFilterRobertsUpBorder function. The result is stored in the pBufferSize parameter.

For an example on how to use this function, see the code example provided with the ippiFilterRobertsUpBorder function description.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pBufferSize is NULL.
ippStsSizeErr	Indicates an error when dstRoiSize is negative, or equal to zero.
ippStsMaskSizeErr	Indicates an error when mask has an illegal value.
ippStsDataTypeErr	Indicates an error when srcDataType or dstDataType has an illegal value.
ippStsNumChannelsError	Indicates an error when numChannels has an illegal value.

See Also

FilterRobertsUpBorder Filters an image using a vertical Roberts edge filter.

FilterRobertsUpBorder

Filters an image using a vertical Roberts edge filter.

Syntax

```
IppStatus ippiFilterRobertsUpBorder_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask,
IppiBorderType borderType, Ipp<srcDatatype> borderValue, Ipp8u* pBuffer);
```

Supported values for mod:

8u16s_C1R	16s_C1R	32f_C1R

Include Files

ippi.h

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starting points of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance in bytes between starting points of consecutive lines in the destination image.
`dstRoiSize`	Size of the source and destination ROI in pixels.
`mask`	Predefined mask of `IppiMaskSize`. Possible value is `ippMskSize3x3`.
`borderType`	Type of border. Possible values are: • `ippBorderConst` • Values of all border pixels are set to constant. • `ippBorderRepl` • Border is replicated from the edge pixels. • `ippBorderMirror` • Mirrored border is used. • `ippBorderMirrorR` • Mirrored border with replication is used. • `ippBorderInMem` • Border is obtained from the source image pixels in memory.
`borderValue`	Constant value to assign to pixels of the constant border. This parameter is applicable only to the `ippBorderConst` border type.
`pBuffer`	Pointer to the work buffer.

Description

This function operates with ROI.

This function applies a vertical Roberts edge filter to the `pSrc` source image ROI. The size of the source image ROI is equal to the destination image ROI size `dstRoiSize`. The values of border pixels are assigned in accordance with the `borderType` and `borderValue` parameters. The kernel of the filter is a matrix of 3x3 size with the following values:

```
0   0   0
0   1   0
-1   0   0
```

The anchor cell is the center cell of the kernel, highlighted in red.

This filter provides the gross approximation of the pixel values gradient in the vertical direction.

Before using this function, you need to compute the size of the work buffer `pBuffer` using the `FilterRobertsUpBorderGetBufferSize` function.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error when `dstRoiSize` is negative, or equal to zero.
`ippStsStepErr`	Indicates an error when `srcStep` or `dstStep` is negative, or equal to zero.
`ippStsNotEvenStepErr`	Indicates an error when one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
`ippStsBorderErr`	Indicates an error when `borderType` has an illegal value.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

//   A simple example of performing linear filtering of an image using one of predefined
convolution kernels
// implemented with Intel(R) Integrated Performance Primitives (Intel(R) IPP) functions:
//     ippiFilterPrewittHorizBorderGetBufferSize
//     ippiFilterPrewittHorizBorder_8u16s_C1R
//     ippiFilterPrewittVertBorderGetBufferSize
//     ippiFilterPrewittVertBorder_8u16s_C1R
//     ippiFilterRobertsDownBorderGetBufferSize
//     ippiFilterRobertsDownBorder_8u16s_C1R
//     ippiFilterRobertsUpBorderGetBufferSize
//     ippiFilterRobertsUpBorder_8u16s_C1R
//     ippiFilterScharrHorizMaskBorderGetBufferSize
//     ippiFilterScharrHorizMaskBorder_8u16s_C1R

#include <stdio.h>
#include "ipp.h"
```

```
#define WIDTH  128 /* image width  */
#define HEIGHT 128 /* image height */

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define check_sts(st)        \
    if ((st) != ippStsNoErr) \
        goto exitLine; /* Go to Exit if Intel(R) IPP function returned status different from
ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main(void)
{
    IppStatus status = ippStsNoErr;
    Ipp8u *pSrc = NULL;
    Ipp16s *pDst = NULL;                          /* Pointers to source/destination images */
    int srcStep = 0, dstStep = 0;                 /* Steps, in bytes, through the source/
destination images */
    IppiSize srcImageSize = {WIDTH + 1, HEIGHT + 1}; /* size of source image in pixels */
    IppiSize roiSize = {WIDTH, HEIGHT};           /* Size of source/destination ROI in pixels
*/
    Ipp8u *pBuffer = NULL;                        /* Pointer to the work buffer */
    int iTmpBufSize = 0;                          /* Common work buffer size */
    IppiMaskSize mask = ippMskSize3x3;            /* Predefined mask of IppiMaskSize type. */
    IppDataType srcDataType = ipp8u, dstDataType = ipp16s;
    IppiBorderType borderType = (IppiBorderType)(ippBorderRepl | ippBorderInMemTop |
ippBorderInMemRight);
    Ipp8u borderValue = 254;
    int numChannels = 1;

    pSrc = ippiMalloc_8u_C1(srcImageSize.width, srcImageSize.height, &srcStep);
    pDst = ippiMalloc_16s_C1(roiSize.width, roiSize.height, &dstStep);

    /* Filters and image using a horizontal Prewitt filter */
    check_sts(status = ippiFilterPrewittHorizBorderGetBufferSize(roiSize, mask, srcDataType,
dstDataType, numChannels, &iTmpBufSize));

    pBuffer = ippsMalloc_8u(iTmpBufSize);

    check_sts(status =
                  ippiFilterPrewittHorizBorder_8u16s_C1R(pSrc + srcStep, srcStep, pDst, dstStep,
roiSize, mask, borderType, borderValue, pBuffer));

    ippsFree(pBuffer);
    pBuffer = NULL;

    /* Filters and image using a vertical Prewitt filter */
    check_sts(status = ippiFilterPrewittVertBorderGetBufferSize(roiSize, mask, srcDataType,
dstDataType, numChannels, &iTmpBufSize));

    pBuffer = ippsMalloc_8u(iTmpBufSize);

    check_sts(status =
                  ippiFilterPrewittVertBorder_8u16s_C1R(pSrc + srcStep, srcStep, pDst, dstStep,
roiSize, mask, borderType, borderValue, pBuffer));
```

```
    ippsFree(pBuffer);
    pBuffer = NULL;

    /* Filters an image using a horizontal Roberts kernel. */
    check_sts(status = ippiFilterRobertsDownBorderGetBufferSize(roiSize, mask, srcDataType,
dstDataType, numChannels, &iTmpBufSize));

    pBuffer = ippsMalloc_8u(iTmpBufSize);

    check_sts(status =
                  ippiFilterRobertsDownBorder_8u16s_C1R(pSrc + srcStep, srcStep, pDst, dstStep,
roiSize, mask, borderType, borderValue, pBuffer));

    ippsFree(pBuffer);
    pBuffer = NULL;

    /* Filters an image using a vertical Roberts edge filter */
    check_sts(status = ippiFilterRobertsUpBorderGetBufferSize(roiSize, mask, srcDataType,
dstDataType, numChannels, &iTmpBufSize));

    pBuffer = ippsMalloc_8u(iTmpBufSize);

    check_sts(status = ippiFilterRobertsUpBorder_8u16s_C1R(pSrc + srcStep, srcStep, pDst,
dstStep, roiSize, mask, borderType, borderValue, pBuffer));

    ippsFree(pBuffer);
    pBuffer = NULL;

    /* Filters an image using a horizontal Scharr kernel */
    check_sts(status = ippiFilterScharrHorizMaskBorderGetBufferSize(roiSize, mask, srcDataType,
dstDataType, numChannels, &iTmpBufSize));

    pBuffer = ippsMalloc_8u(iTmpBufSize);

    check_sts(status =
                  ippiFilterScharrHorizMaskBorder_8u16s_C1R(pSrc + srcStep, srcStep, pDst,
dstStep, roiSize, mask, borderType, borderValue, pBuffer));

    EXIT_MAIN
    ippsFree(pBuffer);
    ippiFree(pSrc);
    ippiFree(pDst);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

See Also

Borders in Neighborhood Operations

Regions of Interest in Intel IPP

User-defined Border Types

FilterRobertsUpBorderGetBufferSize Computes the size of the work buffer for the vertical Roberts edge filter.

FilterRobertsDownBorderGetBufferSize

Computes the size of the work buffer for the horizontal Roberts edge filter.

Syntax

```
IppStatus ippiFilterRobertsDownBorderGetBufferSize (IppiSize dstRoiSize, IppiMaskSize mask, IppDataType srcDataType, IppDataType dstDataType, int numChannels, int* pBufferSize);
```

Include Files

ippi.h

Parameters

dstRoiSize	Size of the destination ROI in pixels.
mask	Predefined mask of IppiMaskSize type. Possible value is ippMskSize3x3.
srcDataType	Data type of the source image.
dstDataType	Data type of the destination image.
numChannels	Number of channels in the image. Possible value is 1.
pBufferSize	Pointer to the size of the external work buffer.

Description

The ippiFilterRobertsDownBorderGetBufferSize function computes the size, in bytes, of the external work buffer needed for the ippiFilterRobertsDownBorder function. The result is stored in the pBufferSize parameter.

For an example on how to use this function, see the code example provided with the ippiFilterRobertsDownBorder function description.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pBufferSize is NULL.
ippStsSizeErr	Indicates an error when dstRoiSize is negative, or equal to zero.
ippStsMaskSizeErr	Indicates an error when mask has an illegal value.
ippStsDataTypeErr	Indicates an error when srcDataType or dstDataType has an illegal value.
ippStsNumChannelsError	Indicates an error when numChannels has an illegal value.

Related information FilterRobertsDownBorder Filters an image using a horizontal Roberts edge filter.

FilterRobertsDownBorder

Filters an image using a horizontal Roberts edge filter.

Syntax

```
IppStatus ippiFilterRobertsDownBorder_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask,
IppiBorderType borderType, Ipp<srcDatatype> borderValue, Ipp8u* pBuffer);
```

Supported values for `mod`:

`8u16s_C1R`	`16s_C1R`	`32f_C1R`

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance, in bytes, between the starting points of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance, in bytes, between the starting points of consecutive lines in the destination image.
`dstRoiSize`	Size of the source and destination ROI in pixels.
`mask`	Predefined mask of `IppiMaskSize`. Possible value is `ippMskSize3x3`.
`borderType`	Type of border. Possible values are: • `ippBorderConst` • Values of all border pixels are set to constant. • `ippBorderRepl` • Border is replicated from the edge pixels. • `ippBorderMirror` • Mirrored border is used. • `ippBorderMirrorR` • Mirrored border with replication is used. • `ippBorderInMem` • Border is obtained from the source image pixels in memory.
`borderValue`	Constant value to assign to pixels of the constant border. This parameter is applicable only to the `ippBorderConst` border type.
`pBuffer`	Pointer to the work buffer.

Description

Before using this function, you need to compute the size of the work buffer `pBuffer` using the `FilterRobertsDownBorderGetBufferSize` function.

This function operates with ROI.

This function applies a horizontal Roberts edge filter to the `pSrc` source image ROI. The size of the source image ROI is equal to the destination image ROI size `dstRoiSize`. The values of border pixels are assigned in accordance with the `borderType` and `borderValue` parameters. The kernel of the filter is a matrix of 3x3 size with the following values:

```
0   0   0

0   1   0

0   0  -1
```

The anchor cell is the center cell of the kernel, highlighted in red.

This filter provides the gross approximation of the pixel values gradient in the horizontal direction.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error when `dstRoiSize` is negative, or equal to zero.
`ippStsStepErr`	Indicates an error when `srcStep` or `dstStep` is negative, or equal to zero.
`ippStsNotEvenStepErr`	Indicates an error when one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
`ippStsBorderErr`	Indicates an error when `borderType` has an illegal value.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

//   A simple example of performing linear filtering of an image using one of predefined
```

```
convolution kernels
// implemented with Intel(R) Integrated Performance Primitives (Intel(R) IPP) functions:
//     ippiFilterPrewittHorizBorderGetBufferSize
//     ippiFilterPrewittHorizBorder_8u16s_C1R
//     ippiFilterPrewittVertBorderGetBufferSize
//     ippiFilterPrewittVertBorder_8u16s_C1R
//     ippiFilterRobertsDownBorderGetBufferSize
//     ippiFilterRobertsDownBorder_8u16s_C1R
//     ippiFilterRobertsUpBorderGetBufferSize
//     ippiFilterRobertsUpBorder_8u16s_C1R
//     ippiFilterScharrHorizMaskBorderGetBufferSize
//     ippiFilterScharrHorizMaskBorder_8u16s_C1R

#include <stdio.h>
#include "ipp.h"

#define WIDTH  128 /* image width  */
#define HEIGHT 128 /* image height */

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define check_sts(st)        \
    if ((st) != ippStsNoErr) \
        goto exitLine; /* Go to Exit if Intel(R) IPP function returned status different from
ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main(void)
{
    IppStatus status = ippStsNoErr;
    Ipp8u *pSrc = NULL;
    Ipp16s *pDst = NULL;                            /* Pointers to source/destination images */
    int srcStep = 0, dstStep = 0;                   /* Steps, in bytes, through the source/
destination images */
    IppiSize srcImageSize = {WIDTH + 1, HEIGHT + 1}; /* size of source image in pixels */
    IppiSize roiSize = {WIDTH, HEIGHT};             /* Size of source/destination ROI in pixels
*/
    Ipp8u *pBuffer = NULL;                          /* Pointer to the work buffer */
    int iTmpBufSize = 0;                            /* Common work buffer size */
    IppiMaskSize mask = ippMskSize3x3;              /* Predefined mask of IppiMaskSize type. */
    IppDataType srcDataType = ipp8u, dstDataType = ipp16s;
    IppiBorderType borderType = (IppiBorderType)(ippBorderRepl | ippBorderInMemTop |
ippBorderInMemRight);
    Ipp8u borderValue = 254;
    int numChannels = 1;

    pSrc = ippiMalloc_8u_C1(srcImageSize.width, srcImageSize.height, &srcStep);
    pDst = ippiMalloc_16s_C1(roiSize.width, roiSize.height, &dstStep);

    /* Filters and image using a horizontal Prewitt filter */
    check_sts(status = ippiFilterPrewittHorizBorderGetBufferSize(roiSize, mask, srcDataType,
dstDataType, numChannels, &iTmpBufSize));

    pBuffer = ippsMalloc_8u(iTmpBufSize);
```

```
    check_sts(status =
                  ippiFilterPrewittHorizBorder_8u16s_C1R(pSrc + srcStep, srcStep, pDst, dstStep,
roiSize, mask, borderType, borderValue, pBuffer));

    ippsFree(pBuffer);
    pBuffer = NULL;

    /* Filters and image using a vertical Prewitt filter */
    check_sts(status = ippiFilterPrewittVertBorderGetBufferSize(roiSize, mask, srcDataType,
dstDataType, numChannels, &iTmpBufSize));

    pBuffer = ippsMalloc_8u(iTmpBufSize);

    check_sts(status =
                  ippiFilterPrewittVertBorder_8u16s_C1R(pSrc + srcStep, srcStep, pDst, dstStep,
roiSize, mask, borderType, borderValue, pBuffer));

    ippsFree(pBuffer);
    pBuffer = NULL;

    /* Filters an image using a horizontal Roberts kernel. */
    check_sts(status = ippiFilterRobertsDownBorderGetBufferSize(roiSize, mask, srcDataType,
dstDataType, numChannels, &iTmpBufSize));

    pBuffer = ippsMalloc_8u(iTmpBufSize);

    check_sts(status =
                  ippiFilterRobertsDownBorder_8u16s_C1R(pSrc + srcStep, srcStep, pDst, dstStep,
roiSize, mask, borderType, borderValue, pBuffer));

    ippsFree(pBuffer);
    pBuffer = NULL;

    /* Filters an image using a vertical Roberts edge filter */
    check_sts(status = ippiFilterRobertsUpBorderGetBufferSize(roiSize, mask, srcDataType,
dstDataType, numChannels, &iTmpBufSize));

    pBuffer = ippsMalloc_8u(iTmpBufSize);

    check_sts(status = ippiFilterRobertsUpBorder_8u16s_C1R(pSrc + srcStep, srcStep, pDst,
dstStep, roiSize, mask, borderType, borderValue, pBuffer));

    ippsFree(pBuffer);
    pBuffer = NULL;

    /* Filters an image using a horizontal Scharr kernel */
    check_sts(status = ippiFilterScharrHorizMaskBorderGetBufferSize(roiSize, mask, srcDataType,
dstDataType, numChannels, &iTmpBufSize));

    pBuffer = ippsMalloc_8u(iTmpBufSize);

    check_sts(status =
                  ippiFilterScharrHorizMaskBorder_8u16s_C1R(pSrc + srcStep, srcStep, pDst,
dstStep, roiSize, mask, borderType, borderValue, pBuffer));

    EXIT_MAIN
    ippsFree(pBuffer);
    ippiFree(pSrc);
```

```
    ippiFree(pDst);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

See Also

Borders in Neighborhood Operations

Regions of Interest in Intel IPP

User-defined Border Types

FilterRobertsDownBorderGetBufferSize Computes the size of the work buffer for the horizontal Roberts edge filter.

FilterScharrHorizMaskBorderGetBufferSize

Computes the size of the work buffer for the Scharr Horizontal filter.

Syntax

```
IppStatus ippiFilterScharrHorizMaskBorderGetBufferSize (IppiSize dstRoiSize,
IppiMaskSize mask, IppDataType srcDataType, IppDataType dstDataType, int numChannels,
int* pBufferSize);
```

Include Files

`ippi.h`

Parameters

`dstRoiSize`	Size of the destination ROI in pixels.
`mask`	Predefined mask of `IppiMaskSize` type. Possible value is `ippMskSize3x3`.
`srcDataType`	Data type of the source image.
`dstDataType`	Data type of the destination image.
`numChannels`	Number of channels in the image. Possible value is 1.
`pBufferSize`	Pointer to the size of the external work buffer.

Description

The `ippiFilterScharrHorizMaskBorderGetBufferSize` function computes the size, in bytes, of the external work buffer needed for the `ippiFilterScharrHorizMaskBorder` function. The result is stored in the `pBufferSize` parameter.

For an example on how to use this function, see the code example provided with the `ippiFilterScharrHorizMaskBorder` function description.

Return Values

`ippStsNoErr`	Indicates no error.

ippStsNullPtrErr	Indicates an error when pBufferSize is NULL.
ippStsSizeErr	Indicates an error when dstRoiSize is negative, or equal to zero.
ippStsMaskSizeErr	Indicates an error when mask has an illegal value.
ippStsDataTypeErr	Indicates an error when srcDataType or dstDataType has an illegal value.
ippStsNumChannelsError	Indicates an error when numChannels has an illegal value.

See Also

FilterScharrHorizMaskBorder Filters an image using a horizontal Scharr filter.

FilterScharrHorizMaskBorder

Filters an image using a horizontal Scharr filter.

Syntax

```
IppStatus ippiFilterScharrHorizMaskBorder_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, Ipp<srcDatatype> borderValue, Ipp8u* pBuffer);
```

Supported values for mod:

8u16s_C1R	16s_C1R	32f_C1R

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance, in bytes, between the starting points of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance, in bytes, between the starting points of consecutive lines in the destination image.
dstRoiSize	Size of the source and destination ROI in pixels.
mask	Predefined mask of IppiMaskSize. Possible value is ippMskSize3x3.

borderType	Type of border. Possible values are: • ippBorderConst • Values of all border pixels are set to constant. • ippBorderRepl • Border is replicated from the edge pixels. • ippBorderMirror • Mirrored border is used. • ippBorderMirrorR • Mirrored border with replication is used. • ippBorderInMem • Border is obtained from the source image pixels in memory.
borderValue	Constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.
pBuffer	Pointer to the work buffer.

Description

Before using this function, you need to compute the size of the work buffer pBuffer using the FilterScharrHorizMaskBorderGetBufferSize function.

This function operates with ROI.

This function applies a horizontal Scharr filter to the pSrc source image ROI. The size of the source image ROI is equal to the destination image ROI size dstRoiSize. The values of border pixels are assigned in accordance with the borderType and borderValue parameters. The kernel of the filter is a matrix of 3x3 size with the following values:

```
3   10   3

0    0   0

-3  -10  -3
```

The anchor cell is the center cell of the kernel, highlighted in red.

This filter enhances horizontal edges of an image.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when dstRoiSize is negative, or equal to zero.
ippStsStepErr	Indicates an error when srcStep or dstStep is negative, or equal to zero.
ippStsNotEvenStepErr	Indicates an error when one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.

ippStsBorderErr	Indicates an error when borderType has an illegal value.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

//   A simple example of performing linear filtering of an image using one of predefined
convolution kernels
// implemented with Intel(R) Integrated Performance Primitives (Intel(R) IPP) functions:
//     ippiFilterPrewittHorizBorderGetBufferSize
//     ippiFilterPrewittHorizBorder_8u16s_C1R
//     ippiFilterPrewittVertBorderGetBufferSize
//     ippiFilterPrewittVertBorder_8u16s_C1R
//     ippiFilterRobertsDownBorderGetBufferSize
//     ippiFilterRobertsDownBorder_8u16s_C1R
//     ippiFilterRobertsUpBorderGetBufferSize
//     ippiFilterRobertsUpBorder_8u16s_C1R
//     ippiFilterScharrHorizMaskBorderGetBufferSize
//     ippiFilterScharrHorizMaskBorder_8u16s_C1R

#include <stdio.h>
#include "ipp.h"

#define WIDTH  128 /* image width  */
#define HEIGHT 128 /* image height */

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define check_sts(st)        \
    if ((st) != ippStsNoErr) \
        goto exitLine; /* Go to Exit if Intel(R) IPP function returned status different from
ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main(void)
{
    IppStatus status = ippStsNoErr;
    Ipp8u *pSrc = NULL;
    Ipp16s *pDst = NULL;                          /* Pointers to source/destination images */
    int srcStep = 0, dstStep = 0;                 /* Steps, in bytes, through the source/
```

```
destination images */
    IppiSize srcImageSize = {WIDTH + 1, HEIGHT + 1}; /* size of source image in pixels */
    IppiSize roiSize = {WIDTH, HEIGHT};              /* Size of source/destination ROI in pixels
*/
    Ipp8u *pBuffer = NULL;                           /* Pointer to the work buffer */
    int iTmpBufSize = 0;                             /* Common work buffer size */
    IppiMaskSize mask = ippMskSize3x3;               /* Predefined mask of IppiMaskSize type. */
    IppDataType srcDataType = ipp8u, dstDataType = ipp16s;
    IppiBorderType borderType = (IppiBorderType)(ippBorderRepl | ippBorderInMemTop |
ippBorderInMemRight);
    Ipp8u borderValue = 254;
    int numChannels = 1;

    pSrc = ippiMalloc_8u_C1(srcImageSize.width, srcImageSize.height, &srcStep);
    pDst = ippiMalloc_16s_C1(roiSize.width, roiSize.height, &dstStep);

    /* Filters and image using a horizontal Prewitt filter */
    check_sts(status = ippiFilterPrewittHorizBorderGetBufferSize(roiSize, mask, srcDataType,
dstDataType, numChannels, &iTmpBufSize));

    pBuffer = ippsMalloc_8u(iTmpBufSize);

    check_sts(status =
                  ippiFilterPrewittHorizBorder_8u16s_C1R(pSrc + srcStep, srcStep, pDst, dstStep,
roiSize, mask, borderType, borderValue, pBuffer));

    ippsFree(pBuffer);
    pBuffer = NULL;

    /* Filters and image using a vertical Prewitt filter */
    check_sts(status = ippiFilterPrewittVertBorderGetBufferSize(roiSize, mask, srcDataType,
dstDataType, numChannels, &iTmpBufSize));

    pBuffer = ippsMalloc_8u(iTmpBufSize);

    check_sts(status =
                  ippiFilterPrewittVertBorder_8u16s_C1R(pSrc + srcStep, srcStep, pDst, dstStep,
roiSize, mask, borderType, borderValue, pBuffer));

    ippsFree(pBuffer);
    pBuffer = NULL;

    /* Filters an image using a horizontal Roberts kernel. */
    check_sts(status = ippiFilterRobertsDownBorderGetBufferSize(roiSize, mask, srcDataType,
dstDataType, numChannels, &iTmpBufSize));

    pBuffer = ippsMalloc_8u(iTmpBufSize);

    check_sts(status =
                  ippiFilterRobertsDownBorder_8u16s_C1R(pSrc + srcStep, srcStep, pDst, dstStep,
roiSize, mask, borderType, borderValue, pBuffer));

    ippsFree(pBuffer);
    pBuffer = NULL;

    /* Filters an image using a vertical Roberts edge filter */
    check_sts(status = ippiFilterRobertsUpBorderGetBufferSize(roiSize, mask, srcDataType,
dstDataType, numChannels, &iTmpBufSize));
```

```
    pBuffer = ippsMalloc_8u(iTmpBufSize);

    check_sts(status = ippiFilterRobertsUpBorder_8u16s_C1R(pSrc + srcStep, srcStep, pDst,
dstStep, roiSize, mask, borderType, borderValue, pBuffer));

    ippsFree(pBuffer);
    pBuffer = NULL;

    /* Filters an image using a horizontal Scharr kernel */
    check_sts(status = ippiFilterScharrHorizMaskBorderGetBufferSize(roiSize, mask, srcDataType,
dstDataType, numChannels, &iTmpBufSize));

    pBuffer = ippsMalloc_8u(iTmpBufSize);

    check_sts(status =
                  ippiFilterScharrHorizMaskBorder_8u16s_C1R(pSrc + srcStep, srcStep, pDst,
dstStep, roiSize, mask, borderType, borderValue, pBuffer));

    EXIT_MAIN
    ippsFree(pBuffer);
    ippiFree(pSrc);
    ippiFree(pDst);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

See Also

Borders in Neighborhood Operations

Regions of Interest in Intel IPP

User-defined Border Types

FilterPrewittHorizBorderGetBufferSize Computes the size of the work buffer for the Prewitt Horizontal filter.

FilterScharrVertMaskBorderGetBufferSize

Computes the size of the work buffer for the Scharr Vertical filter.

Syntax

```
IppStatus ippiFilterScharrVertMaskBorderGetBufferSize (IppiSize dstRoiSize,
IppiMaskSize mask, IppDataType srcDataType, IppDataType dstDataType, int numChannels,
int* pBufferSize);
```

Include Files

`ippi.h`

Parameters

`dstRoiSize`	Size of the destination ROI in pixels.
`mask`	Predefined mask of `IppiMaskSize` type. Possible value is `ippMskSize3x3`.

srcDataType	Data type of the source image.
dstDataType	Data type of the destination image.
numChannels	Number of channels in the image. Possible value is 1.
pBufferSize	Pointer to the size of the external work buffer.

Description

The ippiFilterScharrVertMaskBorderGetBufferSize function computes the size, in bytes, of the external work buffer needed for the ippiFilterScharrVertMaskBorder function. The result is stored in the pBufferSize parameter.

For an example on how to use this function, see the code example provided with the ippiFilterScharrVertMaskBorder function description.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pBufferSize is NULL.
ippStsSizeErr	Indicates an error when dstRoiSize is negative, or equal to zero.
ippStsMaskSizeErr	Indicates an error when mask has an illegal value.
ippStsDataTypeErr	Indicates an error when srcDataType or dstDataType has an illegal value.
ippStsNumChannelsError	Indicates an error when numChannels has an illegal value.

See Also

FilterScharrVertMaskBorder Filters an image using a vertical Scharr kernel.

FilterScharrVertMaskBorder

Filters an image using a vertical Scharr kernel.

Syntax

```
IppStatus ippiFilterScharrVertMaskBorder_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, Ipp<srcDatatype> borderValue, Ipp8u* pBuffer);
```

Supported values for mod:

8u16s_C1R	16s_C1R	32f_C1R

Include Files

ippi.h

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance, in bytes, between starting points of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance, in bytes, between starting points of consecutive lines in the destination image.
`dstRoiSize`	Size of the source and destination ROI in pixels.
`mask`	Predefined mask of `IppiMaskSize`. Possible value is `ippMskSize3x3`.
`borderType`	Type of border. Possible values are: • `ippBorderConst` • Values of all border pixels are set to constant. • `ippBorderRepl` • Border is replicated from the edge pixels. • `ippBorderMirror` • Mirrored border is used. • `ippBorderMirrorR` • Mirrored border with replication is used. • `ippBorderInMem` • Border is obtained from the source image pixels in memory.
`borderValue`	Constant value to assign to pixels of the constant border. This parameter is applicable only to the `ippBorderConst` border type.
`pBuffer`	Pointer to the work buffer.

Description

This function operates with ROI.

This function applies a vertical Scharr filter to the `pSrc` source image ROI. The size of the source image ROI is equal to the destination image ROI size `dstRoiSize`. The values of border pixels are assigned in accordance with the `borderType` and `borderValue` parameters. The kernel of the filter is a matrix of 3x3 size with the following values:

```
-3   0   3
-10  0   10
-3   0   3
```

The anchor cell is the center cell of the kernel, highlighted in red.

This filter enhances horizontal edges of an image.

Before using this function, you need to compute the size of the work buffer `pBuffer` using the `FilterScharrVertMaskBorderGetBufferSize` function.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error when `dstRoiSize` is negative, or equal to zero.
`ippStsStepErr`	Indicates an error when `srcStep` or `dstStep` is negative, or equal to zero.
`ippStsNotEvenStepErr`	Indicates an error when one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
`ippStsBorderErr`	Indicates an error when `borderType` has an illegal value.

Example

The code example below demonstrates how to use the `ippiFilterScharrVertMaskBorderGetBufferSize` and `ippiFilterScharrVertMaskBorder_8u16s_C1R` functions.

```
IppStatus fix_scharrvert_8u16( void ) {
    Ipp8u pSrc[9*8] =
        {
            0, 1, 2, 120, 121, 122, 50, 51, 52,
            1, 2, 3, 121, 122, 123, 52, 52, 53,
            3, 4, 5, 130, 131, 132, 63, 64, 65,
            4, 5, 6, 131, 132, 133, 64, 65, 66,
            5, 6, 7, 132, 133, 134, 65, 66, 67,
                  8, 7, 6, 134, 133, 132, 67, 66, 65,
                  7, 6, 5, 133, 132, 131, 66, 65, 64,
                  6, 5, 4, 132, 131, 130, 65, 64, 63
         };
    Ipp16s  pDst[8*7];
    Ipp8u   *pBuffer;
    IppiSize roiSize = {8, 7};
    IppiBorderType borderType = ippBorderRepl | ippBorderInMemTop | ippBorderInMemRight;
    int    srcStep = 9 * sizeof(Ipp8u);
    int    dstStep = 8 * sizeof(Ipp16s);
    int    bufferSize;
    IppStatus status;
    ippiFilterScharrVertMaskBorderGetBufferSize(roiSize, ippMskSize3x3, ipp8u, ipp16s, 1,
&bufferSize);
    pBuffer = ippsMalloc_8u(bufferSize);
    status = ippiFilterScharrVertMaskBorder_8u16s_C1R(pSrc + srcStep, srcStep, pDst, dstStep,
roiSize, ippMskSize3x3,
             borderType, 0, pBuffer);
    ippsFree(pBuffer);
    return status;
}
```

The result is as follows:

```
pDst -->
 16  32 1925 1925  32 -1117 -1127  22
 16  32 1995 1995  32 -1094 -1097  29
 16  32 2016 2016  32 -1088 -1088  32
 10  20 2019 2019  20 -1082 -1082  20
-10 -20 2029 2029 -20 -1062 -1062 -20
-16 -32 2032 2032 -32 -1056 -1056 -32
-16 -32 2032 2032 -32 -1056 -1056 -32
```

See Also

Borders in Neighborhood Operations

Regions of Interest in Intel IPP

User-defined Border Types

FilterScharrVertMaskBorderGetBufferSize Computes the size of the work buffer for the Scharr Vertical filter.

FilterSharpenBorderGetBufferSize

Computes the size of the work buffer for image sharpening.

Syntax

```
IppStatus ippiFilterSharpenBorderGetBufferSize(IppiSize dstRoiSize, IppiMaskSize mask,
IppDataType srcDataType, IppDataType dstDataType, int numChannels, int* pBufferSize);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`dstRoiSize`	Size of the destination ROI, in pixels.
`mask`	Predefined mask of `IppiMaskSize` type. Possible value is `ippMskSize3x3`.
`srcDataType`	Data type of the source image.
`dstDataType`	Data type of the destination image.
`numChannels`	Number of channels in the image. Possible values are 1, 3, or 4.
`pBufferSize`	Pointer to the size, in bytes, of the external buffer.

Description

This function computes the size, in bytes, of the external work buffer for the ippiFilterSharpenBorder function. The result is stored in the `pBufferSize` parameter.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error when pBufferSize is NULL.
ippStsSizeErr	Indicates an error when dstRoiSize has a field with a zero or negative value.
ippStsMaskSizeErr	Indicates an error when mask has an illegal value.
ippStsDataTypeErr	Indicates an error when srcDataType or dstDataType has an illegal value.
ippStsNumChannelErr	Indicates an error when numChannels has an illegal value.

See Also

FilterSharpenBorder Performs image sharpening with a high-pass filter.

FilterSharpenBorder

Performs image sharpening with a high-pass filter.

Syntax

Case 1: Operating on one-channel data

```
IppStatus ippiFilterSharpenBorder_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, Ipp8u
borderValue, Ipp8u* pBuffer);
```

```
IppStatus ippiFilterSharpenBorder_16s_C1R(const Ipp16s* pSrc, int srcStep, Ipp16s*
pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType,
Ipp16s borderValue, Ipp8u* pBuffer);
```

```
IppStatus ippiFilterSharpenBorder_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f*
pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType,
Ipp32f borderValue, Ipp8u* pBuffer);
```

Case 2: Operating on multi-channel data

```
IppStatus ippiFilterSharpenBorder_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, const
Ipp8u pBorderValue[3], Ipp8u* pBuffer);
```

```
IppStatus ippiFilterSharpenBorder_16s_C3R(const Ipp16s* pSrc, int srcStep, Ipp16s*
pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType,
const Ipp16s pBorderValue[3], Ipp8u* pBuffer);
```

```
IppStatus ippiFilterSharpenBorder_32f_C3R(const Ipp32f* pSrc, int srcStep, Ipp32f*
pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType,
const Ipp32f pBorderValue[3], Ipp8u* pBuffer);
```

```
IppStatus ippiFilterSharpenBorder_8u_AC4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, const
Ipp8u pBorderValue[3], Ipp8u* pBuffer);
```

```
IppStatus ippiFilterSharpenBorder_16s_AC4R(const Ipp16s* pSrc, int srcStep, Ipp16s*
pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType,
const Ipp16s pBorderValue[3], Ipp8u* pBuffer);

IppStatus ippiFilterSharpenBorder_32f_AC4R(const Ipp32f* pSrc, int srcStep, Ipp32f*
pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType,
const Ipp32f pBorderValue[3], Ipp8u* pBuffer);

IppStatus ippiFilterSharpenBorder_8u_C4R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, const
Ipp8u pBorderValue[4], Ipp8u* pBuffer);

IppStatus ippiFilterSharpenBorder_16s_C4R(const Ipp16s* pSrc, int srcStep, Ipp16s*
pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType,
const Ipp16s pBorderValue[4], Ipp8u* pBuffer);

IppStatus ippiFilterSharpenBorder_32f_C4R(const Ipp32f* pSrc, int srcStep, Ipp32f*
pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType,
const Ipp32f pBorderValue[4], Ipp8u* pBuffer);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance, in bytes, between the starting points of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance, in bytes, between the starting points of consecutive lines in the destination image.
`dstRoiSize`	Size of the source and destination ROI, in pixels.
`mask`	Predefined mask of `IppiMaskSize`. Possible value is `ippMskSize3x3`.
`borderType`	Type of border. Possible values are: • `ippBorderConst` • Values of all border pixels are set to constant. • `ippBorderRepl` • Border is replicated from the edge pixels. • `ippBorderInMem` • Border is obtained from the source image pixels in memory.

borderValue	Constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.
pBorderValue[3], pBorderValue[4]	Pointer to constant values to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.
pBuffer	Pointer to the work buffer.

Description

Before using this function, you need to compute the size of the work buffer pBuffer using the ippiFilterSharpenBorderGetBufferSize function.

This function operates with ROI (see Regions of Interest in Intel IPP).

This function applies a high-pass filter to the pSrc source image ROI. The size of the source image ROI is equal to the destination image ROI size dstRoiSize. The values of border pixels are assigned in accordance with the borderType and borderValue parameters. The kernel of the filter is a matrix of 3x3 size. The kernel has the following value:

```
-1/8  -1/8  -1/8
-1/8  16/8  -1/8
-1/8  -1/8  -1/8
```

The anchor cell is the center cell of the kernel, highlighted in red.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when dstRoiSize is negative, or equal to zero.
ippStsBorderErr	Indicates an error when mask has an illegal value.
ippStsBorderErr	Indicates an error when borderType has an illegal value.

See Also

Borders in Neighborhood Operations

Regions of Interest in Intel IPP

User-defined Border Types

FilterSharpenBorder Performs image sharpening with a high-pass filter.

FilterSobelGetBufferSize

Computes the size of the work buffer for the Sobel filter.

Syntax

```
IppStatus ippiFilterSobelGetBufferSize (IppiSize dstRoiSize, IppiMaskSize mask,
IppNormType normType, IppDataType srcDataType, IppDataType dstDataType, int
numChannels, int* pBufferSize);
```

```
IppStatus ippiFilterSobelGetBufferSize_T (IppiSize dstRoiSize, IppiMaskSize mask,
IppNormType normType, IppDataType srcDataType, IppDataType dstDataType, int
numChannels, int* pBufferSize);
```

Include Files

`ippi.h`

`ippi_tl.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`dstRoiSize`	Size of the destination ROI in pixels.
`mask`	Predefined mask of IppiMaskSize type. Possible values are `ippMskSize3x3` or `ippMskSize5x5`.
`normType`	Normalization mode of IppNormType type.
`srcDataType`	Data type of the source image.
`dstDataType`	Data type of the destination image.
`numChannels`	Number of channels in the image. Possible value is 1.
`pBufferSize`	Pointer to the size of the external work buffer.

Description

The `ippiFilterSobelGetBufferSize` function computes the size, in bytes, of the external work buffer needed for the ippiFilterSobel function. The result is stored in the `pBufferSize` parameter.

For an example on how to use this functions, see the code example provided with the ippiFilterSobel function description.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when `pBufferSize` is NULL.
`ippStsSizeErr`	Indicates an error when `dstRoiSize` is less than, or equal to zero.
`ippStsMaskSizeErr`	Indicates an error when `mask` has an illegal value.

ippStsBadArgErr	Indicates an error when normType has an illegal value.
ippStsDataTypeErr	Indicates an error when srcDataType or dstDataType has an illegal value.
ippStsNumChannelsError	Indicates an error when numChannels has an illegal value.

See Also

FilterSobel Filters an image using a Sobel filter.

FilterSobel

Filters an image using a Sobel filter.

Syntax

```
IppStatus ippiFilterSobel_<mod>(const Ipp<srcdatatype>* pSrc, int srcStep,
Ipp<dstdatatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize maskSize,
IppNormType normType, IppiBorderType borderType, Ipp<srcdatatype> borderValue, Ipp8u*
pBuffer);
```

Supported values for mod:

8u16s_C1R	16s32f_C1R	16u32f_C1R	32f_C1R

```
IppStatus ippiFilterSobel_<mod>_T(const Ipp8u* pSrc, int srcStep, Ipp16s* pDst, int
dstStep, IppiSize dstRoiSize, IppiMaskSize maskSize, IppNormType normType,
IppiBorderType borderType, Ipp8u borderValue, Ipp8u* pBuffer);
```

Supported values for mod:

8u16s_C1R_T

Include Files

ippi.h

ippi_tl.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance, in bytes, between the starting points of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.

dstStep	Distance, in bytes, between the starting points of consecutive lines in the destination image.
dstRoiSize	Size of the source and destination ROI in pixels.
maskSize	Size of the predefined mask. Possible values are ippMskSize3x3 or ippMskSize5x5.
normType	Normalization mode of IppNormType.
borderType	Type of border. Possible values are: • ippBorderConst • Values of all border pixels are set to constant. • ippBorderRepl • Border is replicated from the edge pixels. • ippBorderInMem • Border is obtained from the source image pixels in memory.
borderValue	Constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.
pBuffer	Pointer to the work buffer.

Description

Before using this function, you need to compute the size of the work buffer pBuffer using the ippiFilterSobelBufferSize function.

This function applies a Sobel filter to the source image with the specified kernel size and normalization type. The values of border pixels are assigned in accordance with the borderType and borderValue parameters. The kernel of the filter is a matrix of 3x3 or 5x5 size depending on the maskSize value. The formulas below describe the algorithm for the 3x3 and 5x5 Sobel operators.

3x3 Sobel operator:

$$G_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} * A \quad \text{and} \quad G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * A$$

5x5 Sobel operator:

$$G_x = \begin{bmatrix} 1 & 2 & 0 & -2 & -1 \\ 4 & 8 & 0 & -8 & -4 \\ 6 & 12 & 0 & -12 & -6 \\ 4 & 8 & 0 & -8 & -4 \\ 1 & 2 & 0 & -2 & -1 \end{bmatrix} * A \quad \text{and} \quad G_y = \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 2 & 8 & 12 & 8 & 2 \\ 0 & 0 & 0 & 0 & 0 \\ -2 & -8 & -12 & -8 & -2 \\ -1 & -4 & -6 & -4 & -1 \end{bmatrix} * A$$

where

- A is the source image
- * is the 2D convolution operator
- G_x and G_y are horizontal and vertical magnitude of the source image, respectively

Sobel filter output G, as overall gradient magnitude, is generated through L1 and L2 normalization of G_x and G_y.

L1 normalization:

$$G = |G_x| + |G_y|$$

L2 normalization:

$$G = \sqrt{G_x^2 + G_y^2}$$

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error when `dstRoiSize` has a field with a zero or negative value.
`ippStsMaskSizeErr`	Indicates an error when `maskSize` has an illegal value.
`ippStsStepErr`	Indicates an error when `srcStep` or `dstStep` is less than, or equal to zero.
`ippStsNotEvenStepErr`	Indicates an error when one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
`ippStsBorderErr`	Indicates an error when `borderType` has an illegal value.

Example

The code example below demonstrates how to use the `ippiFilterSobelGetBufferSize` and `ippiFilterSobel_8u16s_C1R` functions.

```
IppStatus filter_sobel_8u16s_c1( void ) {
    Ipp8u pSrc[9*8] =
    {
        0, 1, 2, 120, 121, 122, 50, 51, 52,
        1, 2, 3, 121, 122, 123, 52, 52, 53,
```

```
        3, 4, 5, 130, 131, 132, 63, 64, 65,
        4, 5, 6, 131, 132, 133, 64, 65, 66,
        5, 6, 7, 132, 133, 134, 65, 66, 67,
        8, 7, 6, 134, 133, 132, 67, 66, 65,
        7, 6, 5, 133, 132, 131, 66, 65, 64,
        6, 5, 4, 132, 131, 130, 65, 64, 63
    };
    Ipp16s   pDst[8*7];
    IppiSize roiSize = {8, 7};
    IppiMaskSize mask = ippMskSize3x3;
    IppiBorderType borderType = ippBorderConst | ippBorderInMemTop | ippBorderInMemRight;
    int     srcStep = 9 * sizeof(Ipp8u);
    int     dstStep = 8 * sizeof(Ipp16s);
    int     bufferSize;
    IppStatus status;
    Ipp8u   *pBuffer;
    IppNormType normType = ippNormL1;


    ippiFilterSobelGetBufferSize(roiSize, mask, normType, ipp8u, ipp16s, 1, &bufferSize);
    pBuffer = ippsMalloc_8u(bufferSize);
    status = ippiFilterSobel_8u16s_C1R(pSrc+srcStep, srcStep, pDst, dstStep, roiSize, mask,
normType, borderType, 33, pBuffer);
    ippsFree(pBuffer);


    return status;
}
```

The result is as follows:

```
pDst -->
    132     20    502    516     48    322    330     58
    126     20    516    530     48    316    322     58
    118     16    512    512     16    280    280     16
    118     12    510    512      8    272    276      8
    110      4    510    508      8    272    268      8
    114     16    516    516     16    272    272     16
    162    114    398    652    402    526    394    134
```

See Also

Borders in Neighborhood Operations

Regions of Interest in Intel IPP

User-defined Border Types

FilterSobelGetBufferSize Computes the size of the work buffer for the Sobel filter.

FilterSobelHorizBorderGetBufferSize

Computes the size of the work buffer for the Sobel Horizontal filter.

Syntax

```
IppStatus ippiFilterSobelHorizBorderGetBufferSize (IppiSize dstRoiSize, IppiMaskSize
mask, IppDataType srcDataType, IppDataType dstDataType, int numChannels, int*
pBufferSize);
```

```
IppStatus ippiFilterSobelHorizBorderGetBufferSize_T (IppiSize dstRoiSize, IppiMaskSize
mask, IppDataType srcDataType, IppDataType dstDataType, int numChannels, int*
pBufferSize);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`dstRoiSize`	Size of the destination ROI in pixels.
`mask`	Predefined mask of `IppiMaskSize` type. Possible values are `ippMskSize3x3` or `ippMskSize5x5`.
`srcDataType`	Data type of the source image.
`dstDataType`	Data type of the destination image.
`numChannels`	Number of channels in the image. Possible value is 1.
`pBufferSize`	Pointer to the size of the external work buffer.

Description

The `ippiFilterSobelHorizBorderGetBufferSize` function computes the size, in bytes, of the external work buffer needed for the `ippiFilterSobelHorizBorder` function. The result is stored in the `pBufferSize` parameter.

For an example on how to use this functions, see the code example provided with the `ippiFilterSobelHorizBorder` function description.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when `pBufferSize` is NULL.
`ippStsSizeErr`	Indicates an error when `dstRoiSize` is negative, or equal to zero.
`ippStsMaskSizeErr`	Indicates an error when `mask` has an illegal value.
`ippStsDataTypeErr`	Indicates an error when `srcDataType` or `dstDataType` has an illegal value.
`ippStsNumChannelsError`	Indicates an error when `numChannels` has an illegal value.

See Also

FilterSobelHorizBorder Filters an image using a horizontal Sobel filter.

FilterSobelHorizBorder

Filters an image using a horizontal Sobel filter.

Syntax

```
IppStatus ippiFilterSobelHorizBorder_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask,
IppiBorderType borderType, Ipp<srcDatatype> borderValue, Ipp8u* pBuffer);
```

Supported values for `mod`:

`8u16s_C1R`	`16s_C1R`	`32f_C1R`

```
IppStatus ippiFilterSobelHorizBorder_<mod>_T(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask,
IppiBorderType borderType, Ipp<srcDatatype> borderValue, Ipp8u* pBuffer);
```

Supported values for `mod`:

`8u16s_C1R_T`

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance, in bytes, between the starting points of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance, in bytes, between the starting points of consecutive lines in the destination image.
`dstRoiSize`	Size of the source and destination ROI in pixels.
`mask`	Predefined mask of `IppiMaskSize`. Possible values are `ippMskSize3x3` or `ippMskSize5x5`.
`borderType`	Type of border. Possible values are: • `ippBorderConst` • Values of all border pixels are set to constant. • `ippBorderRepl` • Border is replicated from the edge pixels. • `ippBorderMirror` • Mirrored border is used. • `ippBorderMirrorR`

	• Mirrored border with replication is used. • `ippBorderInMem` • Border is obtained from the source image pixels in memory.
`borderValue`	Constant value to assign to pixels of the constant border. This parameter is applicable only to the `ippBorderConst` border type.
`pBuffer`	Pointer to the work buffer.

Description

Before using this function, you need to compute the size of the work buffer `pBuffer` using the `FilterSobelHorizBorderGetBufferSize` function.

This function operates with ROI.

This function applies a horizontal Sobel filter to the `pSrc` source image ROI. The size of the source image ROI is equal to the destination image ROI size `dstRoiSize`. The values of border pixels are assigned in accordance with the `borderType` and `borderValue` parameters. The kernel of the filter is a matrix of 3x3 or 5x5 size depending on the `mask` value. The kernels have the following values:

```
1   4   6   4   1
1   2   1                2   8   12  8   2
0   0   0       or       0   0   0   0   0
-1  -2  -1               -2  -8  -12 -8  -2
-1  -4   -6  -4  -1
```

The anchor cell is the center cell of the kernel, highlighted in red.

This filter enhances horizontal edges of an image.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error when `dstRoiSize` is negative, or equal to zero.
`ippStsStepErr`	Indicates an error when `srcStep` or `dstStep` is negative, or equal to zero.
`ippStsNotEvenStepErr`	Indicates an error when one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
`ippStsBorderErr`	Indicates an error when `borderType` has an illegal value.

Example

The code example below demonstrates how to use the `ippiFilterSobelHorizBorderGetBufferSize` and `ippiFilterSobelHorizBorder_8u16s_C1R` functions to filter an image with the Sobel horizontal kernel.

```
IppStatus fix_sobelhoriz_8u16( void ) {
    Ipp8u pSrc[9*8] =
        {
            0, 1, 2, 120, 121, 122, 50, 51, 52,
            1, 2, 3, 121, 122, 123, 52, 52, 53,
            3, 4, 5, 130, 131, 132, 63, 64, 65,
            4, 5, 6, 131, 132, 133, 64, 65, 66,
            5, 6, 7, 132, 133, 134, 65, 66, 67,
                  8, 7, 6, 134, 133, 132, 67, 66, 65,
                  7, 6, 5, 133, 132, 131, 66, 65, 64,
                  6, 5, 4, 132, 131, 130, 65, 64, 63
         };
    Ipp16s  pDst[8*7];
    Ipp8u   *pBuffer;
    IppiSize roiSize = {8, 7};
    IppiBorderType borderType = ippBorderRepl | ippBorderInMemTop | ippBorderInMemRight;
    int    srcStep = 9 * sizeof(Ipp8u);
    int    dstStep = 8 * sizeof(Ipp16s);
    int    bufferSize;
    IppStatus status;
    ippiFilterSobelHorizBorderGetBufferSize(roiSize, ippMskSize3x3, ipp8u, ipp16s, 1,
&bufferSize);
    pBuffer = ippsMalloc_8u(bufferSize);
    status = ippiFilterSobelHorizBorder_8u16s_C1R(pSrc + srcStep, srcStep, pDst, dstStep,
roiSize, ippMskSize3x3,
             borderType, 0, pBuffer);
    ippsFree(pBuffer);
    return status;
}
```

The result is as follows:

```
pDst -->
 12  12  19  33  40  43  49  52
 12  12  19  33  40  42  47  51
  8   8   8   8   8   8   8   8
 14   8   5   7   4   2   6   4
  6   0  -3  -1  -4  -6  -2  -4
 -8  -8  -8  -8  -8  -8  -8  -8
 -4  -4  -4  -4  -4  -4  -4  -4
```

See Also

Borders in Neighborhood Operations

Regions of Interest in Intel IPP

User-defined Border Types

FilterSobelHorizBorderGetBufferSize Computes the size of the work buffer for the Sobel Horizontal filter.

FilterSobelHorizSecondBorderGetBufferSize

Computes the size of the work buffer for the Sobel Horizontal (second derivative) filter.

Syntax

```
IppStatus ippiFilterSobelHorizSecondBorderGetBufferSize (IppiSize dstRoiSize,
IppiMaskSize mask, IppDataType srcDataType, IppDataType dstDataType, int numChannels,
int* pBufferSize);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`dstRoiSize`	Size of the destination ROI in pixels.
`mask`	Predefined mask of `IppiMaskSize` type. Possible values are `ippMskSize3x3` or `ippMskSize5x5`.
`srcDataType`	Data type of the source image.
`dstDataType`	Data type of the destination image.
`numChannels`	Number of channels in the image. Possible value is 1.
`pBufferSize`	Pointer to the size of the external work buffer.

Description

The `ippiFilterSobelHorizSecondBorderGetBufferSize` function computes the size, in bytes, of the external work buffer needed for the ippiFilterSobelHorizSecondBorder function. The result is stored in the `pBufferSize` parameter.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when `pBufferSize` is NULL.
`ippStsSizeErr`	Indicates an error when `dstRoiSize` is negative, or equal to zero.
`ippStsMaskSizeErr`	Indicates an error when `mask` has an illegal value.
`ippStsDataTypeErr`	Indicates an error when `srcDataType` or `dstDataType` has an illegal value.
`ippStsNumChannelsError`	Indicates an error when `numChannels` has an illegal value.

See Also

FilterSobelHorizSecondBorder Applies horizontal (second derivative) Sobel filter with border.

FilterSobelHorizSecondBorder

Applies horizontal (second derivative) Sobel filter with border.

Syntax

```
IppStatus ippiFilterSobelHorizSecondBorder_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, Ipp<srcDatatype> borderValue, Ipp8u* pBuffer);
```

Supported values for `mod`:

`8u16s_C1R`	`32f_C1R`

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image.
`dstRoiSize`	Size of the source and destination image ROI.
`mask`	Type of the filter kernel.
`borderType`	Type of border (see Borders in Neighborhood Operations); following values are possible: • `ippBorderConst` • Values of all border pixels are set to constant. • `ippBorderRepl` • Replicated border is used. • `ippBorderMirror` • Mirrored border is used • `ippBorderMirrorR` • Mirrored border with replication is used
`borderValue`	The constant value to assign to the pixels in the constant border (not applicable for other border's type).
`pBuffer`	Pointer to the working buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP). This function applies the second derivative horizontal Sobel filter (`y – derivative`) to the source image `pSrc` and stores results to the destination image of the same size `pDst`. Source image can be used as the destination image if they both have the same data type. The values of border pixels are assigned in accordance with the `borderType` and `borderValue` parameters. The kernel of this filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter `mask`.The kernels have the following values with the anchor in the center cell (red):

```
1   4   6   4   1
1   2   1                   0   0   0   0   0
-2  -4  -2        or        -2  -8 -12  -8  -2
1   2   1                   0   0   0   0   0
1   4   6   4   1
```

The function requires the working buffer `pBuffer` which size should be computed by the function ippiFilterSobelHorizSecondBorderGetBufferSize beforehand.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize` has a field with a zero or negative value.
`ippStsStepErr`	Indicates an error condition if `srcStep` or `dstStep` is less than `roiSize.width` * <pixelSize>
`ippStsNotEvenStepErr`	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
`ippStsBadArgErr`	Indicates an error if `borderType` or `divisor` has a wrong value.
`ippStsMaskErr`	Indicates an error condition if `mask` has a wrong value.

FilterSobelVertBorderGetBufferSize

Computes the size of the work buffer for the Sobel Vertical filter.

Syntax

```
IppStatus ippiFilterSobelVertBorderGetBufferSize (IppiSize dstRoiSize, IppiMaskSize mask, IppDataType srcDataType, IppDataType dstDataType, int numChannels, int* pBufferSize);
```

```
IppStatus ippiFilterSobelVertBorderGetBufferSize_T (IppiSize dstRoiSize, IppiMaskSize mask, IppDataType srcDataType, IppDataType dstDataType, int numChannels, int* pBufferSize);
```

Include Files

ippi.h

Parameters

dstRoiSize	Size of the destination ROI in pixels.
mask	Predefined mask of IppiMaskSize type. Possible values are ippMskSize3x3 or ippMskSize5x5.
srcDataType	Data type of the source image.
dstDataType	Data type of the destination image.
numChannels	Number of channels in the image. Possible value is 1.
pBufferSize	Pointer to the size of the external work buffer.

Description

The ippiFilterSobelVertBorderGetBufferSize function computes the size, in bytes, of the external work buffer needed for the ippiFilterSobelVertBorder function. The result is stored in the pBufferSize parameter.

For an example on how to use this function, see the code example provided with the ippiFilterSobelVertBorder function description.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pBufferSize is NULL.
ippStsSizeErr	Indicates an error when dstRoiSize is negative, or equal to zero.
ippStsMaskSizeErr	Indicates an error when mask has an illegal value.
ippStsDataTypeErr	Indicates an error when srcDataType or dstDataType has an illegal value.
ippStsNumChannelsError	Indicates an error when numChannels has an illegal value.

See Also

FilterSobelVertBorder Filters an image using a vertical Sobel filter.

FilterSobelVertBorder

Filters an image using a vertical Sobel filter.

Syntax

```
IppStatus ippiFilterSobelVertBorder_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask,
IppiBorderType borderType, Ipp<srcDatatype> borderValue, Ipp8u* pBuffer);
```

Supported values for `mod`:

`8u16s_C1R`	`16s_C1R`	`32f_C1R`

```
IppStatus ippiFilterSobelVertBorder_<mod>_T(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask,
IppiBorderType borderType, Ipp<srcDatatype> borderValue, Ipp8u* pBuffer);
```

Supported values for `mod`:

`8u16s_C1R_T`

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance, in bytes, between the starting points of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance, in bytes, between the starting points of consecutive lines in the destination image.
`dstRoiSize`	Size of the source and destination ROI in pixels.
`mask`	Predefined mask of `IppiMaskSize`. Possible values are `ippMskSize3x3` or `ippMskSize5x5`.
`borderType`	Type of border. Possible values are: • `ippBorderConst` • Values of all border pixels are set to constant. • `ippBorderRepl` • Border is replicated from the edge pixels. • `ippBorderMirror` • Mirrored border is used. • `ippBorderMirrorR` • Mirrored border with replication is used. • `ippBorderInMem` • Border is obtained from the source image pixels in memory.
`borderValue`	Constant value to assign to pixels of the constant border. This parameter is applicable only to the `ippBorderConst` border type.
`pBuffer`	Pointer to the work buffer.

Description

Before using this function, you need to compute the size of the work buffer `pBuffer` using the `FilterSobelVertBorderGetBufferSize` function.

This function operates with ROI.

This function applies a vertical Sobel filter to the `pSrc` source image ROI. The size of the source image ROI is equal to the destination image ROI size `dstRoiSize`. The values of border pixels are assigned in accordance with the `borderType` and `borderValue` parameters. The kernel of the filter is a matrix of 3x3 or 5x5 size depending on the `mask` value. The kernels have the following values:

```
-1  -2   0   2   1
-1   0   1                -4  -8   0   8   4
-2   0   2       or       -6  -12  0   12   6
-1   0   1                -4  -8   0   8  4
-1  -2   0   2  1
```

The anchor cell is the center cell of the kernel, highlighted in red.

This filter enhances vertical edges of an image.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error when `dstRoiSize` is negative, or equal to zero.
`ippStsStepErr`	Indicates an error when `srcStep` or `dstStep` is negative, or equal to zero.
`ippStsNotEvenStepErr`	Indicates an error when one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
`ippStsBorderErr`	Indicates an error when `borderType` has an illegal value.

Example

The code example below demonstrates how to use the `ippiFilterSobelVertBorderGetBufferSize` and `ippiFilterSobelVertBorder` functions.

```
IppStatus fix_sobel_8u16( void ) {
    Ipp8u pSrc[9*8] =
        {
            0, 1, 2, 120, 121, 122, 50, 51, 52,
            1, 2, 3, 121, 122, 123, 52, 52, 53,
            3, 4, 5, 130, 131, 132, 63, 64, 65,
            4, 5, 6, 131, 132, 133, 64, 65, 66,
            5, 6, 7, 132, 133, 134, 65, 66, 67,
         8, 7, 6, 134, 133, 132, 67, 66, 65,
         7, 6, 5, 133, 132, 131, 66, 65, 64,
         6, 5, 4, 132, 131, 130, 65, 64, 63
```

```
          };
    Ipp16s  pDst[8*7];
    Ipp8u   *pBuffer;
    IppiSize roiSize = {8, 7};
    IppiBorderType borderType = ippBorderRepl | ippBorderInMemTop | ippBorderInMemRight;
    int    srcStep = 9 * sizeof(Ipp8u);
    int    dstStep = 8 * sizeof(Ipp16s);
    int    bufferSize;
    IppStatus status;
    ippiFilterSobelVertBorderGetBufferSize(roiSize, ippMskSize3x3, ipp8u, ipp16s, 1,
&bufferSize);
    pBuffer = ippsMalloc_8u(bufferSize);
    status = ippiFilterSobelVertBorder_8u16s_C1R(pSrc + srcStep, srcStep, pDst, dstStep,
roiSize, ippMskSize3x3,
            borderType, 0, pBuffer);
    ippsFree(pBuffer);
    return status;
}
```

The result is as follows:

```
pDst ->
 -4  -8 -483 -483  -8 279 281  -6
 -4  -8 -497 -497  -8 274 275  -7
 -4  -8 -504 -504  -8 272 272  -8
 -2  -4 -505 -505  -4 270 270  -4
  2   4 -507 -507   4 266 266   4
  4   8 -508 -508   8 264 264   8
  4   8 -508 -508   8 264 264   8
```

See Also

Borders in Neighborhood Operations

Regions of Interest in Intel IPP

User-defined Border Types

FilterSobelVertBorderGetBufferSize Computes the size of the work buffer for the Sobel Vertical filter.

FilterSobelVertSecondBorderGetBufferSize

Computes the size of the work buffer for the Sobel vertical (second derivative) filter.

Syntax

```
IppStatus ippiFilterSobelVertSecondBorderGetBufferSize (IppiSize dstRoiSize,
IppiMaskSize mask, IppDataType srcDataType, IppDataType dstDataType, int numChannels,
int* pBufferSize);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

dstRoiSize	Size of the destination ROI in pixels.
mask	Predefined mask of IppiMaskSize type. Possible values are ippMskSize3x3 or ippMskSize5x5.
srcDataType	Data type of the source image.
dstDataType	Data type of the destination image.
numChannels	Number of channels in the image. Possible value is 1.
pBufferSize	Pointer to the size of the external work buffer.

Description

The ippiFilterSobelVertSecondBorderGetBufferSize function computes the size, in bytes, of the external work buffer needed for the ippiFilterSobelVertSecondBorder function. The result is stored in the pBufferSize parameter.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pBufferSize is NULL.
ippStsSizeErr	Indicates an error when dstRoiSize is negative, or equal to zero.
ippStsMaskSizeErr	Indicates an error when mask has an illegal value.
ippStsDataTypeErr	Indicates an error when srcDataType or dstDataType has an illegal value.
ippStsNumChannelsError	Indicates an error when numChannels has an illegal value.

See Also

FilterSobelVertSecondBorder Applies vertical (second derivative) Sobel filter with border.

FilterSobelNegVertBorderGetBufferSize

Computes the size of the work buffer for the Sobel vertical filter.

Syntax

```
IppStatus ippiFilterSobelNegVertBorderGetBufferSize (IppiSize dstRoiSize, IppiMaskSize mask, IppDataType srcDataType, IppDataType dstDataType, int numChannels, int* pBufferSize);
```

Include Files

ippi.h

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`dstRoiSize`	Size of the destination ROI in pixels.
`mask`	Predefined mask of `IppiMaskSize` type. Possible values are `ippMskSize3x3` or `ippMskSize5x5`.
`srcDataType`	Data type of the source image.
`dstDataType`	Data type of the destination image.
`numChannels`	Number of channels in the image. Possible value is 1.
`pBufferSize`	Pointer to the size of the external work buffer.

Description

The `ippiFilterSobelVertSecondBorderGetBufferSize` function computes the size, in bytes, of the external work buffer needed for the ippiFilterSobelNegVertBorder function. The result is stored in the `pBufferSize` parameter.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when `pBufferSize` is NULL.
`ippStsSizeErr`	Indicates an error when `dstRoiSize` is negative, or equal to zero.
`ippStsMaskSizeErr`	Indicates an error when `mask` has an illegal value.
`ippStsDataTypeErr`	Indicates an error when `srcDataType` or `dstDataType` has an illegal value.
`ippStsNumChannelsError`	Indicates an error when `numChannels` has an illegal value.

See Also

FilterSobelNegVertSecondBorder Applies vertical Sobel filter with border.

FilterSobelNegVertBorder

Applies vertical Sobel filter with border.

Syntax

```
IppStatus ippiFilterSobelNegVertBorder_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask,
IppiBorderType borderType, Ipp<srcDatatype> borderValue, Ipp8u* pBuffer);
```

Supported values for `mod`:

`8u16s_C1R`	`32f_C1R`

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image.
`dstRoiSize`	Size of the destination image ROI.
`mask`	Type of the filter kernel.
`borderType`	Type of border (see Borders in Neighborhood Operations); following values are possible: • `ippBorderConst` • Values of all border pixels are set to constant. • `ippBorderRepl` • Replicated border is used. • `ippBorderMirror` • Mirrored border is used • `ippBorderMirrorR` • Mirrored border with replication is used
`borderValue`	The constant value to assign to the pixels in the constant border (not applicable for other border's type).
`pBuffer`	Pointer to the working buffer.

Description

These functions operate with ROI (see Regions of Interest in Intel IPP). These functions apply the vertical Sobel filter (`x – derivative`) to the source image ROI `pSrc` and stores results to the destination image ROI of the same size `pDst`. Source image can be used as the destination image if they have the same data type. The values of border pixels are assigned in accordance with the `borderType` and `borderValue` parameters. The kernel of this filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter `mask`. The anchor cell is the center cell of the kernel (red).

The function `ippiFilterSobelVertBorde` uses the kernels with the following coefficients:

```
-1   -2   0   2   1

-1   0   1               -4   -8   0   8   4

-2   0   2       or      -6  -12   0  12   6

-1   0   1               -4   -8   0   8   4

-1   -2   0   2   1
```

The function `ippiFilterSobelNegVertBoreder` uses the kernels which coefficients are the same in magnitude but opposite in sign:

```
1   2   0  -2  -1

1   0  -1               4   8   0  -8  -4

2   0  -2       or      6  12   0 -12  -6

1   0  -1               4   8   0  -8  -4

1   2   0  -2  -1
```

Before using this function, compute the size of the work buffer `pBuffer` using the FilterSobelNegVertBorderGetBufferSize function.

Example shows how the function `ippiFilterSobelNegVertBorder_8u16s_C1R` can be used for edge detection.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize` has a field with a zero or negative value.
`ippStsStepErr`	Indicates an error condition if `srcStep` or `dstStep` is less than `roiSize.width` * <`pixelSize`>
`ippStsNotEvenStepErr`	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
`ippStsBadArgErr`	Indicates an error if `borderType` or `divisor` has a wrong value.
`ippStsMaskErr`	Indicates an error condition if `mask` has a wrong value.

See Also

User-defined Border Types

FilterSobelVertSecondBorder

Applies vertical (second derivative) Sobel filter with border.

Syntax

```
IppStatus ippiFilterSobelVertSecondBorder_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiMaskSize mask, IppiBorderType borderType, Ipp<srcDatatype> borderValue, Ipp8u* pBuffer);
```

Supported values for `mod`:

`8u16s_C1R`	`32f_C1R`

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image.
`dstRoiSize`	Size of the destination image ROI.
`mask`	Type of the filter kernel.
`borderType`	Type of border (see Borders in Neighborhood Operations); following values are possible: • `ippBorderConst` • Values of all border pixels are set to constant. • `ippBorderRepl` • Replicated border is used. • `ippBorderMirror` • Mirrored border is used • `ippBorderMirrorR` • Mirrored border with replication is used
`borderValue`	The constant value to assign to the pixels in the constant border (not applicable for other border's type).
`pBuffer`	Pointer to the working buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP). This function applies the second derivative vertical Sobel filter (`x - derivative`) to the source image `pSrc` and stores results to the destination image of the same size `pDst`. Source image can be used as the destination image if they both

have the same data type. The values of border pixels are assigned in accordance with the `borderType` and `borderValue` parameters. The kernel of this filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter `mask`. The kernels have the following values with the anchor in the center cell (red):

```
1   0  -2   0   1
1  -2   1                4   0  -8   0   4
2  -4   2       or       6   0 -12   0   6
1  -2   1                4   0  -8   0   4
1   0  -2   0   1
```

The function requires the working buffer `pBuffer` which size should be computed by the function ippiFilterSobelVertSecondBorderGetBufferSize beforehand.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize` has a field with a zero or negative value.
`ippStsStepErr`	Indicates an error condition if `srcStep` or `dstStep` is less than `roiSize.width` * <`pixelSize`>
`ippStsNotEvenStepErr`	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
`ippStsBadArgErr`	Indicates an error if `borderType` or `divisor` has a wrong value.
`ippStsMaskErr`	Indicates an error condition if `mask` has a wrong value.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

// Perform linear filtering of an image using one of
```

```
//   predefined convolution kernels.
//      ippiFilterSobelVertSecondBorderGetBufferSize
//      ippiFilterSobelVertSecondBorder_8u16s_C1R

#include <stdio.h>
#include "ipp.h"

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define
check_sts(st)
                                  \
    if ((st) !=
ippStsNoErr)
                          \
        goto exitLine; /* Go to Exit if Intel(R) Integrated Performance Primitives (Intel(R)
IPP) function returned status different from \
                           ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main(void)
{

    IppStatus status = ippStsNoErr;
    Ipp8u pSrc[8 * 8] = {4, 4, 4, 4, 4, 4, 4, 4, 4, 3, 3, 3, 3, 3, 3, 4, 4, 3, 2, 2, 2, 2, 3, 4,
4, 3, 2, 1, 1, 2, 3, 4,
                         4, 3, 2, 1, 1, 2, 3, 4, 4, 3, 2, 2, 2, 2, 3, 4, 4, 3, 3, 3, 3, 3, 3, 4,
4, 4, 4, 4, 4, 4, 4, 4};
    Ipp16s pDst[8 * 8];
    IppiSize roiSize = {8, 8};
    Ipp8u borderValue = 3;
    int pBufferSize = 0;
    Ipp8u *pBuffer = NULL;

    check_sts(status = ippiFilterSobelVertSecondBorderGetBufferSize(roiSize, ippMskSize3x3,
ipp8u, ipp16s, 1, &pBufferSize));

    pBuffer = ippiMalloc_8u_C1(8, 8, &pBufferSize);

    check_sts(status = ippiFilterSobelVertSecondBorder_8u16s_C1R(pSrc, 8, pDst, 8 *
sizeof(Ipp16s), roiSize, ippMskSize3x3, ippBorderConst,
                                                                 borderValue, pBuffer));

    EXIT_MAIN
    ippsFree(pBuffer);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return status;
}
/*
    Result:

    4 4 4 4 4 4 4 4
    4 3 3 3 3 3 3 4
    4 3 2 2 2 2 3 4
    4 3 2 1 1 2 3 4
```

```
    4 3 2 1 1 2 3 4     pSrc
    4 3 2 2 2 2 3 4
    4 3 3 3 3 3 3 4
    4 4 4 4 4 4 4 4


    -4 1 0 0 0 0 1 -4
    -7 2 1 0 0 1 2 -7
    -8 1 2 1 1 2 1 -8
    -8 0 1 3 3 1 0 -8
    -8 0 1 3 3 1 0 -8   pDst
    -8 1 2 1 1 2 1 -8
    -7 2 1 0 0 1 2 -7
    -4 1 0 0 0 0 1 -4

*/
```

FilterSobelCrossGetBufferSize

Computes the size of the external buffer for the cross Sobel filter with border.

Syntax

`IppStatus ippiFilterSobelCrossGetBufferSize_<mod>(IppiSize roiSize, IppiMaskSize mask, int* pBufferSize);`

Supported values for `mod`:

`8u16s_C1R`	`32f_C1R`

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`roiSize`	Maximum size of the source and destination image ROI.
`mask`	Predefined mask of `IppiMaskSize` type.
`pBufferSize`	Pointer to the buffer size.

Description

This function computes the size of the external buffer that is required for the filter function ippiFilterSobelCrossBorder. The kernel of the filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter `mask` (see Table "Types of the Fixed Filter Functions"). This buffer `pBufferSize[0]` can be used to filter an image whose width and height are equal to or less than corresponding fields of `roiSize`.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if the pBufferSize pointer is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with a zero or negative value.
ippStsMaskSizeErr	Indicates an error condition if mask has a wrong value.

FilterSobelCrossBorder

Applies second derivative cross Sobel filter with border.

Syntax

```
IppStatus ippiFilterSobelCrossBorder_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize, IppiMaskSize mask,
IppiBorderType borderType, Ipp<srcDatatype> borderValue, Ipp8u* pBuffer);
```

Supported values for mod:

8u16s_C1R	32f_C1R

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination image ROI.
mask	Type of the filter kernel.
borderType	Type of border (see Borders in Neighborhood Operations); following values are possible:

	• ippBorderConst • Values of all border pixels are set to constant. • ippBorderRepl • Replicated border is used.
borderValue	The constant value to assign to the pixels in the constant border (not applicable for other border's type).
pBuffer	Pointer to the working buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function applies the second derivative cross Sobel filter (xy – derivative) to the source image pSrc and stores results to the destination image of the same size pDst. Source image can be used as the destination image if they both have the same data type. The values of border pixels are assigned in accordance with the borderType and borderValue parameters. The kernel of this filter is the matrix of either 3x3 or 5x5 size that is specified by the parameter mask. The kernels have the following values with the anchor in the center cell (red):

```
-1  -2   0   2   1
-1   0   1                -2  -4   0   4   2
0    0   0       or       0   0   0   0   0
1    0  -1                2   4   0  -4  -2
1    2   0  -2  -1
```

The function requires the working buffer pBuffer whose size should be computed by the function ippiFilterSobelCrossGetBufferSize beforehand.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with a zero or negative value.
ippStsStepErr	Indicates an error condition if srcStep or dstStep is less than roiSize.width * <pixelSize>
ippStsNotEvenStepErr	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
ippStsBadArgErr	Indicates an error if borderType or divisor has a wrong value.
ippStsMaskErr	Indicates an error condition if mask has a wrong value.

GenSobelKernel

Computes kernel for the Sobel filter.

Syntax

```
IppStatus ippiGenSobelKernel_16s(Ipp16s* pDst, int kernelSize, int dx, int sign);
IppStatus ippiGenSobelKernel_32f(Ipp32f* pDst, int kernelSize, int dx, int sign);
```

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pDst`	Pointer to the destination vector.
`kernelSize`	Size of the Sobel kernel.
`dx`	Order of derivative.
`sign`	Specifies signs of kernel elements.

Description

This function computes the one-dimensional Sobel kernel. Kernel coefficients are equal to coefficients of the polynomial

$$(1+x)^{kernelSize-dx-1}\cdot(x-1)^{dx}$$

If the `sign` parameter is negative, then signs of kernel coefficients are changed. Kernel calculated by this function can be used to filter images by a high order Sobel filter.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if the `pDst` pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `kernelSize` is less than 3 or is even.
`ippStsBadArgErr`	Indicates an error condition if `dx` is equal to or less than `kernelSize`, or `dx` is negative.

Deinterlacing Filters

This section describes functions that perform image deinterlacing.

- DeinterlaceFilterCAVT
- Median

DeinterlaceFilterCAVT

Performs deinterlacing of two-field image.

Syntax

```
IppStatus ippiDeinterlaceFilterCAVT_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst,
int dstStep, Ipp16u threshold, IppiSize roiSize);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image.
`threshold`	Threshold level value.
`roiSize`	Size of the source and destination image ROI.

Description

This function operates with ROI (see Regions of Interest in Intel IPP). This function performs deinterlacing of a two-field image, pointed to by `pSrc`, using content adaptive vertical temporal (CAVT) filtering.

The field pointed to by `pSrc` is copied to `pDst`, while the other field in the destination image, pointed to by `pDst + dstStep`, is the interpolated one. Note that you can set the pointers to the bottom left corner of the images and use negative steps to have the bottom field unchanged and the top one interpolated.

The `threshold` parameter is the edge detection threshold with the valid range [0-2041] regulating the probability of temporal interpolation: 0 means that all the pixels are interpolated only spatially, from the vertically neighbouring pixels of the copied field, 2041 - that combined spatial-temporal interpolation, involving the pixels from the modified field as well, is applied to all the pixels.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize.width is less than or equal to 0, or roiSize.height is odd or less than 8.

Median

Creates an image consisting of median values of three source images.

Syntax

```
IppStatus ippiMedian_8u_P3C1R(const Ipp8u* pSrc[3], int srcStep, Ipp8u* pDst, int dstStep, IppiSize size);
```

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc	Array of pointers to the ROI in each plane of the source image.
srcStep	Distance in bytes between starts of consecutive lines in each plane of the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
size	Size of the source and destination image ROI.

Description

This function operates with ROI (see Regions of Interest in Intel IPP). This function sets each pixel in the destination image ROI as the median value of correspondent pixels in the each plane of the source image.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.

ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if size has a field with zero or negative value.

Image Linear Transforms

This chapter describes the Intel® IPP image processing functions that perform linear transform operations on an image buffer.

These operations include Fast Fourier Transform (FFT), Discrete Fourier Transform (DFT), and Discrete Cosine Transform (DCT).

To speed up performance, linear transform functions use precomputed auxiliary data that is needed for computation of the transforms (that is, tables of twiddle factors for FFT functions). This data is calculated by the respective initialization functions and passed to the transform functions in context structures specific for each type of transform.

Intel IPP linear transform functions can use external work buffers for storing data and intermediate results, which eliminates the need to allocate and free internal memory buffers and thus helps to further increase function performance. To determine the required work buffer size, use one of the respective support functions specific for each transform type. In case when no external buffer is specified, the transform functions handle memory allocation internally.

All Intel IPP linear transform functions except DCT of 8x8 size work on images with floating-point data only.

- Fourier Transforms
- Windowing Functions
- Discrete Cosine Transforms

Fourier Transforms

Intel IPP functions that compute FFT and DFT can process both real and complex images. Function flavors operating on real data are distinguished by R suffix present in function-specific modifier of their full name, whereas complex flavors' names include C suffix (see Function Naming).

The results of computing the Fourier transform can be normalized by specifying the appropriate value of flag argument for context initialization. This parameter sets up a pair of matched normalization factors to be used in forward and inverse transforms as listed in the following table:

Value of flag Argument	Forward Transform normalization	Inverse Transform normalization
IPP_FFT_DIV_FWD_BY_N	1/MN	1
IPP_FFT_DIV_INV_BY_N	1	1/MN
IPP_FFT_DIV_BY_SQRTN	1/sqrt(MN)	1/sqrt(MN)
IPP_FFT_NODIV_BY_ANY	1	1

In this table, N and M denote the length of Fourier transform in the x- and y-directions, respectively (or, equivalently, the number of columns and rows in the 2D array being transformed).

For the FFT, these lengths must be integer powers of 2, that is $N=2^{orderX}$, $M=2^{orderY}$, where power exponentsare known as order of FFT.

For the DFT, N and M can take on arbitrary integer non-negative values.

- Real - Complex Packed (RCPack2D) Format
- FFTGetSize

- FFTInit
- FFTFwd
- FFTInv
- DFTGetSize
- DFTInit
- DFTFwd
- DFTInv
- MulPack
- MulPackConj
- Magnitude
- MagnitudePackGetBufferSize
- MagnitudePack
- Phase
- PhasePackGetBufferSize
- PhasePack
- PolarToCart
- PackToCplxExtend
- CplxExtendToPack

Real - Complex Packed (RCPack2D) Format

The forward Fourier transform of a real two-dimensional image data yields a matrix of complex results which has conjugate-symmetric properties. Intel IPP functions use packed format RCPack2D for storing and retrieving data of this type. Accordingly, real flavors of the inverse Fourier transform functions convert packed complex conjugate-symmetric data back to its real origin. The RCPack2D format exploits the complex conjugate symmetry of the transformed data to store only a half of the resulting Fourier coefficients. For the `N` by `M` transform, the respective FFT and DFT functions actually store real and imaginary parts of the complex Fourier coefficients `A(i,j)` for `i = 0,...,M-1; j = 0,... N/2` in a single real array of dimensions (`N`, `M`). The RCPack2D storage format is slightly different for odd and even `M` and is arranged in accordance with the following tables:

Re A(0,0)	Re A(0,1)	Im A(0,1)	...	Re A(0, (N-1)/2)	Im A(0, (N-1)/2)	Re A(0,N/2)
Re A(1,0)	Re A(1,1)	Im A(1,1)	...	Re A(1, (N-1)/2)	Im A(1, (N-1)/2)	Re A(1,N/2)
Im A(1,0)	Re A(2,1)	Im A(2,1)	...	Re A(2, (N-1)/2)	Im A(2, (N-1)/2)	Im A(1,N/2)
...	...	...	...	...	...	...
Re A(M/2,0)	Re A(M-2,1)	Im A(M-2,1)	...	Re A(M-2, (N-1)/2)	Im A(M-2, (N-1)/2)	Re A(M/2,N/2)
Im A(M/2,0)	Re A(M-1,1)	Im A(M-1,1)	...	Re A(M-1, (N-1)/2)	Im A(M-1, (N-1)/2)	Im A(M/2,N/2)

Re A(0,0)	Re A(0,1)	Im A(0,1)	...	Re A(0, (N-1)/2)	Im A(0, (N-1)/2)	Re A(0,N/2)
Re A(1,0)	Re A(1,1)	Im A(1,1)	...	Re A(1, (N-1)/2)	Im A(1, (N-1)/2)	Re A(1,N/2)
Im A(1,0)	Re A(2,1)	Im A(2,1)	...	Re A(2, (N-1)/2)	Im A(2, (N-1)/2)	Im A(1,N/2)
...	...	...	...	...	...	...

Re A(M/2-1,0)	Re A(M-3,1)	Im A(M-3,1)	...	Re A(M-3, (N-1)/2)	Im A(M-3, (N-1)/2)	Re A(M/2-1,N/2)
Im A(M/2-1,0)	Re A(M-2,1)	Im A(M-2,1)	...	Re A(M-2, (N-1)/2)	Im A(M-2, (N-1)/2)	Im A(M/2-1,N/2)
Re A(M/2,0)	Re A(M-1,1)	Im A(M-1,1)	...	Re A(M-1, (N-1)/2)	Im A(M-1, (N-1)/2)	Re A(M/2,N/2)

The shaded columns to the right side of the tables indicate values for even `N` only.

Note the above tables show the arrangement of coefficients for one channel. For multichannel images the channel coefficients are clustered and stored consecutively, for example, for 3-channel image they are stored in the following way: `C1-ReA(0,0); C2-R` e `A(0,0); C3-ReA(0,0); C1-ReA(0,1);C2-ReA(0,1); C3-ReA(0,1); C1-ImA(0,1);C2-ImA(0,1);` …

The remaining Fourier coefficients are obtained using the following relationships based on conjugate-symmetric properties:

```
A(i,j) = conj(A(M-i,N-j)), i = 1,..., M-1; j = 1,..., N-1
A(0,j) = conj(A(0,N-j)), j = 1,..., N-1
A(i,0) = conj(A(M-i,0)), i = 1,..., M-1
```

FFTGetSize

Computes the size of the FFT context structure and the size of the work buffer.

Syntax

```
IppStatus ippiFFTGetSize_R_32f (int orderX, int orderY, int flag, IppHintAlgorithm hint, int* pSizeSpec, int* pSizeInit, int* pSizeBuf);
IppStatus ippiFFTGetSize_C_32fc (int orderX, int orderY, int flag, IppHintAlgorithm hint, int* pSizeSpec, int* pSizeInit, int* pSizeBuf);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`orderX, orderY`	Order of the FFT in x- and y- directions, respectively.
`flag`	Flag to choose the option for results normalization.
`hint`	This parameter is deprecated. Set the value to `ippAlgHintNone`.
`pSizeSpec`	Pointer to the size of the FFT context structure.

pSizeInit	Pointer to the size of the buffer for the FFT initialization function.
pSizeBuf	Pointer to the size of the FFT external work buffer.

Description

This function computes the following:

- Size of the FFT context structure. The result in bytes is stored in the pSpecSize parameter.
- Size of the work buffer for the ippiFFTInit functions. The result in bytes is stored in the pSizeInit parameter.
- Size of the work buffer for the ippiFFTFwd and ippiFFTInv functions. The result in bytes is stored in the pSizeBuf parameter.

The suffix after the function name indicates the flavors of the FFT functions: ippiFFTGetSize_C is for complex flavors and ippiFFTGetSize_R is for real flavors.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsFftOrderErr	Indicates an error condition when the FFT order value is illegal.
ippStsFFTFlagErr	Indicates an error condition when the flag value is illegal.

See Also

FFTInit Initializes the context structure for the image FFT functions.

FFTInv Applies an inverse FFT to complex source data and stores results in a destination image.

FFTFwd Applies forward Fast Fourier Transform to an image.

FFTInit

Initializes the context structure for the image FFT functions.

Syntax

```
IppStatus ippiFFTInit_R_32f (int orderX, int orderY, int flag, IppHintAlgorithm hint,
IppiFFTSpec_R_32f* pFFTSpec, Ipp8u* pMemInit);
```

```
IppStatus ippiFFTInit_C_32fc (int orderX, int orderY, int flag, IppHintAlgorithm hint,
IppiFFTSpec_C_32fc* pFFTSpec, Ipp8u* pMemInit);
```

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

orderX, orderY	Order of the FFT in x- and y- directions, respectively.
flag	Flag to choose the option for results normalization.
hint	This parameter is deprecated. Set the value to ippAlgHintNone.
pFFTSpec	Pointer to the FFT context structure.
pMemInit	Pointer to the temporary work buffer.

Description

This function initializes the pFFTSpec context structure needed to compute the forward and inverse FFT of a two-dimensional image data.

Before calling this function, you need to allocate memory for the FFT context structure and temporary work buffer (if it is required). To compute the size of the FFT context structure and temporary work buffer, use the ippiFFTGetSize function.

The pMemInit parameter can be NULL only if the work buffer is not used. After initialization is done, you can free the temporary work buffer.

The ippiFFTFwd and ippiFFTInv functions called with the pointer to the initialized pFFTSpec structure, compute the fast Fourier transform with the following characteristics:

- length $N=2^{orderX}$ in x-direction by $M=2^{orderY}$ in y-direction
- results normalization mode as set by flag (see Table "Normalization Factors for Fourier Transform Results")
- computation algorithm indicated by hint.

The suffix after the function name indicates the type of the context structure to be initialized: ippiFFTInit_C is for the complex FFT context structure and ippiFFTInit_R is for the real FFT context structure.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsFftOrderErr	Indicates an error condition when the FFT order value is illegal.
ippStsFFTFlagErr	Indicates an error condition when the flag value is illegal.

Example

The code example below demonstrates how to use the ippiFFTGetSize and ippiFFTInit functions.

```
/// get sizes for required buffers
    ippiFFTGetSize_R_32f( orderX, orderY, IPP_FFT_DIV_INV_BY_N, ippAlgHintNone, &sizeSpec,
&sizeInit, &sizeBuffer );
```

```
    /// allocate memory for required buffers
    pMemSpec = (IppiFFTSpec_R_32f*) ippMalloc ( sizeSpec );


    if ( sizeInit > 0 )
    {
        pMemInit = ippMalloc ( sizeInit );
    }


    if ( sizeBuffer > 0 )
    {
        pMemBuffer = ippMalloc ( sizeBuffer );
    }


    /// initialize FFT specification structure
    ippiFFTInit_R_32f( orderX, orderY, IPP_FFT_DIV_INV_BY_N, ippAlgHintNone, pMemSpec,
pMemInit );


    /// free initialization buffer
    if ( sizeInit > 0 )
    {
        ippFree( pMemInit );
    }


    /// perform forward FFT to put source data to frequency domain
    ippiFFTFwd_RToPack_32f_C1R( pSrc, srcStep, pDst, dstStep, pMemSpec, pMemBuffer );


    /// ...


    /// free buffers
    if ( sizeBuffer > 0 )
    {
        ippFree( pMemBuffer );
    }


    ippFree( pMemSpec );
```

See Also

FFTGetSize Computes the size of the FFT context structure and the size of the work buffer.

FFTInv Applies an inverse FFT to complex source data and stores results in a destination image.

FFTFwd Applies forward Fast Fourier Transform to an image.

FFTFwd

Applies forward Fast Fourier Transform to an image.

Syntax

Case 1: Not-in-place operation on floating-point data

```
IppStatus ippiFFTFwd_RToPack_<mod> (const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int
dstStep, const IppiFFTSpec_R_32f* pFFTSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

`32f_C1R`
`32f_C3R`
`32f_C4R`
`32f_AC4R`

Case 2: Not-in-place operation on complex data

```
IppStatus ippiFFTFwd_CToC_32fc_C1R(const Ipp32fc* pSrc, int srcStep, Ipp32fc* pDst, int
dstStep, const IppiFFTSpec_C_32fc* pFFTSpec, Ipp8u* pBuffer);
```

Case 3: In-place operation on floating-point data

```
IppStatus ippiFFTFwd_RToPack_<mod>(Ipp32f* pSrcDst, int srcDstStep, const
IppiFFTSpec_R_32f* pFFTSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

`32f_C1IR`
`32f_C3IR`
`32f_C4IR`
`32f_AC4IR`

Case 4: In-place operation on complex data

```
IppStatus ippiFFTFwd_CToC_32fc_C1IR(Ipp32fc* pSrcDst, int srcDstStep, const
IppiFFTSpec_C_32fc* pFFTSpec, Ipp8u* pBuffer);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image.

pSrcDst	Pointer to the source and destination image ROI for the in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
pFFTSpec	Pointer to the FFT specification structure.
pBuffer	Pointer to the external work buffer.

Description

This function operates with ROI.

This function performs a forward FFT on each channel of the source image ROI pSrc(pSrcDst``for in-place flavors) and writes the Fourier coefficients into the corresponding channel of the destination buffer ``pDst(pSrcDst``for in-place flavors). The size of ROI is ``N x M, it is specified by the parameters orderX, orderY.

The function flavor ippiFFTFwd_RToPack that operates on images with real data takes advantage of the symmetry property and stores the output data in RCPack2D format. It supports processing of the 1-, 3-, and 4-channel images. Note that the functions with AC4 descriptor do not process alpha channel.

The function flavor ippiFFTFwd_CToC that operates on images with complex data does not perform any packing of the transform results as no symmetry with respect to frequency domain data is observed in this case. Memory layout of images with complex data follows the same conventions as for real images provided that each pixel value consists of two numbers: imaginary and real part.

Before using the forward FFT functions, you need to compute the size of the work buffer by ippiFFTGetSize and initialize the context structure by the ippiFFTInit function. The forward FFT functions use the pFFTSpec context structure to set the mode of calculations and retrieve support data.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if pSrc, pDst, or pFFTSpec pointer is NULL.
ippStsStepErr	Indicates an error condition if srcStep or dstStep value is zero or negative.
ippStsContextMatchErr	Indicates an error condition if a pointer to an invalid pFFTSpec structure is passed.
ippStsMemAllocErr	Indicates an error condition if memory allocation fails.

Example

FFTFwd_CToC:

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
```

```
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *****************************************************************************/

//   A simple example of performing the forward FFT for processing complex data.
// implemented with Intel(R) Integrated Performance Primitives (Intel(R) IPP) functions:
//     ippiFFTGetSize_C_32fc
//     ippiFFTInit_C_32fc
//     ippiFFTFwd_CToC_32fc_C1R

#include <stdio.h>
#include "ipp.h"

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define check_sts(st)        \
    if ((st) != ippStsNoErr) \
        goto exitLine; /* Go to Exit if Intel(R) IPP function returned status different from
ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main(void)
{
    IppStatus status;
    IppiFFTSpec_C_32fc *pSpec = NULL; /* Pointer to FFT spec structure */

    Ipp32fc srcFwd[8 * 8] = {0}, dstFwd[8 * 8] = {0}; /* Source/destination images */
    Ipp32fc m3 = {-3, 0}, one = {1, 0};

    Ipp8u *pMemInit = NULL, *pBuffer = NULL;     /* Pointer to the work buffers */
    int sizeSpec = 0, sizeInit = 0, sizeBuf = 0; /* Size of FFT spec structure, init and work
buffers */

    srcFwd[0] = m3;
    srcFwd[9] = one;

    check_sts(status = ippiFFTGetSize_C_32fc(3, 3, IPP_FFT_DIV_INV_BY_N, ippAlgHintAccurate,
&sizeSpec, &sizeInit, &sizeBuf));

    /* memory allocation */
    pSpec = (IppiFFTSpec_C_32fc *)ippMalloc(sizeSpec);
    pBuffer = (Ipp8u *)ippMalloc(sizeBuf);
    pMemInit = (Ipp8u *)ippMalloc(sizeInit);

    check_sts(status = ippiFFTInit_C_32fc(3, 3, IPP_FFT_DIV_INV_BY_N, ippAlgHintAccurate, pSpec,
pMemInit));

    /* forward FFT transform */
```

```
    check_sts(status = ippiFFTFwd_CToC_32fc_C1R(srcFwd, 8 * sizeof(Ipp32fc), dstFwd, 8 *
sizeof(Ipp32fc), pSpec, pBuffer));

    EXIT_MAIN
    ippFree(pMemInit);
    ippFree(pSpec);
    ippFree(pBuffer);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

FFTFwd_RToPack:

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

//   A simple example of performing the forward FFT for processing real data.
// implemented with Intel(R) Integrated Performance Primitives (Intel(R) IPP) functions:
//     ippiFFTGetSize_R_32f
//     ippiFFTInit_R_32f
//     ippiFFTFwd_RToPack_32f_C1R

#include <stdio.h>
#include "ipp.h"

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define check_sts(st)        \
    if ((st) != ippStsNoErr) \
        goto exitLine; /* Go to Exit if Intel(R) IPP function returned status different from
ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main(void)
{
    IppStatus status;
    IppiFFTSpec_R_32f *pSpec = NULL; /* Pointer to FFT spec structure */

    /* Pointers to source/destination images */
    Ipp32f pSrcFwd[64] = {0}, pDstFwd[64] = {0};

    /* Pointer to the work buffers */
    Ipp8u *pMemInit = NULL;
```

```
    Ipp8u *pBuffer = NULL;

    int sizeSpec = 0, sizeInit = 0, sizeBuf = 0; /* size of FFT spec structure, Init and work
buffers */

    pSrcFwd[0] = -3;
    pSrcFwd[9] = 1;

    check_sts(status = ippiFFTGetSize_R_32f(3, 3, IPP_FFT_DIV_INV_BY_N, ippAlgHintAccurate,
&sizeSpec, &sizeInit, &sizeBuf));

    /* memory allocation */
    pSpec = (IppiFFTSpec_R_32f *)ippMalloc(sizeSpec);
    pBuffer = (Ipp8u *)ippMalloc(sizeBuf);
    pMemInit = (Ipp8u *)ippMalloc(sizeInit);

    /* inverse FFT transform */
    check_sts(status = ippiFFTInit_R_32f(3, 3, IPP_FFT_DIV_INV_BY_N, ippAlgHintAccurate, pSpec,
pMemInit));

    check_sts(status = ippiFFTFwd_RToPack_32f_C1R(pSrcFwd, 8 * sizeof(Ipp32f), pDstFwd, 8 *
sizeof(Ipp32f), pSpec, pBuffer));

    EXIT_MAIN
    ippFree(pMemInit);
    ippFree(pSpec);
    ippFree(pBuffer);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

See Also

Regions of Interest in Intel IPP

FFTGetSize Computes the size of the FFT context structure and the size of the work buffer.

FFTInit Initializes the context structure for the image FFT functions.

FFTInv

Applies an inverse FFT to complex source data and stores results in a destination image.

Syntax

Case 1: Not-in-place operation on floating-point data

```
IppStatus ippiFFTInv_PackToR_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int
dstStep, const IppiFFTSpec_R_32f* pFFTSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

32f_C1R
32f_C3R
32f_C4R
32f_AC4R

Case 2: Not-in-place operation on complex data

```
IppStatus ippiFFTInv_CToC_32fc_C1R(const Ipp32fc* pSrc, int srcStep, Ipp32fc* pDst, int dstStep, const IppiFFTSpec_C_32fc* pFFTSpec, Ipp8u* pBuffer);
```

Case 3: In-place operation on floating-point data

```
IppStatus ippiFFTInv_PackToR_<mod>(Ipp32f* pSrcDst, int srcDstStep, const IppiFFTSpec_R_32f* pFFTSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

`32f_C1IR`
`32f_C3IR`
`32f_C4IR`
`32f_AC4IR`

Case 4: In-place operation on complex data

```
IppStatus ippiFFTInv_CToC_32fc_C1IR(Ipp32fc* pSrcDst, int srcDstStep, const IppiFFTSpec_C_32fc* pFFTSpec, Ipp8u* pBuffer);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image.
`pSrcDst`	Pointer to the source and destination image ROI for the in-place operation.
`srcDstStep`	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
`pFFTSpec`	Pointer to the previously initialized FFT context structure.
`pBuffer`	Pointer to the external work buffer.

Description

This function operates with ROI.

This function performs an inverse FFT on each channel of the source image `pSrc(pSrcDst``for in-place flavors) and writes the restored image data into the corresponding channel of the destination image buffer ``pDst(pSrcDst``for in-place flavors). The size of ROI is ``N x M`, it is specified by the parameters `orderX`, `orderY`.

For the `ippiFFTInv_PackToR`, function flavor, the input buffer must contain data in RCPack2D format.

Before using the inverse FFT functions, you need to compute the size of the work buffer by `ippiFFTGetSize` and initialize the context structure by the `ippiFFTInit` function. The inverse FFT functions use the `pFFTSpec` context structure to set the mode of calculations and retrieve support data.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if `pSrc`, `pDst`, or `pFFTSpec` pointer is `NULL`.
`ippStsStepErr`	Indicates an error condition if `srcStep` or `dstStep` value is zero or negative.
`ippStsContextMatchErr`	Indicates an error condition if a pointer to an invalid `pFFTSpec` structure is passed.
`ippStsMemAllocErr`	Indicates an error condition if memory allocation fails.

Example

`FFTInv_CToC`:

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

//   A simple example of performing the forward FFT for processing complex data.
// implemented with Intel(R) Integrated Performance Primitives (Intel(R) IPP) functions:
//     ippiFFTGetSize_C_32fc
//     ippiFFTInit_C_32fc
//     ippiFFTFwd_CToC_32fc_C1R
//     ippiFFTInv_CToC_32fc_C1R

#include <stdio.h>
```

```
#include "ipp.h"

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define check_sts(st)        \
    if ((st) != ippStsNoErr) \
        goto exitLine; /* Go to Exit if Intel(R) IPP function returned status different from
ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main(void)
{
    IppStatus status;
    IppiFFTSpec_C_32fc *pSpec = NULL; /* Pointer to FFT spec structure */

    /* Pointers to source/destination images */
    Ipp32fc pSrcFwd[64] = {0}, pDstFwd[64] = {0};
    Ipp32fc pSrcInv[64] = {0}, pDstInv[64] = {0};
    Ipp32fc m3 = {-3, 0}, one = {1, 0};

    Ipp8u *pMemInit = NULL, *pBuffer = NULL; /* Pointer to the work buffers */

    int sizeSpec = 0, sizeInit = 0, sizeBuf = 0; /* size of FFT spec structure, Init and work
buffers */

    pSrcFwd[0] = m3;
    pSrcFwd[9] = one;

    check_sts(status = ippiFFTGetSize_C_32fc(3, 3, IPP_FFT_DIV_INV_BY_N, ippAlgHintAccurate,
&sizeSpec, &sizeInit, &sizeBuf));

    /* memory allocation */
    pSpec = (IppiFFTSpec_C_32fc *)ippMalloc(sizeSpec);
    pBuffer = (Ipp8u *)ippMalloc(sizeBuf);
    pMemInit = (Ipp8u *)ippMalloc(sizeInit);

    check_sts(status = ippiFFTInit_C_32fc(3, 3, IPP_FFT_DIV_INV_BY_N, ippAlgHintAccurate, pSpec,
pMemInit));

    /* forward FFT transform */
    check_sts(status = ippiFFTFwd_CToC_32fc_C1R(pSrcFwd, 8 * sizeof(Ipp32fc), pDstFwd, 8 *
sizeof(Ipp32fc), pSpec, pBuffer));

    check_sts(status = ippsCopy_32fc(pDstFwd, pSrcInv, 64));

    /* inverse FFT transform */
    check_sts(status = ippiFFTInv_CToC_32fc_C1R(pSrcInv, 8 * sizeof(Ipp32fc), pDstInv, 8 *
sizeof(Ipp32fc), pSpec, pBuffer));

    EXIT_MAIN
    ippFree(pMemInit);
    ippFree(pSpec);
    ippFree(pBuffer);
```

```
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

FFTInv_RToPack:

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

//   A simple example of performing the inverse FFT for processing real data.
// implemented with Intel(R) Integrated Performance Primitives (Intel(R) IPP) functions:
//     ippiFFTGetSize_R_32f
//     ippiFFTInit_R_32f
//     ippiFFTFwd_RToPack_32f_C1R
//     ippiFFTInv_PackToR_32f_C1R

#include <stdio.h>
#include "ipp.h"

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define check_sts(st)        \
    if ((st) != ippStsNoErr) \
        goto exitLine; /* Go to Exit if Intel(R) IPP function returned status different from
ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main(void)
{
    IppStatus status;
    IppiFFTSpec_R_32f *pSpec = NULL; /* Pointer to FFT spec structure */

    /* Pointers to source/destination images */
    Ipp32f pSrcFwd[64] = {0}, pDstFwd[64] = {0};
    Ipp32f pSrcInv[64] = {0}, pDstInv[64] = {0};

    /* Pointer to the work buffers */
    Ipp8u *pMemInit = NULL;
    Ipp8u *pBuffer = NULL;

    int sizeSpec = 0, sizeInit = 0, sizeBuf = 0; /* size of FFT spec structure, Init and work
buffers */
```

```
    pSrcFwd[0] = -3;
    pSrcFwd[9] = 1;

    check_sts(status = ippiFFTGetSize_R_32f(3, 3, IPP_FFT_DIV_INV_BY_N, ippAlgHintAccurate,
&sizeSpec, &sizeInit, &sizeBuf));

    /* memory allocation */
    pSpec = (IppiFFTSpec_R_32f *)ippMalloc(sizeSpec);
    pBuffer = (Ipp8u *)ippMalloc(sizeBuf);
    pMemInit = (Ipp8u *)ippMalloc(sizeInit);

    check_sts(status = ippiFFTInit_R_32f(3, 3, IPP_FFT_DIV_INV_BY_N, ippAlgHintAccurate, pSpec,
pMemInit));

    /* forward FFT transform */
    check_sts(status = ippiFFTFwd_RToPack_32f_C1R(pSrcFwd, 8 * sizeof(Ipp32f), pDstFwd, 8 *
sizeof(Ipp32f), pSpec, pBuffer));

    check_sts(status = ippsCopy_32f(pDstFwd, pSrcInv, 64));

    /* inverse FFT transform */
    check_sts(status = ippiFFTInv_PackToR_32f_C1R(pSrcInv, 8 * sizeof(Ipp32f), pDstInv, 8 *
sizeof(Ipp32f), pSpec, pBuffer));

    EXIT_MAIN
    ippFree(pMemInit);
    ippFree(pSpec);
    ippFree(pBuffer);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

See Also

Regions of Interest in Intel IPP

FFTGetSize Computes the size of the FFT context structure and the size of the work buffer.

FFTInit Initializes the context structure for the image FFT functions.

DFTGetSize

Computes the size of the FFT context structure and the size of the work buffer.

Syntax

```
IppStatus ippiDFTGetSize_R_32f (IppiSize roiSize, int flag, IppHintAlgorithm hint, int*
pSizeSpec, int* pSizeInit, int* pSizeBuf);
```

```
IppStatus ippiDFTGetSize_C_32fc (IppiSize roiSize, int flag, IppHintAlgorithm hint,
int* pSizeSpec, int* pSizeInit, int* pSizeBuf);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

roiSize	Size of the source and destination ROI in pixels.
flag	Flag to choose the option for results normalization. For more information, see Table "Normalization Factors for Fourier Transform Results"
hint	This parameter is deprecated. Set the value to ippAlgHintNone.
pSizeSpec	Pointer to the size of the DFT context structure.
pSizeInit	Pointer to the size of the buffer for the DFT initialization function.
pSizeBuf	Pointer to the size of the DFT external work buffer.

Description

This function computes the following:

- Size of the DFT context structure. The result in bytes is stored in the pSpecSize parameter.
- Size of the work buffer for the ippiDFTInit functions. The result, in bytes, is stored in the pSizeInit parameter.
- Size of the work buffer for the ippiDFTFwd and ippiDFTInv functions. The result, in bytes, is stored in the pSizeBuf parameter.

The suffix after the function name indicates the flavors of the DFT functions: ippiDFTGetSize_C is for complex flavors and ippiDFTGetSize_R is for real flavors.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsFftFlagErr	Indicates an error condition when the flag value is illegal.
ippStsFftOrderErr	Indicates an error when the amount of memory needed to compute the DFT for points in the ROI of size roiSize exceeds the limit.
ippStsSizeErr	Indicates an error condition when the roiSize has a field with a zero or negative value.

See Also

DFTInit Initializes the context structure for the image DFT functions.

DFTInit

Initializes the context structure for the image DFT functions.

Syntax

```
IppStatus ippiDFTInit_R_32f (IppiSize roiSize, int flag, IppHintAlgorithm hint,
IppiDFTSpec_R_32f* pDFTSpec, Ipp8u* pMemInit);
```

```
IppStatus ippiDFTInit_C_32fc (IppiSize roiSize, int flag, IppHintAlgorithm hint,
IppiDFTSpec_C_32fc* pDFTSpec, Ipp8u* pMemInit);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`roiSize`	Size of the source and destination ROI in pixels.
`flag`	Flag to choose the option for results normalization.
`hint`	This parameter is deprecated. Set the value to `ippAlgHintNone`.
`pDFTSpec`	Pointer to the DFT context structure.
`pMemInit`	Pointer to the temporary work buffer.

Description

This function initializes the `pDFTSpec` context structure needed to compute the forward and inverse DFT of a two-dimensional image data.

Before calling this function, you need to allocate memory for the FFT context structure and temporary work buffer (if it is required). To compute the size of the FFT context structure and temporary work buffer, use the `ippiDFTGetSize` function.

The `pMemInit` parameter can be `NULL` only if the work buffer is not used. After initialization is done, you can free the temporary work buffer.

The `ippiDFTFwd` and `ippiDFTInv` functions called with the pointer to the initialized `pDFTSpec` structure compute the discrete Fourier transform with the following characteristics:

- ROI of the `roiSize` size
- results normalization mode set by `flag` (see Table "Normalization Factors for Fourier Transform Results")
- computation algorithm indicated by `hint`.

The suffix after the function name indicates the type of the context structure to be initialized: `ippiDFTInit_C` is for the complex DFT context structure and `ippiDFTInit_R` is for the real DFT context structure.

Return Values

`ippStsNoErr`	Indicates no error.

ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsFftFlagErr	Indicates an error condition when the flag value is illegal.
ippStsFftOrderErr	Indicates an error when the amount of memory needed to compute the DFT for points in the ROI of size roiSize exceeds the limit.
ippStsSizeErr	Indicates an error condition when the roiSize has a field with a zero or negative value.

Example

The code example below demonstrates how to use the ippiDFTGetSize and ippiDFTInit functions.

```
/// get sizes for required buffers
ippiDFTGetSize_R_32f( roiSize, IPP_FFT_DIV_INV_BY_N, ippAlgHintNone, &sizeSpec, &sizeInit,
&sizeBuffer );


/// allocate memory for required buffers
pMemSpec = (IppiDFTSpec_R_32f*) ippMalloc ( sizeSpec );


if ( sizeInit > 0 )
{
    pMemInit = ippMalloc ( sizeInit );
}


if ( sizeBuffer > 0 )
{
    pMemBuffer = ippMalloc ( sizeBuffer );
}


/// initialize DFT specification structure
ippiDFTInit_R_32f( roiSize, IPP_FFT_DIV_INV_BY_N, ippAlgHintNone, pMemSpec, pMemInit );


/// free initialization buffer
if ( sizeInit > 0 )
{
    ippFree( pMemInit );
}


/// perform forward DFT to put source data to frequency domain
ippiDFTFwd_RToPack_32f_C1R( pSrc, srcStep, pDst, dstStep, pMemSpec, pMemBuffer );


/// ...


/// free buffers
if ( sizeBuffer > 0 )
```

```
{
    ippFree( pMemBuffer );
}


ippFree( pMemSpec );
```

See Also

Regions of Interest in Intel IPP

DFTGetSize Computes the size of the FFT context structure and the size of the work buffer.

DFTFwd Applies forward discrete Fourier transform to an image.

DFTInv Applies an inverse DFT to complex source data and stores results in a destination image.

DFTFwd

Applies forward discrete Fourier transform to an image.

Syntax

Case 1: Not-in-place operation on floating-point data

```
IppStatus ippiDFTFwd_RToPack_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int
dstStep, const IppiDFTSpec_R_32f* pDFTSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

32f_C1R
32f_C3R
32f_C4R
32f_AC4R

Case 2: Not-in-place operation on complex data

```
IppStatus ippiDFTFwd_CToC_32fc_C1R(const Ipp32fc* pSrc, int srcStep, Ipp32fc* pDst, int
dstStep, const IppiDFTSpec_C_32fc* pDFTSpec, Ipp8u* pBuffer);
```

Case 3: In-place operation on floating-point data

```
IppStatus ippiDFTFwd_RToPack_<mod>(Ipp32f* pSrcDst, int srcDstStep, const
IppiDFTSpec_R_32f* pDFTSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

32f_C1IR
32f_C3IR
32f_C4IR
32f_AC4IR

Case 4: In-place operation on complex data

```
IppStatus ippiDFTFwd_CToC_32fc_C1IR(Ipp32fc* pSrcDst, int srcDstStep, const
IppiDFTSpec_C_32fc* pDFTSpec, Ipp8u* pBuffer);
```

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
pSrcDst	Pointer to the source and destination image ROI for the in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
pDFTSpec	Pointer to the previously initialized DFT context structure.
pBuffer	Pointer to the external work buffer.

Description

This function operates with ROI.

This function performs a forward DFT on each channel of the source image ROI pSrc (pSrcDst for in-place flavors) and writes the Fourier coefficients into the corresponding channel of the destination buffer pDst (pSrcDst for in-place flavors).

The function flavor ippiDFTFwd_RToPack that operates on images with real data takes advantage of the symmetry property and stores the output data in RCPack2D format. It supports processing of the 1-, 3-, and 4-channel images. Note that the functions with AC4 descriptor do not process alpha channel.

The function flavor ippiDFTFwd_CToC that operates on images with complex data performs no packing of the transform results as no symmetry with respect to frequency domain data is observed in this case. Memory layout of images with complex data follows the same conventions as for real images provided that each pixel value consists of two numbers: imaginary and real part.

Before using the forward DFT functions, you need to compute the size of the work buffer by ippiDFTGetSize and initialize the context structure by the ippiDFTInit function. The forward DFT functions use the pDFTSpec context structure to set the mode of calculations and retrieve support data.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr	Indicates an error condition if pSrc, pDst, or pDFTSpec pointer is NULL.
ippStsStepErr	Indicates an error condition if srcStep or dstStep value is zero or negative.
ippStsContextMatchErr	Indicates an error condition if a pointer to an invalid pDFTSpec structure is passed.
ippStsMemAllocErr	Indicates an error condition if memory allocation fails.

See Also

Regions of Interest in Intel IPP

DFTGetSize Computes the size of the FFT context structure and the size of the work buffer.

DFTInit Initializes the context structure for the image DFT functions.

DFTInv

Applies an inverse DFT to complex source data and stores results in a destination image.

Syntax

Case 1: Not-in-place operation on floating-point data

```
IppStatus ippiDFTInv_PackToR_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, const IppiDFTSpec_R_32f* pDFTSpec, Ipp8u* pBuffer);
```

Supported values for mod:

32f_C1R
32f_C3R
32f_C4R
32f_AC4R

Case 2: Not-in-place operation on complex data

```
IppStatus ippiDFTInv_CToC_32fc_C1R(const Ipp32fc* pSrc, int srcStep, Ipp32fc* pDst, int dstStep, const IppiDFTSpec_C_32fc* pDFTSpec, Ipp8u* pBuffer);
```

Case 3: In-place operation on floating-point data

```
IppStatus ippiDFTInv_PackToR_<mod>(Ipp32f* pSrcDst, int srcDstStep, const IppiDFTSpec_R_32f* pDFTSpec, Ipp8u* pBuffer);
```

Supported values for mod:

32f_C1IR
32f_C3IR
32f_C4IR
32f_AC4IR

Case 5: In-place operation on complex data

```
IppStatus ippiDFTInv_CToC_32fc_C1IR(Ipp32fc* pSrcDst, int srcDstStep, const
IppiDFTSpec_C_32fc* pDFTSpec, Ipp8u* pBuffer);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image.
`pSrcDst`	Pointer to the source and destination image ROI for the in-place operation.
`srcDstStep`	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
`pDFTSpec`	Pointer to the previously initialized DFT context structure.
`pBuffer`	Pointer to the external work buffer.

Description

This function operates with ROI.

This function performs an inverse DFT on each channel of the input buffer `pSrc` (`pSrcDst` for in-place flavors) and writes the restored image data into the corresponding channel of the output image buffer `pDst` (`pSrcDst` for in-place flavors).

For function flavor `ippiDFTInv_PackToR`, the input buffer must contain data in RCPack2D format.

Before using the inverse DFT functions, you need to compute the size of the work buffer by `ippiDFTGetSize` and initialize the context structure by the `ippiDFTInit` function. The inverse DFT functions use the `pDFTSpec` context structure to set the mode of calculations and retrieve support data.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if `pSrc`, `pDst`, or `pDFTSpec` pointer is `NULL`.

ippStsStepErr	Indicates an error condition if srcStep or dstStep value is zero or negative.
ippStsContextMatchErr	Indicates an error condition if a pointer to an invalid pDFTSpec structure is passed.
ippStsMemAllocErr	Indicates an error condition if memory allocation fails.

See Also

Regions of Interest in Intel IPP

DFTGetSize Computes the size of the FFT context structure and the size of the work buffer.

DFTInit Initializes the context structure for the image DFT functions.

MulPack

Multiplies two source images in packed format.

Syntax

```
IppStatus ippiMulPack_<mod>(const Ipp32f* pSrc1, int src1Step, const Ipp32f* pSrc2, int src2Step, Ipp32f* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

32f_C1R
32f_C3R
32f_C4R
32f_AC4R

```
IppStatus ippiMulPack_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for mod:

32f_C1IR
32f_C3IR
32f_C4IR
32f_AC4IR

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc1, pSrc2	Pointer to the ROI in the source images.
src1Step, src2Step	Distance in bytes between starts of consecutive lines in the source images.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
pSrc	Pointer to the first source image ROI for the in-place operation.
srcStep	Distance in bytes between starts of consecutive lines in the first source image for the in-place operation.
pSrcDst	Pointer to the second source and destination image ROI for the in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
roiSize	Size of the source and destination ROI in pixels.
scaleFactor	Scale factor (see Integer Result Scaling).

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function multiplies corresponding pixel values of two source images, A and B represented in RCPack2D format and stores the result into the destination image C in packed format also. The multiplying is performed according to the following formulas:

Re C = Re A *Re B – Im A *Im B;

Im C = Im A *Re B + Im B *Re A.

Not-in-place flavors multiply pixel values of ROI in the source images pSrc1 and pSrc2, and store result in the pDst.

In-place flavors multiply pixel values of ROI in the source images pSrc and pSrcDst, and store result in the pSrcDst.

This function can be used in image filtering operations that include FFT transforms.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
```

```
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *****************************************************************************/

//    The code example below illustrates how the horizontal edges of an image can be detected.
// implemented with Intel(R) Integrated Performance Primitives (Intel(R) IPP) functions:
//     ippiSet_32f_C1R
//     ippiAddC_32f_C1IR
//     ippiCopy_32f_C1R
//     ippiFFTGetSize_R_32f
//     ippiFFTInit_R_32f
//     ippiFFTFwd_RToPack_32f_C1R
//     ippiMulPack_32f_C1R
//     ippiFFTInv_PackToR_32f_C1R

#include <stdio.h>
#include "ipp.h"

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define check_sts(st)        \
    if ((st) != ippStsNoErr) \
        goto exitLine; /* Go to Exit if Intel(R) IPP function returned status different from
ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main(void)
{
    IppStatus status = ippStsNoErr;
    IppiFFTSpec_R_32f *pSpec = NULL; /* Pointer to FFT spec structure */

    Ipp32f pSrc[64], pFlt[64], pDst[64];       /* Pointers to source/filter/destination images */
    Ipp32f pSrcFr[64], pFltFr[64], pDstFr[64]; /* Pointers to source/filter/destination images
into the frequency domain */

    Ipp8u *pMemInit = NULL, *pBuffer = NULL;     /* Pointer to the work buffers */
    int sizeSpec = 0, sizeInit = 0, sizeBuf = 0; /* size of FFT spec structure, Init and work
buffers */

    const IppiSize roiSize8x8 = {8, 8}, roiSize3x3 = {3, 3};     /* image and filter roi sizes */
    const Ipp32f filter[3 * 3] = {-1, -1, -1, 0, 0, 0, 1, 1, 1}; /* filter data */

    /* set the image zero */
    check_sts(status = ippiSet_32f_C1R(0, pSrc, 8 * sizeof(Ipp32f), roiSize8x8));

    /* create unit area in the image */
    check_sts(status = ippiAddC_32f_C1IR(1, pSrc + 8 + 1, 8 * sizeof(Ipp32f), roiSize3x3));

    /* set the filter zero */
    check_sts(status = ippiSet_32f_C1R(0, pFlt, 8 * sizeof(Ipp32f), roiSize8x8));

    /* copy the filter coefficients */
```

```
    check_sts(status = ippiCopy_32f_C1R(filter, 3 * sizeof(Ipp32f), pFlt, 8 * sizeof(Ipp32f),
roiSize3x3));

    check_sts(status = ippiFFTGetSize_R_32f(3, 3, IPP_FFT_DIV_INV_BY_N, ippAlgHintAccurate,
&sizeSpec, &sizeInit, &sizeBuf));

    /* memory allocation */
    pSpec = (IppiFFTSpec_R_32f *)ippMalloc(sizeSpec);
    pBuffer = (Ipp8u *)ippMalloc(sizeBuf);
    pMemInit = (Ipp8u *)ippMalloc(sizeInit);

    check_sts(status = ippiFFTInit_R_32f(3, 3, IPP_FFT_DIV_INV_BY_N, ippAlgHintAccurate, pSpec,
pMemInit));

    /* forward FFT transform : the source image is transformed into the frequency domain*/
    check_sts(status = ippiFFTFwd_RToPack_32f_C1R(pSrc, 8 * sizeof(Ipp32f), pSrcFr, 8 *
sizeof(Ipp32f), pSpec, pBuffer));

    /* forward FFT transform: the filter is transformed into the frequency domain */
    check_sts(status = ippiFFTFwd_RToPack_32f_C1R(pFlt, 8 * sizeof(Ipp32f), pFltFr, 8 *
sizeof(Ipp32f), pSpec, pBuffer));

    /* multiplying the packed data */
    check_sts(status = ippiMulPack_32f_C1R(pSrcFr, 8 * sizeof(Ipp32f), pFltFr, 8 *
sizeof(Ipp32f), pDstFr, 8 * 4, roiSize8x8));

    /* inverse FFT transform : the filtered data is transformed to the time domain */
    check_sts(status = ippiFFTInv_PackToR_32f_C1R(pDstFr, 8 * sizeof(Ipp32f), pDst, 8 *
sizeof(Ipp32f), pSpec, pBuffer));

    EXIT_MAIN
    ippFree(pMemInit);
    ippFree(pSpec);
    ippFree(pBuffer);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize` has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if any of the specified buffer step values is zero or negative.

MulPackConj

Multiplies a source image by the complex conjugate image with data in packed format and stores the result in the destination buffer in the packed format.

Syntax

Case 1: Not-in-place operation

```
IppStatus ippiMulPackConj_<mod>(const Ipp32f* pSrc1, int src1Step, const Ipp32f* pSrc2,
int src2Step,Ipp32f* pDst, int dstStep, IppiSize roiSize);
```

Supported values for `mod`:

`32f_C1R`
`32f_C3R`
`32f_C4R`
`32f_AC4R`

Case 2: In-place operation

```
IppStatus ippiMulPackConj_<mod>(const Ipp32f* pSrc, int srcStep, Ipp32f* pSrcDst, int
srcDstStep, IppiSize roiSize);
```

Supported values for `mod`:

`32f_C1IR`
`32f_C3IR`
`32f_C4IR`
`32f_AC4IR`

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc1`, `pSrc2`	Pointer to the ROI in the source images.
`src1Step`, `src2Step`	Distance in bytes between starts of consecutive lines in the source images.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image.
`pSrc`	Pointer to the first source image ROI for the in-place operation.
`srcStep`	Distance in bytes between starts of consecutive lines in the first source image for the in-place operation.

pSrcDst	Pointer to the second source and destination image ROI for the in-place operation.
srcDstStep	Distance in bytes between starts of consecutive lines in the source and destination image for the in-place operation.
roiSize	Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This functionmultiplies pixel values of the source image A by the corresponding pixel values of the complex conjugate image A*, represented in RCPack2D format. The result of the operation is written into the destination buffer in packed format also.

Not-in-place flavors multiply pixel values of ROI in the source images pSrc1 and pSrc2, and store result in the pDst.

In-place flavors multiply pixel values of ROI in the source images pSrc and pSrcDst, and store result in the pSrcDst.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if any of the specified buffer step values is zero or negative.

Magnitude

Computes magnitude of elements of a complex data image.

Syntax

```
IppStatus ippiMagnitude_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

32fc32f_C1R	32fc32f_C3R

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function computes magnitude of elements of the source image pSrc given in complex data format, and stores results in the destination image pDst.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if pSrc or pDst pointer is NULL.
ippStsStepErr	Indicates an error condition if srcStep or dstStep value is zero or negative.
ippStsSizeErr	Indicates an error condition if width or height of images is less than or equal to zero.

MagnitudePackGetBufferSize

Computes the size of the work buffer for the ippiMagnitudePack function.

Syntax

```
IppStatus ippiMagnitudePackGetBufferSize_32f (int numChannels, IppiSize dstRoiSize,
int* pSize);
```

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

numChannels	Number of channels in the image. Possible values are 1 and 3.
dstRoiSize	Size, in pixels, of the destination image ROI.
pSize	Pointer to the computed size of the external work buffer, in bytes.

Description

The ippiMagnitudePackGetBufferSize function computes the size, in bytes, of the external work buffer needed for the ippiMagnitudePack function. The result is stored in the pSize parameter.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pSize is NULL.
ippStsSizeErr	Indicates an error when dstRoiSize has a field with a value less than 1.
ippStsNumChannelsErr	Indicates an error when numChannels has an illegal value.

See Also

MagnitudePack MODIFIED API. Computes magnitude of elements of an image in packed format.

MagnitudePack

MODIFIED API. Computes magnitude of elements of an image in packed format.

Syntax

```
IppStatus ippiMagnitudePack_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, Ipp8u* pBuffer);
```

Supported values for mod:

32f_C1R	32f_C3R

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc	Pointer to the source image ROI.

srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
dstRoiSize	Size of the destination ROI in pixels.
pBuffer	Pointer to the work buffer. To compute the size of the buffer, use the ippiMagnitudePackGetBufferSize function.

Description

NOTE
Important
The API of this function has been modified in Intel IPP 9.0 release.

This function operates with ROI (see Regions of Interest in Intel IPP).

This function computes magnitude of elements of the source image pSrc given in RCPack2D format and stores results in the destination image pDst.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when pSrc or pDst pointer is NULL.
ippStsStepErr	Indicates an error when srcStep or dstStep value is zero or negative.
ippStsSizeErr	Indicates an error when width or height of images is less than, or equal to zero.

See Also

MagnitudePackGetBufferSize Computes the size of the work buffer for the ippiMagnitudePack function.

Regions of Interest in Intel IPP

Real - Complex Packed (RCPack2D) Format

Phase

Computes the phase of elements of a complex data image.

Syntax

```
IppStatus ippiPhase_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, Ipp<dstDatatype>*
pDst, int dstStep, IppiSize roiSize);
```

Supported values for mod:

32fc32f_C1R	32fc32f_C3R

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of the source and destination ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function computes the phase in radians of elements of a source image pSrc given in complex data format, and stores results in the destination image pDst.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if pSrc or pDst pointer is NULL.
ippStsStepErr	Indicates an error condition if srcStep or dstStep value is zero or negative.
ippStsSizeErr	Indicates an error condition if width or height of images is less than or equal to zero.

PhasePackGetBufferSize

Computes the size of the work buffer for the ippiPhasePack function.

Syntax

```
IppStatus ippiPhasePackGetBufferSize_32f (int numChannels, IppiSize dstRoiSize, int*
pSize);
```

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

numChannels	Number of channels in the image. Possible values are 1 and 3.
dstRoiSize	Size, in pixels, of the destination image ROI.
pSize	Pointer to the computed size of the external work buffer, in bytes.

Description

The ippiPhasePackGetBufferSize function computes the size, in bytes, of the external work buffer needed for the ippiPhasePack function. The result is stored in the pSize parameter.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pSize is NULL.
ippStsSizeErr	Indicates an error when dstRoiSize has a field with a value less than 1.
ippStsNumChannelsErr	Indicates an error when numChannels has an illegal value.

See Also

PhasePack MODIFIED API. Computes the phase of elements of an image in packed format.

PhasePack

MODIFIED API. Computes the phase of elements of an image in packed format.

Syntax

```
IppStatus ippiPhasePack_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize dstRoiSize, Ipp8u* pBuffer);
```

Supported values for mod:

32f_C1R	32f_C3R

Include Files

ippi.h

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image.
`dstRoiSize`	Size of the destination ROI in pixels.
`pBuffer`	Pointer to the work buffer. To compute the size of the buffer, use the PhasePackGetBufferSize function.

Description

NOTE
Important
The API of this function has been modified in Intel IPP 9.0 release.

This function operates with ROI (see Regions of Interest in Intel IPP).

This function computes the phase of elements of a source image `pSrc` given in RCPack2D format and stores the results in the destination image `pDst`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error when `pSrc` or `pDst` pointer is `NULL`.
`ippStsStepErr`	Indicates an error when `srcStep` or `dstStep` value is zero or negative.
`ippStsSizeErr`	Indicates an error when width or height of images is less than, or equal to zero.

See Also

PhasePackGetBufferSize Computes the size of the work buffer for the ippiPhasePack function.

Regions of Interest in Intel IPP

Real - Complex Packed (RCPack2D) Format

PolarToCart

Converts an image in the polar coordinate form to Cartesian coordinate form.

Syntax

```
IppStatus ippiPolarToCart_<mod>(const Ipp32f* pSrcMagn, const Ipp32f* pSrcPhase, int srcStep, IppiSize roiSize, Ipp32fc* pDst, int dstStep);
```

Supported values for `mod`:

`32fc_C1R`	`32fc_C3R`

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrcMagn`	Pointer to the buffer containing magnitudes of the source image.
`pSrcPhase`	Pointer to the buffer containing phase values of the source image.
`srcStep`	Distance in bytes between starts of consecutive lines in the source buffers.
`roiSize`	Size of the source and destination image ROI.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the polar coordinate of the source image stored in the arrays of magnitudes `pSrcMagn` and phase values `pSrcPhase` to the destination image `pDst` in complex-data format (in Cartesian coordinate form).

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.

ippStsStepErr	Indicates an error condition if srcStep or dstStep value is zero or negative.
ippStsSizeErr	Indicates an error condition if srcSize has a field with value less than 1.

Example

The code example below shows how to use the function ippiPolarToCart_32fc_C1R.

```
void func_polartocart() {
    Ipp32f pSrcMagn[2*2] = {1.0, 0.0, 2.1, 3.2};
    Ipp32f pSrcPhase[2*2] = {0.0, 2.0, 1.6,-1.0};
    Ipp32fc pDst[2*2] = {0};
    int srcStep = 2*sizeof(Ipp32f);
    int dstStep = 2*sizeof(Ipp32fc);
    IppiSize roiSize = {2, 2};


    ippiPolarToCart_32fc_C1R(pSrcMagn, pSrcPhase, srcStep, roiSize, pDst, dstStep);
}
```

Result:

```
pDst - > (1.0, 0.0)  (0.0, 0.0)  (-0.1, 2.1) (1.7, -2.7)
```

PackToCplxExtend

Converts an image in packed format to a complex data image.

Syntax

```
IppStatus ippiPackToCplxExtend_32f32fc_C1R(const Ipp32f* pSrc, IppiSize srcSize, int srcStep, Ipp32fc* pDst, int dstStep);
```

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc	Pointer to the source image ROI.
srcSize	Size in pixels of the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source buffer.
pDst	Pointer to the destination image buffer.

dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the source image pSrc in RCPack2D format to complex data format and stores the results in pDst, which is a matrix with complete set of the Fourier coefficients. Note that if the pSrc in RCPack2D format is a real array of dimensions (N x M), then the pDst is a real array of dimensions (2xNxM). This should be taken into account when allocating memory for the function operation.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if pSrc or pDst pointer is NULL.
ippStsStepErr	Indicates an error condition if srcStep or dstStep value is zero or negative.
ippStsSizeErr	Indicates an error condition if srcSize has field with zero or negative value.

CplxExtendToPack

Converts a complex data image to an image in packed format.

Syntax

```
IppStatus ippiCplxExtendToPack_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
IppiSize srcSize, Ipp<dstDatatype>* pDst, int dstStep);
```

Supported values for mod:

32fc32f_C1R
32fc32f_C3R

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc	Pointer to the source image ROI.
srcSize	Size in pixels of the source image ROI.

srcStep	Distance in bytes between starts of consecutive lines in the source buffer.
pDst	Pointer to the destination image buffer.
dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function converts the source image pSrc in complex data format to RCPack2D format and stores the results in pDst, which is a real array of dimensions (N x M). The pSrc is a matrix with complete set of the Fourier coefficients.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if pSrc or pDst pointer is NULL.
ippStsStepErr	Indicates an error condition if srcStep or dstStep value is zero or negative.
ippStsSizeErr	Indicates an error condition if srcSize has field with zero or negative value.

Windowing Functions

This section describes Intel® Integrated Performance Primitives (Intel® IPP) windowing functions used in image processing. Windowing is a mathematical function where pixel values are multiplied to prepare an image for subsequent analysis. This procedure is often called windowing. A windowing function approaches zero towards the edges of the image, avoiding strong distortions of spectral densities in the Fourier domain.

The Intel IPP provides two following types of windowing functions:

- Bartlett windowing function
- Hamming windowing function

These functions generate the windowing samples and apply them to the specified image. To obtain the windowing samples, apply the desired function to the image with all pixel values set to 1.0. As the windowing operation is time-consuming, applying the same window to multiple images may be useful. In this case, use one of the image multiplication functions (ippiMul) to multiply the image's pixel values by the windowing samples.

- WinBartlettGetBufferSize, WinBartlettSepGetBufferSize,
- WinBartlett, WinBartlettSep
- WinHammingGetBufferSize, WinHammingSepGetBufferSize,
- WinHamming, WinHammingSep

WinBartlettGetBufferSize, WinBartlettSepGetBufferSize,

Compute the size of the work buffer for the ippiWinBartlett or ippiWinBartlettSep function.

Syntax

```
IppStatus ippiWinBartlettGetBufferSize (IppDataType dataType, IppiSize roiSize, int* pSize);
```

```
IppStatus ippiWinBartlettSepGetBufferSize (IppDataType dataType, IppiSize roiSize, int* pSize);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`dataType`	Data type for the Bartlett window function. Possible values are: `ipp8u`, `ipp16u`, or `ipp32f`.
`roiSize`	Size, in pixels, of the image ROI.
`pSize`	Pointer to the computed size of the external work buffer, in bytes.

Description

The `ippiWinBartlettGetBufferSize` and `ippiWinBartlettSepGetBufferSize` functions compute the size, in bytes, of the external work buffer needed for the ippiWinBartlett or ippiWinBartlettSep function. The result is stored in the `pSize` parameter.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when `pSize` is `NULL`.
`ippStsSizeErr`	Indicates an error when `roiSize` has a field with a value less than 3.
`ippStsNumChannelsErr`	Indicates an error when `dataType` has an illegal value.

See Also

WinBartlett, WinBartlettSep MODIFIED API. Apply Bartlett window function to the image.

WinBartlett, WinBartlettSep

MODIFIED API. Apply Bartlett window function to the image.

Syntax

Case 1: Not-in-place operation

```
IppStatus ippiWinBartlett_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize roiSize, Ipp8u* pBuffer);
```

Supported values for `mod`:

`8u_C1R`	`16u_C1R`	`32f_C1R`

```
IppStatus ippiWinBartlettSep_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, Ipp8u* pBuffer);
```

Supported values for `mod`:

`8u_C1R`	`16u_C1R`	`32f_C1R`

Case 2: In-place operation

```
IppStatus ippiWinBartlett_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize
roiSize, Ipp8u* pBuffer);
```

Supported values for `mod`:

`8u_C1IR`	`16u_C1IR`	`32f_C1IR`

```
IppStatus ippiWinBartlettSep_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize
roiSize, Ipp8u* pBuffer);
```

Supported values for `mod`:

`8u_C1IR`	`16u_C1IR`	`32f_C1IR`

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance, in bytes, between the starting points of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance, in bytes, between the starting points of consecutive lines in the destination image.
`pSrcDst`	Pointer to the source and destination image ROI for the in-place operation.
`srcDstStep`	Distance, in bytes, between the starting points of consecutive lines in the source and destination image for the in-place operation.
`roiSize`	Size of the source and destination ROI, in pixels.

pBuffer	Pointer to the work buffer. To compute the size of the buffer, use the WinBartlettGetBufferSize WinBartlettSepGetBufferSize or WinBartlettGetBufferSize WinBartlettSepGetBufferSize function.

Description

These functions operate with ROI (see Regions of Interest in Intel IPP).

These functions compute the Bartlett (triangle) window samples, multiply pixel values of the source image pSrc (pSrcDst for in-place flavors) with these samples, and store results in the destination image pDst (pSrcDst for in-place flavors).

The Bartlett window function for one-dimensional case with M elements is defined as follows:

$$w_{bartlett}(i) = \begin{cases} \frac{2i}{M-1}, & 0 \le i \le \frac{M-1}{2} \\ 2 - \frac{2i}{M-1}, & \frac{M-1}{2} < i \le M-1 \end{cases}$$

The ippiWinBartlettSep flavor applies the window function successively to the rows and then to the columns of the image.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when pSrc or pDst pointer is NULL.
ippStsStepErr	Indicates an error when srcStep or dstStep value is zero or negative.
ippStsSizeErr	Indicates an error when width or height of images is less than, or equal to zero.

See Also

WinBartlettGetBufferSize, WinBartlettSepGetBufferSize Compute the size of the work buffer for the ippiWinBartlett or ippiWinBartlettSep function.

Regions of Interest in Intel IPP

WinHammingGetBufferSize, WinHammingSepGetBufferSize,

Compute the size of the work buffer for the ippiWinHamming or ippiWinHammingSep function.

Syntax

```
IppStatus ippiWinHammingGetBufferSize (IppDataType dataType, IppiSize roiSize, int*
pSize);
```

```
IppStatus ippiWinHammingSepGetBufferSize (IppDataType dataType, IppiSize roiSize, int*
pSize);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`dataType`	Data type for the Bartlett window function. Possible values are: `ipp8u`, `ipp16u`, or `ipp32f`.
`roiSize`	Size, in pixels, of the image ROI.
`pSize`	Pointer to the computed size of the external work buffer, in bytes.

Description

The `ippiWinHammingGetBufferSize` and `ippiWinHammingSepGetBufferSize` functions compute the size, in bytes, of the external work buffer needed for the ippiWinHamming or ippiWinHammingSep function. The result is stored in the `pSize` parameter.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when `pSize` is `NULL`.
`ippStsSizeErr`	Indicates an error when `roiSize` has a field with a value less than 3.
`ippStsNumChannelsErr`	Indicates an error when `dataType` has an illegal value.

See Also

WinHamming, WinHammingSep MODIFIED API. Apply Hamming window function to the image.

WinHamming, WinHammingSep

MODIFIED API. Apply Hamming window function to the image.

Syntax

Case 1: Not-in-place operation

```
IppStatus ippiWinHamming_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>*
pDst, int dstStep, IppiSize roiSize, Ipp8u* pBuffer);
```

Supported values for `mod`:

8u_C1R	16u_C1R	32f_C1R

```
IppStatus ippiWinHammingSep_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, Ipp8u* pBuffer);
```

Supported values for `mod`:

8u_C1R	16u_C1R	32f_C1R

Case 2: In-place operation

```
IppStatus ippiWinHamming_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize
roiSize, Ipp8u* pBuffer);
```

Supported values for `mod`:

8u_C1IR	16u_C1IR	32f_C1IR

```
IppStatus ippiWinHammingSep_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize
roiSize, Ipp8u* pBuffer);
```

Supported values for `mod`:

8u_C1IR	16u_C1IR	32f_C1IR

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance, in bytes, between the starting points of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance, in bytes, between the starting points of consecutive lines in the destination image.
`pSrcDst`	Pointer to the source and destination image ROI for the in-place operation.
`srcDstStep`	Distance, in bytes, between the starting points of consecutive lines in the source and destination image for the in-place operation.
`roiSize`	Size of the source and destination ROI, in pixels.
`pBuffer`	Pointer to the work buffer. To compute the size of the buffer, use the ippiWinHammingGetBufferSize or ippiWinHammingSepGetBufferSize function.

Description

These functions operate with ROI (see Regions of Interest in Intel IPP).

These functions compute the Hamming (triangle) window samples, multiply pixel values of the source image `pSrc` (`pSrcDst` for in-place flavors) with these samples, and store results in the destination image `pDst` (`pSrcDst` for in-place flavors).

The Hamming window function for one-dimensional case with `M` elements is defined as follows:

$$w_{hamming}(n) = 0.54 - 0.46\cos\left(\frac{2\pi n}{len-1}\right)$$

The `ippiWinHammingSep` flavor applies the window function successively to the rows and then to the columns of the image.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error when `pSrc` or `pDst` pointer is `NULL`.
`ippStsStepErr`	Indicates an error when `srcStep` or `dstStep` value is zero or negative.
`ippStsSizeErr`	Indicates an error when width or height of images is less than, or equal to zero.

See Also

WinHammingGetBufferSize, WinHammingSepGetBufferSize Compute the size of the work buffer for the ippiWinHamming or ippiWinHammingSep function.

Regions of Interest in Intel IPP

Discrete Cosine Transforms

Discrete Cosine Transform (DCT) of a real 2D image yields output results that are also real, which eliminates the need to use packed format for storing the transformed data. However, forward and inverse DCT functions ippiDCTFwd and ippiDCTInv need different context data structures to be initialized and filled in prior to their use. Consequently, the required workspace buffer size is different for these functions. In case of using an external buffer, its size must be determined by previously calling the respective support function. DCT functions that use context structures implement the modified computation algorithm proposed in [Rao90].

The DCT functions ippiDCT8x8Fwd and ippiDCT8x8Inv working on a fixed 8x8 image buffer need no context data or external workspace buffers. Functions `ippiDCT8x8Inv` meet IEEE-1180 standard requirements (see [IEEE]).

Intel IPP Discrete Cosine Transform functions working on a fixed 8x8 image buffer use Feig and Winograd algorithm ([Feig92]) modified for taking advantage of SIMD instructions. For details on algorithms used in DCT transforms and for more references, see [AP922].

- DCTFwdGetSize, DCTInvGetSize
- DCTFwdInit, DCTInvInit
- DCTFwd
- DCTInv

- DCT8x8Fwd
- DCT8x8Inv, DCT8x8Inv_A10
- DCT8x8FwdLS
- DCT8x8InvLSClip
- DCT8x8Inv_2x2, DCT8x8Inv_4x4
- DCT8x8To2x2Inv, DCT8x8To4x4Inv

DCTFwdGetSize, DCTInvGetSize

Compute the size of the DCT context structure and the size of the required work buffers.

Syntax

```
IppStatus ippiDCTFwdGetSize_32f (IppiSize roiSize, int* pSizeSpec, int* pSizeInit, int* pSizeBuf);
```

```
IppStatus ippiDCTInvGetSize_32f (IppiSize roiSize, int* pSizeSpec, int* pSizeInit, int* pSizeBuf);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`roiSize`	Size of the source and destination ROI, in pixels.
`pSizeSpec`	Pointer to the size of the DCT context structure.
`pSizeInit`	Pointer to the size of the buffer for the DCT initialization function.
`pSizeBuf`	Pointer to the size of the DCT external work buffer.

Description

These functions compute the following:

1. Size of the DCT context structure. The result, in bytes, is stored in the `pSizeSpec` parameter.
2. Size of the work buffer for the ippiDCTFwdInit and ippiDCTInvInit functions. The result, in bytes, is stored in the `pSizeInit` parameter.
3. Size of the work buffer for the ippiDCTFwd and ippiDCTInv functions. The result, in bytes, is stored in the `pSizeBuf` parameter.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when one of the specified pointers is `NULL`.

ippStsSizeErr	Indicates an error when roiSize has a field with a zero or negative value.

See Also

DCTFwdInit DCTInvInit Initialize the context structure for the forward or inverse DCT operation.

DCTFwd Applies a forward discrete cosine transform to an image.

DCTInv Applies an inverse discrete cosine transform to an image.

DCTFwdInit, DCTInvInit

Initialize the context structure for the forward or inverse DCT operation.

Syntax

```
IppStatus ippiDCTFwdInit_32f (IppiDCTFwdSpec_32f* pDCTSpec, IppiSize roiSize, Ipp8u* pMemInit);
```

```
IppStatus ippiDCTInvInit_32f (IppiDCTInvSpec_32f* pDCTSpec, IppiSize roiSize, Ipp8u* pMemInit);
```

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pDCTSpec	Pointer to the forward or inverse DCT context structure for initialization.
roiSize	Size of the source and destination ROI, in pixels.
pMemInit	Pointer to the temporary work buffer.

Description

These functions initialize the pDCTSpec context structure to apply the forward or inverse DCT to two-dimensional image data. The ippiDCTFwd and ippiDCTInv functions use the pointer to the initialized DCT context structure as an argument to compute the forward or inverse DCT for points in the ROI of size roiSize.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.

ippStsSizeErr	Indicates an error when roiSize has a field with a zero or negative value.

See Also

DCTFwd Applies a forward discrete cosine transform to an image.

DCTInv Applies an inverse discrete cosine transform to an image.

DCTFwd

Applies a forward discrete cosine transform to an image.

Syntax

```
IppStatus ippiDCTFwd_<mod> (const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep,
const IppiDCTFwdSpec_32f* pDCTSpec, Ipp8u* pBuffer);
```

Supported values for mod:

32f_C1R
32f_C3R
32f_C4R
32f_AC4R

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance, in bytes, between the starting points of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance, in bytes, between the starting points of consecutive lines in the destination image.
pDCTSpec	Pointer to the previously initialized forward DCT context structure.
pBuffer	Pointer to the external work buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP) of the size that is specified by the ippiDFTInit function.

This function performs a forward DCT on each channel of the source image `pSrc` and writes the result into the corresponding channel of the destination image buffer `pDst`. Note that the function flavor with `AC4` descriptor does not process alpha channel. This function uses the previously initialized `pDCTSpec` context structure to set the mode of calculations and retrieve support data.

You can use this function with the external work buffer `pBuffer` to avoid memory allocation within the functions. Once the work buffer is allocated, it can be used for all following calls to the functions computing DCT. As internal allocation of memory is too expensive operation and depends on operating system and/or runtime libraries used - the use of an external buffer improves performance significantly, especially for the small size transforms.

Before using the forward DCT functions, you need to compute the size of the required buffers and the external work buffer using the ippiDCTFwdGetSize function.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if `pSrc`, `pDst`, or `pDCTSpec` pointer is `NULL`.
`ippStsStepErr`	Indicates an error condition if `srcStep` or `dstStep` value is zero or negative.
`ippStsContextMatchErr`	Indicates an error condition if a pointer to an invalid `pDCTSpec` structure is passed.
`ippStsMemAllocErr`	Indicates an error condition if memory allocation fails.

Example

The code example below demonstrates how to use the ippiDCTFwdGetSize, ippiDCTFwdInit, and `ippiDCTFwd` functions.

```
void DCT_example( void )
{
    Ipp32f Src[8*8] = {
        0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0,
        0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0,
        0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0,
        0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0,
        0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0,
        0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0,
        0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0,
        0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0
    };
    Ipp32f Dst[8*8];
    IppiSize roiSize = {8, 8};
    int srcStep;
    int dstStep;


    int sizeSpec;
    int sizeInit;
    int sizeBuffer;
```

```
    IppiDCTFwdSpec_32f *pMemSpec;
    Ipp8u *pMemInit = 0;
    Ipp8u *pMemBuffer = 0;


    srcStep = dstStep = 8 * sizeof(Ipp32f);


    /// get sizes for required buffers
    ippiDCTFwdGetSize_32f( roiSize, &sizeSpec, &sizeInit, &sizeBuffer );


    /// allocate memory for required buffers
    pMemSpec = (IppiDCTFwdSpec_32f*) ippMalloc ( sizeSpec );


    if ( sizeInit > 0 )
    {
        pMemInit = (Ipp8u*) ippMalloc ( sizeInit );
    }


    if ( sizeBuffer > 0 )
    {
        pMemBuffer = (Ipp8u*) ippMalloc ( sizeBuffer );
    }


    /// initialize DCT specification structure
    ippiDCTFwdInit_32f( pMemSpec, roiSize, pMemInit );


    /// free initialization buffer
    if ( sizeInit > 0 )
    {
        ippFree( pMemInit );
    }


    /// perform forward DCT
    ippiDCTFwd_32f_C1R( Src, srcStep, Dst, dstStep, pMemSpec, pMemBuffer );


    /// ...


    /// free buffers
    if ( sizeBuffer > 0 )
    {
        ippFree( pMemBuffer );
    }


    ippFree( pMemSpec );
}
```

Result:

```
Dst ->  28.0 -18.2  0.0 -1.9  0.0 -0.57   0.0 -0.14
         0.0   0.0  0.0  0.0  0.0   0.0   0.0   0.0
      0.0   0.0  0.0  0.0  0.0   0.0   0.0   0.0
         0.0   0.0  0.0  0.0  0.0   0.0   0.0   0.0
         0.0   0.0  0.0  0.0  0.0   0.0   0.0   0.0
         0.0   0.0  0.0  0.0  0.0   0.0   0.0   0.0
         0.0   0.0  0.0  0.0  0.0   0.0   0.0   0.0
      0.0   0.0  0.0  0.0  0.0   0.0   0.0   0.0
```

DCTInv

Applies an inverse discrete cosine transform to an image.

Syntax

```
IppStatus ippiDCTInv_<mod> (const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep,
const IppiDCTInvSpec_32f* pDCTSpec, Ipp8u* pBuffer);
```

Supported values for `mod` :

`32f_C1R`
`32f_C3R`
`32f_C4R`
`32f_AC4R`

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance, in bytes, between the starting points of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance, in bytes, between the starting points of consecutive lines in the destination image.
`pDCTSpec`	Pointer to the previously initialized inverse DCT context structure.
`pBuffer`	Pointer to the external work buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP) of the size that is specified by the ippiDCTInvInit function.

This function performs an inverse DCT on each channel of the input image `pSrc` and writes the result into the corresponding channel of the output image buffer `pDst`. Note that the function flavor with `AC4` descriptor does not process alpha channel. This function uses the previously initialized `pDCTSpec` context structure to set the mode of calculations and retrieve support data.

The function may be used with the external work buffer `pBuffer` to avoid memory allocation within the functions. Once the work buffer is allocated, it can be used for all following calls to the functions computing DCT. As internal allocation of memory is too expensive operation and depends on operating system and/or runtime libraries used - the use of an external buffer improves performance significantly, especially for the small size transforms.

Before using the inverse DCT functions, you need to compute the size of the required buffers and the external work buffer using the ippiDCTInvGetSize function.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if `pSrc`, `pDst`, or `pDCTSpec` pointer is `NULL`.
`ippStsStepErr`	Indicates an error condition if `srcStep` or `dstStep` value is zero or negative.
`ippStsContextMatchErr`	Indicates an error condition if a pointer to an invalid `pDCTSpec` structure is passed.
`ippStsMemAllocErr`	Indicates an error condition if memory allocation fails.

DCT8x8Fwd

Performs a forward DCT on a 2D buffer of 8x8 size.

Syntax

Case 1: Not-in-place operation

```
IppStatus ippiDCT8x8Fwd_<mod>(const Ipp<datatype>* pSrc, Ipp<datatype>* pDst);
```

Supported values for `mod`:

`16s_C1`	`32f_C1`

Case 2: Not-in-place operation with ROI

```
IppStatus ippiDCT8x8Fwd_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pDst);
```

Supported values for `mod`:

`16s_C1R`	`8u16s_C1R`

Case 3: In-place operation

```
IppStatus ippiDCT8x8Fwd_<mod>(Ipp<datatype>* pSrcDst);
```

Supported values for `mod`:

`16s_C1I`	`32f_C1I`

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image buffer.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image buffer for operations with ROI.
`pDst`	Pointer to the destination image buffer.
`pSrcDst`	Pointer to the source and destination image for in-place operations.

Description

Some flavors operate with ROI (see Regions of Interest in Intel IPP).

This function computes the forward discrete cosine transform of short integer or floating-point data in a 2D buffer of 8x8 size. No prerequisites are needed to use this transform function.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if `pSrc`, `pDst`, or `pSrcDst` pointer is `NULL`.
`ippStsStepErr`	Indicates an error condition if `srcStep` value is zero or negative.

Example

The code example below illustrates the use of `ippiDCT8x8Fwd` function.

```
IppStatus dct16s( void ) {
   Ipp16s x[64] = {0};
   IppiSize roi = {8,8};
   int i;
   for( i=0; i<8; ++i ) {
       ippiSet_16s_C1R( (Ipp16s)i, x+8*i+i, 8*sizeof(Ipp16s), roi );
       --roi.width;
       --roi.height;
```

```
    }
    return ippiDCT8x8Fwd_16s_C1I( x );
}
```

The destination image `x` contains:

```
18 -9 -2 -1 -1  0  0  0
-9  7  0  0  0  0  0  0
-2  0  2  0  0  0  0  0
-1  0  0  1  0  0  0  0
 0  0  0  0  1  0  0  0
 0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0
```

DCT8x8Inv, DCT8x8Inv_A10

Performs an inverse DCT on a 2D buffer of 8x8 size.

Syntax

Case 1: Not-in-place operation

```
IppStatus ippiDCT8x8Inv_<mod>(const Ipp<datatype>* pSrc, Ipp<datatype>* pDst);
```

Supported values for `mod`:

`16s_C1`	`32f_C1`

```
IppStatus ippiDCT8x8Inv_A10_16s_C1(const Ipp16s* pSrc, Ipp16s* pDst);
```

Case 2: Not-in-place operation with ROI

```
IppStatus ippiDCT8x8Inv_<mod>(const Ipp<srcDatatype>* pSrc, Ipp<dstDatatype>* pDst, int dstStep);
```

Supported values for `mod`:

`16s_C1R`	`16s8u_C1R`

Case 3: In-place operation

```
IppStatus ippiDCT8x8Inv_<mod>(Ipp<datatype>* pSrcDst);
```

Supported values for `mod`:

`16s_C1I`	`32f_C1I`

```
IppStatus ippiDCT8x8Inv_A10_16s_C1I( Ipp16s* pSrcDst);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

pSrc	Pointer to the source image.
pDst	Pointer to the destination buffer.
dstStep	Distance in bytes between starts of consecutive lines in the destination buffer for operations with ROI.
pSrcDst	Pointer to the source and destination image for in-place operations.

Description

Some flavors operate with ROI (see Regions of Interest in Intel IPP).

This function computes the inverse discrete cosine transform of data in a 2D buffer of 8x8 size. No prerequisites are needed to use this transform function.

NOTE
CAUTION
Source data for 16s functions must be the result of the forward discrete cosine transform of data from the range [-512, 511] for flavors with A10 modifier (ippiDCT8x8Inv_A10), and from the range [-256, 255] for flavors without A10 modifier (ippiDCT8x8Inv); they cannot be arbitrary data from the range [-32768, 32767].

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if pSrc, pDst, or pSrcDst pointer is NULL.
ippStsStepErr	Indicates an error condition if dstStep value is zero or negative.

DCT8x8FwdLS

Performs a forward DCT on a 2D buffer of 8x8 size with prior data conversion and level shift.

Syntax

```
IppStatus ippiDCT8x8FwdLS_8u16s_C1R(const Ipp8u* pSrc, int srcStep, Ipp16s* pDst,
Ipp16s addVal);
```

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc	Pointer to the source image buffer.
pDst	Pointer to the destination buffer.
srcStep	Distance in bytes between starts of consecutive lines in the source image buffer.
addVal	The level shift value.

Description

This function operates with ROI (see Regions of Interest in Intel IPP) that is a 2D buffer of 8x8 size in this case, thus there is no need to specify its size.

This function first converts data in the buffer pSrc from unsigned Ipp8u type to the signed Ipp16s type and then performs level shift operation by adding the constant value addVal to each sample. After that, the function performs the forward discrete cosine transform of the modified data. The result is stored in pDst.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if pSrc or pDst pointer is NULL.
ippStsStepErr	Indicates an error condition if srcStep value is zero or negative.

DCT8x8InvLSClip

Performs an inverse DCT on a 2D buffer of 8x8 size with further data conversion and level shift.

Syntax

```
IppStatus ippiDCT8x8InvLSClip_16s8u_C1R(const Ipp16s* pSrc, Ipp8u* pDst, int dstStep,
Ipp16s addVal, Ipp8u clipDown, Ipp8u clipUp);
```

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc	Pointer to the source image buffer.
pDst	Pointer to the destination image buffer.

dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.
addVal	The level shift value.
clipDown	The lower bound for the range of output values.
clipUp	The upper bound for the range of output values.

Description

This function operates with ROI (see Regions of Interest in Intel IPP) that is a 2D buffer of 8x8 size in this case, thus there is no need to specify its size.

This function performs the inverse discrete cosine transform of the buffer pSrc. After completing the DCT, this function performs level shift operation by adding the constant value addVal to each sample. Finally, the function converts data from the signed Ipp16s type to the unsigned Ipp8u type. The output data are clipped to the range [clipDown..clipUp]. The result is stored in the destination buffer pDst.

NOTE
CAUTION
Source data for 16s flavors must be the result of the forward discrete cosine transform of data from the range [-256, 255], they cannot be arbitrary data from the range [-32768, 32767].

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if pSrc or pDst pointer is NULL.
ippStsStepErr	Indicates an error condition if dstStep value is zero or negative.

DCT8x8Inv_2x2, DCT8x8Inv_4x4

Perform an inverse DCT on a top left quadrant of size 2x2 or 4x4 in the 2D buffer of size 8x8.

Syntax

```
IppStatus ippiDCT8x8Inv_2x2_16s_C1(const Ipp16s* pSrc, Ipp16s* pDst);
IppStatus ippiDCT8x8Inv_4x4_16s_C1(const Ipp16s* pSrc, Ipp16s* pDst);
IppStatus ippiDCT8x8Inv_2x2_16s_C1I(Ipp16s* pSrcDst);
IppStatus ippiDCT8x8Inv_4x4_16s_C1I(Ipp16s* pSrcDst);
```

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc	Pointer to the source image.
pDst	Pointer to the destination buffer.
pSrcDst	Pointer to the source and destination buffer for in-place operations.

Description

These functions compute the inverse discrete cosine transform of non-zero elements in the top left quadrant of size 2x2 or 4x4 in the 2D buffer of 8x8 size. No prerequisites are needed to use this transform function.

NOTE
CAUTION
Source data for 16s flavors must be the result of the forward discrete cosine transform of data from the range [-256, 255], they cannot be arbitrary data from the range [-32768, 32767].

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.

DCT8x8To2x2Inv, DCT8x8To4x4Inv

Perform an inverse DCT on a 2D buffer of 8x8 size with further downsampling to 2x2 or 4x4 size.

Syntax

```
IppStatus ippiDCT8x8To2x2Inv_16s_C1(const Ipp16s* pSrc, Ipp16s* pDst);
IppStatus ippiDCT8x8To4x4Inv_16s_C1(const Ipp16s* pSrc, Ipp16s* pDst);
IppStatus ippiDCT8x8To2x2Inv_16s_C1I(Ipp16s* pSrcDst);
IppStatus ippiDCT8x8To4x4Inv_16s_C1I(Ipp16s* pSrcDst);
```

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc	Pointer to the source image.
pDst	Pointer to the destination buffer.
pSrcDst	Pointer to the source and destination buffer for in-place operations.

Description

These functions compute the inverse discrete cosine transform of the 2D buffer pSrc of 8x8 size. Then the functions perform downsampling of the result by averaging to the destination buffer pDst of size 2x2 or 4x4.

In-place flavors of the functions perform operations on the source and destination buffer pSrcDst.

NOTE
CAUTION
Source data for 16s flavors must be the result of the forward discrete cosine transform of data from the range [-256, 255], they cannot be arbitrary data from the range [-32768, 32767].

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.

Image Statistics Functions

This chapter describes the Intel® IPP image processing functions that can be used to compute the following statistical parameters of an image:

- sum, integrals, mean and standard deviation of pixel values
- intensity histogram of pixel values
- minimum and maximum pixel values
- spatial and central moments of order 0 to 3
- the infinity, L1, and L2 norms of the image pixel values and of the differences between pixel values of two images
- relative error values for the infinity, L1, and L2 norms of differences between pixel values of two images
- universal image quality index
- proximity measures of an image and a template (another image).

- Sum
- Integral
- SqrIntegral
- TiltedIntegral
- TiltedSqrIntegral
- Mean
- Mean_StdDev
- RectStdDev
- TiltedRectStdDev
- HistogramGetBufferSize
- HistogramGetLevels

- HistogramInit, HistogramUniformInit
- Histogram
- CountInRange
- BlockMinMax
- Min
- MinIndx
- Max
- MaxIndx
- MinMax
- MinMaxIndx
- MaxEvery
- MinEvery
- FindPeaks3x3GetBufferSize
- FindPeaks3x3
- Image Moments
- Image Norms
- Image Quality Index
- Image Proximity Measures

Sum

Computes the sum of image pixel values.

Syntax

Case 1: Operation on one-channel integer data

```
IppStatus ippiSum_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp64f* pSum);
```

Supported values for `mod`:

8u_C1R	16u_C1R	16s_C1R

Case 2: Operation on one-channel floating-point data

```
IppStatus ippiSum_32f_C1R(const Ipp32f* pSrc, int srcStep, IppiSize roiSize, Ipp64f*
pSum, IppHintAlgorithm hint);
```

Case 3: Operation on multi-channel integer data

```
IppStatus ippiSum_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp64f sum[3]);
```

Supported values for `mod`:

8u_C3R	16u_C3R	16s_C3R

```
IppStatus ippiSum_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp64f sum[4]);
```

Supported values for `mod`:

8u_C4R	16u_C4R	16s_C4R

Case 4: Operation on multi-channel floating-point data

```
IppStatus ippiSum_32f_C3R(const Ipp32f* pSrc, int srcStep, IppiSize roiSize, Ipp64f
sum[3], IppHintAlgorithm hint);

IppStatus ippiSum_32f_C4R(const Ipp32f* pSrc, int srcStep, IppiSize roiSize, Ipp64f
sum[4], IppHintAlgorithm hint);
```

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
roiSize	Size of the source ROI in pixels.
pSum	Pointer to the computed sum of pixel values.
sum	Array containing computed sums of channel values of pixels in the source buffer.
hint	Option to select the algorithmic implementation of the function.

Description

This function operates with ROI (see Regions of Interest in Intel IPP). This function computes the sum of pixel values pSum for the source image pSrc using algorithm indicated by the hint argument (see Table "Hint Arguments for Image Moment Functions"). In case of a multi-channel image, the sum is computed over each channel and stored in the array sum.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if the pSrc or pSum pointer is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value.

Example

The code example below demonstrates the use of `ippiSum` function:

```
IppStatus sum( void ) {
Ipp64f sum;
Ipp8u x[5*4];
IppiSize roi = {5,4};
ippiSet_8u_C1R( 1, x, 5, roi );
return ippiSum_8u_C1R( x, 5, roi, &sum);
}
```

Integral

Transforms an image to the integral representation.

Syntax

```
IppStatus ippiIntegral_8u32s_C1R(const Ipp8u* pSrc, int srcStep, Ipp32s* pDst, int dstStep, IppiSize srcRoiSize, Ipp32s val);
IppStatus ippiIntegral_8u32f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize srcRoiSize, Ipp32f val);
IppStatus ippiIntegral_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize srcRoiSize);
```

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`pDst`	Pointer to the ROI in the destination integral image.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image.
`srcRoiSize`	Size of source and destination image ROI in pixels.
`val`	The value to add to `pDst` image pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP). This function transforms a source image `pSrc` to the integral image `pDst`. Pixel values of the destination image `pDst` are computed using pixel values of the source image `pSrc` and the specified value `val` in accordance with the following formula:

image:: _images/GUID-71346E88-571C-46DD-B693-2AC6E97B9A11-low.jpg

where `i`, `j` are coordinates of the destination image pixels (see Figure 11-1) varying in the range `i = 1` ,..., `srcRoiSize.height`, `j = 0`,..., `srcRoiSize.width`. Pixel values of zero row and column of `pDst(``i=0)` `is set to ``val`.

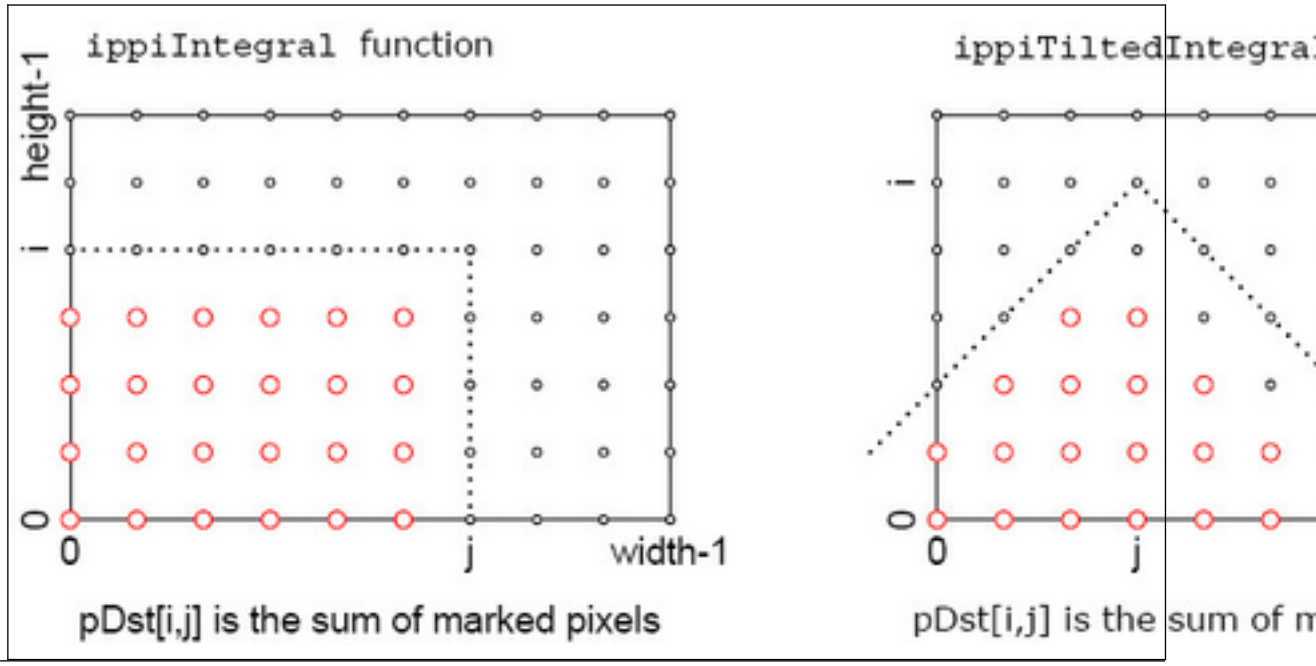
For the `ippiIntegral_32f_C1` function flavor the value of `val` is considered to be equal to zero.

The size of the destination images is (`srcRoiSize.width + 1`) x (`srcRoiSize.height + 1`).

Figure Operation of the Integral and TiltedIntegral functions shows what pixels (red circles) of the source image are used in computation new pixel values in the `i`, `j` coordinates.

For large images the result of summation can exceed the upper bound of the output data type. Table Maximum Image Size for Integral Functions lists the maximum image size for different function flavors and values.

Operation of the Integral and TiltedIntegral functions



Maximum Image Size for Integral Functions

Function Flavor	Value `val`	Maximum Image Size
`ippiIntegral_8u32s_C1R`	0	$(2^{31}-1)/255$
	-2^{31}	$2^{32}/255$
`ippiIntegral_8u32f_C1R`	0	$2^{24}/255$
	-2^{24}	$(2^{25}+1)/255$

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if pSrc or pDst is NULL.
ippStsSizeErr	Indicates an error condition if srcRoiSize has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if srcStep is less than srcRoiSize.width * <``pixelSize>, or ``dstStep is less than (srcRoiSize.width+1) * <pixelSize> .
ippStsNotEvenStepErr	Indicates an error condition if srcStep or dstStep is not divisible by <pixelSize>.

Example

The code example below demonstrates how to use the ippiIntegral_8u32s_C1R function:

```
void func_integral_8u32s_C1R()
{
    Ipp8u pSrc[5*4];
    Ipp32s pDst[6*5];
    IppiSize ROI = {5,4};
    ippiSet_8u_C1R(1,pSrc,5,ROI);
    Ipp32s val = 1;
    ippiIntegral_8u32s_C1R(pSrc, 5, pDst, 6*sizeof(Ipp32s), ROI, val);
}
```

Result:

```
pSrc -> 1 1 1 1 1           pDst -> 1 1 1 1  1  1
        1 1 1 1 1                   1 2 3 4  5  6
        1 1 1 1 1                   1 3 5 7  9  11
        1 1 1 1 1                   1 4 7 10 13 16
                                    1 5 9 13 17 21
```

The code example below demonstrates how to use the ippiIntegral_32f_C1R function:

```
void func_integral_32f_C1R()
{
    Ipp32f pSrc[5*4];
    Ipp32f pDst[6*5];
    IppiSize ROI = {5, 4};


    ippiSet_32f_C1R(1, pSrc, 5*sizeof(Ipp32f), ROI);
    ippiIntegral_32f_C1R(pSrc, 5*sizeof(Ipp32f), pDst, 6*sizeof(Ipp32f), ROI);
}
```

Result:

```
pSrc -> 1.0 1.0 1.0 1.0 1.0        pDst -> 0.0 0.0 0.0 0.0  0.0  0.0
        1.0 1.0 1.0 1.0 1.0                0.0 1.0 2.0 3.0  4.0  5.0
        1.0 1.0 1.0 1.0 1.0                0.0 2.0 4.0 6.0  8.0  10.0
        1.0 1.0 1.0 1.0 1.0                0.0 3.0 6.0 9.0  12.0 15.0
                                           0.0 4.0 8.0 12.0 16.0 20.0
```

SqrIntegral

Transforms an image to integral and integral of pixel squares representations.

Syntax

```
IppStatus ippiSqrIntegral_8u32s64f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32s* pDst,
int dstStep, Ipp64f* pSqr, int sqrStep, IppiSize roiSize, Ipp32s val, Ipp64f valSqr);

IppStatus ippiSqrIntegral_8u32f64f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32f* pDst,
int dstStep, Ipp64f* pSqr, int sqrStep, IppiSize roiSize, Ipp32f val, Ipp64f valSqr);

IppStatus ippiSqrIntegral_8u32s_C1R(const Ipp8u* pSrc, int srcStep, Ipp32s* pDst, int
dstStep, Ipp32s* pSqr, int sqrStep, IppiSize roiSize, Ipp32s val, Ipp32s valSqr);
```

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`pDst`	Pointer to the ROI in the destination integral image.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image.
`pSqr`	Pointer to the ROI of the destination integral image of pixel squares.
`sqrStep`	Distance in bytes between starts of consecutive lines in the destination integral image of pixel squares.
`roiSize`	Size of source image ROI in pixels.
`val`	The value to add to `pDst` image pixels.
`valSqr`	The value to add to `pSqr` image pixels

Description

This function operates with ROI (see Regions of Interest in Intel IPP). This function builds two destination images: integral image `pDst` and integral image of pixel squares `pSqr`. Pixel values of `pDst` are computed using pixel values of the source image `pSrc` and the specified value `val` in accordance with the following formula:

$$pDst[i, j] = val + \sum_{k<i} \sum_{l<j} pSrc[k, l]$$

Pixel values of `pSqr` are computed using pixel values of the source image `pSrc` and the specified value `valSqr` in accordance with the following formula:

$$pSqr[i, j] = valSqr + \sum_{k<i} \sum_{l<j} pSrc[k, l]^2$$

where `i`, `j` are coordinates of the destination image pixels (see Figure "Operation of the Integral and TiltedIntegral functions") varying in the range `i = 1`, …, `roiSize.height`, `j = 0`,…, `roiSize.width`. Pixel values of zero row and column are set to `val` for `pDst`, and to `valSqr` for `pSqr`. The size of both destination images is (`roiSize.width + 1`) x (`roiSize.height` + 1).

Figure "Operation of the Integral and TiltedIntegral functions" shows what pixels (red circles) of the source image are used in computation new pixel values in the `i`, `j` coordinates.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize` has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if `srcStep` is less than `roiSize.width * <``pixelSize>`, or `` `dstStep` or `sqrStep` is less than `(roiSize.width+1) * <pixelSize>`.
`ippStsNotEvenStepErr`	Indicates an error condition if `dstStep` is not divisible by 4, or `sqrStep` is not divisible by 8.

TiltedIntegral

Transforms an image to the tilted integral representation.

Syntax

```
IppStatus ippiTiltedIntegral_8u32f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32f* pDst,
int dstStep, IppiSize roiSize, Ipp32f val);
```

```
IppStatus ippiTiltedIntegral_8u32s_C1R(const Ipp8u* pSrc, int srcStep, Ipp32s* pDst,
int dstStep, IppiSize roiSize, Ipp32s val);
```

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`pDst`	Pointer to the ROI in the destination integral image.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image.
`roiSize`	Size of source image ROI in pixels.
`val`	The value to add to pDst image pixels

Description

This function operates with ROI (see Regions of Interest in Intel IPP). This function transforms a source image `pSrc` to the tilted integral image `pDst`. Pixel values of the destination image `pDst` are computed using pixel values of the source image `pSrc` and the specified value `val` in accordance with the following formula:

$$pDst[i, j] = val + \sum_{\substack{k+l \le i+j-2 \\ k-l \le i-j+1}} pSrc[k, l]$$

where `i`, `j` are coordinates of the destination image pixels (see Figure "Operation of the Integral and TiltedIntegral functions")varying in the range `i = 2,...,roiSize.height + 1,j = 0,…, roiSize.width + 1`. Pixel values of rows 0 and 1 of the destination image `pDst(``i=0) is set to ``val`.

The size of the destination images is (`roiSize.width + 2`) x (`roiSize.height + 2`).

Figure "Operation of the Integral and TiltedIntegral functions" shows what pixels (red circles) of the source image are used in computation new pixel values in the `i`, `j` coordinates.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if `pSrc` or `pDst` is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize` has a field with zero or negative value.

ippStsStepErr	Indicates an error condition if srcStep is less than roiSize.width * <``pixelSize>, or ``dstStep is less than (roiSize.width+2) * <pixelSize>.
ippStsNotEvenStepErr	Indicates an error condition if one dstStep is not divisible by 4.

TiltedSqrIntegral

Transforms an image to tilted integral and tilted integral of pixel squares representations.

Syntax

```
IppStatus ippiTiltedSqrIntegral_8u32s_C1R(const Ipp8u* pSrc, int srcStep, Ipp32s* pDst, int dstStep, Ipp32s* pSqr, int sqrStep, IppiSize roiSize, Ipp32s val, Ipp32s valSqr);
```

```
IppStatus ippiTiltedSqrIntegral_8u32s64f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32s* pDst, int dstStep, Ipp64f* pSqr, int sqrStep, IppiSize roiSize, Ipp32s val, Ipp64f valSqr);
```

```
IppStatus ippiTiltedSqrIntegral_8u32f64f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32f* pDst, int dstStep, Ipp64f* pSqr, int sqrStep, IppiSize roiSize, Ipp32f val, Ipp64f valSqr);
```

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the ROI in the destination integral image.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
pSqr	Pointer to the ROI of the destination integral image of pixel squares.
sqrStep	Distance in bytes between starts of consecutive lines in the destination integral image of pixel squares.
roiSize	Size of source image ROI in pixels.

val	The value to add to pDst image pixels.
valSqr	The value to add to pSqr image pixels

Description

This function operates with ROI (see Regions of Interest in Intel IPP). This function builds two destination image: tilted integral image pDst and tilted integral image of pixel squares pSqr.

Pixel values of pDst are computed using pixel values of the source image pSrc and the specified value val in accordance with the following formula:

$$pDst[i, j] = val + \sum_{\substack{k+l \le i+j-2 \\ k-l \le i-j+1}} pSrc[k, l]$$

Pixel values of pSqr are computed using pixel values of the source image pSrc and the specified value valSqr in accordance with the following formula:

$$pSqr[i, j] = valSqr + \sum_{\substack{k+l \le i+j-2 \\ k-l \le i-j+1}} pSrc[k, l]^2$$

where i, j are coordinates of the destination image pixels (see Figure "Operation of the Integral and TiltedIntegral functions") varying in the range i = 2,...,roiSize.height,j = 0,..., roiSize.width. Pixel values of zero and first rows (i = 0,1) are set to val for pDst, and to valSqr for pSqr. The size of both destination images is (roiSize.width + 2) x (roiSize.height + 2).

Figure "Operation of the Integral and TiltedIntegral functions" shows what pixels (red circles) of the source image are used in computation new pixel values in the i, j coordinates.

Return Values

ippStsNoEr	Indicates no error.
ippStsNullPtrErr	Indicates an error if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if srcStep is less than roiSize.width * <``pixelSize>, or ``dstStep or sqrStep is less than (roiSize.width+2) * <pixelSize> .
ippStsNotEvenStepErr	Indicates an error condition if dstStep is not divisible by 4, or sqrStep is not divisible by 8.

Mean

Computes the mean of image pixel values.

Syntax

Case 1: Operation on one-channel integer data

```
IppStatus ippiMean_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp64f* pMean);
```

Supported values for `mod`:

8u_C1R	16u_C1R	16s_C1R

Case 2: Operation on one-channel floating-point data

```
IppStatus ippiMean_32f_C1R(const Ipp32f* pSrc, int srcStep, IppiSize roiSize, Ipp64f*
pMean, IppHintAlgorithm hint);
```

Case 3: Masked operation on one-channel data

```
IppStatus ippiMean_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp8u* pMask,
int maskStep, IppiSize roiSize, Ipp64f* pMean);
```

Supported values for `mod`:

8u_C1MR	16u_C1MR	32f_C1MR

Case 4: Operation on multi-channel integer data

```
IppStatus ippiMean_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp64f mean[3]);
```

Supported values for `mod`:

8u_C3R	16u_C3R	16s_C3R

```
IppStatus ippiMean_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp64f mean[4]);
```

Supported values for `mod`:

8u_C4R	16u_C4R	16s_C4R

Case 5: Operation on multi-channel floating-point data

```
IppStatus ippiMean_<mod>(const Ipp32f* pSrc, int srcStep, IppiSize roiSize, Ipp64f
mean[3], IppHintAlgorithm hint);
```

Supported values for `mod`:

32f_C3R

```
IppStatus ippiMean_32f_C4R(const Ipp32f* pSrc, int srcStep, IppiSize roiSize, Ipp64f
mean[4], IppHintAlgorithm hint);
```

Case 6: Masked operation on multi-channel data

```
IppStatus ippiMean_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp8u* pMask,
int maskStep, IppiSize roiSize, int coi, Ipp64f* pMean);
```

Supported values for `mod`:

`8u_C3CMR`	`16u_C3CMR`	`32f_C3CMR`

Include Files

`ippi.h`

`ippcv.h`

Domain Dependencies

Flavors declared in `ippi.h`:

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Flavors declared in `ippcv.h`:

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`pMask`	Pointer to the mask image.
`maskStep`	Distance in bytes between starts of consecutive lines in the mask image.
`roiSize`	Size of the source ROI in pixels.
`coi`	Channel of interest (for color images only); can be 1, 2, or 3.
`pMean`	Pointer to the computed mean of pixel values.
`mean`	Array containing computed mean values for each channel of a multi-channel image.
`hint`	Option to select the algorithmic implementation of the function.

Description

The flavors of the function `ippiMean` that perform masked operations are declared in the `ippcv.h` file. All other function flavors are declared in the `ippi.h` file. This function operates with ROI (see Regions of Interest in Intel IPP). It computes the mean (average) of pixel values `pMean` for the source image `pSrc`. Computation algorithm is specified by the `hint` argument (see Table "Hint Arguments for Image Moment Functions"). For non-masked operations on a multi-channel image (Case 4, 5), the mean is computed over each channel and stored in the array `mean`. In the mask multi-channel mode (Case 6), the mean is computed for a single channel of interest specified by `coi`.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value.
ippStsStepErr	Indicates an error condition in mask mode, if srcStep or maskStep is less than roiSize.width * <pixelSize>.
ippStsNotEvenStepErr	Indicates an error condition in mask mode if steps for floating-point images cannot be divided by 4.
ippStsCOIErr	Indicates an error when coi is not 1, 2, or 3.

Example

The code example below shows how to use the ippiMean function.

```
IppStatus mean( void ) {
   Ipp64f mean;
   Ipp8u x[5*4];
   IppiSize roi = {5,4};
   ippiSet_8u_C1R( 3, x, 5, roi );
   return ippiMean_8u_C1R( x, 5, roi, &mean );
}
```

Mean_StdDev

Computes the mean and standard deviation of image pixel values.

Syntax

Case 1: Operation on one-channel data

```
IppStatus ippiMean_StdDev_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize, Ipp64f* pMean, Ipp64f* pStdDev);
```

Supported values for mod:

8u_C1R	16u_C1R	32f_C1R

Case 2: Masked operation on one-channel data

```
IppStatus ippiMean_StdDev_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp8u* pMask, int maskStep, IppiSize roiSize, Ipp64f* pMean, Ipp64f* pStdDev);
```

Supported values for mod:

8u_C1MR	16u_C1MR	32f_C1MR

Case 3: Operation on multi-channel data

```
IppStatus ippiMean_StdDev_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize, int coi, Ipp64f* pMean, Ipp64f* pStdDev);
```

Supported values for `mod`:

`8u_C3CR`	`16u_C3CR`	`32f_C3CR`

Case 4: Masked operation on multi-channel data

```
IppStatus ippiMean_StdDev_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp8u*
pMask, int maskStep, IppiSize roiSize, int coi, Ipp64f* pMean, Ipp64f* pStdDev);
```

Supported values for `mod`:

`8u_C3CMR`	`16u_C3CMR`	`32f_C3CMR`

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`pMask`	Pointer to the mask image.
`maskStep`	Distance in bytes between starts of consecutive lines in the mask image.
`roiSize`	Size of the source ROI in pixels
`coi`	Channel of interest (for color images only); can be 1, 2, or 3.
`pMean`	Pointer to the computed mean of pixel values.
`pStdDev`	Pointer to the computed standard deviation of pixel values in the image.

Description

This function operates with ROI (see Regions of Interest in Intel IPP). This function computes the mean and standard deviation of pixel values in the ROI of the source image `pSrc`. In the mask mode, the computation is done over pixels selected by nonzero mask values. In the multi-channel mode, the mean is computed for a single channel of interest specified by `coi`. If any of the parameters `pMean` or `pStdDev` is not required, the zero pointer is to be passed to the corresponding parameter.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr	Indicates an error when pSrc or pMask pointer is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if srcStep or maskStep is less than roiSize.width * <pixelSize>.
ippStsNotEvenStepErr	Indicates an error condition if steps for floating-point images cannot be divided by 4.
ippStsCOIErr	Indicates an error when coi is not 1, 2, or 3.

RectStdDev

Computes the standard deviation of the integral images.

Syntax

```
IppStatus ippiRectStdDev_32f_C1R(const Ipp32f* pSrc, int srcStep, const Ipp64f* pSqr, int sqrStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiRect rect);
```

```
IppStatus ippiRectStdDev_32s32f_C1R(const Ipp32s* pSrc, int srcStep, const Ipp64f* pSqr, int sqrStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiRect rect);
```

```
IppStatus ippiRectStdDev_32s_C1RSfs(const Ipp32s* pSrc, int srcStep, const Ipp32s* pSqr, int sqrStep, Ipp32s* pDst, int dstStep, IppiSize roiSize, IppiRect rect, int scaleFactor);
```

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Pointer to the ROI in the source integral image.
srcStep	Distance in bytes between starts of consecutive lines in the source integral image.
pSqr	Pointer to the ROI in the source integral image of pixel squares.
sqrStep	Distance in bytes between starts of consecutive lines in the source integral image of pixel squares.
pDst	Pointer to the destination image ROI.

dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of destination image ROI in pixels.
rect	Rectangular window.
scaleFactor	Scale factor (see Integer Result Scaling).

Description

This function operates with ROI (see Regions of Interest in Intel IPP). This function computes the standard deviation for each pixel in the rectangular window `rect` using the integral image `pSrc` and integral image of pixel squares `pSqr`. The computations are performed in accordance with the following formulas:

$$pDst[i, j] = \sqrt{max\left(0, \frac{sumSqr \cdot numPix - sum^2}{numPix^2}\right)}$$

where `i, j` are coordinates of the destination image pixels varying in the range `i = 0,...,roiSize.height - 1,j` = 0,…, `roiSize.width - 1`;

`sum = pSrc[i + rect.y + rect.height, j + rect.x + rect.width] - pSrc[i + rect.y, j + rect.x + rect.width] - pSrc[i + rect.y + rect.height, j + rect.x] + pSrc[i + rect.y, j + rect.x];`

`sumSqr = pSqr[i + rect.y + rect.height, j + rect.x + rect.width] - pSqr[i + rect.y, j + rect.x + rect.width] - pSqr[i + rect.y + rect.height, j + rect.x] + pSqr[i + rect.y, j + rect.x];`

`numPix = rect.height * rect.width.`

The minimum size of each source images `pSrc` and `pSqr` should be (`roiSize.width + rect.x + rect.width`) x (`roiSize.height + rect.y + rect.height`).

The source images `pSrc` and `pSqr` can be obtained by using the functions ippiIntegral or ippiSqrIntegral.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if one of the specified pointers is `NULL`.
ippStsSizeErr	Indicates an error condition if `roiSize` has a field with zero or negative value.
ippStsSizeErr	Indicates an error condition if `rect.width` or `rect.height` is less than or equal to zero, or if `rect.x` or `rect.y` is less than zero.

ippStsStepErr	Indicates an error condition if srcStep or sqrStep is less than (roiSize.width+rect.x+rect.width +1) * <pixelSize>, or dstStep is less than roiSize.width * <pixelSize>.
ippStsNotEvenStepErr	Indicates an error condition if sqrStep is not divisible by 8, or one of pSrc and dstStep is not divisible by 4.

TiltedRectStdDev

Computes the standard deviation of the tilted integral images.

Syntax

```
IppStatus ippiTiltedRectStdDev_32f_C1R(const Ipp32f* pSrc, int srcStep, const Ipp64f* pSqr, int sqrStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiRect rect);
```

```
IppStatus ippiTiltedRectStdDev_32s_C1RSfs(const Ipp32s* pSrc, int srcStep, const Ipp32s* pSqr, int sqrStep, Ipp32s* pDst, int dstStep, IppiSize roiSize, IppiRect rect, int scaleFactor);
```

```
IppStatus ippiTiltedRectStdDev_32s32f_C1R(const Ipp32s* pSrc, int srcStep, const Ipp64f* pSqr, int sqrStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiRect rect);
```

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Pointer to the ROI in the source integral image.
srcStep	Distance in bytes between starts of consecutive lines in the source integral image.
pSqr	Pointer to the ROI in the source integral image of pixel squares.
sqrStep	Distance in bytes between starts of consecutive lines in the source integral image of pixel squares.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of destination image ROI in pixels.
rect	Rectangular window.

scaleFactor	Scale factor (see Integer Result Scaling).

Description

This function operates with ROI (see Regions of Interest in Intel IPP). This function computes the standard deviation for each pixel in the rectangular window rect using the tilted integral image pSrc and tilted integral image of pixel squares pSqr. The computations are performed in accordance with the following formulas:

$$pDst[i, j] = \sqrt{max\left(0, \frac{sumSqr \cdot numPix - sum^2}{numPix^2}\right)}$$

where i, j are coordinates of the destination image pixels varying in the range i = 0,...,roiSize.height - 1,j = 0,..., roiSize.width - 1;

sum = pSrc[i + rect.x - rect.y + rect.height + rect.width, j + rect.x + rect.y - rect.height + rect.width] - pSrc[i + rect.x - rect.y + rect.width, j + rect.x + rect.y + rect.width] - pSrc[i + rect.x - rect.y + rect.height, j + rect.x - rect.y - rect.height] + pSrc[i + rect.x - rect.y, j + rect.x + rect.y];

sumSqr = pSqr[i + rect.x - rect.y + rect.height + rect.width, j + rect.x + rect.y - rect.height + rect.width] - pSqr[i + rect.x - rect.y + rect.width, j + rect.x + rect.y + rect.width] - pSqr[i + rect.x - rect.y + rect.height, j + rect.x - rect.y - rect.height] + pSqr[i + rect.x - rect.y, j + rect.x + rect.y];

numPix = 2 * rect.height * rect.width.

The minimum size of each source images pSrc and pSqr should be (roiSize.width + rect.height + rect.width - 2) x (roiSize.height + rect.x + rect.y + rect.height + rect.width - 2).

The source images pSrc and pSqr can be obtained by using the functions ippiTiltedIntegral or ippiTiltedSqrIntegral.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value.
ippStsSizeErr	Indicates an error condition if rect.width or rect.height is less than or equal to zero, or if rect.x or rect.y is less than zero.
ippStsStepErr	Indicates an error condition if srcStep or sqrStep is less than (roiSize.width+rect.x+rect.width +1) * <pixelSize>, or dstStep is less than roiSize.width * <pixelSize>.

ippStsNotEvenStepErr	Indicates an error condition if sqrStep is not divisible by 8, or one of pSrc and dstStep is not divisible by 4.

HistogramGetBufferSize

Computes the size of the specification structure and work buffer for the ippiHistogram function.

Syntax

```
IppStatus ippiHistogramGetBufferSize(IppDataType dataType, IppiSize roiSize, const int nLevels[], int numChannels, int uniform, int* pSpecSize, int* pBufferSize);
```

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

dataType	Data type of the source image. Supported values are: ipp8u, ipp16u, ipp16s, and ipp32f.
roiSize	Size of the source image ROI, in pixels.
nLevels	Number of level values. Each channel has a separate number of levels.
numChannels	Number of image channels. Supported values are: 1, 3, and 4.
uniform	Type of levels distribution: 0 - with random step, 1 - with uniform step.
pSpecSize	Pointer to the computed value of the specification structure size, in bytes.
pBufferSize	Pointer to the computed value of the external buffer size.

Description

The ippiHistogramGetBufferSize function computes the size of the histogram specification structure and the size of the external work buffer (in bytes) needed for the Histogram function.

For an example on how to use this function, refer to the example provided with the Histogram function description.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when roiSize is less than, or equal to zero.
ippStsHistoNofLevelsErr	Indicates an error when the number of levels is less than 2.
ippStsNumChannelsErr	Indicates an error when the numChannels value differs from 1, 3, or 4.
ippStsDataTypeErr	Indicates an error when the dataType value differs from ipp8u, ipp16u, ipp16s, or ipp32f.

See Also

Histogram Computes the intensity histogram of an image.

HistogramGetLevels

Returns the array with level values stored in the specification structure.

Syntax

```
IppStatus ippiHistogramGetLevels(const IppiHistogramSpec* pSpec, Ipp32f* pLevels[]);
```

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSpec	Pointer to the specification structure.
pLevels	Pointer to the array of pointers to the level values vectors for each channel.

Description

The ippiHistogramGetLevels function returns the level values stored in the histogram specification structure.

For an example on how to use this function, refer to the example provided with the Histogram function description.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsBadArgErr	Indicates an error when the pSpec object is not initialized.

See Also

Histogram Computes the intensity histogram of an image.

HistogramInit, HistogramUniformInit

Initializes the specification structure for the ippiHistogram function.

Syntax

```
IppStatus ippiHistogramInit(IppDataType dataType, const Ipp32f* pLevels[], int nLevels[], int numChannels, IppiHistogramSpec* pSpec);
```

```
IppStatus ippiHistogramUniformInit(IppDataType dataType, Ipp32f lowerLevel[], Ipp32f upperLevel[], int nLevels[], int numChannels, IppiHistogramSpec* pSpec);
```

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

dataType	Data type of the source image. Supported values are: ipp8u, ipp16u, ipp16s, and ipp32f.
pLevels	Pointer to the array of pointers to the level values vectors for each channel.
lowerLevel	Lower levels for uniform histogram, separate for each channel.
upperLevel	Upper levels for uniform histogram, separate for each channel.
nLevels	Number of level values. Each channel has a separate number of levels.
numChannels	Number of image channels. Supported values are: 1, 3, and 4.
pSpec	Pointer to the specification structure.

Description

The ippiHistogramInit function initializes the specification structure for the Histogram function.

For an example on how to use these functions, refer to the example provided with the Histogram function description.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when roiSize is less than, or equal to zero.
ippStsHistoNofLevelsErr	Indicates an error when the number of levels is less than 2.
ippStsNumChannelsErr	Indicates an error when the numChannels value differs from 1, 3, or 4.
ippStsDataTypeErr	Indicates an error when the dataType value differs from ipp8u, ipp16u, ipp16s, or ipp32f.

See Also

Histogram Computes the intensity histogram of an image.

Histogram

Computes the intensity histogram of an image.

Syntax

Case 1: One-channel data

```
IppStatus ippiHistogram_<mod>(const Ipp<dataType>* pSrc, int srcStep, IppiSize roiSize,
Ipp32u* pHist, const IppiHistogramSpec* pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32f_C1R

Case 2: Three-channel data

```
IppStatus ippiHistogram_<mod>(const Ipp<dataType>* pSrc, int srcStep, IppiSize roiSize,
Ipp32u* pHist[3], const IppiHistogramSpec* pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u_C3R	16u_C3R	16s_C3R	32f_C3R

Case 3: Four-channel data

```
IppStatus ippiHistogram_<mod>(const Ipp<dataType>* pSrc, int srcStep, IppiSize roiSize,
Ipp32u* pHist[4], const IppiHistogramSpec* pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u_C4R	16u_C4R	16s_C4R	32f_C4R

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance, in bytes, between the starting points of consecutive lines in the source image.
roiSize	Size of the source image ROI, in pixels.
pHist	Pointer to the computed histogram. In case of multi-channel data, pHist is an array of pointers to the histogram for each channel.
pSpec	Pointer to the specification structure.
pBuffer	Pointer to the work buffer.

Description

The ippiHistogram function operates with ROI (see Regions of Interest in Intel IPP).

This function computes the intensity histogram for each channel of the source image and stores the result in the pHist array.

Before calling this function, initialize the specification structure using the HistogramInit or HistogramUniformInit functions. The specification structure defines the following parameters for histogram calculation:

- Histogram type: with uniform or random levels step
- Number of levels
- Level values

Length of the pHist array is defined by the nLevels parameter passed to the HistogramInit or HistogramUniformInit function.

As nLevels is the number of levels, the number of values in the pHist array, which is the number of histogram bins, is nLevels - 1. The meaning of the pHist and pLevels values can be illustrated by the following example: pHist[k] is the number of the source image pixels pSrc(x, y) that satisfy the condition pLevels[k] <= pSrc(x, y) < pLevels(k+1).

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when roiSize has a zero or negative value.
ippStsStepErr	Indicates an error when srcStep is less than roiSize.width * sizeOf(*pSrc) * nChannels.
ippStsBadArgErr	Indicates an error when the pSpec object is not initialized.

Example

The code example below demonstrates how to use the HistogramGetBufferSize, HistogramUniformInit, HistogramGetLevels, and Histogram functions.

```
void HistogramExample()
{
   const int HEIGHT = 8;
   const int WIDTH = 8;
   Ipp8u pImg[WIDTH*HEIGHT];
   IppiSize roi = {WIDTH, HEIGHT};
   int i;
   IppStatus sts;


   {  // fill image with random values in [0..255] range with uniform distribution.
      IppsRandUniState_8u* pRndObj;
      int sizeRndObj;


      // get spec size
      ippsRandUniformGetSize_8u( &sizeRndObj );
      pRndObj = (IppsRandUniState_8u*)ippsMalloc_8u( sizeRndObj );
      // initialize rnd spec
      ippsRandUniformInit_8u(pRndObj, 0/*low*/, 255/*high*/, 0/*seed*/ );


      // fill image
      for ( i=0; i<HEIGHT; i++ ) {
         sts = ippsRandUniform_8u(pImg + i*WIDTH,  WIDTH, pRndObj);
      }


      ippsFree( pRndObj );
   }


   printf_8u_2D("pImg:", pImg, roi, WIDTH, sts);


   {
      const int nBins = 5;
      int nLevels[] = { nBins+1 };
      Ipp32f lowerLevel[] = {0};
      Ipp32f upperLevel[] = {256};
```

```
      Ipp32f pLevels[nBins+1], *ppLevels[1];
      int sizeHistObj, sizeBuffer;


      IppiHistogramSpec* pHistObj;
      Ipp8u* pBuffer;
      Ipp32u pHistVec[nBins];


      // get sizes for spec and buffer
      ippiHistogramGetBufferSize(ipp8u, roi, nLevels, 1/*nChan*/, 1/*uniform*/, &sizeHistObj,
&sizeBuffer);


      pHistObj = (IppiHistogramSpec*)ippsMalloc_8u( sizeHistObj );
      pBuffer = (Ipp8u*)ippsMalloc_8u( sizeBuffer );
      // initialize spec
      ippiHistogramUniformInit( ipp8u, lowerLevel, upperLevel, nLevels, 1, pHistObj );


      // check levels of bins
      ppLevels[0] = pLevels;
      sts = ippiHistogramGetLevels( pHistObj, ppLevels );
      printf_32f( "pLevels:", pLevels, nBins+1, sts );


      // calculate histogram
      sts = ippiHistogram_8u_C1R( pImg, WIDTH, roi, pHistVec, pHistObj, pBuffer );


      ippsFree( pHistObj );
      ippsFree( pBuffer );


      printf_32u( "Histogram:", pHistVec, nBins, sts );
   }
}
```

Output:

```
pImg:
0 33 53 102 90 188 210 60
195 137 247 137 7 15 65 244
149 44 210 20 170 140 183 144
133 61 191 32 212 108 178 89
86 30 54 93 168 93 2 114
30 145 216 42 86 113 148 205
148 181 217 99 219 31 156 156
237 36 74 80 208 121 118 106


pLevels:
0.0 51.0 102.0 153.0 204.0 255.0


Histogram:
13 14 16 10 11
```

See Also

Regions of Interest in Intel IPP

Histogram Computes the intensity histogram of an image.

HistogramGetBufferSize Computes the size of the specification structure and work buffer for the ippiHistogram function.

HistogramGetLevels Returns the array with level values stored in the specification structure.

HistogramInit, HistogramUniformInit Initializes the specification structure for the ippiHistogram function.

CountInRange

Computes the number of pixels within the given intensity range.

Syntax

Case 1: Operation on one-channel data

```
IppStatus ippiCountInRange_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize
roiSize, int* counts, Ipp<datatype> lowerBound, Ipp<datatype> upperBound);
```

Supported values for `mod`:

`8u_C1R`	`32f_C1R`

Case 2: Operation on multi-channel data

```
IppStatus ippiCountInRange_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize
roiSize, int counts[3], Ipp<datatype> lowerBound[3], Ipp<datatype> upperBound[3]);
```

Supported values for `mod`:

`8u_C3R`	`32f_C3R`
`8u_AC4R`	`32f_AC4R`

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`roiSize`	Size of the source ROI in pixels.
`counts`	The computed number of pixels within the given intensity range. An array of 3 values in case of multi-channel data.

lowerBound	Lower limit of the intensity range.
upperBound	Upper limit of the intensity range.

Description

This function operates with ROI (see Regions of Interest in Intel IPP). This function computes the number of pixels in the image which have intensity values in the range between lowerBound and upperBound (inclusive).

In case of a multi-channel image, pixels are counted within intensity range for each color channel separately, and the array counts of three resulting values is returned. The alpha channel values, if present, are not processed.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if pSrc pointer is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if srcStep has a zero or negative value.
ippStsRangeErr	Indicates an error condition if lowerBound exceeds upperBound.

BlockMinMax

Finds minimum and maximum values for blocks of an image.

Syntax

```
IppStatus ippiBlockMinMax_<dataType>_C1R(const Ipp<dataType>* pSrc, int srcStep,
IppiSize srcSize, Ipp<dataType>* pDstMin, int dstMinStep, Ipp<dataType>* pDstMax, int
dstMaxStep, IppiSize blockSize, Ipp<dataType>* pGlobalMin, Ipp<dataType>* pGlobalMax);
```

Supported values for dataType:

8u	16u	16s	32f

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance, in bytes, between the starting points of consecutive lines in the source image.
srcSize	Size, in pixels, of the source image.
pDstMin	Pointer to the destination image to store minimum values per block.
dstMinStep	Distance, in bytes, between the starting points of consecutive lines in the pDstMin image.
pDstMax	Pointer to the destination image to store maximum values per block.
dstMaxStep	Distance, in bytes, between the starting points of consecutive lines in the pDstMax image.
blockSize	Size, in pixels, of the image block.
pGlobalMin	Destination pointer to the minimum value for the entire source image.
pGlobalMax	Destination pointer to the maximum value for the entire source image.

Description

This function operates with ROI.

This function finds minimum and maximum values for blocks of the source image, which are defined by the blockSize parameter. Minimum and maximum values for blocks are stored in the pDstMin and pDstMax images, respectively. Minimum and maximum values for the entire image are stored in the pGlobalMin and pGlobalMax pointers, respectively.

If pDstMin or pDstMax pointer is NULL, the corresponding component (minimum or maximum value) is not calculated.

The size of the pDstMin and pDstMax images is calculated by the following formulae:

- if srcWidth is divisible by blockWidth, the destination width is equal to:

  dstWidth = srcWidth/blockWidth

  otherwise:

  dstWidth = srcWidth/blockWidth + 1
- if srcHeight is divisible by blockHeight, the destination height is equal to:

  dstHeight = srcHeight/blockHeight

  otherwise:

  dstHeight = srcHeight/blockHeight + 1

Return Values

ippStsNoErr	Indicates no error.

ippStsNullPtrErr	Indicates an error when pSrc, pDstMin, and pDstMax pointers are NULL.
ippStsStepErr	Indicates an error when: • srcStep is less than srcSize.width * < pixelSize``>`` • dstMinStep or dstMaxStep is less than dstSize.width * < pixelSize``>``
ippStsSizeErr	Indicates an error when srcSize or blockSize has a zero or negative value.

See Also

Regions of Interest in Intel IPP

Min

Computes the minimum of image pixel values.

Syntax

Case 1: Operation on one-channel data

```
IppStatus ippiMin_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp<datatype>* pMin);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32f_C1R

Case 2: Operation on multi-channel data

```
IppStatus ippiMin_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp<datatype> min[3]);
```

Supported values for mod:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

```
IppStatus ippiMin_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp<datatype> min[4]);
```

Supported values for mod:

8u_C4R	16u_C4R	16s_C4R	32f_C4R

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
roiSize	Size of the source ROI in pixels.
pMin	Pointer to the minimum pixel value (for one-channel data).
min	Array containing minimum channel values of pixels in the source buffer (for multi-channel data).

Description

This function operates with ROI (see Regions of Interest in Intel IPP). This function computes the minimum pixel value pMin for the source image pSrc. In case of a multi-channel image, the minimum is computed over each channel and stored in the array min.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if pSrc or pMin pointer is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value.

Example

The code example below demonstrates how to use the function ippiMin.

```
Ipp8u src[4*1] = { 40, 20, 60, 80 };
IppiSize roiSize = { 4, 1 };
Ipp8u min;
ippiMin_8u_C1R ( src, 4, roiSize, &min );
```

Result:

```
min = 20
```

MinIndx

Computes the minimum of image pixel values and retrieves the x and y coordinates of pixels with minimal intensity values.

Syntax

Case 1: Operation on one-channel data

```
IppStatus ippiMinIndx_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp<datatype>* pMin, int* pIndexX, int* pIndexY);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32f_C1R

Case 2: Operation on multi-channel data

```
IppStatus ippiMinIndx_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp<datatype> min[3], int indexX[3], int indexY[3]);
```

Supported values for `mod`:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

```
IppStatus ippiMinIndx_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp<datatype> min[4], int indexX[4], int indexY[4]);
```

Supported values for `mod`:

8u_C4R	16u_C4R	16s_C4R	32f_C4R

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`roiSize`	Size of the source ROI in pixels.
`pMin`	Pointer to the minimum pixel value (for one-channel data).
`min`	Array containing minimum color channel values of pixels in the source buffer (for multi-channel data).
`pIndexX, pIndexY`	Pointers to the x and y coordinates of the pixel with minimum value.
`indexX, indexY`	Arrays containing the x and y coordinates of pixels with minimum channel values.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function computes the minimum pixel value `pMin` for the source image `pSrc`. In case of a multi-channel image, the minimum is computed over each channel and stored in the array `min`. The function also retrieves the `x` and `y` coordinates of pixels on which the minimum is reached. If several pixels have equal minimum

values, the coordinates of the first pixel from the start of the source buffer is returned. For multi-channel data, `indexX[k]` and `indexY[k]` are the `x` and `y` coordinates of the pixel that has the minimal intensity value of the `k`-th channel, `k = 1,2,3,4`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of `pSrc,pMin, pIndexX,` or `pIndexY` pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize` has a field with zero or negative value.

Max

Computes the maximum of image pixel values.

Syntax

Case 1: Operation on one-channel data

```
IppStatus ippiMax_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp<datatype>* pMax);
```

Supported values for `mod`:

`8u_C1R`	`16u_C1R`	`16s_C1R`	`32f_C1R`

Case 2: Operation on multi-channel data

```
IppStatus ippiMax_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp<datatype> max[3]);
```

Supported values for `mod`:

`8u_C3R`	`16u_C3R`	`16s_C3R`	`32f_C3R`
`8u_AC4R`	`16u_AC4R`	`16s_AC4R`	`32f_AC4R`

```
IppStatus ippiMax_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp<datatype> max[4]);
```

Supported values for `mod`:

`8u_C4R`	`16u_C4R`	`16s_C4R`	`32f_C4R`

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
roiSize	Size of the source ROI in pixels
pMax	Pointer to the maximum pixel value (for one-channel data).
max	Array containing maximum channel values of pixels in the source buffer (for multi-channel data).

Description

This function operates with ROI (see Regions of Interest in Intel IPP). This function computes the maximum pixel value pMax for the source image pSrc. In case of a multi-channel image, the maximum is computed over each channel and stored in the array max.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if pSrc or pMax pointer is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value.

MaxIndx

Computes the maximum of image pixel values and retrieves the x and y coordinates of pixels with maximal intensity values.

Syntax

Case 1: Operation on one-channel data

```
IppStatus ippiMaxIndx_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp<datatype>* pMax, int* pIndexX, int* pIndexY);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32f_C1R

Case 2: Operation on multi-channel data

```
IppStatus ippiMaxIndx_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp<datatype> max[3], int indexX[3], int indexY[3]);
```

Supported values for mod:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

```
IppStatus ippiMaxIndx_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp<datatype> max[4], int indexX[4], int indexY[4]);
```

Supported values for `mod`:

`8u_C4R`	`16s_C4R`	`16u_C4R`	`32f_C4R`

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`roiSize`	Size of the source ROI in pixels.
`pMax`	Pointer to the maximum pixel value (for one-channel data).
`max`	Array containing maximum channel values of pixels in the source buffer (for multi-channel data).
`pIndexX, pIndexY`	Pointers to the x and y coordinates of the pixel with maximum value.
`indexX, indexY`	Arrays containing the x and y coordinates of pixels with maximum channel values.

Description

This function operates with ROI (see Regions of Interest in Intel IPP). This function computes the maximum pixel value `pMax` for the source image `pSrc`. In case of a multi-channel image, the maximum is computed over each channel and stored in the array `max`. The function also retrieves the `x and y` coordinates of pixels on which the maximum is reached. If several pixels have equal maximum values, the coordinates of the first pixel from the start of the source buffer is returned. For multi-channel data, `indexX[k]andindexY[k]` are the `x and y` coordinates of the pixel that has the maximal intensity value of the `k-th channel`, k = 1,2,3,4.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if any of `pSrc`,`pMax`, `pIndexX`, or `pIndexY` pointers is `NULL`.

ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value.

Example

The code example below demonstrates how to use the function ippiMaxIndx.

```
Ipp8u src[4*1] = { 40, 20, 60, 80 };
IppiSize roiSize = { 4, 1 };
Ipp8u max;
int IndexX;
int IndexY;
ippiMaxIndx_8u_C1R ( src, 4, roiSize, &max, &IndexX, &IndexY );
```

Output:

```
max = 80  IndexX = 3 IndexY = 0
```

MinMax

Computes the minimum and maximum of image pixel values.

Syntax

Case 1: Operation on one-channel data

```
IppStatus ippiMinMax_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp<datatype>* pMin, Ipp<datatype>* pMax);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32f_C1R

Case 2: Operation on multi-channel data

```
IppStatus ippiMinMax_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp<datatype> min[3], Ipp<datatype> max[3]);
```

Supported values for mod:

8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_AC4R	16u_AC4R	16s_AC4R	32f_AC4R

```
IppStatus ippiMinMax_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp<datatype> min[4], Ipp<datatype> max[4]);
```

Supported values for mod:

8u_C4R	16u_C4R	16s_C4R	32f_C4R

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
roiSize	Size of the source ROI in pixels.
pMin, pMax	Pointers to the minimum and maximum pixel values (for one-channel data).
min, max	Arrays containing minimum and maximum channel values of pixels in the source buffer (for multi-channel data).

Description

This function operates with ROI (see Regions of Interest in Intel IPP). This function computes the minimum and maximum pixel values pMin and pMax for the source image pSrc. In case of a multi-channel image, the minimum and maximum is computed over each channel and stored in the arrays min and max.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if pSrc, pMin, or pMax pointer is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value.

MinMaxIndx

Calculates minimum and maximum pixel values and their indexes in selected image rectangle.

Syntax

Case 1: Operation on one-channel data

```
IppStatus ippiMinMaxIndx_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize
roiSize, Ipp32f* pMinVal, Ipp32f* pMaxVal, IppiPoint* pMinIndex, IppiPoint* pMaxIndex);
```

Supported values for mod:

8u_C1R	16u_C1R	32f_C1R

Case 2: Masked operation on one-channel data

```
IppStatus ippiMinMaxIndx_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp8u*
pMask, int maskStep, IppiSize roiSize, Ipp32f* pMinVal, Ipp32f* pMaxVal, IppiPoint*
pMinIndex, IppiPoint* pMaxIndex);
```

Supported values for mod:

8u_C1MR	16u_C1MR	32f_C1MR

Case 3: Operation on multi-channel data

```
IppStatus ippiMinMaxIndx_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize
roiSize, int coi, Ipp32f* pMinVal, Ipp32f* pMaxVal, IppiPoint* pMinIndex, IppiPoint*
pMaxIndex);
```

Supported values for `mod`:

`8u_C3CR`	`16u_C3CR`	`32f_C3CR`

Case 4: Masked operation on multi-channel data

```
IppStatus ippiMinMaxIndx_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp8u*
pMask, int maskStep, IppiSize roiSize, int coi, Ipp32f* pMinVal, Ipp32f* pMaxVal,
IppiPoint* pMinIndex, IppiPoint* pMaxIndex);
```

Supported values for `mod`:

`8u_C3CMR`	`16u_C3CMR`	`32f_C3CMR`

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`pMask`	Pointer to the mask image.
`maskStep`	Distance in bytes between starts of consecutive lines in the mask image.
`roiSize`	Size of the image ROI in pixels.
`coi`	Channel of interest (for color images only); can be 1, 2, or 3.
`pMinVal`	Pointer to the variable that returns the value of the minimum pixel.
`pMaxVal`	Pointer to the variable that returns the value of the maximum pixel.
`pMinIndex`	Pointer to the variable that returns the index of the minimum value found.
`pMaxIndex`	Pointer to the variable that returns the index of the maximum value found.

Description

This function operates with ROI (see Regions of Interest in Intel IPP). This function finds minimum and maximum pixel values and their indexes in an image ROI or in an arbitrary image region defined by nonzero mask values. If there are several minima and maxima in the selected area, the function returns the top leftmost positions. If the specified region in the mask mode is empty, that is, the mask image is filled with zeros, then the function returns {`minIndex`, `maxIndex`} = {0, 0}, `minVal=maxVal=` 0. If any of the parameters `pMinVal`, `pMaxVal`, `pMinIndex`,or `pMaxIndex` is not required, the zero pointer is to be passed to the corresponding parameter.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if `pSrc` or `pMask` pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize` has a field with zero or negative value.
`ippStsStepErr`	Indicates an error for masked operations when `srcStep` or `maskStep` is less than `roiSize.width * <pixelSize>`.
`ippStsNotEvenStepErr`	Indicates an error when steps for floating-point images cannot be divided by 4.
`ippStsCOIErr`	Indicates an error when `coi` is not 1, 2, or 3.

MaxEvery

Computes maximum value for each pair of pixels of two images.

Syntax

Case 1: Not-in-place operation

```
IppStatus ippiMaxEvery_8u_C1R(const Ipp8u* pSrc1, int src1Step, const Ipp8u* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiMaxEvery_16u_C1R(const Ipp16u* pSrc1, int src1Step, const Ipp16u* pSrc2, int src2Step, Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiMaxEvery_32f_C1R(const Ipp32f* pSrc1, int src1Step, const Ipp32f* pSrc2, int src2Step, Ipp32f* pDst, int dstStep, IppiSize roiSize);
```

Case 2: In-place operation

```
IppStatus ippiMaxEvery_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for `mod`:

`8u_C1IR`	`16u_C1IR`	`16s_C1IR`	`32f_C1IR`
`8u_C3IR`	`16u_C3IR`	`16s_C3IR`	`32f_C3IR`

8u_C4IR	16u_C4IR	16s_C4IR	32f_C4IR
8u_AC4IR	16u_AC4IR	16s_AC4IR	32f_AC4IR

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc1, pSrc2	Pointer to the first and second source image, respectively (for not-in-place operation).
src1Step, src2Step	Distance, in bytes, between the starting points of consecutive lines in the first and second source image, respectively (for not-in-place operation).
pDst	Pointer to the destination image (for not-in-place operation).
pSrc	Pointer to the first source image ROI (for in-place operation).
srcStep	Distance, in bytes, between the starting points of consecutive lines in the first source image (for in-place operation).
dstStep	Distance, in bytes, between the starting points of consecutive lines in the destination image.
pSrcDst	Pointer to the second source and destination image ROI (for in-place operation).
srcDstStep	Distance, in bytes, between the starting points of consecutive lines in the second source and destination image (for in-place operation).
roiSize	Size of the image ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

Not-in-place operation:

This function computes the maximum value for each pair of the corresponding pixels of two source images (pSrc1 and pSrc2 for not-in-place operation or pSrc and pSrcDst for in-place), and stores the result in pDst.

In-place operation:

This function computes the maximum value for each pair of the corresponding pixels of two source images pSrc and pSrcDst, and stores the result in pSrcDst:

pSrcDst(i, j) = max(pSrc(i, j), pSrcDst(i, j)).

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if srcStep, src1Step, src2Step or srcDstStep is less than roiSize.width*<pixelSize>.
ippStsNotEvenStepErr	Indicates an error condition if one of step values for floating-point images are not divisible by 4.

MinEvery

Computes minimum value for each pair of pixels of two images.

Syntax

Case 1: Not-in-place operation

```
IppStatus ippiMinEvery_8u_C1R(const Ipp8u* pSrc1, int src1Step, const Ipp8u* pSrc2, int src2Step, Ipp8u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiMinEvery_16u_C1R(const Ipp16u* pSrc1, int src1Step, const Ipp16u* pSrc2, int src2Step, Ipp16u* pDst, int dstStep, IppiSize roiSize);
```

```
IppStatus ippiMinEvery_32f_C1R(const Ipp32f* pSrc1, int src1Step, const Ipp32f* pSrc2, int src2Step, Ipp32f* pDst, int dstStep, IppiSize roiSize);
```

Case 2: In-place operation

```
IppStatus ippiMinEvery_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize);
```

Supported values for mod:

8u_C1IR	16u_C1IR	16s_C1IR	32f_C1IR
8u_C3IR	16u_C3IR	16s_C3IR	32f_C3IR
8u_C4IR	16u_C4IR	16s_C4IR	32f_C4IR
8u_AC4IR	16u_AC4IR	16s_AC4IR	32f_AC4IR

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc1, pSrc2	Pointer to the first and second source image, respectively (for not-in-place operation).
src1Step, src2Step	Distance, in bytes, between the starting points of consecutive lines in the first and second source image, respectively (for not-in-place operation).
pDst	Pointer to the destination image (for not-in-place operation).
pSrc	Pointer to the first source image ROI (for in-place operation).
srcStep	Distance, in bytes, between the starting points of consecutive lines in the first source image (for in-place operation).
pSrcDst	Pointer to the second source and destination image ROI (for in-place operation).
dstStep	Distance, in bytes, between the starting points of consecutive lines in the destination image.
srcDstStep	Distance, in bytes, between the starting points of consecutive lines in the second source and destination image (for in-place operation).
roiSize	Size of the image ROI in pixels.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

Not-in-place operation:

This function computes the minimum value for each pair of the corresponding pixels of two source images (pSrc1 and pSrc2 for not-in-place operation or pSrc and pSrcDst for in-place), and stores the result in pDst.

In-place operation:

This function computes the minimum value for each pair of the corresponding pixels of two source images pSrc and pSrcDst, and stores the result in pSrcDst:

pSrcDst(i, j) = min(pSrc(i, j), pSrcDst(i, j)).

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value.

ippStsStepErr	Indicates an error condition if srcStep, src1Step, src2Step or srcDstStep is less than roiSize.width*<pixelSize>.
ippStsNotEvenStepErr	Indicates an error condition if one of step values for floating-point images are not divisible by 4.

FindPeaks3x3GetBufferSize

Computes the size of the working buffer for the peak search.

Syntax

```
IppStatus ippiFindPeaks3x3GetBufferSize_32s_C1R(int roiWidth, int* pBufferSize);
IppStatus ippiFindPeaks3x3GetBufferSize_32f_C1R(int roiWidth, int* pBufferSize);
```

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

roiWidth	Maximum width of the image, in pixels.
pBufferSize	Pointer to the size of the working buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function computes the size of the working buffer required for the function ippiFindPeaks3x3. The buffer with the length pBufferSize[0] can be used to filter images with width that is less than or equal to roiWidth.

Example "Simplified Peak Search for Calculation of SIFT Features" shows how to use the function ippiFindPeaks3x3GetBufferSize_32f_C1R.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if the pointer pBufferSize is NULL.
ippStsSizeErr	Indicates an error condition if roiWidth is less than 1.

FindPeaks3x3

Finds coordinates of peaks (maximums or minimums) with absolute value exceeding threshold value.

Syntax

```
IppStatus ippiFindPeaks3x3_32s_C1R(const Ipp32s* pSrc, int srcStep, IppiSize roiSize,
Ipp32s threshold, IppiPoint* pPeak, int maxPeakCount, int* pPeakCount, IppiNorm norm,
int border, Ipp8u* pBuffer);
```

```
IppStatus ippiFindPeaks3x3_32f_C1R(const Ipp32f* pSrc, int srcStep, IppiSize roiSize,
Ipp32f threshold, IppiPoint* pPeak, int maxPeakCount, int* pPeakCount, IppiNorm norm,
int border, Ipp8u* pBuffer);
```

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the first source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the first source image.
`roiSize`	Size of the image ROI in pixels.
`threshold`	Threshold value.
`pPeak`	Pointer to the coorditanes peaks [`maxPeakCount`].
`maxPeakCount`	Maximum number of peaks.
`pPeakCount`	Pointer to the number of the detected peaks.
`border`	Border value, only pixel with distance from the edge of the image greater than `border` are processed.
`norm`	Specifies type of the norm to form the mask for extremum search: • ippiNormInf • Infinity norm (8-connectivity, 3x3 rectangular mask); • ippiNormL1 • L1 norm (4-connectivity, 3x3 cross mask).
`pBuffer`	Pointer to the working buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function detects local maximum and minimum pixels in the source image:

$$pSrc(i_m, j_m) = \max_{(k, l) \in O(i_m, j_m)} pSrc(k, l), \quad pSrc(i_m, j_m) \geq thresho$$

$$pSrc(i_m, j_m) = \min_{(k, l) \in O(i_m, j_m)} pSrc(k, l), \quad |pSrc(i_m, j_m)| \geq thresho$$

and stores their coordinates in the `pPeak` array `pPeak[m].x = jm, pPeak[m].y = im, m = 0, ...pPeakCount[0], pPeakCount[0] ≤ maxPeakCount`

The neighborhood `O(i, j)` for the extremum search is defined by the parameter `norm`. The number of detected extremums is returned in `pPeakCount[0]`. The operation is stopped when the `maxPeakCount` extremums are found.

The function requires the working buffer `pBuffer` whose size should be computed by the function ippiFindPeaks3x3GetBufferSize beforehand.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize` has a field with zero or negative value; or if `maxPeakCount` is less than or equal to 0; or if `border` is less than 1 or greater than one of 0.5* `roiSize.width` or of 0.5* `roiSize.height`.
`ippStsStepErr`	Indicates an error condition if `srcStep` is less than `roiSize.width*<pixelSize>`.
`ippStsNotEvenStepErr`	Indicates an error condition if `srcStep` is not divisible by 4.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
```

```
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *****************************************************************************/

// A simple example of finding coordinates of peaks
//     ippiFindPeaks3x3GetBufferSize_32f_C1R
//     ippiFindPeaks3x3_32f_C1R

#include <stdio.h>
#include "ipp.h"

#define WIDTH   128 /* Image width  */
#define HEIGHT  128 /* Image height */
#define KERSIZE 3   /* Kernel size  */

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define
check_sts(st)
                                  \
    if ((st) !=
ippStsNoErr)
                          \
        goto exitLine; /* Go to Exit if Intel(R) Integrated Performance Primitives (Intel(R)
IPP) function returned status different from \
                           ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main(void)
{
    IppStatus status = ippStsNoErr;
    Ipp32f *pSrc = NULL;                /*  initial image  */
    Ipp8u *pBuffer = NULL;              /*  working buffer */
    IppiPoint *pPeak = NULL;            /*  array of peaks */
    IppiSize roiSize = {WIDTH, HEIGHT}; /* image size */
    int srcStep = 0;                    /*  row step in bytes */
    int bufSize = 0;                    /*  working buffer size */
    int maxPeaks = 500;                 /*  max peaks number for image */
    int peakCount = 0;                  /*  peaks number for image */
    Ipp32f threshold = 0.02f;           /*  peak threshold */
    Ipp32s borderValue = 40;            /*  Border Value */

    pSrc = ippiMalloc_32f_C1(roiSize.width, roiSize.height, &srcStep);

    check_sts(status = ippiFindPeaks3x3GetBufferSize_32f_C1R(roiSize.width, &bufSize));

    pBuffer = ippsMalloc_8u(bufSize);
    pPeak = (IppiPoint *)ippsMalloc_8u(maxPeaks * sizeof(IppiPoint));

    check_sts(status = ippiFindPeaks3x3_32f_C1R(pSrc, srcStep, roiSize, threshold, pPeak,
```

```
maxPeaks, &peakCount, ippiNormL1, borderValue, pBuffer));

    EXIT_MAIN
    ippsFree(pBuffer);
    ippsFree(pPeak);
    ippiFree(pSrc);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return status;
}
```

Image Moments

Spatial and central moments are important statistical properties of an image. The spatial moment M_U (m, n) of order (m, n) is defined as follows:

$$M_U(m,\ n) = \sum_{j}\sum_{k} x_k^m y_j^n P_{j,\ k}$$

where the summation is performed for all rows and columns in the image; $P_{j,k}$ are pixel values; x_k and y_j are pixel coordinates; m and n are integer power exponents that define the moment order.

The central moment U_U (m, n) is the spatial moment computed relative to the "center of gravity" (x_0, y_0):

$$U_U(m,\ n) = \sum_{j}\sum_{k} (x_k - x_0)^m (y_j - y_0)^n P_{j,\ k}$$

where $x_0 = M_U(1,0)/ M_U(0,0)$ and $y_0 = M_U(0,1)/ M_U(0,0)$.

The normalized spatial moment M(m, n) and central moment U(m, n) are defined as follows:

$$M(m,\ n) = \frac{M_U(m,\ n)}{M_U(0,\ 0)^{\frac{m+n+2}{2}}}$$

$$U(m,\ n) = \frac{U_U(m,\ n)}{U_U(0,\ 0)^{\frac{m+n+2}{2}}}$$

The Intel IPP functions support moments of order (m, n) with $0 \le m + n \le 3$. The computation of seven invariant Hu moments derived from the second and third order moments is also supported. All computed moments are stored in context structures of type IppiMomentState_64s (for integer versions)or IppiMomentState_64f (for floating point versions).

Most Intel IPP functions for computing image moments have code branches that implement different algorithms to compute the results. You can choose the desired code variety to be used by the given function by setting the `hint` argument to one of the following values that are listed in Table Hint Arguments for Image Moment Functions:

Hint Arguments for Image Moment Functions

Value	Description
`ippAlgHintNone`	The computation algorithm will be chosen by the internal function logic.
`ippAlgHintFast`	Fast algorithm must be used. The output results will be less accurate.
`ippAlgHintAccurate`	High accuracy algorithm must be used. The function will need more time to execute.

- MomentGetStateSize
- MomentInit
- Moments
- GetSpatialMoment
- GetCentralMoment
- GetNormalizedSpatialMoment
- GetNormalizedCentralMoment
- GetHuMoments

MomentGetStateSize

Computes the size of the external buffer for the moment context structure.

Syntax

```
IppStatus ippiMomentGetStateSize_64f(IppHintAlgorithm hint, int* pSize);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSize`	Pointer to the computed value of the buffer size.
`hint`	Option to select the algorithmic implementation of the function.

Description

Use this function to determine the size of the external work buffer for the moment context structure to be initialized by the function ippiMomentInit. Computation algorithm is specified by `hint` argument (see Table "Hint Arguments for Image Moment Functions").

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if pSize pointer is NULL.

MomentInit

Initializes the moment context structure.

Syntax

```
IppStatus ippiMomentInit_64f(IppiMomentState_64f* pState, IppHintAlgorithm hint);
```

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pState	Pointer to the structure for storing moment values.
hint	Option to select the algorithmic implementation of the function.

Description

This function initializes the structure that is needed for the function ippiMoments to store the computed image moments. Computation algorithm is specified by hint argument (see Table "Hint Arguments for Image Moment Functions").

The structure is allocated in the external buffer. The size of this buffer can be computed by the function ippiMomentGetStateSize.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if pState pointer is NULL.

Moments

Computes all image moments of order 0 to 3 and Hu moment invariants.

Syntax

Case 1: Computation of floating-point results

```
IppStatus ippiMoments64f_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize
roiSize, IppiMomentState_64f* pCtx);
```

Supported values for `mod`:

`8u_C1R`	`16u_C1R`	`32f_C1R`
`8u_C3R`	`16u_C3R`	`32f_C3R`
`8u_AC4R`	`16u_AC4R`	`32f_AC4R`

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`roiSize`	Size of the source ROI in pixels.
`pCtx`	Pointer to the structure that stores image moments.

Description

This function operates with ROI (see Regions of Interest in Intel IPP). This function computes all spatial and central moments of order 0 to 3 for the source image `pSrc`. The seven Hu moment invariants are also computed. Different functions, `ippiMoments64s` and `ippiMoments64f`, are used to compute image moments in integer and floating-point formats, respectively.

The `ippiMoments` function computes spatial moment values relative to the image point referred to by `pSrc`. Note that this point is the ROI origin and may not coincide with the entire image origin. If you need to obtain spatial moment values relative to the actual image origin, use ippiGetSpatialMoment functions to recalculate them.

The moments' values are stored in the `pCtx` structure. To retrieve a particular moment value, use one of the functions described in the sections that follow.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if `pSrc` or `pCtx` pointer is `NULL`.

ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if srcStep has a zero or negative value.
ippStsContextMatchErr	Indicates an error condition if a pointer to an invalid structure is passed.

GetSpatialMoment

Retrieves image spatial moment of the specified order, computed by ippiMoments.

Syntax

```
IppStatus ippiGetSpatialMoment_64f(const IppiMomentState_64f* pState, int mOrd, int nOrd, int nChannel, IppiPoint roiOffset, Ipp64f* pValue);
```

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pState	Pointer to the structure that stores image moments.
mOrd, nOrd	Integer power exponents defining the moment order. These arguments must satisfy the condition 0≤ mOrd + nOrd ≤ 3.
nChannel	The channel for which the moment is returned.
roiOffset	Offset in pixels of the ROI origin (top left corner) from the image origin.
pValue	Pointer to the retrieved moment value.

Description

This function returns the pointer pValue to the spatial moment that was previously computed by the ippiMoments function. All spatial moment values are computed by ippiMoments relative to the image ROI origin. You may also obtain spatial moment values relative to different point in the image, using the appropriate roiOffset settings.

The moment order is specified by the integer exponents mOrd, nOrd.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if pState or pValue pointer is NULL.
ippStsContextMatchErr	Indicates an error condition if a pointer to an invalid structure is passed.
ippStsSizeErr	Indicates an error condition if mOrd + nOrd is greater than 3, or nChannel has an illegal value.

GetCentralMoment

Retrieves image central moment computed by ippiMoments.

Syntax

```
IppStatus ippiGetCentralMoment_64f(const IppiMomentState_64f* pState, int mOrd, int nOrd, int nChannel, Ipp64f* pValue);
```

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pState	The structure that stores image moments.
mOrd, nOrd	Integer power exponents defining the moment order. These arguments must satisfy the condition $0 \le mOrd + nOrd \le 3$.
nChannel	The channel for which the moment is returned.
pValue	Pointer to the returned moment value.

Description

This function returns the pointer pValue to the central moment previously computed by the ippiMoments function. The moment order is specified by the integer exponents mOrd, nOrd.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr	Indicates an error condition if pState or pValue pointer is NULL.
ippStsContextMatchErr	Indicates an error condition if a pointer to an invalid structure is passed.
ippStsSizeErr	Indicates an error condition if mOrd + nOrd is greater than 3, or nChannel has an illegal value.

GetNormalizedSpatialMoment

Retrieves the normalized value of the image spatial moment computed by ippiMoments.

Syntax

```
IppStatus ippiGetNormalizedSpatialMoment_64f(const IppiMomentState_64f* pState, int mOrd, int nOrd, int nChannel, IppiPoint roiOffset, Ipp64f* pValue);
```

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pState	The structure that stores image moments.
mOrd, nOrd	Integer power exponents defining the moment order. These arguments must satisfy the condition $0 \le mOrd + nOrd \le 3$.
nChannel	The channel for which the moment is returned.
roiOffset	Offset in pixels of the ROI origin (top left corner) from the image origin.
pValue	Pointer to the returned normalized moment value.

Description

This function normalizes the spatial moment value that was previously computed by the ippiMoments function, and returns the pointer pValue to the normalized moment. See Image Moments for details of moments normalization. The moment order (mOrd, nOrd) is specified by integer power exponents. All spatial moment values are computed by ippiMoments relative to the image ROI origin. You may also obtain normalized spatial moment values relative to different point in the image, using the appropriate roiOffset settings.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if pState or pValue pointer is NULL.
ippStsContextMatchErr	Indicates an error condition if a pointer to an invalid structure is passed.
ippStsMoment00ZeroErr	Indicates an error condition if M(0,0) value is close to zero.
ippStsSizeErr	Indicates an error condition if mOrd + nOrd is greater than 3, or nChannel has an illegal value.

GetNormalizedCentralMoment

Retrieves the normalized value of the image central moment computed by ippiMoments.

Syntax

```
IppStatus ippiGetNormalizedCentralMoment_64f(const IppiMomentState_64f* pState, int mOrd, int nOrd, int nChannel, Ipp64f* pValue);
```

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pState	The structure that stores image moments.
mOrd, nOrd	Integer power exponents defining the moment order. These arguments must satisfy the condition $0 \le mOrd + nOrd \le 3$.
nChannel	The channel for which the moment is returned.
pValue	Pointer to the returned moment value.

Description

This function normalizes the central moment value that was previously computed by the ippiMoments function, and returns the pointer pValue to the normalized moment. The moment order (mOrd, nOrd) is specified by the integer power exponents. See Image Moments for details of moments normalization.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if pState or pValue pointer is NULL.
ippStsContextMatchErr	Indicates an error condition if a pointer to an invalid structure is passed.
ippStsMoment00ZeroErr	Indicates an error condition if M(0,0) value is close to zero.

GetHuMoments

Retrieves image Hu moment invariants computed by ippiMoments function.

Syntax

```
IppStatus ippiGetHuMoments_64f(const IppiMomentState_64f* pState, int nChannel,
IppiHuMoment_64f pHm);
```

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pState	Pointer to the structure that stores image moments.
nChannel	The channel for which the moment is returned.
pHm	Pointer to the array containing the Hu moment invariants.

Description

This function returns the pointer pHm to the array of seven Hu moment invariants previously computed by the ippiMoments function.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if pState or pHm pointer is NULL.

ippStsContextMatchErr	Indicates an error condition if a pointer to an invalid structure is passed.
ippStsMoment00ZeroErr	Indicates an error condition if M(0,0) value is close to zero.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

//    A simple example of computing Hu moment values for images.
// implemented with Intel(R) Integrated Performance Primitives (Intel(R) IPP) functions:
//     ippiSet_8u_C1R
//     ippiMomentGetStateSize_64f
//     ippiMomentInit_64f
//     ippiMoments64f_8u_C1R
//     ippiGetHuMoments_64f

#include <stdio.h>
#include "ipp.h"

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define check_sts(st)        \
    if ((st) != ippStsNoErr) \
        goto exitLine; /* Go to Exit if Intel(R) IPP function returned status different from
ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main(void)
{
    IppStatus status = ippStsNoErr;
    Ipp8u pSrc[64];                                     /* Pointer to source image */
    IppiSize roiA = {8, 8}, roiH = {8, 1}, roiV = {1, 8}; /* Used image ROI sizes */
    int sizeState = 0;                                  /* State size */
    IppiHuMoment_64f hmH = {0}, hmV = {0};              /* Horizontal and vertical moments */
    IppiMomentState_64f *pState = NULL;                 /* Pointers to state structure */

    check_sts(status = ippiMomentGetStateSize_64f(ippAlgHintNone, &sizeState));

    /* memory allocation */
```

```
    pState = (IppiMomentState_64f *)ippMalloc(sizeState);

    check_sts(status = ippiMomentInit_64f(pState, ippAlgHintNone));

    /* set horizontal edge */
    check_sts(status = ippiSet_8u_C1R(0, pSrc, 8, roiA));
    check_sts(status = ippiSet_8u_C1R(3, pSrc, 8, roiH));

    /* compute spatial moment values */
    check_sts(status = ippiMoments64f_8u_C1R(pSrc, 8, roiA, pState));

    /* compute seven Hu moment invariants */
    check_sts(status = ippiGetHuMoments_64f(pState, 0, hmH));

    /* set vertical edge */
    check_sts(status = ippiSet_8u_C1R(0, pSrc, 8, roiA));
    check_sts(status = ippiSet_8u_C1R(3, pSrc, 8, roiV));

    /* compute spatial moment values */
    check_sts(status = ippiMoments64f_8u_C1R(pSrc, 8, roiA, pState));

    /* compute seven Hu moment invariants */
    check_sts(status = ippiGetHuMoments_64f(pState, 0, hmV));

    EXIT_MAIN
    ippFree(pState);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

Image Norms

The functions described in this section compute the following norms of the image pixel values:

- Infinity norm (the largest absolute pixel value)
- L1 norm (the sum of absolute pixel values)
- L2 norm (the square root of the sum of squared pixel values).

Functions of this group also help you compute the norm of differences in pixel values of two input images as well as the relative error for two input images.

- Norm_Inf
- Norm_L1
- Norm_L2
- NormDiff_Inf
- NormDiff_L1
- NormDiff_L2
- NormRel_Inf
- NormRel_L1
- NormRel_L2

Norm_Inf

Computes the infinity norm of image pixel values.

Syntax

Case 1: Operation on one-channel data

```
IppStatus ippiNorm_Inf_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp64f* pValue);
```

Supported values for `mod`:

`8u_C1R`	`16u_C1R`	`16s_C1R`	`32f_C1R`

Case 2: Masked operation on one-channel data

```
IppStatus ippiNorm_Inf_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp8u*
pMask, int maskStep, IppiSize roiSize, Ipp64f* pNorm);
```

Supported values for `mod`:

`8u_C1MR`	`16u_C1MR`	`32f_C1MR`

Case 3: Operation on multi-channel data

```
IppStatus ippiNorm_Inf_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp64f value[3]);
```

Supported values for `mod`:

`8u_C3R`	`16u_C3R`	`16s_C3R`	`32f_C3R`

```
IppStatus ippiNorm_Inf_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp64f value[4]);
```

Supported values for `mod`:

`8u_C4R`	`16u_C4R`	`16s_C4R`	`32f_C4R`

Case 4: Masked operation on multi-channel data

```
IppStatus ippiNorm_Inf_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp8u*
pMask, int maskStep, IppiSize roiSize, int coi, Ipp64f* pNorm);
```

Supported values for `mod`:

`8u_C3CMR`	`16u_C3CMR`	`32f_C3CMR`

Include Files

`ippcv.h`

`ippi.h`

Domain Dependencies

Flavors declared in `ippcv.h`:

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Flavors declared in `ippi.h`:

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pMask	Pointer to the mask image.
maskStep	Distance in bytes between starts of consecutive lines in the mask image.
roiSize	Size of the source ROI in pixels.
coi	Channel of interest (for color images only); can be 1, 2, or 3.
pValue	Pointer to the computed infinity norm of pixel values.
value	An array containing the computed infinity norms of channel values in case of multi-channel data.
pNorm	Pointer to the computed norm value in the mask mode.

Description

The flavors of the function ippiNorm_Inf that perform masked operation are declared in the ippcv.h file. All other function flavors are declared in the ippi.h file. This function operates with ROI (see Regions of Interest in Intel IPP) and computes the infinity norm pValue(pNorm``for the mask mode) for the source image ``pSrc. This norm is defined as the largest absolute pixel value in an image. In the mask mode, the computation is done over pixels selected by non-zero mask values.

For non-masked operations on a multi-channel image (Case 3), the norm is computed separately for each channel and stored in the array value .

In the mask multi-channel mode (Case 4), the norm is computed for a single channel of interest specified by coi.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value.
ippStsStepErr	Indicates an error condition in mask mode, if srcStep or maskStep is less than roiSize.width * <pixelSize>.
ippStsNotEvenStepErr	Indicates an error condition in mask mode if steps for floating-point images cannot be divided by 4.

ippStsCOIErr	Indicates an error when coi is not 1, 2, or 3.

Norm_L1

Computes the L1- norm of image pixel values.

Syntax

Case 1: Operation on one-channel integer data

```
IppStatus ippiNorm_L1_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp64f* pValue);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R

Case 2: Operation on one-channel floating-point data

```
IppStatus ippiNorm_L1_32f_C1R(const Ipp32f* pSrc, int srcStep, IppiSize roiSize,
Ipp64f* pValue, IppHintAlgorithm hint);
```

Case 3: Masked operation on one-channel data

```
IppStatus ippiNorm_L1_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp8u* pMask,
int maskStep, IppiSize roiSize, Ipp64f* pNorm);
```

Supported values for mod:

8u_C1MR	16u_C1MR	32f_C1MR

Case 4: Operation on multi-channel integer data

```
IppStatus ippiNorm_L1_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp64f value[3]);
```

Supported values for mod:

8u_C3R	16u_C3R	16s_C3R

```
IppStatus ippiNorm_L1_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp64f value[4]);
```

Supported values for mod:

8u_C4R	16u_C4R	16s_C4R

Case 5: Operation on multi-channel floating-point data

```
IppStatus ippiNorm_L1_32f_C3R(const Ipp32f* pSrc, int srcStep, IppiSize roiSize, Ipp64f
value[3], IppHintAlgorithm hint);
```

Supported values for mod:

```
IppStatus ippiNorm_L1_32f_C4R(const Ipp32f* pSrc, int srcStep, IppiSize roiSize, Ipp64f
value[4], IppHintAlgorithm hint);
```

Case 6: Masked operation on multi-channel data

```
IppStatus ippiNorm_L1_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp8u* pMask,
int maskStep, IppiSize roiSize, int coi, Ipp64f* pNorm);
```

Supported values for mod:

8u_C3CMR	16u_C3CMR	32f_C3CMR

Include Files

ippcv.h

ippi.h

Domain Dependencies

Flavors declared in ippcv.h:

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Flavors declared in ippi.h:

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pMask	Pointer to the mask image.
maskStep	Distance in bytes between starts of consecutive lines in the mask image.
roiSize	Size of the source ROI in pixels.
coi	Channel of interest (for color images only); can be 1, 2, or 3.
pValue	Pointer to the computed L1- norm of pixel values.
value	An array containing the computed L1- norms of channel values in case of multi-channel data.
pNorm	Pointer to the computed norm value in the mask mode.
hint	Option to select the algorithmic implementation of the function.

Description

The flavors of the function ippiNorm_L1 that perform masked operation are declared in the ippcv.h file. All other function flavors are declared in the ippi.h file. The function operates with ROI (see Regions of Interest in Intel IPP). It computes the L1- norm pValue(pNorm``in mask mode) for the source image ``pSrc. This norm is defined as the sum of absolute pixel values in an image. Computation algorithm is specified by the hint argument (see Table "Hint Arguments for Image Moment Functions"). In the mask mode, the computation is done over pixels selected by nonzero mask values.

For non-masked operations on a multi-channel image (`Case 4, 5`), the norm is computed separately for each channel and stored in the array `value` .

In the mask multi-channel mode (`Case 6`), the norm is computed for a single channel of interest specified by `coi`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error when any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize` has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition in mask mode, if `srcStep` or `maskStep` is less than `roiSize.width * <pixelSize>`.
`ippStsNotEvenStepErr`	Indicates an error condition in mask mode if steps for floating-point images cannot be divided by 4.
`ippStsCOIErr`	Indicates an error when `coi` is not 1, 2, or 3.

Example

The code example below demonstrates how an image norm can be computed.

```
IppStatus norm( void ){
   Ipp64f sum, normL1;

   Ipp8u x[5*4];
   IppiSize roi = {5,4};
   ippiSet_8u_C1R( 1, x, 5, roi );
   ippiSum_8u_C1R( x, 5, roi, &sum);
   return ippiNorm_L1_8u_C1R( x, 5, roi, &normL1 );
}
```

Norm_L2

Computes the L2- norm of image pixel values.

Syntax

Case 1: Operation on one-channel integer data

```
IppStatus ippiNorm_L2_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp64f* pValue);
```

Supported values for `mod`:

`8u_C1R`	`16u_C1R`	`16s_C1R`

Case 2: Operation on one-channel floating-point data

```
IppStatus ippiNorm_L2_32f_C1R(const Ipp32f* pSrc, int srcStep, IppiSize roiSize,
Ipp64f* pValue, IppHintAlgorithm hint);
```

Case 3: Masked operation on one-channel data

```
IppStatus ippiNorm_L2_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp8u* pMask,
int maskStep, IppiSize roiSize, Ipp64f* pNorm);
```

Supported values for `mod`:

`8u_C1MR`	`16u_C1MR`	`32f_C1MR`

Case 4: Operation on multi-channel integer data

```
IppStatus ippiNorm_L2_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp64f value[3]);
```

Supported values for `mod`:

`8u_C3R`	`16u_C3R`	`16s_C3R`

```
IppStatus ippiNorm_L2_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize roiSize,
Ipp64f value[4]);
```

Supported values for `mod`:

`8u_C4R`	`16u_C4R`	`16s_C4R`

Case 5: Operation on multi-channel floating-point data

```
IppStatus ippiNorm_L2_32f_C3R(const Ipp32f* pSrc, int srcStep, IppiSize roiSize, Ipp64f
value[3], IppHintAlgorithm hint);
```

```
IppStatus ippiNorm_L2_32f_C4R(const Ipp32f* pSrc, int srcStep, IppiSize roiSize, Ipp64f
value[4], IppHintAlgorithm hint);
```

Case 6: Masked operation on multi-channel data

```
IppStatus ippiNorm_L2_<mod>(const Ipp<datatype>* pSrc, int srcStep, const Ipp8u* pMask,
int maskStep, IppiSize roiSize, int coi, Ipp64f* pNorm);
```

Supported values for `mod`:

`8u_C3CMR`	`16u_C3CMR`	`32f_C3CMR`

Include Files

`ippcv.h`

`ippi.h`

Domain Dependencies

Flavors declared in `ippcv.h`:

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Flavors declared in `ippi.h`:

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pMask	Pointer to the mask image.
maskStep	Distance in bytes between starts of consecutive lines in the mask image.
roiSize	Size of the source ROI in pixels.
coi	Channel of interest (for color images only); can be 1, 2, or 3.
pValue	Pointer to the computed L2- norm of pixel values.
value	An array containing the computed L2- norms of channel values in case of multi-channel data.
pNorm	Pointer to the computed norm value in the mask mode.
hint	Option to select the algorithmic implementation of the function.

Description

The flavors of the function ippiNorm_L2 that perform masked operation are declared in the ippcv.h file. All other function flavors are declared in the ippi.h file. The function operates with ROI (see Regions of Interest in Intel IPP). It computes the L2- norm pValue(pNorm``in mask mode) for the source image ``pSrc. This norm is defined as the square root of the sum of squared pixel values in an image. Computation algorithm is specified by the hint argument (see Table "Hint Arguments for Image Moment Functions"). In the mask mode, the computation is done over pixels selected by nonzero mask values.

For non-masked operations on a multi-channel image (Case 4,5), the norm is computed separately for each channel and stored in the array value.

In the mask multi-channel mode (Case 6), the norm is computed for a single channel of interest specified by coi.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value.
ippStsStepErr	Indicates an error condition in mask mode, if srcStep or maskStep is less than roiSize.width * <pixelSize>.

ippStsNotEvenStepErr	Indicates an error condition in mask mode if steps for floating-point images cannot be divided by 4.
ippStsCOIErr	Indicates an error when coi is not 1, 2, or 3.

NormDiff_Inf

Computes the infinity norm of differences between pixel values of two images.

Syntax

Case 1: Operation on one-channel data

```
IppStatus ippiNormDiff_Inf_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pValue);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32f_C1R

Case 2: Masked operation on one-channel data

```
IppStatus ippiNormDiff_Inf_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep, IppiSize roiSize,
Ipp64f* pNorm);
```

Supported values for mod:

8u_C1MR	16u_C1MR	32f_C1MR

Case 3: Operation on multi-channel data

```
IppStatus ippiNormDiff_Inf_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[3]);
```

Supported values for mod:

8u_C3R	16u_C3R	16s_C3R	32f_C3R

```
IppStatus ippiNormDiff_Inf_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[4]);
```

Supported values for mod:

8u_C4R	16u_C4R	16s_C4R	32f_C4R

Case 4: Masked operation on multi-channel data

```
IppStatus ippiNormDiff_Inf_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep, IppiSize roiSize,
int coi, Ipp64f* pNorm);
```

Supported values for mod:

8u_C3CMR	16u_C3CMR	32f_C3CMR

Include Files

ippcv.h

ippi.h

Domain Dependencies

Flavors declared in `ippcv.h`:

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Flavors declared in `ippi.h`:

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc1, pSrc2`	Pointers to the source images ROI.
`src1Step, src2Step`	Distance in bytes between starts of consecutive lines in the source images.
`pMask`	Pointer to the mask image.
`maskStep`	Distance in bytes between starts of consecutive lines in the mask image.
`roiSize`	Size of the source ROI in pixels.
`pValue`	Pointer to the computed infinity norm of difference between pixel values.
`value`	An array containing the computed infinity norms of difference between corresponding channel values in case of multi-channel data.
`coi`	Channel of interest (for color images only); can be 1, 2, or 3.
`pNorm`	Pointer to the computed norm value in the mask mode.

Description

The flavors of the function `ippiNormDiff_Inf` that perform masked operation are declared in the `ippcv.h` file. All other function flavors are declared in the `ippi.h` file. The function operates with ROI (see Regions of Interest in Intel IPP). It computes the infinity norm `pValue(pNorm``in the mask mode) of differences between pixel values of the two source images ``pSrc1` and `pSrc2`. This norm is defined as the largest absolute value of differences:

norm = max | `pSrc1-pSrc2` |

In the mask mode, the computation is done over pixels selected by nonzero mask values.

For non-masked operations on multi-channel images (`Case 3`), the norm is computed separately for each pair of corresponding channels and stored in the array `value` .

In the mask multi-channel mode (`Case 4`), the norm is computed for a single channel of interest specified by `coi`.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value.
ippStsStepErr	Indicates an error condition in mask mode, if src1Step, src2Step, or maskStep is less than roiSize.width * <pixelSize>.
ippStsNotEvenStepErr	Indicates an error condition in mask mode if steps for floating-point images cannot be divided by 4.
ippStsCOIErr	Indicates an error when coi is not 1, 2, or 3.

NormDiff_L1

Computes the L1- norm of differences between pixel values of two images.

Syntax

Case 1: Operation on one-channel integer data

```
IppStatus ippiNormDiff_L1_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pValue);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R

Case 2: Operation on one-channel floating-point data

```
IppStatus ippiNormDiff_L1_32f_C1R(const Ipp32f* pSrc1, int src1Step, const Ipp32f*
pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pValue, IppHintAlgorithm hint);
```

Case 3: Masked operation on one-channel data

```
IppStatus ippiNormDiff_L1_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep, IppiSize roiSize,
Ipp64f* pNorm);
```

Supported values for mod:

8u_C1MR	16u_C1MR	32f_C1MR

Case 4: Operation on multi-channel integer data

```
IppStatus ippiNormDiff_L1_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[3]);
```

Supported values for mod:

8u_C3R	16u_C3R	16s_C3R

```
IppStatus ippiNormDiff_L1_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[4]);
```

Supported values for `mod`:

`8u_C4R`	`16u_C4R`	`16s_C4R`

Case 5: Operation on multi-channel floating-point data

```
IppStatus ippiNormDiff_L1_32f_C3R(const Ipp32f* pSrc1, int src1Step, const Ipp32f*
pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[3], IppHintAlgorithm hint);
```

```
IppStatus ippiNormDiff_L1_32f_C4R(const Ipp32f* pSrc1, int src1Step, const Ipp32f*
pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[4], IppHintAlgorithm hint);
```

Case 6: Masked operation on multi-channel data

```
IppStatus ippiNormDiff_L1_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const Ipp32f*
pSrc2, int src2Step, const Ipp8u* pMask, int maskStep, IppiSize roiSize, int coi,
Ipp64f* pNorm);
```

Supported values for `mod`:

`8u_C3CMR`	`16u_C3CMR`	`32f_C3CMR`

Include Files

`ippcv.h`

`ippi.h`

Domain Dependencies

Flavors declared in `ippcv.h`:

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Flavors declared in `ippi.h`:

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc1, pSrc2`	Pointers to the source images ROI.
`src1Step, src2Step`	Distance in bytes between starts of consecutive lines in the source images.
`pMask`	Pointer to the mask image.
`maskStep`	Distance in bytes between starts of consecutive lines in the mask image.
`roiSize`	Size of the source ROI in pixels.
`coi`	Channel of interest (for color images only); can be 1, 2, or 3.
`pValue`	Pointer to the computed L1- norm of difference between pixel values.

value	An array containing the computed L1- norms of difference between channel values in case of multi-channel data.
pNorm	Pointer to the computed norm value in the mask mode.
hint	Option to select the algorithmic implementation of the function.

Description

The flavors of the function ippiNormDiff_L1 that perform masked operation are declared in the ippcv.h file. All other function flavors are declared in the ippi.h file. The function operates with ROI (see Regions of Interest in Intel IPP). It computes the L1-norm pValue(pNorm``in the mask mode) of differences between pixel values of the two source image buffers ``pSrc1 and pSrc2. This norm is defined as the sum of absolute values of differences:

norm = Σ | pSrc1–pSrc2 |

Computation algorithm is specified by the hint argument (see Table "Hint Arguments for Image Moment Functions"). In the mask mode, the computation is done over pixels selected by nonzero mask values.

For non-masked operations on multi-channel images (Case 4,5), the norm is computed separately for each pair of the corresponding channels and stored in the array value .

In the mask multi-channel mode (Case 6), the norm is computed for a single channel of interest specified by coi.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value.
ippStsStepErr	Indicates an error condition in mask mode, if src1Step, src2Step, or maskStep is less than roiSize.width * <pixelSize>.
ippStsNotEvenStepErr	Indicates an error condition in mask mode if steps for floating-point images cannot be divided by 4.
ippStsCOIErr	Indicates an error when coi is not 1, 2, or 3.

Example

The code example below shows how to use the function ippiNormDiff_L1_8u_C1R.

```
void func_normdiff_l1() {
    Ipp8u pSrc1[8*4];
    Ipp8u pSrc2[8*4];
    Ipp64f Value;
    int src1Step = 8;
    int src2Step = 8;
```

```
    IppiSize roi = {8,4};
    IppiSize roiSize = {5,4};


    ippiSet_8u_C1R(1, pSrc1, src1Step, roi);
    ippiSet_8u_C1R(2, pSrc2, src2Step, roi);


    ippiNormDiff_L1_8u_C1R( pSrc1, src1Step, pSrc2, src2Step, roiSize, &Value);
}
```

Result:

```
-> 20.0
```

NormDiff_L2

Computes the L2- norm of differences between pixel values of two images.

Syntax

Case 1: Operation on one-channel integer data

```
IppStatus ippiNormDiff_L2_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pValue);
```

Supported values for `mod`:

8u_C1R	16u_C1R	16s_C1R

Case 2: Operation on one-channel floating-point data

```
IppStatus ippiNormDiff_L2_32f_C1R(const Ipp32f* pSrc1, int src1Step, const Ipp32f*
pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pValue, IppHintAlgorithm hint);
```

Case 3: Masked operation on one-channel data

```
IppStatus ippiNormDiff_L2_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep, IppiSize roiSize,
Ipp64f* pNorm);
```

Supported values for `mod`:

8u_C1MR	16u_C1MR	32f_C1MR

Case 4: Operation on multi-channel integer data

```
IppStatus ippiNormDiff_L2_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[3]);
```

Supported values for `mod`:

8u_C3R	16u_C3R	16s_C3R

```
IppStatus ippiNormDiff_L2_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[4]);
```

Supported values for `mod`:

8u_C4R	16u_C4R	16s_C4R

Case 5: Operation on multi-channel floating-point data

```
IppStatus ippiNormDiff_L2_32f_C3R(const Ipp32f* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[3], IppHintAlgorithm
hint);

IppStatus ippiNormDiff_L2_32f_C4R(const Ipp32f* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[4], IppHintAlgorithm
hint);
```

Case 6: Masked operation on multi-channel data

```
IppStatus ippiNormDiff_L2_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep, IppiSize roiSize,
int coi, Ipp64f* pNorm);
```

Supported values for `mod`:

`8u_C3CMR`	`16u_C3CMR`	`32f_C3CMR`

Include Files

`ippcv.h`

`ippi.h`

Domain Dependencies

Flavors declared in `ippcv.h`:

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Flavors declared in `ippi.h`:

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc1, pSrc2`	Pointers to the source images ROI.
`src1Step, src2Step`	Distance in bytes between starts of consecutive lines in the source images.
`pMask`	Pointer to the mask image.
`maskStep`	Distance in bytes between starts of consecutive lines in the mask image.
`roiSize`	Size of the source ROI in pixels.
`coi`	Channel of interest (for color images only); can be 1, 2, or 3.
`pValue`	Pointer to the computed L2- norm of difference between pixel values.
`value`	An array containing the computed L2- norms of difference between channel values in case of multi-channel data.

pNorm	Pointer to the computed norm value in the mask mode.
hint	Option to select the algorithmic implementation of the function.

Description

The flavors of the function ippiNormDiff_L2 that perform masked operation are declared in the ippcv.h file. All other function flavors are declared in the ippi.h file. The function operates with ROI (see Regions of Interest in Intel IPP).
It computes the L2-norm pValue(pNorm``in the mask mode) of differences between pixel values of the two source image buffers ``pSrc1 and pSrc2. This norm is defined as the square root of the sum of squared differences:

$$norm = \sqrt{\sum |pSrc1 - pSrc2|^2} .$$

Computation algorithm is specified by the hint argument (see Table "Hint Arguments for Image Moment Functions"). In the mask mode, the computation is done over pixels selected by nonzero mask values.

For non-masked operations on multi-channel images (Case 4,5), the norm is computed separately for each pair of the corresponding channels and stored in the array value .

In the mask multi-channel mode (Case 6), the norm is computed for a single channel of interest specified by coi.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value.
ippStsStepErr	Indicates an error condition in mask mode, if src1Step, src2Step, or maskStep is less than roiSize.width * <pixelSize>.
ippStsNotEvenStepErr	Indicates an error condition in mask mode if steps for floating-point images cannot be divided by 4.
ippStsCOIErr	Indicates an error when coi is not 1, 2, or 3.

NormRel_Inf

Computes the relative error for the infinity norm of differences between pixel values of two images.

Syntax

Case 1: Operation on one-channel data

```
IppStatus ippiNormRel_Inf_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pValue);
```

Supported values for `mod`:

8u_C1R	16u_C1R	16s_C1R	32f_C1R

Case 2: Masked operation on one-channel data

```
IppStatus ippiNormRel_Inf_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep, IppiSize roiSize,
Ipp64f* pNorm);
```

Supported values for `mod`:

8u_C1MR	16u_C1MR	32f_C1MR

Case 3: Operation on multi-channel data

```
IppStatus ippiNormRel_Inf_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[3]);
```

Supported values for `mod`:

8u_C3R	16u_C3R	16s_C3R	32f_C3R

```
IppStatus ippiNormRel_Inf_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[4]);
```

Supported values for `mod`:

8u_C4R	16u_C4R	16s_C4R	32f_C4R

Case 4: Masked operation on multi-channel data

```
IppStatus ippiNormRel_Inf_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep, IppiSize roiSize,
int coi, Ipp64f* pNorm);
```

Supported values for `mod`:

8u_C3CMR	16u_C3CMR	32f_C3CMR

Include Files

`ippcv.h`

`ippi.h`

Domain Dependencies

Flavors declared in `ippcv.h`:

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Flavors declared in `ippi.h`:

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc1, pSrc2	Pointers to the source images ROI.
src1Step, src2Step	Distance in bytes between starts of consecutive lines in the source images.
pMask	Pointer to the mask image.
maskStep	Distance in bytes between starts of consecutive lines in the mask image.
roiSize	Size of the source ROI in pixels.
pValue	Pointer to the computed relative error value.
value	An array containing the computed relative error values for separate channels in case of multi-channel data.
coi	Channel of interest (for color images only); can be 1, 2, or 3.
pNorm	Pointer to the computed relative norm value in the mask mode.

Description

The flavors of the function ippiNormRel_Inf that perform masked operation are declared in the ippcv.h file. All other function flavors are declared in the ippi.h file. The function operates with ROI (see Regions of Interest in Intel IPP). It computes the infinity norm of differences between pixel values of two source buffers pSrc1 and pSrc2. This norm is defined as the largest absolute pixel value in an image. The output relative error pValue(pNorm``in the mask mode) is then formed by dividing the computed norm of differences by the infinity norm of the second source image buffer ``pSrc2. In the mask mode, the computation is done over pixels selected by nonzero mask values.

For non-masked operations on multi-channel images (Case 3), the relative norm is computed separately for each pair of corresponding channels and stored in the array value.

In the mask multi-channel mode (Case 4), the relative norm is computed for a single channel of interest specified by coi.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value.
ippStsStepErr	Indicates an error condition in mask mode, if src1Step, src2Step, or maskStep is less than roiSize.width * <pixelSize>.

ippStsNotEvenStepErr	Indicates an error condition in mask mode if steps for floating-point images cannot be divided by 4.
ippStsCOIErr	Indicates an error when coi is not 1, 2, or 3.
ippStsDivByZero	Indicates a warning when the infinity norm of pSrc2 has a zero value.

NormRel_L1

Computes the relative error for the L1 norm of differences between pixel values of two images.

Syntax

Case 1: Operation on one-channel integer data

```
IppStatus ippiNormRel_L1_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pValue);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R

Case 2: Operation on one-channel floating-point data

```
IppStatus ippiNormRel_L1_32f_C1R(const Ipp32f* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pValue, IppHintAlgorithm
hint);
```

Case 3: Masked operation on one-channel data

```
IppStatus ippiNormRel_L1_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep, IppiSize roiSize,
Ipp64f* pNorm);
```

Supported values for mod:

8u_C1MR	16u_C1MR	32f_C1MR

Case 4: Operation on multi-channel integer data

```
IppStatus ippiNormRel_L1_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[3]);
```

Supported values for mod:

8u_C3R	16u_C3R	16s_C3R

```
IppStatus ippiNormRel_L1_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[4]);
```

Supported values for mod:

8u_C4R	16u_C4R	16s_C4R

Case 5: Operation on multi-channel floating-point data

```
IppStatus ippiNormRel_L1_32f_C3R(const Ipp32f* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[3], IppHintAlgorithm
hint);
```

```
IppStatus ippiNormRel_L1_32f_C4R(const Ipp32f* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[4], IppHintAlgorithm
hint);
```

Case 6: Masked operation on multi-channel data

```
IppStatus ippiNormRel_L1_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep, IppiSize roiSize,
int coi, Ipp64f* pNorm);
```

Supported values for `mod`:

`8u_C3CMR`	`16u_C3CMR`	`32f_C3CMR`

Include Files

`ippcv.h`

`ippi.h`

Domain Dependencies

Flavors declared in `ippcv.h`:

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Flavors declared in `ippi.h`:

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc1, pSrc2`	Pointers to the source images ROI.
`src1Step, src2Step`	Distance in bytes between starts of consecutive lines in the source images.
`pMask`	Pointer to the mask image.
`maskStep`	Distance in bytes between starts of consecutive lines in the mask image.
`roiSize`	Size of the source ROI in pixels.
`coi`	Channel of interest (for color images only); can be 1, 2, or 3.
`pValue`	Pointer to the computed relative error value.
`value`	An array containing the computed relative error values for separate channels in case of multi-channel data.
`pNorm`	Pointer to the computed relative norm value in the mask mode.

hint	Option to select the algorithmic implementation of the function.

Description

The flavors of the function ippiNormRel_L1 that perform masked operation are declared in the ippcv.h file. All other function flavors are declared in the ippi.h file. The function operates with ROI (see Regions of Interest in Intel IPP). It computes the L1- norm of differences between pixel values of two source buffers pSrc1 and pSrc2. This norm is defined as the sum of absolute pixel values in an image. The output relative error pValue (pNorm in the mask mode) is then formed by dividing the computed norm of differences by the L1- norm of the second source image buffer pSrc2. Computation algorithm is specified by the hint argument (see Table "Hint Arguments for Image Moment Functions").In the mask mode, the computation is done over pixels selected by nonzero mask values.

For non-masked operations on multi-channel images (Cases 4, 5), the relative norm is computed separately for each pair of corresponding channels and stored in the array value.

In the mask multi-channel mode (Case 6), the relative norm is computed for a single channel of interest specified by coi.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value.
ippStsStepErr	Indicates an error condition in mask mode, if src1Step, src2Step, or maskStep is less than roiSize.width * <pixelSize>.
ippStsNotEvenStepErr	Indicates an error condition in mask mode if steps for floating-point images cannot be divided by 4.
ippStsCOIErr	Indicates an error when coi is not 1, 2, or 3.
ippStsDivByZero	Indicates a warning when the L1 norm of pSrc2 has a zero value.

Example

The code example below shows how to use the function ippiNormRel_L1_8u_C1R.

```
void func_normrel_l1() {
    Ipp8u pSrc1[8*4];
    Ipp8u pSrc2[8*4];
    Ipp64f Value;
    int src1Step = 8;
    int src2Step = 8;
    IppiSize roi = {8,4};
    IppiSize roiSize = {5,4};


    ippiSet_8u_C1R(1, pSrc1, src1Step, roi);
```

```
    ippiSet_8u_C1R(2, pSrc2, src2Step, roi);
    ippiNormRel_L1_8u_C1R( pSrc1, src1Step, pSrc2, src2Step, roiSize, &Value);
}
```

Output:

```
-> 0.5
```

NormRel_L2

Computes the relative error for the L2 norm of differences between pixel values of two images.

Syntax

Case 1: Operation on one-channel integer data

```
IppStatus ippiNormRel_L2_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pValue);
```

Supported values for `mod`:

8u_C1R	16u_C1R	16s_C1R

Case 2: Operation on one-channel floating-point data

```
IppStatus ippiNormRel_L2_32f_C1R(const Ipp32f* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f* pValue, IppHintAlgorithm
hint);
```

Case 3: Masked operation on one-channel data

```
IppStatus ippiNormRel_L2_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep, IppiSize roiSize,
Ipp64f* pNorm);
```

Supported values for `mod`:

8u_C1MR	16u_C1MR	32f_C1MR

Case 4: Operation on multi-channel integer data

```
IppStatus ippiNormRel_L2_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[3]);
```

Supported values for `mod`:

8u_C3R	16u_C3R	16s_C3R

```
IppStatus ippiNormRel_L2_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[4]);
```

Supported values for `mod`:

8u_C4R	16u_C4R	16s_C4R

Case 5: Operation on multi-channel floating-point data

```
IppStatus ippiNormRel_L2_32f_C3R(const Ipp32f* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[3], IppHintAlgorithm
hint);
```

```
IppStatus ippiNormRel_L2_32f_C4R(const Ipp32f* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp64f value[4], IppHintAlgorithm
hint);
```

Case 6: Masked operation on multi-channel data

```
IppStatus ippiNormRel_L2_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, const Ipp8u* pMask, int maskStep, IppiSize roiSize,
int coi, Ipp64f* pNorm);
```

Supported values for `mod`:

`8u_C3CMR`	`16u_C3CMR`	`32f_C3CMR`

Include Files

`ippcv.h`

`ippi.h`

Domain Dependencies

Flavors declared in `ippcv.h`:

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Flavors declared in `ippi.h`:

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc1, pSrc2`	Pointers to the source images ROI.
`src1Step, src2Step`	Distance in bytes between starts of consecutive lines in the source images.
`pMask`	Pointer to the mask image.
`maskStep`	Distance in bytes between starts of consecutive lines in the mask image.
`roiSize`	Size of the source ROI in pixels.
`coi`	Channel of interest (for color images only); can be 1, 2, or 3.
`pValue`	Pointer to the computed relative error value.
`value`	An array containing the computed relative error values for separate channels in case of multi-channel data.
`pNorm`	Pointer to the computed relative norm value in the mask mode.

hint	Option to select the algorithmic implementation of the function.

Description

The flavors of the function ippiNormRel_L2 that perform masked operation are declared in the ippcv.h file. All other function flavors are declared in the ippi.h file. The function operates with ROI (see Regions of Interest in Intel IPP). It computes the L2- norm of differences between pixel values of two source buffers pSrc1 and pSrc2. This norm is defined as the square root of the sum of squared pixel values in an image. The output relative error pValue(pNorm``in the mask mode) is then formed by dividing the computed norm of differences by the L2- norm of the second source image buffer ``pSrc2. Computation algorithm is specified by the hint argument (see Table "Hint Arguments for Image Moment Functions").In the mask mode, the computation is done over pixels selected by nonzero mask values.

For non-masked operations on multi-channel images (Cases 4, 5), the relative norm is computed separately for each pair of corresponding channels and stored in the array value.

In the mask multi-channel mode (Case 6), the relative norm is computed for a single channel of interest specified by coi.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value.
ippStsStepErr	Indicates an error condition in mask mode, if src1Step, src2Step, or maskStep is less than roiSize.width * <pixelSize>.
ippStsNotEvenStepErr	Indicates an error condition in mask mode if steps for floating-point images cannot be divided by 4.
ippStsCOIErr	Indicates an error when coi is not 1, 2, or 3.
ippStsDivByZero	Indicates a warning when the L2 norm of pSrc2 has a zero value.

Example

The code example below shows how to use the function ippiNormRel_L2_8u_C1R.

```
void func_normrel_l1() {
    Ipp8u pSrc1[8*4];
    Ipp8u pSrc2[8*4];
    Ipp64f Value;
    int src1Step = 8;
    int src2Step = 8;
    IppiSize roi = {8,4};
    IppiSize roiSize = {5,4};


    ippiSet_8u_C1R(1, pSrc1, src1Step, roi);
```

```
    ippiSet_8u_C1R(10, pSrc2, src2Step, roi);
    ippiNormRel_L2_8u_C1R( pSrc1, src1Step, pSrc2, src2Step, roiSize, &Value);
}
```

Result:

```
-> 0.9
```

Image Quality Index

Intel IPP functions described in this section compute the universal image quality index [Wang02] that may be used as image and video quality distortion measure. It is mathematically defined by modeling the image distortion relative to the reference image as a combination of three factors: loss of correlation, luminance distortion, and contrast distortion.

If two images `f` and `g` are considered as a matrices with `M` column and `N` rows containing pixel values `f[i,j],g[i,j]`, respectively (`0≥ i > M`, `0≥ j > N`), the universal image quality index `Q` may be calculated as a product of three components:

$$Q = \frac{\sigma_{fg}}{\sigma_f \sigma_g} \cdot \frac{2\bar{f}\bar{g}}{(\bar{f})^2 + (\bar{g})^2} \cdot \frac{2\sigma_f \sigma_g}{\sigma_f^2 + \sigma_g^2}$$

where

$$\bar{f} = \frac{1}{MN} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} f[i,j] \qquad \bar{g} = \frac{1}{MN} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} g[i,j]$$

$$\sigma_{fg} = \frac{1}{M+N-1} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} (f[i,j] - \bar{f})(g[i,j] - \bar{g})$$

$$\sigma_f^2 = \frac{1}{M+N-1} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} (f[i,j] - \bar{f})^2 \qquad \sigma_g^2 = \frac{1}{M+N-1} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} (g[i,j] - \bar{g})^2$$

The first component is the correlation coefficient, which measures the degree of linear correlation between images `f` and `g`. It varies in the range [-1, 1]. The best value 1 is obtained when `f` and `g` are linearly related, which means that `g[i,j] =af [ i,j]` + b for all possible values of `i` and `j`. The second component, with a value range of [0, 1], measures how close the mean luminance is between images. Since σ_f and σ_g can be considered as estimates of the contrast of `f` and `g`, the third component measures how similar the contrasts of the images are. The value range for this component is also [0, 1].

The range of values for the index `Q` is [-1, 1]. The best value 1 is achieved if and only if the images are identical.

- QualityIndexGetBufferSize

- QualityIndex

QualityIndexGetBufferSize

Computes the size of the work buffer for the `ippiQualityIndex` function.

Syntax

```
IppStatus ippiQualityIndexGetBufferSize(IppDataType srcType, IppChannels ippChan,
IppiSize roiSize, int* pBufferSize);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`srcType`	Data type of the source images. Possible values: `ipp8u`, `ipp16u`, or `ipp32f`.
`ippChan`	Number of channels in the source images. Possible values: `ippC1`, `ippC3`, or `ippAC4`.
`roiSize`	Size, in pixels, of the source images.
`pBufferSize`	Pointer to the computed value of the buffer size, in bytes.

Description

The function computes the size of the work buffer, in bytes, for the ippiQualityIndex function and stores the result in the `pBufferSize` parameter.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error when any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error when `roiSize` is less than, or equal to zero.
`ippStsDataTypeErr`	Indicates an error when `srcType` has an illegal value.
`ippStsChannelErr`	Indicates an error when `ippChan` has an illegal value.

See Also

QualityIndex Computes the universal image quality index.

QualityIndex

Computes the universal image quality index.

Syntax

Case 1: Operation on one-channel data

```
IppStatus ippiQualityIndex_<mod>(const Ipp<srcDatatype>* pSrc1, int src1Step, const
Ipp<srcDatatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp<dstDatatype>
pQualityIndex[1], Ipp8u* pBuffer);
```

Supported values for `mod`:

`8u32f_C1R`	`16u32f_C1R`	`32f_C1R`

Case 2: Operation on multi-channel data

```
IppStatus ippiQualityIndex_<mod>(const Ipp<srcDatatype>* pSrc1, int src1Step, const
Ipp<srcDatatype>* pSrc2, int src2Step, IppiSize roiSize, Ipp<dstDatatype>
pQualityIndex[3], Ipp8u* pBuffer);
```

Supported values for `mod`:

`8u32f_C3R`	`16u32f_C3R`	`32f_C3R`
`8u32f_AC4R`	`16u32f_AC4R`	`32f_AC4R`

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc1, pSrc2`	Pointers to the source images ROI.
`src1Step, src2Step`	Distance in bytes between starts of consecutive lines in the source images.
`roiSize`	Size of the source ROI in pixels.
`pQualityIndex`	Pointer to the computed quality index value.
`pBuffer`	Pointer to the buffer for internal calculations. To compute the size of the buffer, use the QualityIndexGetBufferSize function.

Description

This function operates with ROI (see Regions of Interest in Intel IPP). This function computes the universal image quality index for two images `pSrc1` and `pSrc2` according to the formula in the introduction section above. The computed value of the index is stored in `pQualityIndex`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error when any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize` has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if `src1Step` or `src2Step` has a zero or negative value.
`ippStsQualityIndexErr`	Indicates an error condition if pixel values of one of the images are constant.
`ippStsMemAllocErr`	Indicates an error condition if memory allocation fails.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

//    A simple example of computing the universal image quality index for two images.
// implemented with Intel(R) Integrated Performance Primitives (Intel(R) IPP) functions:
//     ippiImageJaehne_32f_C1R
//     ippiImageJaehne_32f_C1R
//     ippiAddRandUniform_32f_C1IR
//     ippiQualityIndexGetBufferSize
//     ippiQualityIndex_32f_C1R

#include <stdio.h>
#include "ipp.h"

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define check_sts(st)        \
```

```
    if ((st) != ippStsNoErr) \
        goto exitLine; /* Go to Exit if Intel(R) IPP function returned status different from
ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main(void)
{
    IppStatus status = ippStsNoErr;
    /* Pointers to source images */
    Ipp32f pSrc1[256 * 256];
    Ipp32f pSrc2[256 * 256];
    /* Distance in bytes between starts of consecutive lines in the source images */
    int src1Step = 256 * sizeof(Ipp32f);
    int src2Step = 256 * sizeof(Ipp32f);
    Ipp32f qIndex = 0; /* quality index value */
    IppiSize roiSize = {256, 256};
    unsigned int pSeed = 3;
    int bufSize = 0;       /* work buffer size */
    Ipp8u *pBuffer = NULL; /* pointer to the work buffer */

    /* create reference image (image1) */
    check_sts(status = ippiImageJaehne_32f_C1R(pSrc1, src1Step, roiSize));

    /* ... and distorted image (image2 = image1 + RandUniform) */
    check_sts(status = ippiImageJaehne_32f_C1R(pSrc2, src2Step, roiSize));

    check_sts(status = ippiAddRandUniform_32f_C1IR(pSrc2, src2Step, roiSize, 0, 1, &pSeed));

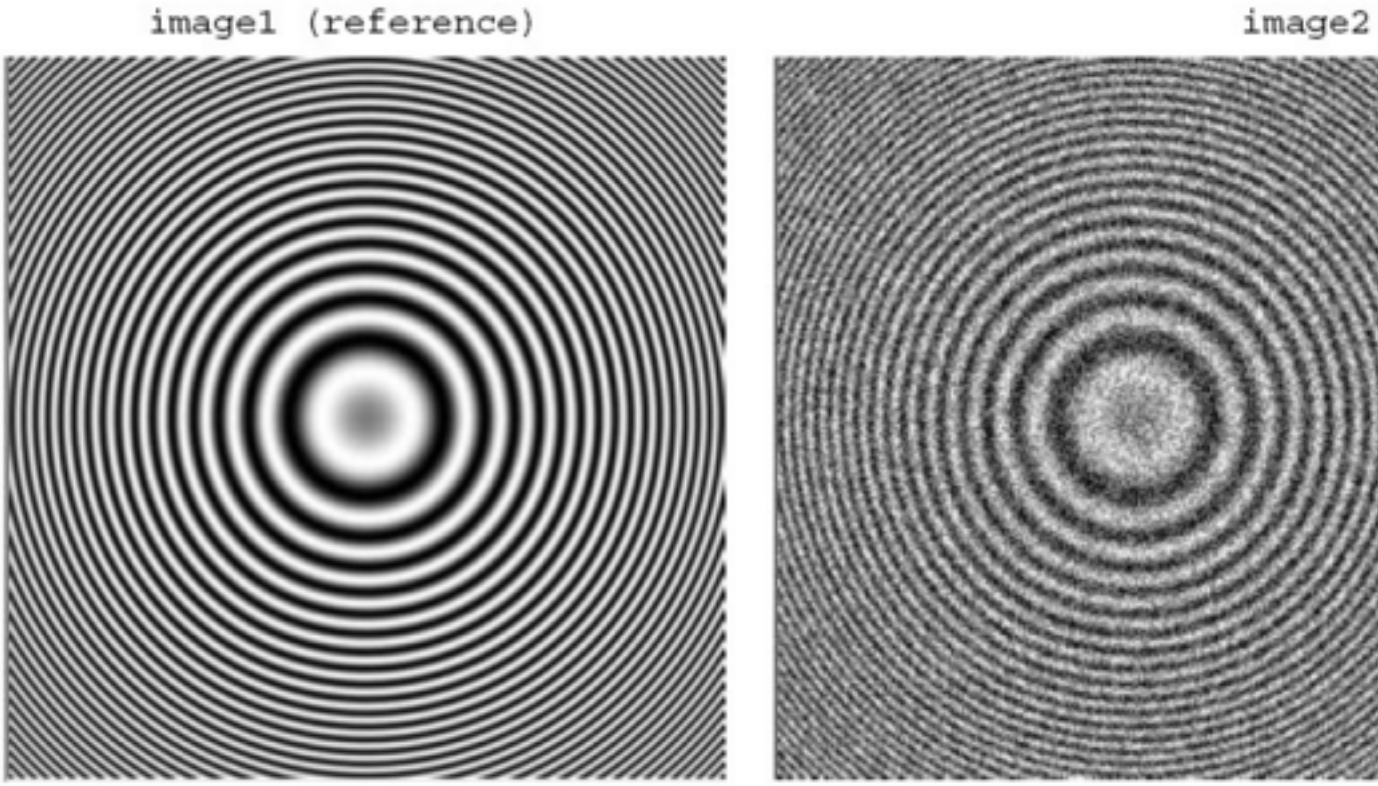
    check_sts(status = ippiQualityIndexGetBufferSize(ipp32f, ippC1, roiSize, &bufSize));

    /* work buffer memory allocation */
    pBuffer = (Ipp8u *)ippMalloc(bufSize);

    /* computes the universal image quality index for two images pSrc1 and pSrc2 */
    check_sts(status = ippiQualityIndex_32f_C1R(pSrc1, src1Step, pSrc2, src2Step, roiSize,
&qIndex, pBuffer));

    EXIT_MAIN
    ippFree(pBuffer);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

```
Result:
```




```
Universal image quality index of the distorted image2 -> pQIndex =
```

See Also

QualityIndexGetBufferSize Computes the size of the work buffer for the ippiQualityIndex function.

Image Proximity Measures

The functions described in this section compute the proximity (similarity) measure between an image and a template (another image). These functions may be used as feature detection functions, as well as the components of more sophisticated techniques.

There are several ways to compute the measure of similarity between two images. One way is to compute the Euclidean distance, or sum of the squared distances (SSD), of an image and a template. The smaller is the value of SSD at a particular pixel, the more similarity exists between the template and the image in the neighborhood of that pixel.

The squared Euclidean distance $S_{tx}(r,c)$ between a template and an image for the pixel in row r and column c is given by the equation:

$$S_{tx}(r,c) = \sum_{j=0}^{tplRows-1} \sum_{i=0}^{tplCols-1} \left[t(j,i) - x\left(r + j - \frac{tplRows}{2}, c + i - \frac{tplCols}{2}\right)\right]^2$$

where x(r,c) is the image pixel value in row r and column c, and t(j,i) is the template pixel value in row j and column i; template size is tplCols by tplRows and its center is positioned at (r,c).

The other similarity measure is the cross-correlation function: the higher is the cross-correlation at a particular pixel, the more similarity exists between the template and the image in the neighborhood of that pixel.

The cross-correlation R_{tx} (r,c) between a template and an image at the pixel in row r and column c is computed by the equation :

$$R_{tx}(r,\ c) = \sum_{j=0}^{tplRows-1} \sum_{i=0}^{tplCols-1} t(j,\ i) \cdot x\left(r+j-\frac{tplRows}{2},\ c+i-\frac{tplCols}{2}\right)$$

The cross-correlation function is dependent on the brightness variation across the image. To avoid this dependence, the correlation coefficient function is used instead. It is defined as:

$$G_{tx}(r,\ c) = \sum_{j=0}^{tplRows-1} \sum_{i=0}^{tplCols-1} [t(j,\ i)-\bar{t}] \cdot \left[x\left(r+j-\frac{tplRows}{2},\ c+i-\frac{tplCols}{2}\right)-\bar{x}\right.$$

where t with the overline is the mean of the template, and x with the overline is the mean of the image in the region just under the template.

All Intel IPP proximity functions compute normalized values of SSD, cross-correlation and correlation coefficient that are defined as follows:

normalized SSD: σ_{tx} (r,c)

$$\sigma_{tx}(r,\ c) = \frac{S_{tx}(r,\ c)}{\sqrt{R_{xx}(r,\ c)R_{tt}\left(\frac{tplRows}{2},\ \frac{tplCols}{2}\right)}}$$

normalized cross-correlation ρ_{tx} (r,c):

$$\rho_{tx}(r,\ c) = \frac{R_{tx}(r,\ c)}{\sqrt{R_{xx}(r,\ c)R_{tt}\left(\frac{tplRows}{2},\ \frac{tplCols}{2}\right)}}$$

Here R_{xx} and R_{tt} denote the autocorrelation of the image and the template, respectively:

$$R_{xx}(r,\ c) = \sum_{j=r-\frac{tplRows-1}{2}}^{r+\frac{tplRows-1}{2}} \sum_{i=c-\frac{tplCols-1}{2}}^{c+\frac{tplCols-1}{2}} x_{j,\ i}x_{j,\ i}$$

$$R_{tt}\left(\frac{tplRows}{2},\ \frac{tplCols}{2}\right) = \sum_{j=0}^{tplRows-1} \sum_{i=0}^{tplCols-1} t_{j,\ i}t_{j,\ i}$$

Normalized correlation coefficient γ_{tx} (`r,c`):

$$\gamma_{tx}(r,\ c) = \frac{G_{tx}(r,\ c)}{\sqrt{G_{xx}(r,\ c)G_{tt}\left(\frac{tplRows}{2},\ \frac{tplCols}{2}\right)}}$$

Here G_{xx} and G_{tt} denote the autocorrelations
of the image and the template without constant brightness component,
respectively:

$$G_{xx}(r,\ c) = \sum_{j=r-\frac{tplRows-1}{2}}^{r+\frac{tplRows-1}{2}} \sum_{i=c-\frac{tplCols-1}{2}}^{c+\frac{tplCols-1}{2}} [x_{j,\ i} - \bar{x}(r,\ c)]^2$$

$$G_{tt}\left(\frac{tplRows}{2},\ \frac{tplCols}{2}\right) = \sum_{j=0}^{tplRows-1} \sum_{i=0}^{tplCols-1} (t_{j,\ i} - \bar{t})^2$$

- SqrDistanceNormGetBufferSize
- SqrDistanceNorm
- CrossCorrNormGetBufferSize
- CrossCorrNorm
- SADGetBufferSize
- SAD

SqrDistanceNormGetBufferSize

Computes the size of the work buffer for the `ippiSqrDistanceNorm` function.

Syntax

```
IppStatus ippiSqrDistanceNormGetBufferSize (IppiSize srcRoiSize, IppiSize tplRoiSize,
IppEnum algType, int* pBufferSize);
```

Include Files

`ippi.h`

Parameters

`srcRoiSize`, `tplRoiSize`	Size of the source/template ROI in pixels.
`algType`	Bit-field mask for the algorithm type definition. Possible values are the results of composition of the `IppAlgType`, `IppiROIShape`, and `IppiNormOp` values.
`pBufferSize`	Pointer to the size of the work buffer.

Description

The `ippiSqrDistanceNormGetBufferSize` function computes the size, in bytes, of the external work buffer needed for the function that computes the Euclidean distance between an image and a template. The result is stored in the `pBufferSize` parameter.

Return Values

`ippStsNoErr`	Indicates no error.	
`ippStsSizeErr`	Indicates an error when:	• `srcRoiSize` or `tplRoiSize` is negative, or equal to zero • the value of `srcRoiSize` is less than the corresponding value of `tplRoiSize`
`ippStsAlgTypeErr`	Indicates an error when:	• `srcRoiSize` or `tplRoiSize` is negative, or equal to zero • the value of `srcRoiSize` is less than the corresponding value of `tplRoiSize`
`ippStsNullErr`	Indicates an error when the `pBufferSize` is NULL.	

See Also

Structures and Enumerators

SqrDistanceNorm Computes Euclidean distance between an image and a template.

SqrDistanceNorm

Computes Euclidean distance between an image and a template.

Syntax

Case 1: Operating with integer output

```
IppStatus ippiSqrDistanceNorm_8u_C1RSfs(const Ipp8u* pSrc, int srcStep, IppiSize
srcRoiSize, const Ipp8u* pTpl, int tplStep, IppiSize tplRoiSize, Ipp8u* pDst, int
dstStep, int scaleFactor, IppEnum algType, Ipp8u* pBuffer);
```

Case 2: Operating on data with floating-point output

```
IppStatus ippiSqrDistanceNorm_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, IppiSize
srcRoiSize, const Ipp<srcDatatype>* pTpl, int tplStep, IppiSize tplRoiSize, Ipp32f*
pDst, int dstStep, IppEnum algType, Ipp8u* pBuffer);
```

Supported values for `mod`:

`32f_C1R`	`8u32f_C1R`	`16u32f_C1R`

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance, in bytes, between the starting points of consecutive lines in the source image.
`srcRoiSize`	Size of the source ROI in pixels.
`pTpl`	Pointer to the template image.
`tplStep`	Distance, in bytes, between the starting points of consecutive lines in the template image.
`tplRoiSize`	Size of the template ROI in pixels.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance, in bytes, between the starting points of consecutive lines in the destination image.
`scaleFactor`	Scale factor.
`algType`	Bit-field mask for the algorithm type definition. Possible values are the results of composition of the `IppAlgType`, `IppiROIShape`, and `IppiNormOp` values.
`pBuffer`	Pointer to the work buffer.

Description

Before using this function, you need to compute the size of the work buffer using the `ippiSqrDistanceNormGetBufferSize` function.

This function operates with ROI.

Depending on the `IppiNormOp` value set to the `algType` parameter, the function calculates the following results:

`IppiNormOp` Value	Result
`ippiNormNone`	Squared Euclidean distances S_{tx} (`r`,`c`)
`ippiNorm`	Normalized squared Euclidean distances σ_{tx} (`r`,`c`)

For more information on how each value is calculated, see Image Proximity Measures.

The size of the resulting matrix depends on the `IppiROIShape` value:

`IppiROIShape` Value	Matrix Size
`ippiROIFull`	(W_s + W_t - 1) * (H_s + H_t - 1)
`ippiROISame`	W_s * H_s
`ippiROIValid`	(W_s -W_t + 1) * (H_s -H_t +1)

where

- W_s , H_s is the width and height of the source image
- W_t , H_t is the width and height of the template image

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when any of the specified pointers is NULL.
`ippStsStepErr`	Indicates an error when the value of `srcStep`, `tplStep`, or `dstStep` is negative, or equal to zero.
`ippStsSizeErr`	Indicates an error when: • `srcRoiSize` or `tplRoiSize` is negative, or equal to zero • the value of `srcRoiSize` is less than the corresponding value of `tplRoiSize`
`ippStsAlgTypeErr`	Indicates an error when:

Example

The code example below demonstrates how to use the `ippiSqrDistanceNormGetBufferSize` and `ippiSqrDistanceNorm` functions.

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
```

```
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *****************************************************************************/

#include "ipps.h"
#include "ippi.h"
#include <iostream>
#include <iomanip>

void print_dst_2D_32f(Ipp32f *pDst, int dstStep, IppiSize roiSize)
{
    std::cout << "pDst ->\n";
    std::cout.precision(2);
    for (int h = 0; h < roiSize.height; h++) {
        for (int w = 0; w < roiSize.width; w++) {
            std::cout << std::setw(6) << std::fixed << pDst[w];
        }
        pDst = (Ipp32f *)((Ipp8u *)pDst + dstStep);
        std::cout << std::endl;
    }
}

IppStatus main()
{
    IppStatus status;
    Ipp32f pSrc[5 * 4];
    Ipp32f pTpl[5 * 4];
    Ipp32f pDst[9 * 7]; //(5+5-1) x (4+4-1)

    IppiSize srcRoiSize = {5, 4};
    IppiSize tplRoiSize = {5, 4};
    IppiSize dstRoiSize = {9, 7};
    int srcStep = 5 * sizeof(Ipp32f);
    int tplStep = 5 * sizeof(Ipp32f);
    int dstStep = 9 * sizeof(Ipp32f);
    IppEnum funCfg = (IppEnum)(ippAlgAuto | ippiNorm | ippiROIFull);
    Ipp8u *pBuffer;
    int bufSize = 0;

    ippiSet_32f_C1R(2.0, pSrc, srcStep, srcRoiSize);
    ippiSet_32f_C1R(1.0, pTpl, tplStep, tplRoiSize);

    status = ippiSqrDistanceNormGetBufferSize(srcRoiSize, tplRoiSize, funCfg, &bufSize);
    if (status != ippStsNoErr)
        return status;

    pBuffer = ippsMalloc_8u(bufSize);

    status = ippiSqrDistanceNorm_32f_C1R(pSrc, srcStep, srcRoiSize, pTpl, tplStep, tplRoiSize,
pDst, dstStep, funCfg, pBuffer);
    print_dst_2D_32f(pDst, dstStep, dstRoiSize);

    ippsFree(pBuffer);
    return status;
}
```

The result is as follows:

```
pDst ->
 2.24 1.58 1.29 1.12 1.00 1.12 1.29 1.58 2.24
 1.58 1.12 0.91 0.79 0.71 0.79 0.91 1.12 1.58
 1.29 0.91 0.75 0.65 0.58 0.65 0.75 0.91 1.29
 1.12 0.79 0.65 0.56 0.50 0.56 0.65 0.79 1.12
 1.29 0.91 0.75 0.65 0.58 0.65 0.75 0.91 1.29
 1.58 1.12 0.91 0.79 0.71 0.79 0.91 1.12 1.58
 2.24 1.58 1.29 1.12 1.00 1.12 1.29 1.58 2.24
```

See Also

Integer Result Scaling

Image Proximity Measures

Regions of Interest in Intel IPP

Structures and Enumerators

SqrDistanceNormGetBufferSize Computes the size of the work buffer for the ippiSqrDistanceNorm function.

CrossCorrNormGetBufferSize

Computes the size of the work buffer for the `ippiCrossCorrNorm` function.

Syntax

```
IppStatus ippiCrossCorrNormGetBufferSize (IppiSize srcRoiSize, IppiSize tplRoiSize,
IppEnum algType, int* pBufferSize);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`srcRoiSize`, `tplRoiSize`	Size of the source/template ROI in pixels.
`algType`	Bit-field mask for the algorithm type definition. Possible values are the results of composition of the `IppAlgType`, `IppiROIShape`, and `IppiNormOp` values.
`pBufferSize`	Pointer to the size of the work buffer.

Description

The `ippiCrossCorrNormGetBufferSize` function computes the size, in bytes, of the external work buffer needed for the function that performs two-dimensional cross-correlation. The result is stored in the `pBufferSize` parameter.

Return Values

ippStsNoErr	Indicates no error.	
ippStsSizeErr	Indicates an error when:	• srcRoiSize or tplRoiSize is negative, or equal to zero • the value of srcRoiSize is less than the corresponding value of tplRoiSize.
ippStsAlgTypeErr	Indicates an error when:	• srcRoiSize or tplRoiSize is negative, or equal to zero • the value of srcRoiSize is less than the corresponding value of tplRoiSize.
ippStsNullPtrErr	Indicates an error when pBufferSize is NULL.	

See Also

Structures and Enumerators

CrossCorrNorm Computes a normalized cross-correlation between an image and a template.

CrossCorrNorm

Computes a normalized cross-correlation between an image and a template.

Syntax

Case 1: Operating on data with integer output

```
IppStatus ippiCrossCorrNorm_8u_C1RSfs(const Ipp8u* pSrc, int srcStep, IppiSize
srcRoiSize, const Ipp8u* pTpl, int tplStep, IppiSize tplRoiSize, Ipp8u* pDst, int
dstStep, int scaleFactor, IppEnum algType, Ipp8u* pBuffer);
```

Case 2: Operating on data with floating-point output

```
IppStatus ippiCrossCorrNorm_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, IppiSize
srcRoiSize, const Ipp<srcDatatype>* pTpl, int tplStep, IppiSize tplRoiSize, Ipp32f*
pDst, int dstStep, IppEnum algType, Ipp8u* pBuffer);
```

Supported values for mod:

32f_C1R	8u32f_C1R	16u32f_C1R

Case 3: Operating on data with integer output with TL functions

```
IppStatus ippsCrossCorrNorm_ 8u_C1RSfs_T (const Ipp8u* pSrc, int srcStep,
IppiSizesrcRoiSize, const Ipp8u* pTpl, int tplStep, IppiSize tplRoiSize, Ipp8u* pDst,
int dstStep, int scaleFactor, IppEnum algType, Ipp8u* pBuffer);
```

Case 4: Operating on data with floating-point output with TL functions

```
IppStatus ippsCrossCorrNorm_ <mod>_T (const Ipp<srcDatatype>* pSrc, int srcStep,
IppiSizesrcRoiSize, const Ipp<srcDatatype>* pTpl, int tplStep, IppiSize tplRoiSize,
Ipp32f* pDst, int dstStep, IppEnum algType, Ipp8u* pBuffer);
```

Supported values for mod:

32f_C1R	8u32f_C1R	16u32f_C1R

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance, in bytes, between the starting points of consecutive lines in the source image.
srcRoiSize	Size of the source ROI in pixels.
pTpl	Pointer to the template image.
tplStep	Distance, in bytes, between the starting points of consecutive lines in the template image.
tplRoiSize	Size of the template ROI in pixels.
pDst	Pointer to the destination image ROI.
dstStep	Distance, in bytes, between the starting points of consecutive lines in the destination image.
scaleFactor	Scale factor.
algType	Bit-field mask for the algorithm type definition. Possible values are the results of composition of the IppAlgType, IppiROIShape, and IppiNormOp values.
pBuffer	Pointer to the work buffer.

Description

This function operates with ROI.

Depending on the IppiNormOp value set to the algType parameter, the function calculates the following results:

IppiNormOp Value	Result
ippiNormNone	Cross-correlation values $R_{tx}(r,c)$
ippiNorm	Normalized cross-correlation values $\rho_{tx}(r,c)$
ippiNormCoefficient	Normalized correlation coefficients $\gamma_{tx}(r,c)$

For more information about how each value is calculated, see Image Proximity Measures.

The size of the resulting matrix depends on the IppiROIShape value:

IppiROIShape Value	Matrix Size
ippiROIFull	$(W_s + W_t - 1) * (H_s + H_t - 1)$
ippiROISame	$W_s * H_s$
ippiROIValid	$(W_s - W_t + 1) * (H_s - H_t + 1)$

where

- W_s , H_s is the width and height of the source image
- W_t , H_t is the width and height of the template image

Before using this function, you need to compute the size of the work buffer using the ippiCrossCorrNormGetBufferSize function.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.
ippStsStepErr	Indicates an error when the value of srcStep, tplStep, or dstStep is negative, or equal to zero.
ippStsSizeErr	Indicates an error when: • the srcRoiSize or tplRoiSize is negative, or equal to zero. • the value of srcRoiSize is less than the corresponding value of the tplRoiSize.
ippStsAlgTypeErr	Indicates an error when:

Example

The code example below demonstrates how to use the ippiCrossCorrNormGetBufferSize and ippiCrossCorrNorm functions.

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

// The code example below demonstrates how to use the
//  ippiCrossCorrNormGetBufferSize and ippiCrossCorrNorm functions.
```

```
#include "ipps.h"
#include "ippi.h"
#include <iostream>
#include <iomanip>

void print_dst_2D_32f(Ipp32f *pDst, int dstStep, IppiSize roiSize)
{
    std::cout << "pDst ->\n";
    std::cout.precision(2);
    for (int h = 0; h < roiSize.height; h++) {
        for (int w = 0; w < roiSize.width; w++) {
            std::cout << std::setw(6) << std::fixed << pDst[w];
        }
        pDst = (Ipp32f *)((Ipp8u *)pDst + dstStep);
        std::cout << std::endl;
    }
}

IppStatus main()
{
    IppStatus status;
    IppiSize srcRoiSize = {5, 4};
    IppiSize tplRoiSize = {3, 3};
    IppiSize dstRoiSize = {5, 4}; // same as src

    Ipp32f pSrc[5 * 4] = {1.0f, 2.0f, 1.5f, 4.1f, 3.6f, 0.2f, 3.2f, 2.5f, 1.5f, 10.0f, 5.0f,
6.8f, 0.5f, 4.1f, 1.1f, 7.1f, 4.2f, 2.2f, 8.7f, 10.0f};

    Ipp32f pTpl[3 * 3] = {2.1f, 3.5f, 7.7f, 0.4f, 2.3f, 5.5f, 1.4f, 2.8f, 3.1f};
    Ipp32f pDst[5 * 4];
    int srcStep = 5 * sizeof(Ipp32f);
    int tplStep = 3 * sizeof(Ipp32f);
    int dstStep = 5 * sizeof(Ipp32f);
    IppEnum funCfg = (IppEnum)(ippAlgAuto | ippiROISame | ippiNorm);
    Ipp8u *pBuffer;
    int bufSize;

    status = ippiCrossCorrNormGetBufferSize(srcRoiSize, tplRoiSize, funCfg, &bufSize);
    if (status != ippStsNoErr)
        return status;

    pBuffer = ippsMalloc_8u(bufSize);

    status = ippiCrossCorrNorm_32f_C1R(pSrc, srcStep, srcRoiSize, pTpl, tplStep, tplRoiSize,
pDst, dstStep, funCfg, pBuffer);
    print_dst_2D_32f(pDst, dstStep, dstRoiSize);

    ippsFree(pBuffer);
    return status;
}
```

The result is as follows:

```
pDst ->
 0.53 0.54 0.58 0.50 0.30
 0.68 0.62 0.68 0.83 0.38
 0.77 0.55 0.60 0.81 0.42
 0.81 0.46 0.70 0.62 0.24
```

See Also

Integer Result Scaling

Regions of Interest in Intel IPP

Structures and Enumerators

Image Proximity Measures

SADGetBufferSize

Computes the size of the work buffer for the `ippiSAD` function.

Syntax

```
IppStatus ippiSADGetBufferSize(IppiSize srcRoiSize, IppiSize tplRoiSize, IppDataType dataType, int numChannels, IppiROIShape shape, int* pBufferSize);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`srcRoiSize`	Size of the source ROI in pixels.
`tplRoiSize`	Size of the template ROI in pixels.
`shape`	Enumeration that defines the shape of the result of the SAD operation (see Structures and Enumerators).
`dataType`	Type of the input data.
`numChannels`	Number of channels in the images.
`pBufferSize`	Pointer to the computed value of the external buffer size.

Description

The `ippiSADGetBufferSize` function computes the size of the external work buffer (in bytes) needed for the ippiSAD function and stores the result in the `*pBufferSize` parameter.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if the `pBufferSize` pointer is `NULL`.

ippStsSizeErr	Indicates an error condition if srcRoiSize or tplRoiSize has a field with a zero or negative value.
ippStsNotSupportedModeErr	Indicates an error condition if either shape does not equal ippiROIValid or numChannels does not equal 1.

SAD

Computes sums of absolute differences (SAD) for a template image and different locations within a source image where the template image can fit.

Syntax

```
IppStatus ippiSAD_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, IppiSize srcRoiSize,
const Ipp<srcDatatype>* pTpl, int tplStep, IppiSize tplRoiSize, Ipp<dstDatatype>* pDst,
int dstStep, IppiROIShape shape, int scaleFactor, Ipp8u* pBuffer);
```

```
IppStatus ippiSAD_32f_C1R(const Ipp<srcDatatype>* pSrc, int srcStep, IppiSize
srcRoiSize, const Ipp<srcDatatype>* pTpl, int tplStep, IppiSize tplRoiSize,
Ipp<dstDatatype>* pDst, int dstStep, IppiROIShape shape, Ipp8u* pBuffer);
```

Supported values for mod:

8u32s_C1RSfs	16u32s_C1RSfs	16s32s_C1RSfs

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance, in bytes, between the starts of consecutive lines in the source image.
srcRoiSize	Size of the source ROI in pixels.
pTpl	Pointer to the template image ROI.
tplStep	Distance, in bytes, between the starts of consecutive lines in the template image.
tplRoiSize	Size of the template ROI in pixels.
pDst	Pointer to the destination image ROI.
dstStep	Distance, in bytes, between the starts of consecutive lines in the destination image.

shape	Enumeration that defines the shape of the result of the SAD operation (see Structures and Enumerators).
scaleFactor	Scale factor (see Integer Result Scaling).
pBuffer	Pointer to the buffer for internal calculations.

Description

The ippiSAD function operates with ROI (see Regions of Interest in Intel IPP).

The function only supports the ippiROIValid value of shape, and so the sizes of the destination ROI are different from the sizes of both the source ROI and template ROI and depend on those sizes as follows:

dstRoiSize.width = W = srcRoiSize.width - tplRoiSize.width + 1

dstRoiSize.height = H = srcRoiSize.height - tplRoiSize.height + 1

So there are exactly W * H unique locations within the source image where the template image can fit. The top-left pixel determines each of these locations. For each location, the function computes absolute differences between corresponding pixel values of the source and template images, sums these differences to make the SAD value of the top-left pixel, and assigns the SAD value to the appropriate pixel in the destination buffer.

For integer flavors, the resulting values are also scaled by scaleFactor.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if the pSrc, pTpl, pDst, or pBuffer pointer is NULL.
ippStsSizeErr	Indicates an error condition if srcRoiSize or tplRoiSize has a field with a zero or negative value or if srcRoiSize in any direction is less than tplRoiSize.
ippStsStepErr	Indicates an error condition if srcStep, tplStep, or dstStep has a zero or negative value or is not a multiple of the image data size (4 for floating-point images or 2 for short-integer images)
ippStsBadArgErr	Indicates an error condition if scaleFactor < 0.
ippStsNotSupportedModeErr	Indicates an error condition if intersection of the source and destination ROI is detected or if shape differs from ippiROIValid.

Example

The code example below shows how to use the ippiSAD_8u32s_C1RSfs function.

```
Ipp8u src[8*8] = {1, 2, 3, 4, 8, 8, 8, 8,
1, 2, 3, 4, 8, 8, 8, 8,
1, 2, 3, 4, 8, 8, 8, 8,
1, 2, 3, 4, 8, 8, 8, 8,
```

```
1, 2, 3, 4, 8, 8, 8, 8,
1, 2, 3, 4, 8, 8, 8, 8,
1, 2, 3, 4, 8, 8, 8, 8,
1, 2, 3, 4, 8, 8, 8, 8};


IppiSize srcRoi = { 7, 7 };


Ipp8u template[3*3] = {
10, 10, 10,
10, 10, 10,
10, 10, 10};


IppiSize tplRoi = { 3, 3 };


Ipp32s dst[8*8] = {0, 0, 0, 0, 0, 0, 0, 0,
                   0, 0, 0, 0, 0, 0, 0, 0,
                   0, 0, 0, 0, 0, 0, 0, 0,
                   0, 0, 0, 0, 0, 0, 0, 0,
                   0, 0, 0, 0, 0, 0, 0, 0,
                   0, 0, 0, 0, 0, 0, 0, 0,
                   0, 0, 0, 0, 0, 0, 0, 0,
                   0, 0, 0, 0, 0, 0, 0, 0};


IppiSize srcRoi = { 3, 3 };


ippiSADGetBufferSize( srcRoi,  tplRoi, ippiROIVaid , 1, &bufferSize);
Ipp8u* pBuffer = new Ipp8u [bufferSize];
ippiSAD_8u32s_C1RSfs(src, 8, srcRoi, template, 3, tplRoi, dst, 8, ippiROIVaid , 1, pBuffer);


Result:
1 2 3 4 8 8 8 8        10 10 10        36 32 22 15 9 0  0  0
1 2 3 4 8 8 8 8 src    10 10 10 tpl    36 32 22 15 9 0  0  0 dst
1 2 3 4 8 8 8 8        10 10 10        36 32 22 15 9 0  0  0
1 2 3 4 8 8 8 8                        36 32 22 15 9 0  0  0
1 2 3 4 8 8 8 8                        36 32 22 15 9 0  0  0
1 2 3 4 8 8 8 8                        0  0  0  0  0  0  0  0
1 2 3 4 8 8 8 8                        0  0  0  0  0  0  0  0
1 2 3 4 8 8 8 8                        0  0  0  0  0  0  0  0
```

Image Geometry Transforms

This chapter describes the Intel® IPP image processing functions that perform geometric operations of resizing, rotating, warping and remapping an image.

Most functions performing geometric transform of an image use an interpolation algorithm to resample the image. The type of interpolation method to be used is passed to the function in the `interpolation` parameter for rotate, warp, and remap. For resize transform, the interpolation type is part of the function name.

The following interpolation algorithms are used:

- nearest neighbor
- linear interpolation
- cubic convolution
- supersampling
- interpolation using Lanczos window function
- interpolation with two-parameter cubic filters
- optional edge smoothing of the destination image.

The nearest neighbor algorithm is the fastest, while other methods yield higher quality results, but are slower.

Use one of the following constant identifiers for the applicable interpolation methods:

`IPPI_INTER_NN/ippNearest`	Nearest neighbor interpolation.
`IPPI_INTER_LINEAR/ippLinear`	Linear interpolation.
`IPPI_INTER_CUBIC`	Cubic interpolation.
`IPPI_INTER_LANCZOS/ippLanczos`	Interpolation using 3-lobed Lanczos window function.
`IPPI_INTER_CUBIC2P_BSPLINE`	Interpolation using B-spline.
`IPPI_INTER_CUBIC2P_CATMULLROM`	Interpolation using Catmull-Rom spline.
`IPPI_INTER_CUBIC2P_B05C03`	Interpolation using special cubic filter.
`ippSuper`	Supersampling interpolation.
`ippCubic`	Interpolation with two-parameter cubic filters.

For certain functions, you can combine the above interpolation algorithms with additional smoothing (antialiasing) of edges to which the original image borders are transformed. To use this edge smoothing, set the parameter `interpolation` to the bitwise `OR` of `IPPI_SMOOTH_EDGE` or `IPPI_SUBPIXEL_EDGE` and the desired interpolation mode, or use the special function flags.

NOTE
CAUTION
You can use interpolation with edge smoothing option only in those geometric transform functions where this option is explicitly listed in the parameters definition section.

See appendix B "Interpolation in Image Geometric Transform Functions" for more information on the interpolation algorithms that are used in the library.

Super Sampling mode of resize transform has several limitations. It can be used only:

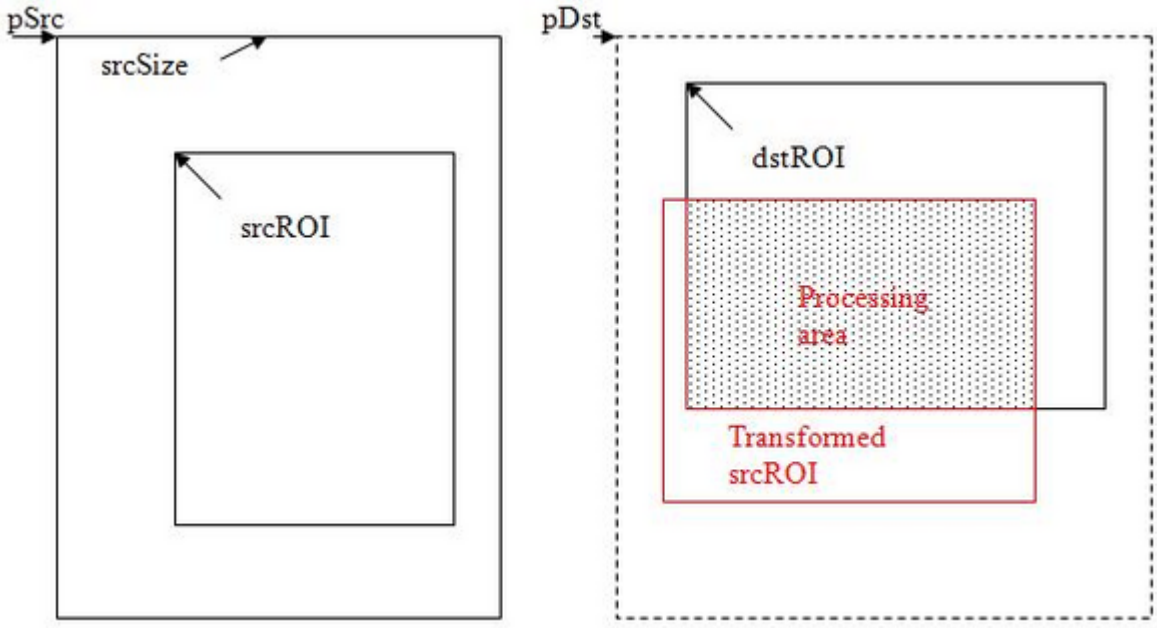
- for reducing image size
- for full images resize, while other interpolation modes can be used for full and tiled images for better speed/memory performance.

- ROI Processing in Geometric Transforms
- Geometric Transform Functions

ROI Processing in Geometric Transforms

All the transform functions described in this chapter operate in rectangular regions of interest (ROIs) that are defined in both the source and destination images. The procedures for handling ROIs in geometric transform functions differ from those used in other functions (see Regions of Interest in Intel IPP). The main difference is that operations take place in the intersection of the *transformed* source ROI and the destination ROI. More specifically, all geometric transform functions (except those which perform inverse warping operations) handle ROIs with the following sequence of operations (see figure below):

- transform the rectangular ROI of the source image to quadrangle in the destination image;
- find the intersection of this quadrangle and the rectangular ROI of the destination image;
- update the destination image in the intersection area.



The coordinates in the source and destination images must have the same origin.

When using functions with ROI, every scan line of a source image has a stride. It is padded with zeroes for alignment, so the actual size of a scan line in bytes is often greater than the image width. The size of each row of image data in bytes is determined by the value of `srcStep` parameter, which gives distance in bytes between the starts of consecutive lines of an image.

To fully describe a rectangular ROI, both its origin (coordinates of top left corner) and size must be referenced. For geometrical transform functions, the source image ROI is specified by `srcRoi` parameter of `IppiRect` type, meaning that all four values describing the rectangular ROI are given explicitly.

The destination image ROI for different functions can be specified either by `dstRoi` parameter of `IppiRect` type, or `dstRoiSize` parameter of `IppiSize` type. In the latter case, only the destination ROI size is passed, while its origin is referenced by `pDst` pointer.

The destination image origin ROI for different functions can be specified by `dstOffset` parameter of `IppiPoint` type and `dstSize` parameter of `IppiSize` type. In this case, the processed destination image coressponds to the processed ROI of the destination image origin.

Geometric Transform Functions

- ResizeYCbCr422GetBufSize
- ResizeYCbCr422
- Resize Functions with Prior Initialization
- Warp Functions with Prior Initialization
- GetBilinearQuad
- GetBilinearBound
- GetBilinearTransform
- WarpBilinearGetBufferSize
- WarpBilinear
- WarpBilinearBack

- WarpBilinearQuadGetBufferSize
- WarpBilinearQuad
- Mirror
- Remap

ResizeYCbCr422GetBufSize

Computes the size of the external buffer for the NV12 resize transform.

Syntax

```
IppStatus ippiResizeYCbCr422GetBufSize(IppiRect srcROI, IppiSize dstRoiSize, int interpolation, int* pSize);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`srcROI`	Region of interest of the source image.
`dstRoiSize`	Region of interest of the destination image.
`interpolation`	Type of interpolation to apply to the source image: • `IPP_INTER_NN` • Nearest neighbor interpolation • `IPP_INTER_LINEAR` • Linear interpolation • `IPP_INTER_CUBIC` • Cubic interpolation • `IPPI_INTER_CUBIC2P_CATMULLROM` • Catmull-Rom cubic filter • `IPP_INTER_LANCZOS` • Lanczos filter with size 6x6
`pSize`	Pointer to the size, in bytes, of the external buffer.

Description

This function computes the size of the external buffer for the YCbCr resize transform.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if `pSize` is `NULL`.

ippStsSizeErr	Indicates an error in the following cases: • width of either source or destination ROI is less than 2 • height of either source or destination ROI is less than 1
ippStsDoubleSize	Indicates a warning if width of wither source or destination ROI is not a multiple of 2.
ippStsInterpolationErr	Indicates an error if `interpolation` has an illegal value.

ResizeYCbCr422

Performs resizing of a 4:2:2 two-channel image.

Syntax

```
IppStatus ippiResizeYCbCr422_8u_C2R(const Ipp8u* pSrc, IppiSize srcSize, int srcStep,
IppiRect srcROI, Ipp8u* pDst, int dstStep, IppiSize dstRoiSize, int interpolation,
Ipp8u* pBuffer);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

pSrc	Pointer to the source image.
srcSize	Size, in pixels, of the source image.
srcStep	Distance, in bytes, between the starting points of consecutive lines in the source image buffer.
srcROI	Region of interest in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance, in bytes, between the starting points of consecutive lines in the destination image buffer.
dstRoiSize	Size of the destination ROI in pixels.
interpolation	Type of interpolation to apply to the source image: • `IPP_INTER_NN` • Nearest neighbor interpolation • `IPP_INTER_LINEAR` • Linear interpolation

	• IPP_INTER_CUBIC • Cubic interpolation • IPPI_INTER_CUBIC2P_CATMULLROM • Catmull-Rom cubic filter • IPP_INTER_LANCZOS • Lanczos filter with size 6x6
pBuffer	Pointer to the external buffer.

Description

This function operates with ROI (see ROI Processing in Geometric Transforms).

This function resizes the sourceimage ROI to the destination image ROI origin. The function performs resampling of the result using the interpolation mode specified by the interpolation parameter, and stores it in thedestination image ROI.

The source image is a two-channelimage in the 4:2:2 sampling format in color spaces with decoupledluminance and chrominance components, for example, YUV422 orYCrCb422.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if pSrc, pDst, or pBuffer is NULL.
ippStsSizeErr	Indicates an error in the following cases: • width of either source or destination ROI is less than 2 • height of either source or destination ROI is less than 1
ippStsDoubleSize	Indicates a warning if width of wither source or destination ROI is not a multiple of 2.
ippStsInterpolationErr	Indicates an error if interpolation has an illegal value.
ippStsWrongIntersectROI	Indicates a warning if srcROI does not intersect with the source image; no operation is required.

Resize Functions with Prior Initialization

This section describes the Intel® IPP resize functions that use the specification structure in operation. Before using these functions, you need to initialize the structure.

- Using Intel® IPP Resize Functions with Prior Initialization
- ResizeGetSize
- ResizeGetBufferSize
- ResizeGetBorderSize
- ResizeGetSrcOffset
- ResizeGetSrcRoi
- ResizeSetMode
- ResizeNearestInit
- ResizeNearest

- ResizeLinearInit
- ResizeLinear
- ResizeCubicInit
- ResizeCubic
- ResizeLanczosInit
- ResizeLanczos
- ResizeSuperShiftInit
- ResizeSuperInit
- ResizeSuper
- ResizeAntialiasingLinearInit
- ResizeAntialiasingCubicInit
- ResizeAntialiasingLanczosInit
- ResizeAntialiasing
- ResizeFilterGetSize
- ResizeFilterInit
- ResizeFilter
- ResizeYUV420GetSize
- ResizeYUV420GetSrcRoi
- ResizeYUV420LanczosInit
- ResizeYUV420SuperInit
- ResizeYUV420GetBorderSize
- ResizeYUV420GetSrcOffset
- ResizeYUV420GetBufferSize
- ResizeYUV420Lanczos
- ResizeYUV420Super
- ResizeYUV422GetSize
- ResizeYUV422GetBorderSize
- ResizeYUV422GetSrcOffset
- ResizeYUV422GetBufSize
- ResizeYUV422GetSrcRoi
- ResizeYUV422NearestInit
- ResizeYUV422LinearInit
- ResizeYUV422Nearest
- ResizeYUV422Linear

Using Intel® IPP Resize Functions with Prior Initialization

You can use one of the following approaches to image resizing:

- Resizing the whole image
- Resizing a tiled image with one prior initialization
- Resizing a tiled image with prior initialization for each tile

Interpolation algorithms of the Lanczos, Linear, and Cubic types use edge pixels of the source image that are out of the image origin. When calling the `ippiResize<Filter>` function with one of these interpolation algorithms applied, you need to specify the appropriate border type. The following border types are supported:

- Replicated borders: border pixels are replicated from the source image boundary pixels;
- Borders in memory: the source image border pixels are obtained from the source image pixels in memory;
- Mixed borders: a combined approach is applied.

NOTE If you want to resize an image with antialiasing, follow the same instructions as provided below, but use `ippiResizeAntialiasing<Filter>Init` instead of `ippiResize<Filter>Init` for initialization, and ippiResizeAntialiasing<Filter> instead of `ippiResize<Filter>`, as a processing function.

Resizing the Whole Image

You can apply the approach described below to resize when source and destination images are fully accessible in memory. However, this method only runs on a single thread.

To resize the whole image:

1. Call the `ippiResizeGetSize` function with the appropriate interpolation type. This function uses source and destination image sizes to calculate how much memory must be allocated for the `IppResizeSpec` structure and initialization work buffer.
2. Initialize the `IppResizeSpec` structure by calling the `ippiResize<Filter>Init`, where < Filter``>`` can take one of the following values: `Nearest`, `Linear`, `Cubic`, `Lanczos`, and `Super`. These prerequisite steps allow resize to be called multiple times without recalculations.
3. Call the `ippiResizeGetBufferSize` function for the initialized `IppResizeSpec` structure. This function uses the destination image size to calculate how much memory must be allocated for the resize work buffer.
4. Call `ippiResize<Filter>` with the appropriate image type.
5. If you call the `ippiResize<Filter>` function with a `ippBorderInMem` border or any mixed border type, the applied interpolation algorithm uses weighted values from edge pixels of the source image when outside the image boundaries. To obtain the size of the border required for correct edge calculation, call the `ippiResizeGetBorderSize` function for the appropriate flavor. In case of mixed border type, out of image pixels are used only behind the non-replicated edge.
6. You can use mixed borders by using the bitwise `OR` operation between the `ippBorderRepl` type and the following border types: `ippBorderInMemTop`, `ippBorderInMemBottom`, `ippBorderInMemLeft`, `ippBorderInMemRight`.

Figure *Simple Image Resize* shows a simple image resizing example, in which image resolution is increased by 1.5x.

Simple Image Resize



Example

The code example below demonstrates whole image resizing with the Lanczos interpolation method:

```
IppStatus resizeExample_C3R(Ipp8u* pSrc, IppiSize srcSize, Ipp32s srcStep, Ipp8u* pDst, IppiSize
dstSize, Ipp32s dstStep)
{
    IppiResizeSpec_32f* pSpec = 0;
    int specSize = 0, initSize = 0, bufSize = 0;
    Ipp8u* pBuffer  = 0;
    Ipp8u* pInitBuf = 0;
    Ipp32u numChannels = 3;
    IppiPoint dstOffset = {0, 0};
    IppStatus status = ippStsNoErr;
    IppiBorderType border = ippBorderRepl;


    /* Spec and init buffer sizes */
    status = ippiResizeGetSize_8u(srcSize, dstSize, ippLanczos, 0, &specSize, &initSize);


    if (status != ippStsNoErr) return status;


    /* Memory allocation */
    pInitBuf = ippsMalloc_8u(initSize);
    pSpec    = (IppiResizeSpec_32f*)ippsMalloc_8u(specSize);


    if (pInitBuf == NULL || pSpec == NULL)
    {
        ippsFree(pInitBuf);
        ippsFree(pSpec);
        return ippStsNoMemErr;
    }


    /* Filter initialization */
    status = ippiResizeLanczosInit_8u(srcSize, dstSize, 3, pSpec, pInitBuf);
    ippsFree(pInitBuf);


    if (status != ippStsNoErr)
    {
        ippsFree(pSpec);
        return status;
    }


    /* work buffer size */
    status = ippiResizeGetBufferSize_8u(pSpec, dstSize, numChannels, &bufSize);
    if (status != ippStsNoErr)
    {
        ippsFree(pSpec);
        return status;
    }


    pBuffer = ippsMalloc_8u(bufSize);
    if (pBuffer == NULL)
    {
```

```
        ippsFree(pSpec);
        return ippStsNoMemErr;
    }


    /* Resize processing */
    status = ippiResizeLanczos_8u_C3R(pSrc, srcStep, pDst, dstStep, dstOffset, dstSize, border,
0, pSpec, pBuffer);


    ippsFree(pSpec);
    ippsFree(pBuffer);


    return status;
}
```

Resizing a Tiled Image with One Prior Initialization

You can apply the approach described below to resize when source and destination images are not fully accessible in memory, or to improve the performance of resizing by external threading.

The main difference between this approach and whole image resizing is that the processing is split into sections of the image called tiles. Each call of the `Resize<Filter>` function works with the destination image origin region of interest (ROI) that is defined by `dstOffset` and `dstSize` parameters. The destination and source ROI must be fully accessible in memory.

To resize an image with the tiled approach:

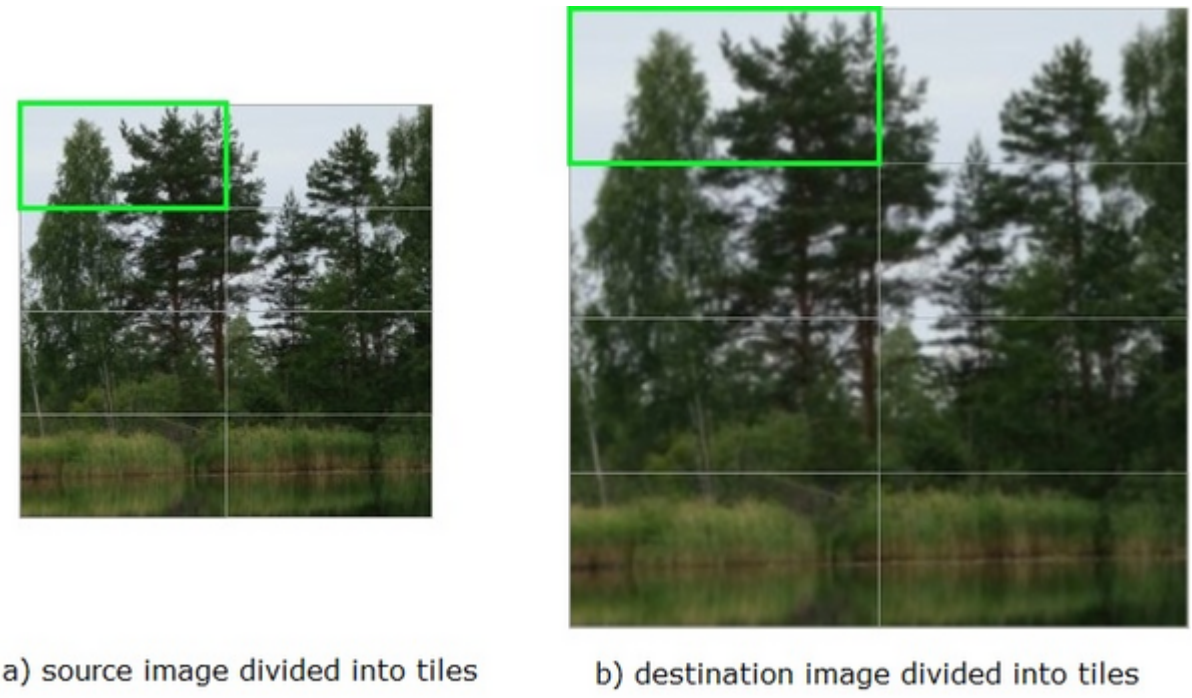
1. Call the `ippiResizeGetSize` function with the appropriate interpolation type. This function uses the source and destination image sizes to calculate how much memory must be allocated for the `IppResizeSpec` structure and initialization work buffer.
2. Initialize the `IppResizeSpec` structure by calling `ippiResize<Filter>Init`, where < Filter``>`` can take one of the following values: `Nearest`, `Cubic`, `Linear`, and `Lanczos`.
3. Determine an appropriate partitioning scheme to divide the destination image into tiles. Tiles can be sets of rows or a regular grid of subimages. A simple vertical subdivision into sets of lines is often sufficient.
4. Obtain the source ROI for the defined destination tile by calling the `ippiResizeGetSrcRoi` function for the corresponding flavor. The algorithm uses edge pixels that are out of the source ROI to calculate edge pixels of the destination ROI. These out of the source ROI edge pixels must be accessible in memory.
5. If the source ROI is an interior field of the source image origin, obtain the border ROI size by calling the `ippiResizeGetBorderSize` function for the corresponding flavor.
6. If the source ROI is an edge tile, the algorithm can interpolate pixels beyond the image boundary as in the previous method.
7. If the source and destination images are fully accessible in memory, you can use the source ROI offset for the `pSrc` calculation. To obtain the offset, call the `ippiResizeGetSrcOffset` function for the corresponding flavor.
8. Call the `ippiResizeGetBufferSize` function to obtain the size of the resize work buffer required for each tile processing. The `dstSize` parameter must be equal to the tile size.
9. Call ippiResize<Filter> for each tile (ROI). The `dstOffset` parameter must specify the image ROI offset with respect to the destination image origin. The `dstSize` parameter must be equal to the ROI size. Parameters `pSrc` and `pDst` must point to the beginning of the source and destination ROI in memory respectively. The source and destination ROIs must be fully accessible in memory.

You can process tiles in any order. When using multitple threads you can process all tiles simultaneously.

NOTE If you resize a tiled image with the Super Sampling algorithm, and the source image width to destination image width ratio is `m/n`, you can reach better performance of resize operation if all destination tiles have width that is a multiple of `n`.

Figure *Tiling Image Resize* shows the resize of the image divided into tiles.

Tiling Image Resize



Example

The code example below demonstrates a multithreading resize operation using OpenMP* with parallelization only in the y direction:

```
#define MAX_NUM_THREADS 16


IppStatus tileResizeExample_C3R(Ipp8u* pSrc, IppiSize srcSize, Ipp32s srcStep, Ipp8u* pDst,
IppiSize dstSize, Ipp32s dstStep)
{
    IppiResizeSpec_32f* pSpec = 0;
    int specSize = 0, initSize = 0, bufSize = 0;
    Ipp8u* pBuffer  = 0;
    Ipp8u* pInitBuf = 0;
    Ipp32u numChannels = 3;
    IppiPoint dstOffset = {0, 0};
    IppiPoint srcOffset = {0, 0};
    IppStatus status = ippStsNoErr;
    IppiBorderSize borderSize = {0, 0, 0, 0};
    IppiBorderType border = ippBorderRepl;
    int numThreads, slice, tail;
```

```
int bufSize1, bufSize2;
IppiSize dstTileSize, dstLastTileSize;
IppStatus pStatus[MAX_NUM_THREADS];



/* Spec and init buffer sizes */
status = ippiResizeGetSize_8u(srcSize, dstSize, ippLinear, 0, &specSize, &initSize);


if (status != ippStsNoErr) return status;


/* Memory allocation */
pInitBuf = ippsMalloc_8u(initSize);
pSpec    = (IppiResizeSpec_32f*)ippsMalloc_8u(specSize);


if (pInitBuf == NULL || pSpec == NULL)
{
    ippsFree(pInitBuf);
    ippsFree(pSpec);
    return ippStsNoMemErr;
}


/* Filter initialization */
status = ippiResizeLinearInit_8u(srcSize, dstSize, pSpec);
ippsFree(pInitBuf);


if (status != ippStsNoErr)
{
    ippsFree(pSpec);
    return status;
}


status = ippiResizeGetBorderSize_8u(pSpec, &borderSize);
if (status != ippStsNoErr)
{
    ippsFree(pSpec);
    return status;
}


/* General transform function */
/* Parallelized only by Y-direction here */
#pragma omp parallel num_threads(MAX_NUM_THREADS)
{
    #pragma omp master
    {
        numThreads = omp_get_num_threads();
        slice = dstSize.height / numThreads;
        tail  = dstSize.height % numThreads;
```

```
            dstTileSize.width = dstLastTileSize.width = dstSize.width;
            dstTileSize.height = slice;
            dstLastTileSize.height = slice + tail;


            ippiResizeGetBufferSize_8u(pSpec, dstTileSize, ippC3, &bufSize1);
            ippiResizeGetBufferSize_8u(pSpec, dstLastTileSize, ippC3, &bufSize2);


            pBuffer = ippsMalloc_8u(bufSize1 * (numThreads - 1) + bufSize2);
        }


        #pragma omp barrier
        {
            if (pBuffer)
            {
                Ipp32u  i;
                Ipp8u  *pSrcT, *pDstT;
                Ipp8u  *pOneBuf;
                IppiPoint srcOffset = {0, 0};
                IppiPoint dstOffset = {0, 0};
                IppiSize  srcSizeT = srcSize;
                IppiSize  dstSizeT = dstTileSize;


                i = omp_get_thread_num();
                dstSizeT.height = slice;
                dstOffset.y += i * slice;


                if (i == numThreads - 1) dstSizeT = dstLastTileSize;


                pStatus[i] = ippiResizeGetSrcRoi_8u(pSpec, dstOffset, dstSizeT, &srcOffset,
&srcSizeT);


                if (pStatus[i] == ippStsNoErr)
                {
                    pSrcT = (Ipp8u*)((char*)pSrc + srcOffset.y * srcStep);
                    pDstT = (Ipp8u*)((char*)pDst + dstOffset.y * dstStep);


                    pOneBuf = pBuffer + i * bufSize1;


                    pStatus[i] = ippiResizeLinear_8u_C3R (pSrcT, srcStep, pDstT, dstStep,
dstOffset, dstSizeT, border, 0, pSpec, pOneBuf);
                }
            }
        }
    }


    ippsFree(pSpec);
```

```
    if (pBuffer == NULL) return ippStsNoMemErr;


    ippsFree(pBuffer);


    for (Ipp32u i = 0; i < numThreads; ++i)
    {
        /* Return bad status */
        if(pStatus[i] != ippStsNoErr) return pStatus[i];
    }


    return status;
}
```

Resizing a Tiled Image with Prior Initialization for Each Tile

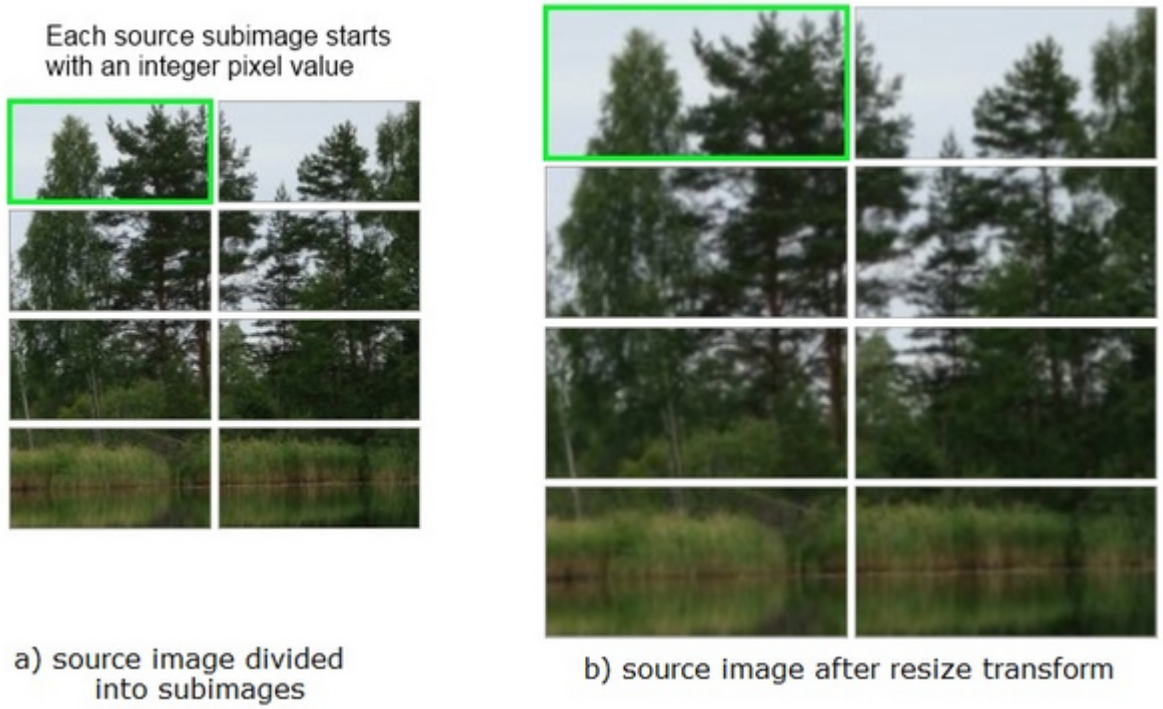
You can apply this approach only in cases when the destination image can be divided into tiles so that each destination tile corresponds to a source image tile that starts with an integer pixel value origin. For example, if the ratio of the source and destination images sizes is 2/3, the destination image can be divided into 3x3 tiles, each of which corresponds to the source image tile 2x2.

This approach is useful if there are restrictions on memory size when processing an image, or if the image size is large and `ippiResizeGetBufferSize` function returns `ippStsSizeErr` error. The initialization data for a tile is less than the same data for the whole image.

Each tile of the source image can be considered as an independent image that can be resized. For interior tile processing, the border must be always of the `ippBorderInMem` type. If you need to replicate any borders of the source image origin, you should combine the border type of the outer tiles so that interior tiles edges have border in memory and external tile borders are of the specified border type. This approach enables the right linking order of tiles.

Figure *Resize of the Image Divided into Subimages* shows the approach, when the source image is divided into several subimages that are resized independently.

Resize of the Image Divided into Subimages



Example

The code example below divides the source image into tiles and resizes each image independently:

```
IppStatus separateTileResizeExample_C3R(Ipp8u* pSrc, IppiSize srcTileSize, Ipp32s srcStep,
Ipp8u* pDst, IppiSize dstTileSize, Ipp32s dstStep, Ipp32s xNumTiles, Ipp32s yNumTiles)
{
    IppiResizeSpec_32f* pSpec = 0;
    int specSize = 0, initSize = 0, bufSize = 0;
    Ipp8u* pBuffer  = 0;
    Ipp8u* pInitBuf = 0;
    Ipp32u numChannels = 3;
    IppStatus status = ippStsNoErr;


    /* tiles cicle */
    for (Ipp32s j = 0; j < xNumTiles; j ++)
    {
        for (Ipp32s i = 0; i < yNumTiles; i ++)
        {
            /* calculation of the destination image ROI offset */
            IppiPoint dstOffset = {j * dstTileSize.width, i * dstTileSize.height};
            Ipp8u* pDstT = pDst + dstStep * dstOffset.y + dstOffset.x * numChannels *
sizeof(Ipp8u);


            /* calculation of the source image ROI offset */
            IppiPoint srcOffset = {j * srcTileSize.width, i * srcTileSize.height};
            Ipp8u* pSrcT = pSrc + srcStep * srcOffset.y + srcOffset.x * numChannels *
sizeof(Ipp8u);


            IppiBorderType borderT = ippBorderRepl;
```

```
            IppiPoint dstOffsetZero = {0, 0};


            /* correction of the border type for the tile processing */
            if (j > 0) /* the processed tile is not on the left image origin edge*/
            {
                borderT = (IppiBorderType)((int)borderT | (int)ippBorderInMemLeft);
            }
            if (j < xNumTiles - 1) /* the processed tile is not on the right image origin edge*/
            {
                borderT = (IppiBorderType)((int)borderT | (int)ippBorderInMemRight);
            }
            if (i > 0) /* the processed tile is not on the top image origin edge*/
            {
                borderT = (IppiBorderType)((int)borderT | (int)ippBorderInMemTop);
            }
            if (i < yNumTiles - 1) /* the processed tile is not on the bottom image origin edge*/
            {
                borderT = (IppiBorderType)((int)borderT | (int)ippBorderInMemBottom);
            }


            /* Spec and init buffer sizes */
            status = ippiResizeGetSize_8u(srcTileSize, dstTileSize, ippLanczos, 0, &specSize,
&initSize);


            if (status != ippStsNoErr) return status;


            /* Memory allocation */
            pInitBuf = ippsMalloc_8u(initSize);
            pSpec    = (IppiResizeSpec_32f*)ippsMalloc_8u(specSize);


            if (pInitBuf == NULL || pSpec == NULL)
            {
                ippsFree(pInitBuf);
                ippsFree(pSpec);
                return ippStsNoMemErr;
            }


            /* Filter initialization */
            status = ippiResizeLanczosInit_8u(srcTileSize, dstTileSize, 3, pSpec, pInitBuf);
            ippsFree(pInitBuf);


            if (status != ippStsNoErr)
            {
                ippsFree(pSpec);
                return status;
            }


            /* work buffer size */
```

```
            status = ippiResizeGetBufferSize_8u(pSpec, dstTileSize, numChannels, &bufSize);
            if (status != ippStsNoErr)
            {
                ippsFree(pSpec);
                return status;
            }


            pBuffer = ippsMalloc_8u(bufSize);
            if (pBuffer == NULL)
            {
                ippsFree(pSpec);
                return ippStsNoMemErr;
            }


            /* Resize processing */
            status = ippiResizeLanczos_8u_C3R(pSrcT, srcStep, pDstT, dstStep, dstOffsetZero,
dstTileSize, borderT, 0, pSpec, pBuffer);


            ippsFree(pSpec);
            ippsFree(pBuffer);


            if (status != ippStsNoErr) return status;
        }
    }


    return ippStsNoErr;
}
```

See Also

User-defined Border Types

ResizeGetSize

Computes the size of the specification structure and the size of the external temporary buffer for the resize transform initialization.

Syntax

Case 1: Processing images of 32-bit sizes

```
IppStatus ippiResizeGetSize_<mod>(IppiSize srcSize, IppiSize dstSize,
IppiInterpolationType interpolation, Ipp32u antialiasing, int* pSpecSize, int*
pInitBufSize);
```

Supported values for `mod`:

`8u`	`16u`	`32f`	`64f`

```
IppStatus ippiResizeGetSize_16s(IppiSize srcSize, IppiSize dstSize,
IppiInterpolationType interpolation, Ipp32u antialiasing, int* pSpecSize, Ipp32s*
pInitBufSize);
```

Case 2: Processing images with platform-aware functions

```
IppStatus ippiResizeGetSize_L(IppiSizeL srcSize, IppiSizeL dstSize, IppDataType
dataType, IppiInterpolationType interpolation, Ipp32u antialiasing, IppSizeL*
pSpecSize, IppSizeL* pInitBufSize);
```

Case 3: Processing images with threading layer (TL) functions

```
IppStatus ippiResizeGetSize_LT(IppiSizeL srcSize, IppiSizeL dstSize, IppDataType
dataType, IppiInterpolationType interpolation, Ipp32u antialiasing, IppSizeL*
pSpecSize, IppSizeL* pInitBufSize);
```

Include Files

`ippi.h`

Flavors with the `_LT` suffix: `ippi_tl.h`

Flavors with the `_L` suffix: `ippi_l.h`

Domain Dependencies

Flavors declared in `ippi.h`:

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Flavors declared in `ippi_tl.h`:

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`, `ippcore_tl.lib`, `ippi_tl.lib`

Parameters

`srcSize`	Size, in pixels, of the source image.
`dstSize`	Size, in pixels, of the destination image.
`interpolation`	Interpolation method. Supported values: `ippNearest`, `ippLinear`, `ippCubic`, `ippLanczos`, and `ippSuper`.
`antialiasing`	Supported values: 1- resizing with antialiasing, 0 - resizing without antialiasing.
`dataType`	Data type of the image. Supported values: `ipp8u`, `ipp16u`, `ipp16s`, `ipp32f`, `ipp64f` (only for linear interpolation).
`pSpecSize`	Pointer to the size, in bytes, of the specification structure.
`pInitBufSize`	Pointer to the size, in bytes, of the temporary buffer required for initialization of the specification structure.

Description

This function computes the size of the specification structure and the size of the external buffer for the following functions depending on the `interpolation` parameter value:

- ippiResizeAntialiasingCubicInit, ippiResizeAntialiasingLanczosInit, or ippiResizeAntialiasingLinearInit for resizing with antialiasing

- ippiResizeNearestInit, ippiResizeLinearInit, ippiResizeCubicInit, ippiResizeLanczosInit, or ippiResizeSuperInit for resizing without antialiasing

Interpolation algorithms have the following filter sizes:

Nearest Neighbor	1x1
Linear	2x2
Cubic	4x4
2-lobed Lanczos	4x4

NOTE The `ippiResizeGetSize` function always returns non-zero value for the `pInitBufSize` parameter, even if the temporary buffer is not required for the specification structure initialization. The temporary buffer is only required when initializing the specification structure for the following functions: `ippiResizeAntialiasingCubic`, `ippiResizeAntialiasingLanczos`, `ippiResizeAntialiasingLanczos`, `ippiResizeLanczos`, and `ippiResizeCubic`.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if one of the specified pointers is `NULL`.
`ippStsResizeNoOperation`	Indicates an error if width or height of the image is equal to zero.
`ippStsSizeErr`	Indicates an error in the following cases: • If the source image size is less than the filter size of the chosen interpolation method (except `ippSuper`). • If one of the specified dimensions of the source image is less than the corresponding dimension of the destination image (for `ippSuper` method only). • If the width or height of the source or destination image is negative.
`ippStsExceededSizeErr`	Indicates an error in the following cases: • If the source image size is less than the filter size of the chosen interpolation method (except `ippSuper`). • If one of the specified dimensions of the source image is less than the corresponding dimension of the destination image (for `ippSuper` method only).

	• If the width or height of the source or destination image is negative.
ippStsInterpolationErr	Indicates an error if interpolation has an illegal value.
ippStsNoAntialising	Indicates a warning if the specified interpolation method does not support antialiasing.
ippStsNotSupportedModeErr	Indicates an error if the requested mode is not supported.
ippStsDataTypeErr	Indicates an error if dataType has an illegal value.

ResizeGetBufferSize

Computes the size of the external buffer for image resizing.

Syntax

Case 1: Single precision

```
IppStatus ippiResizeGetBufferSize_<mod>(const IppiResizeSpec_32f* pSpec, IppiSize dstSize, Ipp32u numChannels, int* pBufSize);
```

Supported values for mod:

8u	16u	16s	32f

Case 2: Double precision

```
IppStatus ippiResizeGetBufferSize_64f(const IppiResizeSpec_64f* pSpec, IppiSize dstSize, Ipp32u numChannels, int* pBufSize);
```

Case 3: Processing images with platform-aware functions

```
IppStatus ippiResizeGetBufferSize_L(const IppiResizeSpec* pSpec, IppiSizeL dstSize, Ipp32u numChannels, IppSizeL* pBufSize);
```

Case 4: Processing images with threading layer (TL) functions

```
IppStatus ippiResizeGetBufferSize_LT(const IppiResizeSpec_LT* pSpec, IppSizeL* pBufSize);
```

Include Files

ippi.h

Flavors with the _LT suffix: ippi_tl.h

Flavors with the _L suffix: ippi_l.h

Domain Dependencies

Flavors declared in ippi.h:

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Flavors declared in `ippi_tl.h`:

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`, `ippcore_tl.lib`, `ippi_tl.lib`

Parameters

`pSpec`	Pointer to the spec structure for the resize filter.
`dstSize`	Size in pixels of the destination image.
`numChannels`	Number of channels, possible values: 1, 3, or 4.
`pBufSize`	Pointer to the size, in bytes, of the external buffer.

Description

This function computes the size of the external buffer for image resizing. The `pSpec` parameter defines the resize algorithm parameters. Prior to using the `ippiResizeGetBufferSize` function, you need to initialize the `pSpec` parameter by calling one of the following functions: ippiResizeNearestInit, ippiResizeLinearInit, ippiResizeCubicInit, ippiResizeLanczosInit, and ippiResizeSuperInit.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if `pBufferSize` pointer is `NULL`.
`ippStsNoOperation`	Indicates a warning if width or height of the destination image is equal to zero.
`ippStsContextMatchErr`	Indicates an error if pointer to the spec structure is invalid.
`ippStsNumChannelErr`	Indicates an error if the value of `numChannels` is illegal.
`ippStsSizeErr`	Indicates an error condition If width or height of the destination image is negative.
`ippStsSizeWrn`	Indicates a warning if the destination image size is more than the destination image origin size.
`ippStsExceededSizeErr`	Indicates an error If at least one of the computed values exceeds maximum of `IppSizeL` type positive value (the size of one of the processed images is too large).

ResizeGetBorderSize

Computes the size of possible borders for the resize transform.

Syntax

Case 1: Interpolation with single precision

```
IppStatus ippiResizeGetBorderSize_<mod>(const IppiResizeSpec_32f* pSpec,
IppiBorderSize* borderSize);
```

Supported values for `mod`:

`8u`	`16u`	`16s`	`32f`

Case 2: Interpolation with double precision

```
IppStatus ippiResizeGetBorderSize_64f(const IppiResizeSpec_64f* pSpec, IppiBorderSize* borderSize);
```

Case 3: Processing images with platform-aware functions

```
IppStatus ippiResizeGetBorderSize_L(const IppiResizeSpec* pSpec, IppiBorderSize* borderSize);
```

Case 4: Processing images with threading layer (TL) functions

```
IppStatus ippiResizeGetBorderSize_LT(const IppiResizeSpec_LT* pSpec, IppiBorderSize* borderSize);
```

Include Files

`ippi.h`

Flavors with the `_LT` suffix: `ippi_tl.h`

Flavors with the `_L` suffix: `ippi_l.h`

Domain Dependencies

Flavors declared in `ippi.h`:

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Flavors declared in `ippi_tl.h`:

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`, `ippcore_tl.lib`, `ippi_tl.lib`

Parameters

`pSpec`	Pointer to the spec structure for the resize filter.
`borderSize`	Size in pixels of necessary borders.

Description

This function computes the size of the source image ROI that is used by the corresponding resize transform and is out of the processing boundaries. The `pSpec` parameter defines the resize algorithm parameters. Prior to using the `ippiResizeGetBorderSize` function, you need to initialize the `pSpec` parameter by calling one of the following functions: ippiResizeNearestInit, ippiResizeLinearInit, ippiResizeCubicInit, ippiResizeLanczosInit, and ippiResizeSuperInit.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if one of the specified pointers is `NULL`.

ippStsContextMatchErr	Indicates an error if pointer to the spec structure is invalid.
ippStsBorderErr	Indicates an error if border has an illegal value.

ResizeGetSrcOffset

Computes the offset of the source image for resizing by tile processing.

Syntax

Single precision

```
IppStatus ippiResizeGetSrcOffset_<mod>(const IppiResizeSpec_32f* pSpec, IppiPoint dstOffset, IppiPoint* srcOffset);
```

Supported values for mod:

8u	16u	16s	32f

Double precision

```
IppStatus ippiResizeGetSrcOffset_64f(const IppiResizeSpec_64f* pSpec, IppiPoint dstOffset, IppiPoint* srcOffset);
```

Processing images with platform-aware functions

```
IppStatus ippiResizeGetSrcOffset_L(const IppiResizeSpec* pSpec, IppiPointL dstOffset, IppiPointL* srcOffset);
```

Include Files

ippi.h

Flavors with the _L suffix: ippi_l.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSpec	Pointer to the spec structure for the resize filter.
dstOffset	Offset of the tiled destination image with respect to the destination image origin.
srcOffset	Offset of the source image.

Description

This function computes the offset of the processed source image ROI using the offset of the processed destination image ROI for the corresponding resize transform by tile processing. The pSpec parameter defines the resize algorithm parameters. Prior to using the ippiResizeGetSrcOffset function, you need to initialize the pSpec parameter by calling one of the following functions: ippiResizeNearestInit, ippiResizeLinearInit, ippiResizeCubicInit, ippiResizeLanczosInit, or ippiResizeSuperInit.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if one of the specified pointers is NULL.
ippStsContextMatchErr	Indicates an error if pointer to the spec structure is invalid.
ippStsOutOfRangeErr	Indicates an error if the destination image offset point is outside the destination image origin.

ResizeGetSrcRoi

Computes the ROI of the source image for resizing by tile processing.

Syntax

Single precision

```
IppStatus ippiResizeGetSrcRoi_<mod>(const IppiResizeSpec_32f* pSpec, IppiPoint
dstRoiOffset, IppiSize dstRoiSize, IppiPoint* srcRoiOffset, IppiSize* srcRoiSize);
```

Supported values for mod:

8u	16u	16s	32f

Double precision

```
IppStatus ippiResizeGetSrcRoi_64f(const IppiResizeSpec_64f* pSpec, IppiPoint
dstRoiOffset, IppiSize dstRoiSize, IppiPoint* srcRoiOffset, IppiSize* srcRoiSize);
```

Processing images with platform-aware functions

```
IppStatus ippiResizeGetSrcRoi_L(const IppiResizeSpec* pSpec, IppiPointL dstRoiOffset,
IppiSizeL dstRoiSize, IppiPointL* srcRoiOffset, IppiSizeL* srcRoiSize);
```

Include Files

ippi.h

Flavors with the _L suffix: ippi_l.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSpec	Pointer to the spec structure for the resize filter.
dstRoiOffset	Offset of the tiled destination image ROI.
dstRoiSize	Size of the tiled destination image ROI.
srcRoiOffset	Offset of the source image ROI.

srcRoiSize	Pointer to the size of the source image ROI.

Description

This function computes the ROI of the processed source image using the processed ROI of the destination image for the corresponding resize transform by tile processing. The pSpec parameter defines the resize algorithm parameters. Prior to using the ippiResizeGetSrcRoi function, you need to initialize the pSpec parameter by calling one of the following functions: ippiResizeNearestInit, ippiResizeLinearInit, ippiResizeCubicInit, ippiResizeLanczosInit, or ippiResizeSuperInit.

NOTE If the destination ROI size exceeds the image origin, the source ROI will be obtained for an intersection of the destination ROI and image origin.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if one of the specified pointers is NULL.
ippStsContextMatchErr	Indicates an error if pointer to the spec structure is invalid.
ippStsOutOfRangeErr	Indicates an error if the destination image offset point is outside the destination image origin.
ippStsSizeWrn	Indicates a warning if the destination ROI exceeds the destination image origin.

ResizeSetMode

Sets the rounding mode for resize functions

Syntax

```
IppStatus ippiResizeSetMode(IppHintAlgorithm hint, IppiResizeSpec* pSpec
```

Include Files

ippi.h

Parameters

hint	Rounding mode for processing Ipp8u data. Possible values are: • ippAlgHintFast • Fast rounding mode (default). • ippAlgHintAccurate • Accurate rounding mode.
pSpec	Pointer to the specification structure for the resize filter.

Description

This function sets the `roundMode` for the resize algorithm.

If you provide the `hint` parameter with the `ippAlgHintFast` value, a faster but less accurate mode will be used. In this case, output pixel values can differ from the exact result by 1. If you choose `ippAlgHintAccurate`, a more accurate but slower mode will be used and all output pixel values will be exact.

Before using this function, initialize the specification structure using the initialization function for a required interpolation method.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when one of the specified pointers is `NULL`.
`ippStsAccurateModeNotSupported`	Indicates an error when the rounding mode is not supported for the selected data type. The rounding result can be inexact.

ResizeNearestInit

Initializes the specification structure for image resizing with the nearest neighbor interpolation method.

Syntax

```
IppStatus ippiResizeNearestInit_<mod>(IppiSize srcSize, IppiSize dstSize,
IppiResizeSpec_32f* pSpec);
```

Supported values for `mod`:

`8u`	`16u`	`16s`	`32f`

Platform-aware function

```
IppStatus ippiResizeNearestInit_L(IppiSizeL srcSize, IppiSizeL dstSize, IppDataType
dataType, IppiResizeSpec* pSpec);
```

Threading layer (TL) function

```
IppStatus ippiResizeNearestInit_LT(IppiSizeL srcSize, IppiSizeL dstSize, IppDataType
dataType, Ipp32u numChannels, IppiResizeSpec_LT* pSpec);
```

Include Files

`ippi.h`

Flavors with the `_LT` suffix: `ippi_tl.h`

Flavors with the `_L` suffix: `ippi_l.h`

Domain Dependencies

Flavors declared in `ippi.h`:

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Flavors declared in `ippi_tl.h`:

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`, `ippcore_tl.lib`, `ippi_tl.lib`

Parameters

`srcSize`	Size, in pixels, of the source image.
`dstSize`	Size, in pixels, of the destination image.
`dataType`	Data type of the image. Supported values: `ipp8u`, `ipp16u`, `ipp16s`, `ipp32f`, `ipp64f`.
`pSpec`	Pointer to the specification structure for the resize filter.
`numChannels`	Number of channels. Possible values are 1, 3, and 4.

Description

This function initializes the specification structure for the resize algorithm with the nearest neighbor interpolation method. To calculate the size of the specification structure object, call the `ippiResizeGetSize` function.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when one of the specified pointers is `NULL`.
`ippStsNoOperation`	Indicates a warning when width or height of the image is equal to zero.
`ippStsSizeErr`	Indicates an error when width or height of the source or destination image is negative.
`ippStsDataTypeErr`	Indicates an error if `dataType` has an illegal value.
`ippStsExceededSizeErr`	Indicates an error If width or height of the source or destination image exceeds 33554431 (0x1FFFFFF) (only for platform-aware and TL functions).

See Also

ResizeGetSize Computes the size of the specification structure and the size of the external temporary buffer for the resize transform initialization.

ResizeNearest

Changes an image size using the nearest neighbor interpolation method.

Syntax

```
IppStatus ippiResizeNearest_<mod>(const Ipp<datatype>* pSrc, Ipp32s srcStep,
Ipp<datatype>* pDst, Ipp32s dstStep, IppiPoint dstOffset, IppiSize dstSize, const
IppiResizeSpec_32f* pSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_C4R	16u_C4R	16s_C4R	32f_C4R

Platform-aware functions

```
IppStatus ippiResizeNearest_<mod>_L(const Ipp<datatype>* pSrc, IppSizeL srcStep,
Ipp<datatype>* pDst, IppSizeL dstStep, IppiPointL dstOffset, IppiSizeL dstSize, const
IppiResizeSpec* pSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_C4R	16u_C4R	16s_C4R	32f_C4R

Threading layer (TL) functions

```
IppStatus ippiResizeNearest_<mod>_LT(const Ipp<datatype>* pSrc, IppSizeL srcStep,
Ipp<datatype>* pDst, IppSizeL dstStep, const IppiResizeSpec_LT* pSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_C4R	16u_C4R	16s_C4R	32f_C4R

Include Files

`ippi.h`

Flavors with the `_LT` suffix: `ippi_tl.h`

Flavors with the `_L` suffix: `ippi_l.h`

Domain Dependencies

Flavors declared in `ippi.h`:

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Flavors declared in `ippi_tl.h`:

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`, `ippcore_tl.lib`, `ippi_tl.lib`

Parameters

pSrc	Pointer to the source image.
srcStep	Distance, in bytes, between the starting points of consecutive lines in the source image buffer.
pDst	Pointer to the destination image.
dstStep	Distance, in bytes, between the starting points of consecutive lines in the destination image buffer.
dstOffset	Offset of the tiled destination image with respect to the destination image origin.
dstSize	Size of the destination image in pixels.
pSpec	Pointer to the specification structure for the resize filter.
pBuffer	Pointer to the work buffer.

Description

This function changes an image size using the nearest neighbor interpolation method. The image size can be either reduced or increased in each direction, depending on the destination image size.

This function operates with ROI. It resizes the source image ROI origin to the destination image ROI origin. You need to define the destination image ROI origin by the following parameters: the offset of the tiled destination image with respect to the destination image origin and the destination image size. The source image ROI origin is defined automatically. To obtain the source image ROI, use the `ippiResizeGetSrcRoi` function with the corresponding `mod` value. To obtain the source image ROI origin offset, call the `ippiResizeGetSrcOffset` function with the corresponding `mod` value. Parameters `pSrc` and `pDst` must point to the processed source and destination image ROI origins, respectively.

Function flavors operating on images of 64-bit sizes (with the `L` suffix) can process only whole images.

The interpolation algorithm applied uses only pixels of the source image origin that are inside of the image boundaries.

Before using the `ippiResizeNearest` function, you need to initialize the resize specification structure using the `ippiResizeNearestInit` function and compute the size of the external buffer `pBuffer` using the `ippiResizeGetBufferSize` function for the corresponding flavor.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is `NULL`.
ippStsNoOperation	Indicates a warning when width or height of the destination image is equal to zero.
ippStsContextMatchErr	Indicates an error when pointer to the spec structure is invalid.
ippStsSizeErr	Indicates an error when width or height of the source or destination image is negative.

ippStsStepErr	Indicates an error when the step value is not data type multiple.
ippStsOutOfRangeErr	Indicates an error when the destination image offset point is outside the destination image origin.
ippStsSizeWrn	Indicates a warning when the destination image size is more than the destination image origin size.

See Also

ROI Processing in Geometric Transforms

ResizeGetSrcRoi Computes the ROI of the source image for resizing by tile processing.

ResizeGetSrcOffset Computes the offset of the source image for resizing by tile processing.

ResizeNearestInit Initializes the specification structure for image resizing with the nearest neighbor interpolation method.

ResizeGetBufferSize Computes the size of the external buffer for image resizing.

ResizeLinearInit

Initializes the specification structure for image resizing with the linear interpolation method.

Syntax

```
IppStatus ippiResizeLinearInit_<mod>(IppiSize srcSize, IppiSize dstSize,
IppiResizeSpec_32f* pSpec);
```

Supported values for `mod`:

8u	16u	16s	32f

```
IppStatus ippiResizeLinearInit_64f(IppiSize srcSize, IppiSize dstSize,
IppiResizeSpec_64f* pSpec);
```

Platform-aware function

```
IppStatus ippiResizeLinearInit_L(IppiSizeL srcSize, IppiSizeL dstSize, IppDataType
dataType, IppiResizeSpec* pSpec);
```

```
IppStatus ippiResizeLinearInit_8u_L(IppiSizeL srcSize, IppiSizeL dstSize,
IppHintAlgorithm hint, IppiResizeSpec* pSpec);
```

Threading layer (TL) function

```
IppStatus ippiResizeLinearInit_LT(IppiSizeL srcSize, IppiSizeL dstSize, IppDataType
dataType, Ipp32u numChannels, IppiResizeSpec_LT* pSpec);
```

Include Files

`ippi.h`

Flavors with the `_LT` suffix: `ippi_tl.h`

Flavors with the `_L` suffix: `ippi_l.h`

Domain Dependencies

Flavors declared in `ippi.h`:

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Flavors declared in ippi_tl.h:

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib, ippcore_tl.lib, ippi_tl.lib

Parameters

srcSize	Size, in pixels, of the source image.
dstSize	Size, in pixels, of the destination image.
dataType	Data type of the image. Supported values: ipp8u, ipp16u, ipp16s, ipp32f, ipp64f.
hint	Computation algorithm for processing Ipp8u data. Possible values are: • ippAlgHintNone • The hint is absent. Equivalent to ippAlgHintFast. • ippAlgHintFast • A faster but less accurate algorithm. • ippAlgHintAccurate • A slower but more accurate algorithm.
pSpec	Pointer to the specification structure for the resize filter.
numChannels	Number of channels. Possible values are 1, 3, and 4.

Description

This function initializes the specification structure for the resize algorithm with the linear interpolation method. To calculate the size of the specification structure, call the ippiResizeGetSize function.

NOTE The function with the parameter hint allows users to choose between a faster but less accurate algorithm and a slower but more accurate one. Without the parameter hint this function initializes the specification structure for the faster but less accurate algorithm for Ipp8u data type.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsNoOperation	Indicates a warning when width or height of the image is equal to zero.
ippStsSizeErr	Indicates an error if: • width or height of the source or destination image is negative

	• the source image size is less than the size of a 2x2 linear filter
ippStsDataTypeErr	Indicates an error if dataType has an illegal value.
ippStsExceededSizeErr	Indicates an error If width or height of the source or destination image exceeds 33554431 (0x1FFFFFF) (only for platform-aware or TL functions).
ippStsNumChannelErr	Indicates an error if numChannel has an illegal value.

See Also

ResizeGetSize Computes the size of the specification structure and the size of the external temporary buffer for the resize transform initialization.

ResizeLinear

Changes an image size using the linear interpolation method.

Syntax

Case 1: Single precision

```
IppStatus ippiResizeLinear_<mod>(const Ipp<datatype>* pSrc, Ipp32s srcStep,
Ipp<datatype>* pDst, Ipp32s dstStep, IppiPoint dstOffset, IppiSize dstSize,
IppiBorderType border, const Ipp<datatype>* pBorderValue, const IppiResizeSpec_32f*
pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
8u_C3R	16u_C3R	16s_C3R	
8u_C4R	16u_C4R	16s_C4R	32f_C4R

```
IppStatus ippiResizeLinear_32f_C3R(const Ipp32f* pSrc, const Ipp32s srcStep, Ipp32f*
pDst, Ipp32s dstStep, IppiPoint dstOffset, IppiSize dstSize, IppiBorderType border,
const Ipp32f* pBorderValue, const IppiResizeSpec_32f* pSpec, Ipp8u* pBuffer);
```

Case 2: Double precision

```
IppStatus ippiResizeLinear_<mod>(const Ipp<datatype>* pSrc, Ipp32s srcStep,
Ipp<datatype>* pDst, Ipp32s dstStep, IppiPoint dstOffset, IppiSize dstSize,
IppiBorderType border, const Ipp<datatype>* pBorderValue, const IppiResizeSpec_64f*
pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

64f_C1R
64f_C3R
64f_C4R

Case 3: Platform-aware functions

```
IppStatus ippiResizeLinear_<mod>_L(const Ipp<datatype>* pSrc, IppSizeL srcStep,
Ipp<datatype>* pDst, IppSizeL dstStep, IppiPointL dstOffset, IppiSizeL dstSize,
IppiBorderType border, const Ipp<datatype>* pBorderValue, const IppiResizeSpec* pSpec,
Ipp8u* pBuffer);
```

Supported values for `mod`:

8u_C1R	16u_C1R	16s_C1R	32f_C1R	64f_C1R
8u_C3R	16u_C3R	16s_C3R	32f_C3R	64f_C3R
8u_C4R	16u_C4R	16s_C4R	32f_C4R	64f_C4R

Case 3: Threading layer (TL) functions

```
IppStatus ippiResizeLinear_<mod>_LT(const Ipp<datatype>* pSrc, IppSizeL srcStep,
Ipp<datatype>* pDst, IppSizeL dstStep, IppiBorderType border, const Ipp8u*
pBorderValue, const IppiResizeSpec_LT* pSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

8u_C1R	16u_C1R	16s_C1R	32f_C1R	64f_C1R
8u_C3R	16u_C3R	16s_C3R	32f_C3R	64f_C3R
8u_C4R	16u_C4R	16s_C4R	32f_C4R	64f_C4R

Include Files

`ippi.h`

Flavors with the `_LT` suffix: `ippi_tl.h`

Flavors with the `_L` suffix: `ippi_l.h`

Domain Dependencies

Flavors declared in `ippi.h`:

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Flavors declared in `ippi_tl.h`:

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`, `ippcore_tl.lib`, `ippi_tl.lib`

Parameters

`pSrc`	Pointer to the source image.
`srcStep`	Distance, in bytes, between the starting points of consecutive lines in the source image buffer.
`pDst`	Pointer to the destination image.
`dstStep`	Distance, in bytes, between the starting points of consecutive lines in the destination image buffer.
`dstOffset`	Offset of the tiled destination image with respect to the destination image origin.

dstSize	Size of the destination image in pixels.
border	Type of border. Possible values are: • ippBorderRepl • Border is replicated from the edge pixels. • ippBorderInMem • Border is obtained from the source image pixels in memory. • ippBorderMirror • Border is obtained from the source image boundary pixels. Value can • ippBorderMirrorR • Border is obtained from the source image boundary pixels. The anchor
pBorderValue	Pointer to the constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.
pSpec	Pointer to the specification structure for the resize filter.
pBuffer	Pointer to the work buffer.

Description

This function changes the size of an image using the linear interpolation method. The image size can be either reduced or increased in each direction, depending on the destination image size.

This function operates with ROI. It resizes the source image ROI origin to the destination image ROI origin. You need to define the destination image ROI origin by the following parameters: the offset of the tiled destination image with respect to the destination image origin and the destination image size. The source image ROI origin is defined automatically. To obtain the source image ROI, use the ippiResizeGetSrcRoi function with the corresponding mod value. To obtain the source image ROI origin offset, call the ippiResizeGetSrcOffset function with the corresponding mod value. Parameters pSrc and pDst must point to the processed source and destination image ROI origins, respectively.

The interpolation algorithm applied uses edge pixels of the source image that are out of the image origin. The function ippiResizeLinear uses in calculation the weighted values of these outer pixels. To obtain the size of the out of the source image origin, call the ippiResizeGetBorderSize function for the corresponding flavor.

If the border type is equal to ippBorderRepl, the source image edge pixels are replicated out of the image origin. If the border type is equal to ippBorderInMem, the outer pixels are obtained from the out of the source image origin space. For the mixed border types, the combined approach is applied.

Before using the ippiResizeLinear function, you need to initialize the specification structure using the ippiResizeLinearInit function and compute the size of the external buffer pBuffer using the ippiResizeGetBufferSize function for the corresponding flavor.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.

ippStsNoOperation	Indicates a warning when width or height of the destination image is equal to zero.
ippStsBorderErr	Indicates an error when the border value is illegal.
ippStsContextMatchErr	Indicates an error when pointer to the specification structure is invalid.
ippStsNotSupportedModeErr	Indicates an error when the requested mode is not supported.
ippStsSizeErr	Indicates an error when width or height of the source or destination image is negative.
ippStsStepErr	Indicates an error when the step value is not a multiple of data type.
ippStsOutOfRangeErr	Indicates an error when the destination image offset point is outside the destination image origin.
ippStsSizeWrn	Indicates a warning when the destination image size is more than the destination image origin size.

See Also

ROI Processing in Geometric Transforms

ResizeGetSrcRoi Computes the ROI of the source image for resizing by tile processing.

ResizeGetSrcOffset Computes the offset of the source image for resizing by tile processing.

User-defined Border Types

ResizeGetBorderSize Computes the size of possible borders for the resize transform.

ResizeLinearInit Initializes the specification structure for image resizing with the linear interpolation method.

ResizeGetBufferSize Computes the size of the external buffer for image resizing.

ResizeCubicInit

Initializes the specification structure for image resizing using interpolation with two-parameter cubic filters.

Syntax

```
IppStatus ippiResizeCubicInit_<mod>(IppiSize srcSize, IppiSize dstSize, Ipp32f valueB,
Ipp32f valueC, IppiResizeSpec_32f* pSpec, Ipp8u* pInitBuf);
```

Supported values for `mod`:

8u	16u	16s	32f

Platform-aware functions

```
IppStatus ippiResizeCubicInit_L(IppiSizeL srcSize, IppiSizeL dstSize, IppDataType
dataType, Ipp32f valueB, Ipp32f valueC, IppiResizeSpec* pSpec, Ipp8u* pInitBuf);
```

```
IppStatus ippiResizeCubicInit_8u_L(IppiSizeL srcSize, IppiSizeL dstSize, Ipp32f valueB,
Ipp32f valueC, IppHintAlgorithm hint, IppiResizeSpec* pSpec, Ipp8u* pInitBuf);
```

Threading layer (TL) functions

```
IppStatus ippiResizeCubicInit_LT(IppiSizeL srcSize, IppiSizeL dstSize, IppDataType
dataType, Ipp32u numChannels, Ipp32f valueB, Ipp32f valueC, IppiResizeSpec_LT* pSpec,
Ipp8u* pInitBuf);
```

Include Files

`ippi.h`

Flavors with the `_LT` suffix: `ippi_tl.h`

Flavors with the `_L` suffix: `ippi_l.h`

Domain Dependencies

Flavors declared in `ippi.h`:

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Flavors declared in `ippi_tl.h`:

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`, `ippcore_tl.lib`, `ippi_tl.lib`

Parameters

`srcSize`	Size, in pixels, of the source image.
`dstSize`	Size, in pixels, of the destination image.
`dataType`	Data type of the image. Supported values: `ipp8u`, `ipp16u`, `ipp16s`, `ipp32f`, `ipp64f`.
`hint`	Computation algorithm for processing `Ipp8u` data. Possible values are: • `ippAlgHintNone` • The hint is absent. Equivalent to `ippAlgHintFast`. • `ippAlgHintFast` • A faster but less accurate algorithm. • `ippAlgHintAccurate` • A slower but more accurate algorithm.
`numChannels`	Number of channels, possible values: 1, 3, or 4.
`valueB`	The first parameter for cubic filters.
`valueC`	The second parameter for cubic filters.
`pSpec`	Pointer to the specification structure for the resize filter.
`pInitBuf`	Pointer to the temporary buffer for the cubic filter initialization.

Description

This function initializes the specification structure for the resize algorithm with interpolation with two-parameter cubic filters. To calculate the size of the specification structure, call the `ippiResizeGetSize` function.

Before using this function, you need to calculate the size of the specification structure and the external buffer `pInitBuf` using the `ippiResizeGetSize` function for the corresponding flavor.

NOTE The function with the parameter `hint` allows users to choose between a faster but less accurate algorithm and a slower but more accurate one. Without the parameter `hint` this function initializes the specification structure for the faster but less accurate algorithm for `Ipp8u` data type.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when one of the specified pointers is `NULL`.
`ippStsNoOperation`	Indicates a warning when width or height of the image is equal to zero.
`ippStsSizeErr`	Indicates an error if: • width or height of the source or destination image is negative • the source image size is less than the size of a 2x2 linear filter
`ippStsDataTypeErr`	Indicates an error if `dataType` has an illegal value.
`ippStsExceededSizeErr`	Indicates an error If width or height of the source or destination image exceeds 33554431 (0x1FFFFFF) (only for platform-aware and TL functions).
`ippStsNumChannelErr`	Indicates an error if `numChannel` has an illegal value.

See Also

ResizeGetSize Computes the size of the specification structure and the size of the external temporary buffer for the resize transform initialization.

ResizeCubic

Changes an image size using interpolation with two-parameter cubic filters.

Syntax

```
IppStatus ippiResizeCubic_<mod>(const Ipp<datatype>* pSrc, Ipp32s srcStep,
Ipp<datatype>* pDst, Ipp32s dstStep, IppiPoint dstOffset, IppiSize dstSize,
IppiBorderType border, const Ipp<datatype>* pBorderValue, const IppiResizeSpec_32f*
pSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_C4R	16u_C4R	16s_C4R	32f_C4R

Platform-aware functions

```
IppStatus ippiResizeCubic_<mod>_L(const Ipp<datatype>* pSrc, IppSizeL srcStep,
Ipp<datatype>* pDst, IppSizeL dstStep, IppiPointL dstOffset, IppiSizeL dstSize,
IppiBorderType border, const Ipp<datatype>* pBorderValue, const IppiResizeSpec* pSpec,
Ipp8u* pBuffer);
```

Supported values for `mod`:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_C4R	16u_C4R	16s_C4R	32f_C4R

Threading layer (TL) functions

```
IppStatus ippiResizeCubic_<mod>_LT(const Ipp<datatype>* pSrc, IppSizeL srcStep,
Ipp<datatype>* pDst, IppSizeL dstStep, IppiBorderType border, const Ipp<datatype>*
pBorderValue, const IppiResizeSpec_LT* pSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_C4R	16u_C4R	16s_C4R	32f_C4R

Include Files

`ippi.h`

Flavors with the `_LT` suffix: `ippi_tl.h`

Flavors with the `_L` suffix: `ippi_l.h`

Domain Dependencies

Flavors declared in `ippi.h`:

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Flavors declared in `ippi_tl.h`:

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`, `ippcore_tl.lib`, `ippi_tl.lib`

Parameters

`pSrc`	Pointer to the source image.
`srcStep`	Distance, in bytes, between the starting points of consecutive lines in the source image buffer.

pDst	Pointer to the destination image.
dstStep	Distance, in bytes, between the startings of consecutive lines in the destination image buffer.
dstOffset	Offset of the tiled destination image with respect to the destination image origin.
dstSize	Size of the destination image in pixels.
border	Type of border. Possible values are: • ippBorderRepl • Border is replicated from the edge pixels. • ippBorderInMem • Border is obtained from the source image pixels in memory. • ippBorderMirror • Border is obtained from the source image boundary pixels. Value can • ippBorderMirrorR • Border is obtained from the source image boundary pixels. The anchor
pBorderValue	Pointer to the constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.
pSpec	Pointer to the spec structure for the resize filter.
pBuffer	Pointer to the work buffer.

Description

This function changes an image size using interpolation with two-parameter cubic filters. The image size can be either reduced or increased in each direction, depending on the destination image size.

This function operates with ROI. It resizes the source image ROI origin to the destination image ROI origin. You need to define the destination image ROI origin by the following parameters: the offset of the tiled destination image with respect to the destination image origin and the destination image size. The source image ROI origin is defined automatically. To obtain the source image ROI, use the ippiResizeGetSrcRoi function with the corresponding mod value. To obtain the source image ROI origin offset, call the ippiResizeGetSrcOffset function with the corresponding mod value. Parameters pSrc and pDst must point to the processed source and destination image ROI origins, respectively.

The interpolation algorithm applied uses edge pixels of the source image that are out of the image origin. The function ippiResizeCubic uses in calculation the weighted values of these outer pixels. To obtain the size of the out of the source image origin, call the ippiResizeGetBorderSize function for the corresponding flavor.

If the border type is equal to ippBorderRepl, the source image edge pixels are replicated out of the image origin. If the border type is equal to ippBorderInMem, the outer pixels are obtained from the out of the source image origin space. For the mixed border types, the combined approach is applied.

Before using the ippiResizeLinear function, you need to initialize the specification structure using the ippiResizeCubicInit function and compute the size of the external buffer pBuffer using the ippiResizeGetBufferSize function for the corresponding flavor.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsNoOperation	Indicates a warning when width or height of the destination image is equal to zero.
ippStsBorderErr	Indicates an error when the border value is illegal.
ippStsContextMatchErr	Indicates an error when pointer to the spec structure is invalid.
ippStsNotSupportedModeErr	Indicates an error when the requested mode is not supported.
ippStsSizeErr	Indicates an error when width or height of the source or destination image is negative.
ippStsStepErr	Indicates an error when the step value is not data type multiple.
ippStsOutOfRangeErr	Indicates an error when the destination image offset point is outside the destination image origin.
ippStsSizeWrn	Indicates a warning when the destination image size is more than the destination image origin size.

See Also

ROI Processing in Geometric Transforms

ResizeGetSrcRoi Computes the ROI of the source image for resizing by tile processing.

ResizeGetSrcOffset Computes the offset of the source image for resizing by tile processing.

User-defined Border Types

ResizeGetBorderSize Computes the size of possible borders for the resize transform.

ResizeCubicInit Initializes the specification structure for image resizing using interpolation with two-parameter cubic filters.

ResizeGetBufferSize Computes the size of the external buffer for image resizing.

ResizeLanczosInit

Initializes the specification structure for image resizing with the Lanczos interpolation method.

Syntax

```
IppStatus ippiResizeLanczosInit_<mod>(IppiSize srcSize, IppiSize dstSize, Ipp32u numLobes, IppiResizeSpec_32f* pSpec, Ipp8u* pInitBuf);
```

Supported values for mod:

8u	16u	16s	32f

Platform-aware functions

```
IppStatus ippiResizeLanczosInit_L(IppiSizeL srcSize, IppiSizeL dstSize, IppDataType
dataType, Ipp32u numLobes, IppiResizeSpec* pSpec, Ipp8u* pInitBuf);

IppStatus ippiResizeLanczosInit_8u_L(IppiSizeL srcSize, IppiSizeL dstSize, Ipp32u
numLobes, IppHintAlgorithm hint, IppiResizeSpec* pSpec, Ipp8u* pInitBuf);
```

Threading layer (TL) functions

```
IppStatus ippiResizeLanczosInit_LT(IppiSizeL srcSize, IppiSizeL dstSize, IppDataType
dataType, Ipp32u numChannels, Ipp32u numLobes, IppiResizeSpec_LT* pSpec, Ipp8u*
pInitBuf);
```

Include Files

`ippi.h`

Flavors with the `_LT` suffix: `ippi_tl.h`

Flavors with the `_L` suffix: `ippi_l.h`

Domain Dependencies

Flavors declared in `ippi.h`:

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Flavors declared in `ippi_tl.h`:

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`, `ippcore_tl.lib`, `ippi_tl.lib`

Parameters

`srcSize`	Size, in pixels, of the source image.
`dstSize`	Size, in pixels, of the destination image.
`dataType`	Data type of the image. Supported values: `ipp8u`, `ipp16u`, `ipp16s`, `ipp32f`, `ipp64f`.
`hint`	Computation algorithm for processing `Ipp8u` data. Possible values are: • `ippAlgHintNone` • The hint is absent. Equivalent to `ippAlgHintFast`. • `ippAlgHintFast` • A faster but less accurate algorithm. • `ippAlgHintAccurate` • A slower but more accurate algorithm.
`numChannels`	Number of image channels. Possible values are 1, 3, or 4.
`numLobes`	Parameter for Lanczos filters. Possible values are 2 or 3.
`pSpec`	Pointer to the specification structure for the resize filter.

pInitBuf	Pointer to the temporary buffer for the cubic filter initialization.

Description

This function initializes the specification structure for the resize algorithm with the Lanczos filter interpolation method. This method is based on the 2-lobed or 3-lobed Lanczos window function as an interpolation function, depending on the value of the numLobes parameter.

Before using this function, you need to calculate the size of the specification structure and the external buffer pInitBuf using the ippiResizeGetSize function for the corresponding flavor.

NOTE The function with the parameter hint allows users to choose between a faster but less accurate algorithm and a slower but more accurate one. Without the parameter hint this function initializes the specification structure for the faster but less accurate algorithm for Ipp8u data type.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsResizeNoOperation	Indicates an error when width or height of the image is equal to zero.
ippStsNotSupportedModeErr	Indicates an error when the requested mode is not supported.
ippStsDataTypeErr	Indicates an error if dataType has an illegal value.
ippStsExceededSizeErr	Indicates an error If width or height of the source or destination image exceeds 33554431 (0x1FFFFFF) (only for platform-aware and TL functions).
ippStsSizeErr	Indicates an error in the following cases:

See Also

ResizeGetSize Computes the size of the specification structure and the size of the external temporary buffer for the resize transform initialization.

ResizeLanczos

Changes an image size using interpolation with the Lanczos filter.

Syntax

```
IppStatus ippiResizeLanczos_<mod>(const Ipp<datatype>* pSrc, Ipp32s srcStep,
Ipp<datatype>* pDst, Ipp32s dstStep, IppiPoint dstOffset, IppiSize dstSize,
IppiBorderType border, const Ipp<datatype>* pBorderValue, const IppiResizeSpec_32f*
pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_C4R	16u_C4R	16s_C4R	32f_C4R

Platform-aware functions

```
IppStatus ippiResizeLanczos_<mod>_L(const Ipp<datatype>* pSrc, IppSizeL srcStep,
Ipp<datatype>* pDst, IppSizeL dstStep, IppiPointL dstOffset, IppiSizeL dstSize,
IppiBorderType border, const Ipp<datatype>* pBorderValue, const IppiResizeSpec* pSpec,
Ipp8u* pBuffer);
```

Supported values for `mod`:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_C4R	16u_C4R	16s_C4R	32f_C4R

Threading layer (TL) functions

```
IppStatus ippiResizeLanczos_<mod>_LT(const Ipp<datatype>* pSrc, IppSizeL srcStep,
Ipp<datatype>* pDst, IppSizeL dstStep, IppiBorderType border, const Ipp<datatype>*
pBorderValue, const IppiResizeSpec_LT* pSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_C4R	16u_C4R	16s_C4R	32f_C4R

Include Files

`ippi.h`

Flavors with the `_LT` suffix: `ippi_tl.h`

Flavors with the `_L` suffix: `ippi_l.h`

Domain Dependencies

Flavors declared in `ippi.h`:

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Flavors declared in `ippi_tl.h`:

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`, `ippcore_tl.lib`, `ippi_tl.lib`

Parameters

`pSrc`	Pointer to the source image.
`srcStep`	Distance, in bytes, between the starting points of consecutive lines in the source image buffer.

pDst	Pointer to the destination image.
dstStep	Distance, in bytes, between the starting points of consecutive lines in the destination image buffer.
dstOffset	Offset of the tiled destination image with respect to the destination image origin.
dstSize	Size of the destination image in pixels.
border	Type of border. Possible values are: • ippBorderRepl • Border is replicated from the edge pixels. • ippBorderInMem • Border is obtained from the source image pixels in memory.
pBorderValue	Pointer to the constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.
pSpec	Pointer to the specification structure for the resize filter.
pBuffer	Pointer to the work buffer.

Description

This function changes an image size using interpolation with the Lanczos filter. The image size can be either reduced or increased in each direction, depending on the destination image size.

This function operates with ROI. It resizes the source image ROI origin to the destination image ROI origin. You need to define the destination image ROI origin by the following parameters: the offset of the tiled destination image with respect to the destination image origin and the destination image size. The source image ROI origin is defined automatically. To obtain the source image ROI, use the ippiResizeGetSrcRoi function with the corresponding mod value. To obtain the source image ROI origin offset, call the ippiResizeGetSrcOffset function with the corresponding mod value. Parameters pSrc and pDst must point to the processed source and destination image ROI origins, respectively.

The interpolation algorithm applied uses edge pixels of the source image that are out of the image origin. The function ippiResizeLanczos uses in calculation the weighted values of these outer pixels. To obtain the size of the out of the source image origin, call the ippiResizeGetBorderSize function for the corresponding flavor.

If the border type is equal to ippBorderRepl, the source image edge pixels are replicated out of the image origin. If the border type is equal to ippBorderInMem, the outer pixels are obtained from the out of the source image origin space. For the mixed border types, a combined approach is applied.

Before using the ippiResizeLanczos function, you need to initialize the specification structure using the ippiResizeLanczosInit function and compute the size of the external buffer pBuffer using the ippiResizeGetBufferSize function for the corresponding flavor.

Return Values

ippStsNoErr	Indicates no error.

ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsNoOperation	Indicates a warning when width or height of the destination image is equal to zero.
ippStsBorderErr	Indicates an error when the border value is illegal.
ippStsContextMatchErr	Indicates an error when pointer to the specification structure is invalid.
ippStsNotSupportedModeErr	Indicates an error when the requested mode is not supported.
ippStsSizeErr	Indicates an error when width or height of the destination image is negative.
ippStsStepErr	Indicates an error when the step value is not data type multiple.
ippStsOutOfRangeErr	Indicates an error when the destination image offset point is outside the destination image origin.
ippStsSizeWrn	Indicates a warning when the destination image size is more than the destination image origin size.

See Also

ROI Processing in Geometric Transforms

ResizeGetSrcRoi Computes the ROI of the source image for resizing by tile processing.

ResizeGetSrcOffset Computes the offset of the source image for resizing by tile processing.

User-defined Border Types

ResizeGetBorderSize Computes the size of possible borders for the resize transform.

ResizeLanczosInit Initializes the specification structure for image resizing with the Lanczos interpolation method.

ResizeGetBufferSize Computes the size of the external buffer for image resizing.

ResizeSuperShiftInit

Initializes the specification structure for image resizing with the super sampling interpolation method and floating point shifts.

Syntax

```
IppStatus ippiResizeSuperInit_<mod>(IppiSize srcSize, IppiSize dstSize, Ipp64f xShift, Ipp64f yShift, IppiBorderType borderType, const Ipp8u * borderVal, int smoothEdge, int numChannels, IppiResizeSpec_32f* pSpec);
```

Supported values for mod:

8u	16u	16s	32f

Include Files

ippi.h

Domain Dependencies

Flavors declared in `ippi.h`:

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`srcSize`	Size, in pixels, of the source image.
`dstSize`	Size, in pixels, of the destination image.
`xShift`	Shift value in the `x` direction.
`yShift`	Shift value in the `y` direction.
`borderType`	Type of border. Supported values: `ippBorderTransp`. Outer pixels are not processed.
`borderVal`	Not used. This parameter is applicable only to the `ippBorderConst` border type.
`smoothEdge`	Flag for edge smoothing. Supported values: 0 - transformation without edge smoothing.1 - transformation with edge smoothing.
`pSpec`	Pointer to the specification structure for the resize filter.
`numChannels`	Number of channels. Possible values are 1, 3, and 4.

Description

This function initializes the specification structure for the resize algorithm with the super sampling interpolation method. To calculate the size of the specification structure, call the `ippiResizeGetSize` function.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if one of the specified pointers is `NULL`.
`ippStsNoOperation`	Indicates a warning if: - width or height of the destination image is equal to zero.- smoothing edge is on and width or height of the destination image is equal to one.
`ippStsSizeErr`	Indicates an error in the following cases: • Indicates that one of the specified dimensions of the source image is less than the corresponding dimension of the destination

	image, or that the width or height of the source or destination image is negative. • If the width or height of the source or destination image is negative.
ippStsExceededSizeErr	Indicates an error If width or height of the source or destination image exceeds 33554431 (0x1FFFFFF) (only for platform-aware and TL functions).
ippStsDataTypeErr	Indicates an error if dataType has an illegal value.
ippStsNumChannelsErr	Indicates an error if the number of channels is not supported.
ippStsBorderErr	Indicates an error if specified borderType is not supported.

See Also

ResizeGetSize Computes the size of the specification structure and the size of the external temporary buffer for the resize transform initialization.

ResizeGetBufferSize Computes the size of the external buffer for image resizing.

ResizeSuperInit

Initializes the specification structure for image resizing with the super sampling interpolation method.

Syntax

```
IppStatus ippiResizeSuperInit_<mod>(IppiSize srcSize, IppiSize dstSize,
IppiResizeSpec_32f* pSpec);
```

Supported values for mod:

8u	16u	16s	32f

Platform-aware function

```
IppStatus ippiResizeSuperInit_L(IppiSizeL srcSize, IppiSizeL dstSize, IppDataType
dataType, IppiResizeSpec* pSpec);
```

Threading layer (TL) function

```
IppStatus ippiResizeSuperInit_LT(IppiSizeL srcSize, IppiSizeL dstSize, IppDataType
dataType, Ipp32u numChannels, IppiResizeSpec_LT* pSpec);
```

Include Files

ippi.h

Flavors with the _LT suffix: ippi_tl.h

Flavors with the _L suffix: ippi_l.h

Domain Dependencies

Flavors declared in `ippi.h`:

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Flavors declared in `ippi_tl.h`:

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`, `ippcore_tl.lib`, `ippi_tl.lib`

Parameters

`srcSize`	Size, in pixels, of the source image.
`dstSize`	Size, in pixels, of the destination image.
`dataType`	Data type of the image. Supported values: `ipp8u`, `ipp16u`, `ipp16s`, `ipp32f`, `ipp64f`.
`pSpec`	Pointer to the specification structure for the resize filter.
`numChannels`	Number of channels. Possible values are 1, 3, and 4.

Description

This function initializes the specification structure for the resize algorithm with the super sampling interpolation method. To calculate the size of the specification structure, call the `ippiResizeGetSize` function.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if one of the specified pointers is `NULL`.
`ippStsNoOperation`	Indicates a warning if width or height of the image is equal to zero.
`ippStsSizeErr`	Indicates an error in the following cases: • Indicates that one of the specified dimensions of the source image is less than the corresponding dimension of the destination image, or that the width or height of the source or destination image is negative. • If the width or height of the source or destination image is negative.
`ippStsExceededSizeErr`	Indicates an error If width or height of the source or destination image exceeds 33554431 (0x1FFFFFF) (only for platform-aware and TL functions).

ippStsDataTypeErr	Indicates an error if dataType has an illegal value.

See Also

ResizeGetSize Computes the size of the specification structure and the size of the external temporary buffer for the resize transform initialization.

ResizeSuper

Changes an image size using the super sampling interpolation method.

Syntax

```
IppStatus ippiResizeSuper_<mod>(const Ipp<datatype>* pSrc, Ipp32s srcStep,
Ipp<datatype>* pDst, Ipp32s dstStep, IppiPoint dstOffset, IppiSize dstSize, const
IppiResizeSpec_32f* pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_C4R	16u_C4R	16s_C4R	32f_C4R

Platform-aware functions

```
IppStatus ippiResizeSuper_<mod>_L(const Ipp<datatype>* pSrc, IppSizeL srcStep,
Ipp<datatype>* pDst, IppSizeL dstStep, IppiPointL dstOffset, IppiSizeL dstSize, const
IppiResizeSpec* pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_C4R	16u_C4R	16s_C4R	32f_C4R

Threading layer (TL) functions

```
IppStatus ippiResizeSuper_<mod>_LT(const Ipp<datatype>* pSrc, IppSizeL srcStep,
Ipp<datatype>* pDst, IppSizeL dstStep, const IppiResizeSpec_LT* pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_C4R	16u_C4R	16s_C4R	32f_C4R

Include Files

ippi.h

Flavors with the _LT suffix: ippi_tl.h

Flavors with the _L suffix: ippi_l.h

Domain Dependencies

Flavors declared in `ippi.h`:

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Flavors declared in `ippi_tl.h`:

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`, `ippcore_tl.lib`, `ippi_tl.lib`

Parameters

`pSrc`	Pointer to the source image.
`srcStep`	Distance, in bytes, between the starting points of consecutive lines in the source image buffer.
`pDst`	Pointer to the destination image.
`dstStep`	Distance, in bytes, between the starting points of consecutive lines in the destination image buffer.
`dstOffset`	Offset of the tiled destination image with respect to the destination image origin.
`dstSize`	Size of the destination image in pixels.
`pSpec`	Pointer to the specification structure for the resize filter.
`pBuffer`	Pointer to the work buffer.

Description

This function changes an image size using the super sampling interpolation method. This method only reduces the image size.

This function operates with ROI. It resizes the source image ROI origin to the destination image ROI origin. You need to define the destination image ROI origin by the following parameters: the offset of the tiled destination image with respect to the destination image origin and the destination image size. The source image ROI origin is defined automatically. To obtain the source image ROI, use the `ippiResizeGetSrcRoi` function with the corresponding `mod` value. To obtain the source image ROI origin offset, call the `ippiResizeGetSrcOffset` function with the corresponding `mod` value. Parameters `pSrc` and `pDst` must point to the processed source and destination image ROI origins, respectively.

The interpolation algorithm applied uses only pixels of the source image origin that are inside of the image boundaries.

Before using the `ippiResizeLinear` function, you need to initialize the resize structure using the `ippiResizeSuperInit` function and compute the size of the external buffer `pBuffer` using the `ippiResizeGetBufferSize` function for the corresponding flavor.

NOTE You can get better performance if you use the following scaling factors along the `x` and `y` axes: 1/2, 2/3, 3/4, 4/5, 5/6, 8/9, 1/3, 2/5, 3/5, 3/7, 4/9, 7/10, 1/4, 2/7, 3/8, 1/8.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsNoOperation	Indicates a warning when width or height of the destination image is equal to zero.
ippStsContextMatchErr	Indicates an error when pointer to the spec structure is invalid.
ippStsStepErr	Indicates an error when the step value is not data type multiple.
ippStsOutOfRangeErr	Indicates an error when the destination image offset point is outside the destination image origin.
ippStsSizeWrn	Indicates a warning when the destination image size is more that the destination image origin size.

See Also

ROI Processing in Geometric Transforms

ResizeGetSrcRoi Computes the ROI of the source image for resizing by tile processing.

ResizeGetSrcOffset Computes the offset of the source image for resizing by tile processing.

ResizeSuperInit Initializes the specification structure for image resizing with the super sampling interpolation method.

ResizeGetBufferSize Computes the size of the external buffer for image resizing.

ResizeAntialiasingLinearInit

Initializes the specification structure for image resizing with antialiasing using linear interpolation.

Syntax

```
IppStatus ippiResizeAntialiasingLinearInit(IppiSize srcSize, IppiSize dstSize,
IppiResizeSpec_32f* pSpec, Ipp8u* pInitBuf);
```

64f equivalent function

```
IppStatus ippiResizeAntialiasingLinearInit_64f(IppiSize srcSize, IppiSize dstSize,
IppiResizeSpec_64f* pSpec, Ipp8u* pInitBuf);
```

Platform-aware function

```
IppStatus ippiResizeAntialiasingLinearInit_L(IppiSizeL srcSize, IppiSizeL dstSize,
IppDataType dataType, IppiResizeSpec* pSpec, Ipp8u* pInitBuf);
```

Threading layer (TL) function

```
IppStatus ippiResizeAntialiasingLinearInit_LT(IppiSizeL srcSize, IppiSizeL dstSize,
IppDataType dataType, Ipp32u numChannels, IppiResizeSpec_LT* pSpec, Ipp8u* pInitBuf);
```

Include Files

ippi.h

Flavors with the _LT suffix: ippi_tl.h

Flavors with the _L suffix: ippi_l.h

Domain Dependencies

Flavors declared in ippi.h:

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Flavors declared in ippi_tl.h:

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib, ippcore_tl.lib, ippi_tl.lib

Parameters

srcSize	Size of the source image, in pixels.
dstSize	Size of the destination image, in pixels.
dataType	Data type of the image. Supported values: ipp8u, ipp16u, ipp16s, ipp32f, ipp64f.
numChannels	Number of image channels. Possible values: 1, 3, or 4.
pSpec	Pointer to the specification structure for the resize filter.
pInitBuf	Pointer to the temporary buffer for initialization of the linear filter.

Description

This function initializes the IppiResizeSpec_32f structure for the resize operation with antialiasing using the linear interpolation method. Note, that ippiResizeAntialiasingLinearInit_64f will initialize the IppiResizeSpec_64f structure.

Before using these functions, calculate the size of the temporary buffer and specification structure using the ippiResizeGetSize function with the antialiasing parameter equal to 1.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsNoOperation	Indicates a warning when width or height of the image is equal to zero.
ippStsSizeErr	Indicates an error if width or height of the source or destination image is negative.
ippStsExceededSizeErr	Indicates an error If width or height of the source or destination image exceeds 33554431 (0x1FFFFFF).
ippStsDataTypeErr	Indicates an error if dataType has an illegal value.

See Also

ResizeGetSize Computes the size of the specification structure and the size of the external temporary buffer for the resize transform initialization.

Linear Interpolation

ResizeAntialiasingCubicInit

Initializes the specification structure for image resizing with antialiasing using interpolation with the two-parameter cubic filters.

Syntax

```
IppStatus ippiResizeAntialiasingCubicInit(IppiSize srcSize, IppiSize dstSize, Ipp32f valueB, Ipp32f valueC, IppiResizeSpec_32f* pSpec, Ipp8u* pInitBuf);
```

64f equivalent function

```
IppStatus ippiResizeAntialiasingCubicInit_64f(IppiSize srcSize, IppiSize dstSize, Ipp64f valueB, Ipp64f valueC, IppiResizeSpec_64f* pSpec, Ipp8u* pInitBuf);
```

Platform-aware function

```
IppStatus ippiResizeAntialiasingCubicInit_L(IppiSizeL srcSize, IppiSizeL dstSize, IppDataType dataType, Ipp32f valueB, Ipp32f valueC, IppiResizeSpec* pSpec, Ipp8u* pInitBuf);
```

Threading layer (TL) function

```
IppStatus ippiResizeAntialiasingCubicInit_LT(IppiSizeL srcSize, IppiSizeL dstSize, IppDataType dataType, Ipp32u numChannels, Ipp32f valueB, Ipp32f valueC, IppiResizeSpec_LT* pSpec, Ipp8u* pInitBuf);
```

Include Files

`ippi.h`

Flavors with the `_LT` suffix: `ippi_tl.h`

Flavors with the `_L` suffix: `ippi_l.h`

Domain Dependencies

Flavors declared in `ippi.h`:

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Flavors declared in `ippi_tl.h`:

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`, `ippcore_tl.lib`, `ippi_tl.lib`

Parameters

`srcSize`	Size of the source image, in pixels.
`dstSize`	Size of the destination image, in pixels.
`valueB`	The first parameter for cubic filters.
`valueC`	The second parameter for cubic filters.

numChannels	Number of image channels. Possible values: 1, 3, or 4.
dataType	Data type of the image. Supported values: ipp8u, ipp16u, ipp16s, ipp32f, ipp64f.
pSpec	Pointer to the specification structure for the resize filter.
pInitBuf	Pointer to the temporary buffer for initialization of the cubic filter.

Description

This function initializes the IppiResizeSpec_32f structure for the resize operation with antialiasing using interpolation with the two-parameter cubic filters. Note, that ippiResizeAntialiasingCubicInit_64f will initialize the IppiResizeSpec_64f structure.

Before using these functions, calculate the size of the temporary buffer and specification structure using the ippiResizeGetSize function with the antialiasing parameter equal to 1.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsNoOperation	Indicates a warning when width or height of the image is equal to zero.
ippStsSizeErr	Indicates an error if width or height of the source or destination image is negative.
ippStsExceededSizeErr	Indicates an error If width or height of the source or destination image exceeds 33554431 (0x1FFFFFF) (only for platform-aware and TL functions).
ippStsDataTypeErr	Indicates an error if dataType has an illegal value.

See Also

ResizeGetSize Computes the size of the specification structure and the size of the external temporary buffer for the resize transform initialization.

Cubic Interpolation

ResizeAntialiasingLanczosInit

Initializes the specification structure for image resizing with antialiasing using interpolation with the Lanczos filter.

Syntax

```
IppStatus ippiResizeAntialiasingLanczosInit(IppiSize srcSize, IppiSize dstSize, Ipp32u numLobes, IppiResizeSpec_32f* pSpec, Ipp8u* pInitBuf);
```

64f equivalent function

```
IppStatus ippiResizeAntialiasingLanczosInit_64f(IppiSize srcSize, IppiSize dstSize,
Ipp32u numLobes, IppiResizeSpec_64f* pSpec, Ipp8u* pInitBuf);
```

Platform-aware function

```
IppStatus ippiResizeAntialiasingLanczosInit_L(IppiSizeL srcSize, IppiSizeL dstSize,
IppDataType dataType, Ipp32u numLobes, IppiResizeSpec* pSpec, Ipp8u* pInitBuf);
```

Threading layer (TL) function

```
IppStatus ippiResizeAntialiasingLanczosInit_LT(IppiSizeL srcSize, IppiSizeL dstSize,
IppDataType dataType, Ipp32u numChannels, Ipp32u numLobes, IppiResizeSpec_LT* pSpec,
Ipp8u* pInitBuf);
```

Include Files

`ippi.h`

Flavors with the `_LT` suffix: `ippi_tl.h`

Flavors with the `_L` suffix: `ippi_l.h`

Domain Dependencies

Flavors declared in `ippi.h`:

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Flavors declared in `ippi_tl.h`:

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`, `ippcore_tl.lib`, `ippi_tl.lib`

Parameters

`srcSize`	Size of the source image, in pixels.
`dstSize`	Size of the destination image, in pixels.
`numLobes`	Number of lobes for the Lanczos window. Possible values: 2 or 3.
`numChannels`	Number of image channels. Possible values: 1, 3, or 4.
`dataType`	Data type of the image. Supported values: `ipp8u`, `ipp16u`, `ipp16s`, `ipp32f`, `ipp64f`.
`pSpec`	Pointer to the specification structure for the resize filter.
`pInitBuf`	Pointer to the temporary buffer for initialization of the cubic filter.

Description

This function initializes the `IppiResizeSpec_32f` structure for the resize operation with antialiasing using interpolation with the Lanczos filter. The Lanczos interpolation method is based on the 2-lobed or 3-lobed Lanczos window function as an interpolation function depending on the value of the `numLobes` parameter. Note, that `ippiResizeAntialiasingLanczosInit_64f` will initialize the `IppiResizeSpec_64f` structure.

Before using these functions, calculate the size of the temporary buffer and specification structure using the ippiResizeGetSize function with the `antialiasing` parameter equal to 1.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when one of the specified pointers is `NULL`.
`ippStsNoOperation`	Indicates a warning when width or height of the image is equal to zero.
ippStsSizeErr	Indicates an error if width or height of the source or destination image is negative.
`ippStsExceededSizeErr`	Indicates an error If width or height of the source or destination image exceeds 33554431 (0x1FFFFFF) (only for platform-aware and TL functions).
`ippStsDataTypeErr`	Indicates an error if `dataType` has an illegal value.

See Also

ResizeGetSize Computes the size of the specification structure and the size of the external temporary buffer for the resize transform initialization.

Lanczos Interpolation

ResizeAntialiasing

Changes an image size using the chosen interpolation method with antialiasing.

Syntax

```
IppStatus ippiResizeAntialiasing_<data_type>_<chan>(const Ipp<data_type>* pSrc, Ipp32s srcStep, Ipp<data_type>* pDst, Ipp32s dstStep, IppiPoint dstOffset, IppiSize dstSize, IppiBorderType border, const Ipp<data_type>* pBorderValue, const IppiResizeSpec_32f* pSpec, Ipp8u* pBuffer);
```

Supported values for `data_type`:

`8u`	`16u`	`16s`	`32f`

Supported values for `chan`:

`C1R`	`C3R`	`C4R`

Also supported `64f_C1R` mode:

```
IppStatus ippiResizeAntialiasing_64f_C1R(const Ipp64f* pSrc, Ipp32s srcStep, Ipp64f* pDst, Ipp32s dstStep, IppiPoint dstOffset, IppiSize dstSize, IppiBorderType border, const Ipp64f* pBorderValue, const IppiResizeSpec_64f* pSpec, Ipp8u* pBuffer);
```

Platform-aware functions

```
IppStatus ippiResizeAntialiasing_<data_type>_<chan>_L(const Ipp<data_type>* pSrc,
IppSizeL srcStep, Ipp<data_type>* pDst, IppSizeL dstStep, IppiPointL dstOffset,
IppiSizeL dstSize, IppiBorderType border, const Ipp<data_type>* pBorderValue, const
IppiResizeSpec* pSpec, Ipp8u* pBuffer);
```

Supported values for `data_type`:

`8u`	`16u`	`16s`	`32f`

Supported values for `chan`:

`C1R`	`C3R`	`C4R`

Threading layer (TL) functions

```
IppStatus ippiResizeAntialiasing_<data_type>_<chan>_LT(const Ipp<data_type>* pSrc,
IppSizeL srcStep, Ipp<data_type>* pDst, IppSizeL dstStep, IppiBorderType border, const
Ipp<data_type>* pBorderValue, const IppiResizeSpec_LT* pSpec, Ipp8u* pBuffer);
```

Supported values for `data_type`:

`8u`	`16u`	`16s`	`32f`

Supported values for `chan`:

`C1R`	`C3R`	`C4R`

Include Files

`ippi.h`

Flavors with the `_LT` suffix: `ippi_tl.h`

Flavors with the `_L` suffix: `ippi_l.h`

Domain Dependencies

Flavors declared in `ippi.h`:

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Flavors declared in `ippi_tl.h`:

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`, `ippcore_tl.lib`, `ippi_tl.lib`

Parameters

`pSrc`	Pointer to the source image.
`srcStep`	Distance, in bytes, between the starting points of consecutive lines in the source image buffer.
`pDst`	Pointer to the destination image.
`dstStep`	Distance, in bytes, between the starting points of consecutive lines in the destination image buffer.
`dstOffset`	Offset of the tiled destination image with respect to the destination image origin.

dstSize	Size of the destination image, in pixels.
border	Type of border. Possible values are: • ippBorderRepl • Border is replicated from the edge pixels. • ippBorderConst • Border pixels are set to constants. • ippBorderInMem • Border is obtained from the source image pixels in memory. • ippBorderMirror • Border is obtained from the source image boundary pixels. Value can • ippBorderMirrorR • Border is obtained from the source image boundary pixels. The anchor
pBorderValue	Pointer to the constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.
pSpec	Pointer to the specification structure for the resize filter.
pBuffer	Pointer to the external buffer.

Description

The ippiResizeAntialiasing function changes the size of an image using the chosen interpolation method with antialiasing. The interpolation method to be applied is defined by the function that you use for the resize filter initialization. Use this function to reduce the image size with minimization of moire artifacts. For more information about the implemented algorithm, refer to [SCHU92].

If you use ippiResizeAntialiasing to increase the image size, the function applies the same algorithm as one of the following resize functions, depending on the interpolation type chosen at the initialization stage: ippiResizeLinear, ippiResizeCubic, or ippiResizeLanczos.

This function operates with ROI. It resizes the source image ROI origin to the destination image ROI origin. Define the destination image ROI origin by the following parameters: the offset of the tiled destination image with respect to the destination image origin and the destination image size. The source image ROI origin is defined automatically. To obtain the source image ROI, use the corresponding flavor of the ippiResizeGetSrcRoi function. To obtain the source image ROI origin offset, call the corresponding flavor of the ippiResizeGetSrcOffset function. Parameters pSrc and pDst must point to the processed source and destination image ROI origins, respectively.

The interpolation algorithm applied uses edge pixels of the source image that are out of the image origin. The ippiResizeAntialiasing function uses the weighted values of these outer pixels. To obtain the size of the out of the source image origin, call the corresponding flavor of the ippiResizeGetBorderSize function.

Before using the ippiResizeAntialiasing function, you need to initialize the IppiResizeSpec_32f structure using one of the following functions, depending on the interpolation method to be applied: ippiResizeAntialiasingLinearInit, ippiResizeAntialiasingCubicInit, or ippiResizeAntialiasingLanczosInit, and compute the size of the external buffer pBuffer using the corresponding flavor of the ippiResizeGetBufferSize function. The same instructions work for ippiResizeAntialiasing_64f, but with 64f equivalent init functions.

For more information about the supported border types, see User-defined Border Types.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsNoOperation	Indicates a warning when width or height of the destination image is equal to zero.
ippStsBorderErr	Indicates an error when the border value is illegal.
ippStsContextMatchErr	Indicates an error when a pointer to the specification structure is invalid.
ippStsNotSupportedModeErr	Indicates an error when the requested mode is not supported.
ippStsSizeErr	Indicates an error when width or height of the destination image is negative.
ippStsStepErr	Indicates an error when the step value is not data type multiple.
ippStsOutOfRangeErr	Indicates an error when the destination image offset point is outside the destination image origin.
ippStsSizeWrn	Indicates a warning when the destination image size is more than the destination image origin size.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

// A simple example of changing an image size by different interpolation
// methods with antialiasing technique using Intel(R) Integrated Performance Primitives
(Intel(R) IPP) functions:
//     ippiResizeGetSize_8u
//     ippiResizeAntialiasingLinearInit
//     ippiResizeGetBufferSize_8u
//     ippiResizeAntialiasing_8u_C3R

#include <stdio.h>
```

```
#include "ipp.h"

#define SRC_WIDTH  128 /* source image width */
#define SRC_HEIGHT 64  /* source image height */
#define DST_WIDTH  64  /* source image width */
#define DST_HEIGHT 128 /* source image height */

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define check_sts(st)        \
    if ((st) != ippStsNoErr) \
        goto exitLine; /* Go to Exit if Intel(R) IPP function returned status different from
ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main(void)
{
    IppStatus status = ippStsNoErr;
    Ipp8u *pSrc = NULL, *pDst = NULL;           /* Pointers to source/destination images */
    int srcStep = 0, dstStep = 0;               /* Steps, in bytes through the source/
destination images */
    IppiSize srcSize = {SRC_WIDTH, SRC_HEIGHT}; /* Size of source ROI in pixels */
    IppiSize dstSize = {DST_WIDTH, DST_HEIGHT}; /* Size of destination ROI in pixels */
    Ipp8u *pBuffer = NULL;                      /* Pointer to the work buffer */
    IppiResizeSpec_32f *pSpec = NULL;
    Ipp8u *pInit = NULL;
    int specSize = 0, initSize = 0, bufSize = 0;
    IppiPoint dstOffset = {0, 0};
    IppiInterpolationType interpolation = ippLinear;
    IppiBorderType borderType = ippBorderRepl;

    pSrc = ippiMalloc_8u_C3(srcSize.width, srcSize.height, &srcStep);
    pDst = ippiMalloc_8u_C3(dstSize.width, dstSize.height, &dstStep);

    /* Calculation of work buffer size */
    check_sts(status = ippiResizeGetSize_8u(srcSize, dstSize, interpolation, 1, &specSize,
&initSize));

    /* Memory allocation */
    pSpec = (IppiResizeSpec_32f *)ippsMalloc_8u(specSize + initSize);

    pInit = (Ipp8u *)pSpec + initSize;

    /* Filter initialization */
    check_sts(status = ippiResizeAntialiasingLinearInit(srcSize, dstSize, pSpec, pInit));

    check_sts(status = ippiResizeGetBufferSize_8u(pSpec, dstSize, 3, &bufSize));

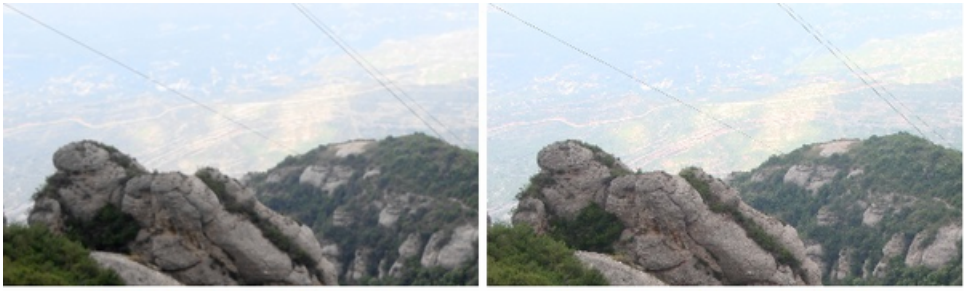
    pBuffer = ippsMalloc_8u(bufSize);

    /* Function call */
    check_sts(status = ippiResizeAntialiasing_8u_C3R(pSrc, srcStep, pDst, dstStep, dstOffset,
dstSize, borderType, 0, pSpec, pBuffer));

    EXIT_MAIN
```

```
    ippiFree(pSrc);
    ippiFree(pDst);
    ippsFree(pSpec);
    ippsFree(pBuffer);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

The figure below demonstrates the results of reducing a 1751x1044 image by five times with antialiasing (a) and without (b).



a) resize with antialiasing b) resize without antialiasing

See Also

ROI Processing in Geometric Transforms

User-defined Border Types

ResizeGetSrcRoi Computes the ROI of the source image for resizing by tile processing.

ResizeGetSrcOffset Computes the offset of the source image for resizing by tile processing.

ResizeGetBorderSize Computes the size of possible borders for the resize transform.

ResizeGetBufferSize Computes the size of the external buffer for image resizing.

ResizeAntialiasingLinearInit Initializes the specification structure for image resizing with antialiasing using linear interpolation.

ResizeAntialiasingCubicInit Initializes the specification structure for image resizing with antialiasing using interpolation with the two-parameter cubic filters.

ResizeAntialiasingLanczosInit Initializes the specification structure for image resizing with antialiasing using interpolation with the Lanczos filter.

ResizeFilterGetSize

Calculates the size of the state structure for resizing filter.

Syntax

```
IppStatus ippiResizeFilterGetSize_8u_C1R(IppiSize srcRoiSize, IppiSize dstRoiSize,
IppiResizeFilterType filter, Ipp32u* pSize);
```

Include Files

```
ippi.h
```

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`srcRoiSize`	Size of the source image ROI in pixels.
`dstRoiSize`	Size of the destination image ROI in pixels.
`filter`	Type of filter used in resizing; possible values: `ippResizeFilterHann`, `ippResizeFilterLanczos`.
`pSize`	Pointer to the size (in bytes) of the state structure.

Description

This function operates with ROI (see ROI Processing in Geometric Transforms).

This function calculates the size `pSize` of the state structure required for the function ippiResizeFilter. The type of filter is specified by the parameter `filter`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if the `pSize` pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `srcRoiSize` or `dstRoiSize` has a field with zero or negative value.
`ippStsNotSupportedModeErr`	Indicates an error condition if `filter` has an invalid value.

ResizeFilterInit

Initializes the state structure for the resize filter.

Syntax

```
IppStatus ippiResizeFilterInit_8u_C1R(IppiResizeFilterState* pState, IppiSize
srcRoiSize, IppiSize dstRoiSize, IppiResizeFilterType filter);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

pState	Pointer to the state structure for resize filter.
srcRoiSize	Size of the source image ROI in pixels.
dstRoiSize	Size of the destination image ROI in pixels.
filter	Type of filter used in resizing; possible values: ippResizeFilterHann, ippResizeFilterLanczos

Description

This function operates with ROI (see ROI Processing in Geometric Transforms).

This function initializes the state structure pState for the resizing filter used by the function ippiResizeFilter. The size of the structure must be computed by the function ippiResizeFilterGetSize beforehand. The type of filter is specified by the parameter filter and it must be the same as in the function ippiResizeFilterGetSize.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if the pState pointer is NULL.
ippStsSizeErr	Indicates an error condition if srcRoiSize or dstRoiSize has a field with zero or negative value.
ippStsNotSupportedModeErr	Indicates an error condition if filter has an invalid value.

ResizeFilter

Changes the size of an image using a generic filter.

Syntax

```
IppStatus ippiResizeFilter_8u_C1R(const Ipp8u* pSrc, int srcStep, IppiSize srcRoiSize,
Ipp8u* pDst, int dstStep, IppiSize dstRoiSize, IppiResizeFilterState* pState);
```

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image buffer.
srcRoiSize	Size of the source image ROI in pixels.
pDst	Pointer to the destination image ROI.
dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.
dstRoiSize	Size of the destination image ROI in pixels.
pState	Pointer to the state structure for the resize filter.

Description

This function operates with ROI (see ROI Processing in Geometric Transforms).

This function resizes the source image pSrc using the special generic filters. The state structure pState contains the parameters of filtering and must be initialized by the function ippiResizeFilterInit beforehand.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsStepErr	Indicates an error condition if srcStep or dstStep has a zero or negative value.
ippStsSizeErr	Indicates an error condition if srcRoiSize or dstRoiSize has a field with zero or negative value.
ippStsContextMatchErr	Indicates an error condition if a pointer to an invalid state structure is passed.

ResizeYUV420GetSize

Computes sizes of the spec structure and the external buffer for the NV12 resize transform initialization.

Syntax

```
IppStatus ippiResizeYUV420GetSize(IppiSize srcSize, IppiSize dstSize,
IppiInterpolationType interpolation, Ipp32u antialiasing, Ipp32s* pSpecSize, Ipp32s*
pInitBufSize);
```

Include Files

ippi.h

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`srcSize`	Size of the source image in pixels.
`dstSize`	Size of the destination image in pixels.
`interpolation`	Interpolation method. Supported values are `ippLanczos` and `ippSuper`.
`antialiasing`	Antialiasing method.
`pSpecSize`	Pointer to the size in bytes of the spec structure.
`pInitBufSize`	Pointer to the size in bytes of the temporal buffer.

Description

This function computes sizes of the spec structure and the external buffer that are required for one of the following functions depending on the interpolation method parameter: ResizeYUV420LanczosInit and ResizeYUV420SuperInit.

The size of the 2-Lobed Lanczos filter is 8x8.

NOTE Antialising is currently not supported. The value for the `antialiasing` parameter must be equal to zero.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsNoOperation`	Indicates a warning if width or height of the image is equal to zero.
`ippStsSizeErr`	Indicates an error in the following cases: • if width or height of the image is equal to 1, • if the source image size is less than a filter size of the chosen interpolation method (except `ippSuper`), • if one of the specified dimensions of the source image is less than the corresponding dimension of the destination image (for `ippSuper` method only),

	• if width or height of the source or destination image is negative, • if one of the calculated sizes exceeds maximum 32 bit signed integer positive value (the size of the one of the processed images is too large). if width or height of the source or destination image is negative.
`ippStsSizeWrn`	Indicates a warning if width or height of the image is odd.
`ippStsInterpolationErr`	Indicates an error if `interpolation` has an illegal value.
`ippStsNoAntialiasing`	Indicates a warning if the specified interpolation method does not support antialiasing.
`ippStsNotSupportedModeErr`	Indicates an error if the requested mode is currently not supported.

ResizeYUV420GetSrcRoi

Computes the ROI of the source image for NV12 resizing by tile processing.

Syntax

Single precision

```
IppStatus ippiResizeYUV420GetSrcRoi(const IppiResizeYUV420Spec* pSpec, IppiPoint
dstRoiOffset, IppiSize dstRoiSize, IppiPoint* srcRoiOffset, IppiSize* srcRoiSize);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSpec`	Pointer to the spec structure for the resize filter.
`dstRoiOffset`	Offset of the tiled destination image ROI.
`dstRoiSize`	Size of the tiled destination image ROI.
`srcRoiOffset`	Offset of the source image ROI.
`srcRoiSize`	Pointer to the size of the source image ROI.

Description

This function computes the ROI of the processed source image using the processed ROI of the destination image for the corresponding resize transform by tile processing. The `pSpec` parameter defines the resize algorithm parameters. Prior to using the `ippiResizeYUV420GetSrcRoi` function, you need to initialize the `pSpec` parameter by calling one of the following functions: ippiResizeYUV420LanczosInit or ippiResizeYUV420SuperInit functions.

NOTE If the destination ROI size exceeds the image origin, the source ROI will be obtained for an intersection of the destination ROI and image origin.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if one of the specified pointers is `NULL`.
`ippStsContextMatchErr`	Indicates an error if pointer to the spec structure is invalid.
`ippStsOutOfRangeErr`	Indicates an error if the destination image offset point is outside the destination image origin.
`ippStsMisalignedOffsetErr`	Indicates an error if one of the fields of the `dstRoiOffset` parameter is odd.
`ippStsSizeErr`	Indicates an error if one of the fields of the `dstRoiSize` is less than 2.
`ippStsSizeWrn`	Indicates a warning if the destination ROI exceeds the destination image origin or contains odd values.

ResizeYUV420LanczosInit

Initializes the spec structure for the NV12 resize transform by interpolation with the Lanczos filter.

Syntax

```
IppStatus ippiResizeYUV420LanczosInit(IppiSize srcSize, IppiSize dstSize, Ipp32u numLobes, IppiResizeYUV420Spec* pSpec, Ipp8u* pInitBuf);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

srcSize	Size in pixels of the source image.
dstSize	Size in pixels of the destination image.
numLobes	Parameter for Lanczos filters. Possible values are 2 or 3.
pSpec	Pointer to the spec structure for the resize filter.
pInitBuf	Pointer to the temporal buffer for the cubic filter initialization.

Description

This function initializes the IppiResizeYUV420Spec structure for the resize algorithm with the Lanczos filter interpolation method. This method is based on the 2-lobed or 3-lobed Lanzos window function as an interpolation function depending on the value of the numLobes parameter.

To calculate the size of the spec structure object, call the ippiResizeYUV420GetSize function.

The function ippiResizeYUV420LanczosInit requires the external buffer pInitBuf. Prior to using this function, you need to call ippiResizeYUV420GetSize for the corresponding flavors to compute the size of the buffer.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if one of the specified pointers is NULL.
ippStsResizeNoOperation	Indicates an error if width or height of the image is equal to zero.
ippStsSizeWrn	Indicates a warning if width or height of the image is odd.
ippStsSizeErr	Indicates an error in the following cases: • if width or height of the source or destination image is equal to 1, • if width or height of the source or destination image is negative, • if the source image size is less than the Lanczos interpolation filter size: 8x8 for 2-lobed Lanczos function, or 12x12 for 3-lobed Lanczos function.
ippStsNotSupportedModeErr	Indicates an error if the requested mode is not supported.

ResizeYUV420SuperInit

Initializes the spec structure for the NV12 resize transform by interpolation with the super sampling algorithm.

Syntax

```
IppStatus ippiResizeYUV420SuperInit(IppiSize srcSize, IppiSize dstSize,
IppiResizeYUV420Spec* pSpec);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`srcSize`	Size in pixels of the source image.
`dstSize`	Size in pixels of the destination image.
`pSpec`	Pointer to the spec structure for the resize filter.

Description

This function initializes the `IppiResizeYUV420Spec` structure for the resize algorithm using interpolation with the super sampling algorithm.

To calculate the size of the spec structure object, call the ippiResizeYUV420GetSize function.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if one of the specified pointers is `NULL`.
`ippStsResizeNoOperation`	Indicates an error if width or height of the image is equal to zero.
`ippStsSizeWrn`	Indicates a warning if width or height of the image is odd.
`ippStsSizeErr`	Indicates an error in the following cases:

ResizeYUV420GetBorderSize

Computes the size of possible borders for the NV12 resize transform.

Syntax

```
IppStatus ippiResizeYUV420GetBorderSize(const IppiResizeYUV420Spec* pSpec,
IppiBorderSize* borderSize);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSpec`	Pointer to the spec structure for the resize filter.
`borderSize`	Size in pixels of necessary borders.

Description

This function computes the size of the source image ROI that is used by the corresponding resize transform and is out of the processing boundaries for Luma and Chroma planes. The `pSpec` parameter defines the resize algorithm parameters. Prior to using the `ippiResizeYUV420GetBorderSize` function, you need to initialize the `pSpec` parameter by calling one of the following functions: ippiResizeYUV420LanczosInit or ippiResizeYUV420SuperInit.

NOTE The returned border size is in Luma/Chroma plane pixels. This means that the chosen resize algorithm uses the returned outer size of the source image ROI for each plane.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if one of the specified pointers is `NULL`.
`ippStsContextMatchErr`	Indicates an error if pointer to the spec structure is invalid.

ResizeYUV420GetSrcOffset

Computes the offset of the source image for the NV12 resize transform by tile processing.

Syntax

```
IppStatus ippiResizeYUV420GetSrcOffset(const IppiResizeYUV420Spec* pSpec, IppiPoint dstOffset, IppiPoint* srcOffset);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

pSpec	Pointer to the spec structure for the resize filter.
dstOffset	Offset of the tiled destination image with respect to the destination image origin.
srcOffset	Offset of the source image.

Description

This function computes the offset of the processed source image ROI using the offset of the processed destination image ROI for the corresponding resize transform by tile processing. The pSpec parameter defines the resize algorithm parameters. Prior to using the ippiResizeGetSrcOffset function, you need to initialize the pSpec parameter by calling one of the following functions: ippiResizeYUV420LanczosInit and ippiResizeYUV420SuperInit.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if one of the specified pointers is NULL.
ippStsContextMatchErr	Indicates an error if pointer to the spec structure is invalid.
ippStsOutOfRangeErr	Indicates an error if the destination imageoffset point is outside the destination image origin.
ippStsMisalignedOffsetErr	Indicates an error if one of the fields of the dstOffset parameter is odd.

ResizeYUV420GetBufferSize

Computes the size of the external buffer for the NV12 resize transform.

Syntax

```
IppStatus ippiResizeYUV420GetBufferSize(const IppiResizeYUV420Spec* pSpec, IppiSize dstSize, Ipp32s* pBufSize);
```

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSpec	Pointer to the spec structure for the resize filter.

dstSize	Size in pixels of the destination image.
pBufSize	Pointer to the size in bytes of the external buffer.

Description

This function computes the size of the external buffer for the NV12 resize transform. The pSpec parameter defines the resize algorithm parameters. Prior to using the ippiResizeYUV420GetBufferSize function, you need to initialize the pSpec parameter by calling one of the following functions: ippiResizeYUV420LanczosInit and ippiResizeYUV420SuperInit.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if pBufferSize pointer is NULL.
ippStsNoOperation	Indicates a warning if width or height of the destination image is equal to zero.
ippStsContextMatchErr	Indicates an error if pointer to the spec structure is invalid.
ippStsSizeWrn	Indicates a warning in the following cases: • if width or height of the image is odd, • if the destination image size is more than the destination image origin size.
ippStsSizeErr	Indicates an error in the following cases:

ResizeYUV420Lanczos

Changes the size of the NV12 image by interpolation with the Lanczos filter.

Syntax

```
IppStatus ippiResizeYUV420Lanczos_8u_P2R(const Ipp8u* pSrcY, Ipp32s srcYStep, const Ipp8u* pSrcUV, Ipp32s srcUVStep, Ipp8u* pDstY, Ipp32s dstYStep, Ipp8u* pDstUV, Ipp32s dstUVStep, IppiPoint dstOffset, IppiSize dstSize, IppiBorderType border, const Ipp8u* pBorderValue, const IppiResizeYUV420Spec* pSpec, Ipp8u* pBuffer);
```

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrcY	Pointer to the source image Y plane.
srcYStep	Distance in bytes between starts of consecutive lines in the source image Y plane.
pSrcUV	Pointer to the source image UV plane.
srcUVStep	Distance in bytes between starts of consecutive lines in the source image UV plane.
pDstY	Pointer to the destination image Y plane.
dstYStep	Distance in bytes between starts of consecutive lines in the destination image Y plane.
pDstUV	Pointer to the destination image UV plane.
dstUVStep	Distance in bytes between starts of consecutive lines in the destination image UV plane.
dstOffset	Offset of the tiled destination image with respect to the destination image origin.
dstSize	Size of the destination image in pixels.
border	Type of the border.
pBorderValue	Pointer to the constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.
pSpec	Pointer to the spec structure for the resize filter.
pBuffer	Pointer to the work buffer.

Description

This function changes an image size using interpolation with the Lanczos filter. The image size can be either reduced or increased in each direction, depending on the destination image size.

This function operates with ROI (see ROI Processing in Geometric Transforms). It resizes the source image ROI origin to the destination image ROI origin. The destination image ROI origin must be defined by the following parameters: the offset of the tiled destination image with respect to the destination image origin and the destination image size. The source image ROI origin is defined automatically. To obtain the source image ROI origin offset, call the ippiResizeYUV420GetSrcOffset function. Parameters pSrcY, pSrcUV and pDstY, pDstUV must point to the processed source and destination image ROI origins respectively.

The source and destination images are in the 4:2:0 two-plane image format (NV12): all Y samples (pSrcY) are found first in memory as an array of unsigned chars with an even number of lines memory alignment, followed by an array (pSrcY) of unsigned chars containing interleaved U and V samples. Supported values for border are ippBorderRepl and ippBorderInMem.

Applied interpolation algorithm uses edge pixels of the source image that are out of the image origin. The function ippiResizeYUV420Lanczos uses in calculation the weighted values of these outer pixels. To obtain the size of the out of the source image origin, call the function ippiResizeYUV420GetBorderSize for the corresponding flavor.

If the border type is equal to `ippBorderRepl`, the source image edge pixels are replicated out of the image origin. If the border type is equal to `ippBorderInMem`, the outer pixels are obtained from the out of the source image origin space. For the mixed border types, the combined approach is applied.

Prior to using the `ippiResizeLinear` function, initialize the `IppiResizeYUV420Spec` structure by calling the ippiResizeYUV420LanczosInit and compute the size of the external buffer `pBuffer` by calling the ippiResizeYUV420GetBufferSize for the corresponding flavor.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if one of the specified pointers is `NULL`.
`ippStsNoOperation`	Indicates a warning if width or height of the destination image is equal to zero.
`ippStsBorderErr`	Indicates an error if the `border` value is illegal.
`ippStsContextMatchErr`	Indicates an error if pointer to the spec structure is invalid.
`ippStsMisalignedOffsetErr`	Indicates an error if one of the fields of the `dstOffset` parameter is odd.
`ippStsSizeWrn`	Indicates a warning in the following cases: • if width of the image is odd, • if the destination image size is more than the destination image origin.
`ippStsSizeErr`	Indicates an error if width of the destination image is equal to 1, or if width or height of the source or destination image is negative.
`ippStsOutOfRangeErr`	Indicates an error if the destination imageoffset point is outside the destination image origin.
`ippStsNotSupportedModeErr`	Indicates an error if the requested mode is not supported.

ResizeYUV420Super

Changes the size of the NV12 image by the super sampling interpolation method.

Syntax

```
IppStatus ippiResizeYUV420Super_8u_P2R(const Ipp8u* pSrcY, Ipp32s srcYStep, const Ipp8u* pSrcUV, Ipp32s srcUVStep, Ipp8u* pDstY, Ipp32s dstYStep, Ipp8u* pDstUV, Ipp32s dstUVStep, IppiPoint dstOffset, IppiSize dstSize, const IppiResizeYUV420Spec* pSpec, Ipp8u* pBuffer);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrcY`	Pointer to the source image Y plane.
`srcYStep`	Distance in bytes between starts of consecutive lines in the source image Y plane.
`pSrcUV`	Pointer to the source image UV plane.
`srcUVStep`	Distance in bytes between starts of consecutive lines in the source image UV plane.
`pDstY`	Pointer to the destination image Y plane.
`dstYStep`	Distance in bytes between starts of consecutive lines in the destination image Y plane.
`pDstUV`	Pointer to the destination image UV plane.
`dstUVStep`	Distance in bytes between starts of consecutive lines in the destination image UV plane.
`dstOffset`	Offset of the tiled destination image with respect to the destination image origin.
`dstSize`	Size of the destination image in pixels.
`pSpec`	Pointer to the spec structure for the resize filter.
`pBuffer`	Pointer to the work buffer.

Description

This function changes an image size using interpolation with the super sampling algorithm. The image size can be either reduced or increased in each direction, depending on the destination image size.

This function operates with ROI (see ROI Processing in Geometric Transforms). It resizes the source image ROI origin to the destination image ROI origin. The destination image ROI origin must be defined by the following parameters: the offset of the tiled destination image with respect to the destination image origin and the destination image size. The source image ROI origin is defined automatically. To obtain the source image ROI origin offset, call the ippiResizeYUV420GetSrcOffset function. Parameters `pSrcY`, `pSrcUV` and `pDstY`, `pDstUV` must point to the processed source and destination image ROI origins respectively.

The interpolation algorithm applied uses only pixels of the source image origin that are inside of the image boundaries.

The source and destination images are in the 4:2:0 two-plane image format (NV12): all Y samples (`pSrcY`) are found first in memory as an array of unsigned chars with an even number of lines memory alignment, followed by an array (`pSrcY`) of unsigned chars containing interleaved U and V samples.

Prior to using the `ippiResizeLinear` function, initialize the `IppiResizeYUV420Spec` structure by calling the ippiResizeYUV420LanczosInit and compute the size of the external buffer `pBuffer` by calling the ippiResizeYUV420GetBufferSize for the corresponding flavor.

NOTE This function provides optimized code paths for the following scaling factors along the `x` and `y` axes: 1/2, 2/3, 3/4, 4/5, 5/6, 8/9, 1/3, 2/5, 3/5, 3/7, 4/9, 7/10, 1/4, 2/7, 3/8, 1/8.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if one of the specified pointers is `NULL`.
`ippStsNoOperation`	Indicates a warning if width or height of the destination image is equal to zero.
`ippStsBorderErr`	Indicates an error if the `border` value is illegal.
`ippStsContextMatchErr`	Indicates an error if pointer to the spec structure is invalid.
`ippStsSizeWrn`	Indicates a warning in the following cases: • if width of the image is odd, • if the destination image size is more than the destination image origin.
`ippStsSizeErr`	Indicates an error if width or height of the destination image is equal to 1; or if width or height of the source or destination image is negative.
`ippStsMisalignedOffsetErr`	Indicates an error if one of the fields of the `dstOffset` parameter is odd.
`ippStsOutOfRangeErr`	Indicates an error if the destination imageoffset point is outside the destination image origin.

ResizeYUV422GetSize

Computes sizes of the spec structure and the external buffer for YUY2 resize transform initialization.

Syntax

```
IppStatus ippiResizeYUV422GetSize(IppiSize srcSize, IppiSize dstSize,
IppiInterpolationType interpolation, Ipp32u antialiasing, Ipp32s* pSpecSize, Ipp32s*
pInitBufSize);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

srcSize	Size of the source image in pixels.
dstSize	Size of the destination image in pixels.
interpolation	Interpolation method. Supported values are ippNearest and ippLinear.
antialiasing	Antialiasing method.
pSpecSize	Pointer to the size in bytes of the spec structure.
pInitBufSize	Pointer to the size in bytes of the temporal buffer.

Description

This function computes sizes of the spec structure and the external buffer that are required for one of the following functions depending on the interpolation method parameter: ResizeYUV422NearestInit and ResizeYUV422LinearInit.

The filter sizes of the Nearest Neighbor and Linear interpolation algorithms are 2x1 and 4x2 respectively.

NOTE Antialiasing is currently not supported. The value for the antialiasing parameter must be equal to zero.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsNoOperation	Indicates a warning if width or height of the image is equal to zero.
ippStsSizeErr	Indicates an error in the following cases: • if the source image size is less than the filter size for the chosen interpolation method, • if one of the calculated sizes exceeds maximum 32 bit signed integer positive value (the size of one of the processed images is too large).
ippStsSizeWrn	Indicates a warning if width of the image is odd.
ippStsInterpolationErr	Indicates an error if interpolation has an illegal value.
ippStsNoAntialiasing	Indicates a warning if the specified interpolation method does not support antialiasing.
ippStsNotSupportedModeErr	Indicates an error if the requested mode is currently not supported.

ResizeYUV422GetBorderSize

Computes the size of possible borders for the YUY2 resize transform.

Syntax

```
IppStatus ippiResizeYUV422GetBorderSize(const IppiResizeYUV422Spec* pSpec,
IppiBorderSize* borderSize);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSpec`	Pointer to the spec structure for the resize filter.
`borderSize`	Size in pixels of necessary borders.

Description

This function computes the size of the source image ROI that is used by the corresponding resize transform and is out of the processing boundaries. The `pSpec` parameter defines the resize algorithm parameters. Prior to using the `ippiResizeYUV422GetBorderSize` function, you need to initialize the `pSpec` parameter by calling one of the following functions: ippiResizeYUV422NearestInit, and ippiResizeYUV422LinearInit.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if one of the specified pointers is `NULL`.
`ippStsContextMatchErr`	Indicates an error if pointer to the spec structure is invalid.

ResizeYUV422GetSrcOffset

Computes the offset of the source image for the YUY2 resize transform by tile processing.

Syntax

```
IppStatus ippiResizeYUV422GetSrcOffset(const IppiResizeYUV422Spec* pSpec, IppiPoint
dstOffset, IppiPoint* srcOffset);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSpec`	Pointer to the spec structure for the resize filter.
`dstOffset`	Offset of the tiled destination image with respect to the destination image origin.
`srcOffset`	Offset of the source image.

Description

This function computes the offset of the processed source image ROI using the offset of the processed destination image ROI for the corresponding resize transform by tile processing. The `pSpec` parameter defines the resize algorithm parameters. Prior to using the `ippiResizeYUV422GetSrcOffset` function, you need to initialize the `pSpec` parameter by calling one of the following functions: ippiResizeYUV422NearestInit and ippiResizeYUV422LinearInit.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if one of the specified pointers is `NULL`.
`ippStsMisalignedOffsetErr`	Indicates an error if the `x` field of the `dstOffset` parameter is odd.
`ippStsOutOfRangeErr`	Indicates an error if the destination imageoffset point is outside the destination image origin.
`ippStsContextMatchErr`	Indicates an error if pointer to the spec structure is invalid.

ResizeYUV422GetBufSize

Computes the size of the external buffer for the NV12 resize transform.

Syntax

```
IppStatus ippiResizeYUV422GetBufSize(const IppiResizeYUV422Spec* pSpec, IppiSize dstSize, Ipp32s* pBufSize);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

pSpec	Pointer to the spec structure for the resize filter.
dstSize	Size in pixels of the destination image.
pBufSize	Pointer to the size in bytes of the external buffer.

Description

This function computes the size of the external buffer for the YUY2 resize transform. The pSpec parameter defines the resize algorithm parameters. Prior to using the ippiResizeYUV420GetBufferSize function, you need to initialize the pSpec parameter by calling one of the following functions: ippiResizeYUV422NearestInit and ippiResizeYUV422LinearInit.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if one of the specified pointers is NULL.
ippStsNoOperation	Indicates a warning if width or height of the destination image is equal to zero.
ippStsContextMatchErr	Indicates an error if pointer to the spec structure is invalid.
ippStsSizeWrn	Indicates a warning in the following cases: • if width of the image is odd, • if the destination image size is more than the destination image origin size.
ippStsSizeErr	Indicates an error in the following cases:

ResizeYUV422GetSrcRoi

Computes the ROI of the source image for YUV422 resizing by tile processing.

Syntax

```
IppStatus ippiResizeYUV422GetSrcRoi(const IppiResizeYUV422Spec* pSpec, IppiPoint
dstRoiOffset, IppiSize dstRoiSize, IppiPoint* srcRoiOffset, IppiSize* srcRoiSize);
```

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSpec	Pointer to the spec structure for the resize filter.
dstRoiOffset	Offset of the tiled destination image ROI.
dstRoiSize	Size of the tiled destination image ROI.
srcRoiOffset	Offset of the source image ROI.
srcRoiSize	Pointer to the size of the source image ROI.

Description

This function computes the ROI of the processed source image using the processed ROI of the destination image for the corresponding resize transform by tile processing. The pSpec parameter defines the resize algorithm parameters. Prior to using the ippiResizeYUV422GetSrcRoi function, you need to initialize the pSpec parameter by calling one of the following functions: ippiResizeYUV422NearestInit or ippiResizeYUV422LinearInit.

NOTE If the destination ROI size exceeds the image origin, the source ROI will be obtained for an intersection of the destination ROI and image origin.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if one of the specified pointers is NULL.
ippStsContextMatchErr	Indicates an error if pointer to the specification structure is invalid.
ippStsOutOfRangeErr	Indicates an error if the destination image offset point is outside the destination image origin.
ippStsMisalignedOffsetErr	Indicates an error if x-value of the parameter dstRoiOffset is odd.
ippStsSizeErr	Indicates an error in the following cases: • If the height of the destination ROI is zero or negative. • If the width of the destination ROI is less than 2.
ippStsSizeWrn	Indicates a warning in the following cases:

ResizeYUV422NearestInit

Initializes the spec structure for the YUY2 resize transform by the nearest neighbor interpolation method.

Syntax

```
IppStatus ippiResizeYUV422NearestInit(IppiSize srcSize, IppiSize dstSize,
IppiResizeYUV422Spec* pSpec);
```

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

srcSize	Size in pixels of the source image.
dstSize	Size in pixels of the destination image.
pSpec	Pointer to the spec structure for the resize filter.

Description

This function initializes the IppiResizeYUV422Spec structure for the resize algorithm with the nearest neighbor interpolation method. To calculate the size of the spec structure object, call the ippiResizeYUV422GetSize function.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if one of the specified pointers is NULL.
ippStsNoOperation	Indicates a warning if width or height of the image is equal to zero.
ippStsSizeWrn	Indicates a warning if width of the image is odd.
ippStsSizeErr	Indicates an error if width of the image is equal to 1, or if width or height of the source or destination image is negative.

ResizeYUV422LinearInit

Initializes the spec structure for the YUY2 resize transform by the linear interpolation method.

Syntax

```
IppStatus ippiResizeYUV422LinearInit(IppiSize srcSize, IppiSize dstSize,
IppiResizeYUV422Spec* pSpec);
```

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

srcSize	Size in pixels of the source image.
dstSize	Size in pixels of the destination image.
pSpec	Pointer to the spec structure for the resize filter.

Description

This function initializes the IppiResizeYUV422Spec structure for the resize algorithm with the linear interpolation method. To calculate the size of the spec structure object, call the ippiResizeYUV422GetSize function.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if one of the specified pointers is NULL.
ippStsNoOperation	Indicates a warning if width or height of the image is equal to zero.
ippStsSizeWrn	Indicates a warning if width of the image is odd.
ippStsSizeErr	Indicates an error in the following cases:

ResizeYUV422Nearest

Changes an YUY2 image size by the nearest neighbor interpolation method.

Syntax

```
IppStatus ippiResizeYUV422Nearest_8u_C2R(const Ipp8u* pSrc, Ipp32s srcStep, Ipp8u*
pDst, Ipp32s dstStep, IppiPoint dstOffset, IppiSize dstSize, const
IppiResizeYUV422Spec* pSpec, Ipp8u* pBuffer);
```

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc	Pointer to the source image.
srcStep	Distance in bytes between starts of consecutive lines in the source image buffer.
pDst	Pointer to the destination image.

dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.
dstOffset	Offset of the tiled destination image with respect to the destination image origin.
dstSize	Size of the destination image in pixels.
pSpec	Pointer to the spec structure for the resize filter.
pBuffer	Pointer to the work buffer.

Description

This function changes an image size using the nearest neighbor interpolation method. The image size can be either reduced or increased in each direction, depending on the destination image size.

This function operates with ROI (see ROI Processing in Geometric Transforms). It resizes the source image ROI origin to the destination image ROI origin. The destination image ROI origin must be defined by the following parameters: the offset of the tiled destination image with respect to the destination image origin and the destination image size. The source image ROI origin is defined automatically. To obtain the source image ROI origin offset, call the ippiResizeYUV422GetSrcOffset function. Parameters pSrc and pDst must point to the processed source and destination image ROI origins respectively.

The source and destination images are in the YUY2 pixel format (Y0U0Y1V0,Y2U1Y3V1,.. or Y0Cb0Y1Cr0,Y2Cb1Y3Cr1,..).

The interpolation algorithm applied uses only pixels of the source image origin that are inside of the image boundaries.

Prior to using the ippiResizeYUV422Nearest function, initialize the IppiResizeYUV422Spec structure by calling the ippiResizeYUV422NearestInit and compute the size of the external buffer pBuffer by calling the ippiResizeYUV422GetBufSize for the corresponding flavor.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if one of the specified pointers is NULL.
ippStsNoOperation	Indicates a warning if width or height of the destination image is equal to zero.
ippStsContextMatchErr	Indicates an error if pointer to the spec structure is invalid.
ippStsSizeWrn	Indicates a warning in the following cases: • if width of the image is odd, • if the destination image size is more than the destination image origin.
ippStsMisalignedOffsetErr	Indicates an error if the x field of the dstOffset parameter is odd.

ippStsSizeErr	Indicates an error if width of the destination image is equal to 1, or if width or height of the source or destination image is negative.
ippStsOutOfRangeErr	Indicates an error if the destination imageoffset point is outside the destination image origin.

ResizeYUV422Linear

Changes an YUY2 image size by the linear interpolation method.

Syntax

```
IppStatus ippiResizeYUV422Linear_8u_C2R(const Ipp8u* pSrc, Ipp32s srcStep, Ipp8u* pDst,
Ipp32s dstStep, IppiPoint dstOffset, IppiSize dstSize, IppiBorderType border, const
Ipp8u* pBorderValue, const IppiResizeYUV422Spec* pSpec, Ipp8u* pBuffer);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

pSrc	Pointer to the source image.
srcStep	Distance in bytes between starts of consecutive lines in the source image buffer.
pDst	Pointer to the destination image.
dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.
dstOffset	Offset of the tiled destination image with respect to the destination image origin.
dstSize	Size of the destination image in pixels.
border	Type of the border.
pBorderValue	Pointer to the constant value to assign to pixels of the constant border. This parameter is applicable only to the `ippBorderConst` border type.
pSpec	Pointer to the spec structure for the resize filter.
pBuffer	Pointer to the work buffer.

Description

This function changes an image size using the linear interpolation method. The image size can be either reduced or increased in each direction, depending on the destination image size.

This function operates with ROI (see ROI Processing in Geometric Transforms). It resizes the source image ROI origin to the destination image ROI origin. The destination image ROI origin must be defined by the following parameters: the offset of the tiled destination image with respect to the destination image origin and the destination image size. The source image ROI origin is defined automatically. To obtain the source image ROI origin offset, call the ippiResizeYUV422GetSrcOffset function. The source and destination images are in the YUY2 pixel format (Y0U0Y1V0,Y2U1Y3V1,.. or Y0Cb0Y1Cr0,Y2Cb1Y3Cr1,..).

Supported values for `border` are `ippBorderRepl` and `ippBorderInMem`.

The interpolation algorithm applied uses edge pixels of the source image that are out of the ROI. The function `ippiResizeYUV422Linear` uses the weighted values of these outer pixels in calculation. To obtain the size of the out of the ROI source image, call the function ippiResizeYUV422GetBorderSize .

If the border type is equal to `ippBorderRepl`, the source image edge pixels are replicated out of the ROI. If the border type is equal to `ippBorderInMem`, the outer pixels are obtained from the out of the ROI image space.

Prior to using the `ippiResizeYUV422Linear` function, initialize the `IppiResizeYUV422Spec` structure by calling the ippiResizeYUV422LinearInit and compute the size of the external buffer `pBuffer` by calling the ippiResizeYUV422GetBufSize for the corresponding flavor.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if one of the specified pointers is `NULL`.
`ippStsNoOperation`	Indicates a warning if width or height of the destination image is equal to zero.
`ippStsBorderErr`	Indicates an error if the `border` value is illegal.
`ippStsContextMatchErr`	Indicates an error if pointer to the spec structure is invalid.
`ippStsSizeWrn`	Indicates a warning in the following cases: • if width of the destination image is odd, • if the destination image size is more than the destination image origin.
`ippStsMisalignedOffsetErr`	Indicates an error if the `x` field of the `dstOffset` parameter is odd.
`ippStsSizeErr`	Indicates an error if width of the destination image is equal to 1, or if width or height of the source or destination image is negative.
`ippStsOutOfRangeErr`	Indicates an error if the destination imageoffset point is outside the destination image origin.

Warp Functions with Prior Initialization

This section describes the Intel® IPP warping functions that use the specification structure in operation. Before using these functions, you need to initialize the structure.

- Using Intel® IPP Warp Affine Functions with Prior Initialization
- Edge Smoothing
- GetAffineQuad
- GetAffineBound
- GetAffineSrcRoi
- GetAffineTransform
- GetRotateTransform
- GetRotateShift
- WarpAffineGetSize
- WarpQuadGetSize
- WarpGetBufferSize
- WarpAffineNearestInit
- WarpQuadNearestInit
- WarpAffineNearest
- WarpAffineLinearInit
- WarpQuadLinearInit
- WarpAffineLinear
- WarpAffineCubicInit
- WarpQuadCubicInit
- WarpAffineCubic
- GetPerspectiveQuad
- GetPerspectiveBound
- GetPerspectiveTransform
- WarpGetRectInfinite
- WarpPerspectiveGetSize
- WarpPerspectiveNearestInit
- WarpPerspectiveNearest
- WarpPerspectiveLinearInit
- WarpPerspectiveLinear
- WarpPerspectiveCubicInit
- WarpPerspectiveCubic

Using Intel® IPP Warp Affine Functions with Prior Initialization

You can use one of the following approaches to image warping:

- Warping the whole image
- Warping a tiled image with one prior initialization

Interpolation algorithms of the Nearest Neighbor, Linear, and Cubic types can use edge pixels of the source image that are out of the image origin. When calling the `ippiWarpAffine<Filter>` function with one of these interpolation algorithms applied, you need to specify the appropriate border type. The following border types are supported:

- Replicated borders: border pixels are replicated from the source image boundary pixels
- Constant border: values of all border pixels are set to a constant
- Transparent borders: destination pixels that have inverse transformed location out of the source image are not processed
- Borders in memory: the source image border pixels are obtained from the source image pixels in memory
- Mixed borders: combination of transparent borders and borders in memory is applied

Warping the Whole Image

You can follow the approach described below to apply affine transformation when source and destination images are fully accessible in memory. However, this method only runs on a single thread.

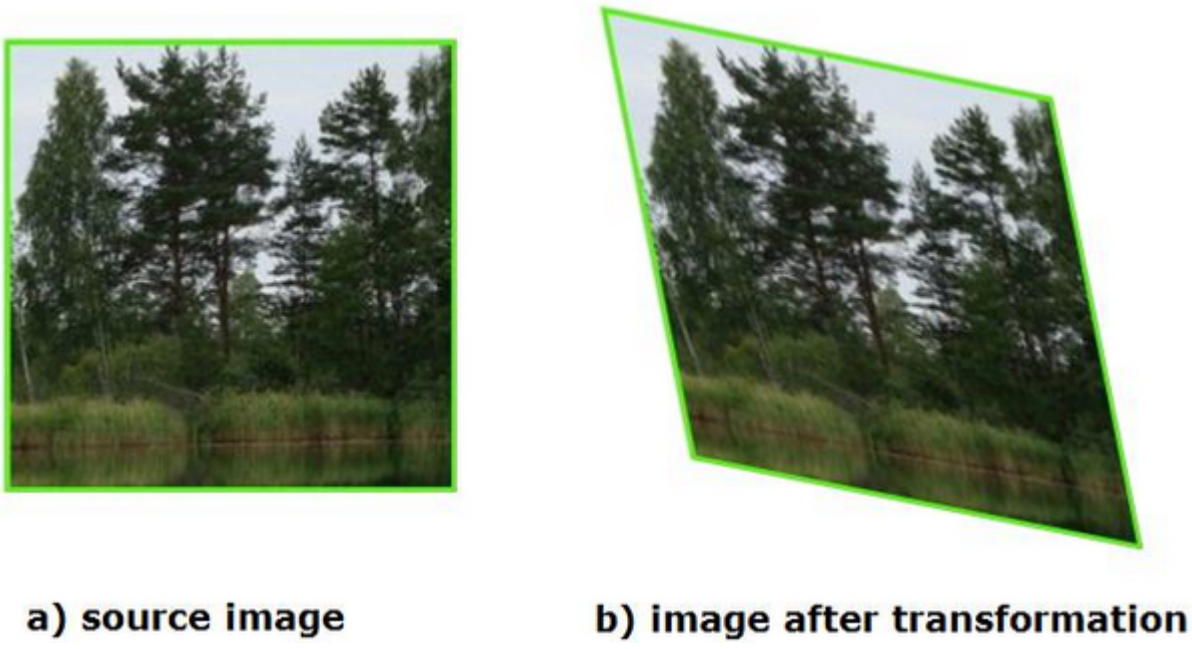
To transform the whole image:

1. Call the WarpAffineGetSize function with the appropriate interpolation type. This function uses source and destination image sizes to calculate how much memory must be allocated for the `IppWarpSpec` structure and work buffer.

2. Initialize the `IppWarpSpec` structure by calling the `ippiWarp<Filter>Init`, where < Filter``>`` can take one of the following values: `Nearest`, `Linear`, and `Cubic`. These prerequisite steps enable calling the warp functions multiple times without recalculations.
3. Call the WarpGetBufferSize function for the initialized `IppWarpSpec` structure. This function uses the destination image size to calculate how much memory must be allocated for the warp work buffer.
4. Call `ippiWarpAffine<Filter>` with the appropriate image type.
5. Specify the algorithm for borders processing by setting the `borderType` and `pBorderValue` parameters when initializing the `IppiWarpSpec` structure. The data type of `pBorderValue` is automatically converted from `Ipp64f` to the data type of the processed images. The function supports the following algorithms for borders processing:
   - If the border type is equal to `ippBorderRepl`, the source image outer pixels are replicated from the edge pixels.
   - If the border type is equal to `ippBorderConst`, the outer pixels are set to the constant value specified in `pBorderValue`.
   - If the border type is equal to `ippBorderTransp`, destination image pixels mapped to the outer source image pixels are not changed. The outer pixels are replicated from the edge pixels, if they are required by interpolation algorithm.
   - If the border type is equal to `ippBorderInMem`, destination image pixels mapped to the outer source image pixels are not changed. The outer pixels are obtained from the out of of the source image origin space, if they are required by interpolation algorithm.
   - The mixed border types can be obtained by the bitwise operation `OR` between `ippBorderTransp` and the `ippBorderInMemTop`, `ippBorderInMemBottom`, `ippBorderInMemLeft`, `ippBorderInMemRight` values.

Figure *Whole Image Warping* shows a simple image affine transformation example. Transformation coefficients are {{1.0, 0.5, 0.0}, {0.5, 1.0, 0.0}}, border type is `ippBorderConst`, `pBorderValue` is a white color pixel. The size of the destination image is 1.2x of the source image size.

Whole Image Warping



Example

The code example below demonstrates affine transformation of the whole image with the linear interpolation method:

```
IppStatus warpAffineExample_8u_C3R(Ipp8u* pSrc, IppiSize srcSize, Ipp32s srcStep, Ipp8u* pDst,
IppiSize dstSize,
    Ipp32s dstStep, const double coeffs[2][3])
{
    IppiWarpSpec* pSpec = 0;
    int specSize = 0, initSize = 0, bufSize = 0; Ipp8u* pBuffer  = 0;
    const Ipp32u numChannels = 3;
    IppiPoint dstOffset = {0, 0};
    IppStatus status = ippStsNoErr;
    IppiBorderType borderType = ippBorderConst;
    IppiWarpDirection direction = ippWarpForward;
    Ipp64f pBorderValue[numChannels];


    for (int i = 0; i < numChannels; ++i) pBorderValue[i] = 255.0;


    /* Spec and init buffer sizes */
    status = ippiWarpAffineGetSize(srcSize, dstSize, ipp8u, coeffs, ippLinear, direction,
borderType, &specSize, &initSize);


    if (status != ippStsNoErr) return status;


    /* Memory allocation */
    pSpec = (IppiWarpSpec*)ippsMalloc_8u(specSize);


    if (pSpec == NULL)
    {
        return ippStsNoMemErr;
    }


    /* Filter initialization */
    status = ippiWarpAffineLinearInit(srcSize, dstSize, ipp8u, coeffs, direction, numChannels,
borderType, pBorderValue, 0, pSpec);


    if (status != ippStsNoErr)
    {
        ippsFree(pSpec);
        return status;
    }


    /* work buffer size */
    status = ippiWarpGetBufferSize(pSpec, dstSize, &bufSize);
    if (status != ippStsNoErr)
    {
        ippsFree(pSpec);
        return status;
    }
```

```
    pBuffer = ippsMalloc_8u(bufSize);
    if (pBuffer == NULL)
    {
        ippsFree(pSpec);
        return ippStsNoMemErr;
    }


    /* Resize processing */
    status = ippiWarpAffineLinear_8u_C3R(pSrc, srcStep, pDst, dstStep, dstOffset, dstSize,
pSpec, pBuffer);


    ippsFree(pSpec);
    ippsFree(pBuffer);


    return status;
}
```

Warping a Tiled Image with One Prior Initialization

You can follow the approach described below to apply affine transformation when the source image is fully accessible in memory and destination image is not fully accessible in memory, or to improve performance of warping by external threading.

The main difference between this approach and whole image warping is that the image is split into sections called tiles. Each call of the `WarpAffine<Filter>` function works with the destination image origin region of interest (ROI) that is defined by `dstRoiOffset` and `dstRoiSize` parameters. The destination ROI must be fully accessible in memory.

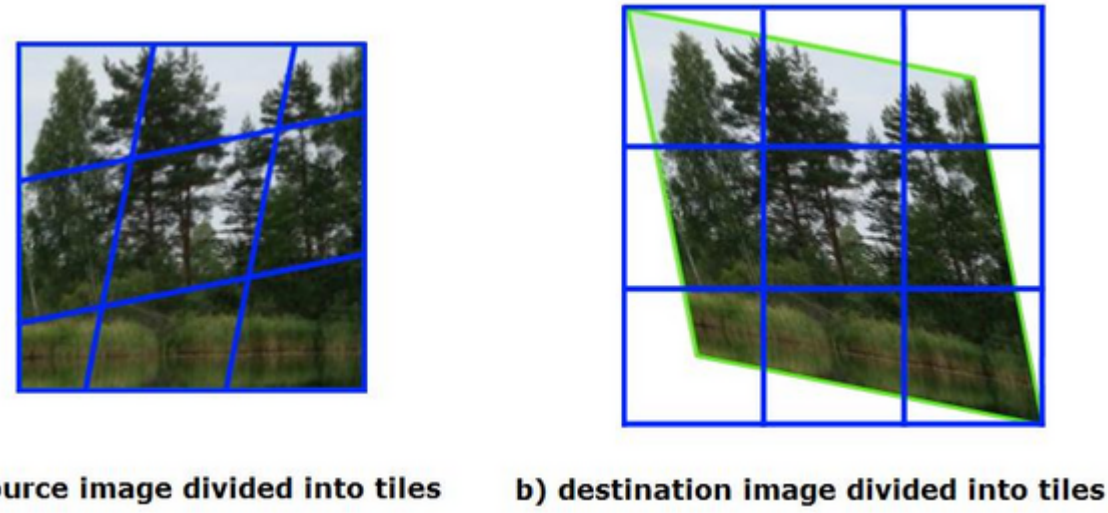
To resize an image with the tiled approach:

1. Call the WarpAffineGetSize function with the appropriate interpolation type. This function uses the size of the source and destination images and transformation parameters to calculate how much memory must be allocated for the `IppWarpSpec` structure and initialization work buffer.
2. Initialize the `IppWarpSpec` structure by calling `ippiWarpAffine<Filter>Init`, where < Filter``>`` can take one of the following values: `Nearest`, `Linear`, and `Cubic`.
3. Determine an appropriate partitioning scheme to divide the destination image into tiles. Tiles can be sets of rows or a regular grid of subimages. A simple vertical subdivision into sets of lines is sufficient in most cases.
4. Call the WarpGetBufferSize function to obtain the size of the work buffer required for each tile processing. The `dstRoiSize` parameter must be equal to the tile size.
5. Call `ippiWarpAffine<Filter>` for each tile (ROI). The `dstRoiOffset` parameter must specify the image ROI offset with respect to the destination image origin. The `dstRoiSize` parameter must be equal to the ROI size. The `pDst` parameter must point to the beginning of the destination ROI in memory. The source and destination ROIs must be fully accessible in memory.

   You can process tiles in any order. When using multiple threads you can process all tiles simultaneously.

Figure *Tiled Image Warping* shows the affine transformation of the image divided into tiles. Transformation coefficients are {{1.0, 0.5, 0.0}, {0.5, 1.0, 0.0}}, applied border type is `ippBorderConst`, pBorderValue is a white color pixel. The size of the destination image is 1.2x of the source image size.

Tiled Image Warping



a) source image divided into tiles b) destination image divided into tiles

Example

The code example below demonstrates a multithreading affine transformation using OpenMP* with parallelization only in y direction:

```
IppStatus tileWarpAffineExample_C3R(Ipp8u* pSrc, IppiSize srcSize, Ipp32s srcStep, Ipp8u* pDst,
IppiSize dstSize, Ipp32s dstStep, const double coeffs[2][3])
{
    IppiWarpSpec* pSpec = 0;
    int specSize = 0, initSize = 0, bufSize = 0; Ipp8u* pBuffer  = 0;
    Ipp8u* pInitBuf = 0;
    const Ipp32u numChannels = 3;
    IppiPoint dstOffset = {0, 0};
    IppiPoint srcOffset = {0, 0};
    IppStatus status = ippStsNoErr;
    IppiBorderType borderType = ippBorderConst;
    IppiWarpDirection direction = ippWarpForward;
    int numThreads, slice, tail;
    int bufSize1, bufSize2;
    IppiSize dstTileSize, dstLastTileSize; IppStatus pStatus[MAX_NUM_THREADS];
    Ipp64f pBorderValue[numChannels];


    for (int i = 0; i < numChannels; ++i) pBorderValue[i] = 255.0;


    /* Spec and init buffer sizes */
    status = ippiWarpAffineGetSize(srcSize, dstSize, ipp8u, coeffs, ippLinear, direction,
borderType, &specSize, &initSize);


    if (status != ippStsNoErr) return status;


    /* Memory allocation */
    pSpec = (IppiWarpSpec*)ippsMalloc_8u(specSize);


    if (pSpec == NULL)
    {
        return ippStsNoMemErr;
```

```
    }


    /* Filter initialization */
    status = ippiWarpAffineLinearInit(srcSize, dstSize, ipp8u, coeffs, direction, numChannels,
borderType, pBorderValue, 0, pSpec);


    if (status != ippStsNoErr)
    {
        ippsFree(pSpec);
        return status;
    }


    /* General transform function */
    /* Parallelized only by Y-direction here */
    #pragma omp parallel num_threads(MAX_NUM_THREADS)
    {
    #pragma omp master
        {
            numThreads = omp_get_num_threads();


            slice = dstSize.height / numThreads; tail  = dstSize.height % numThreads;


            dstTileSize.width = dstLastTileSize.width = dstSize.width;
            dstTileSize.height = slice;
            dstLastTileSize.height = slice + tail;


            ippiWarpGetBufferSize(pSpec, dstTileSize, &bufSize1);
            ippiWarpGetBufferSize(pSpec, dstLastTileSize, &bufSize2);


            pBuffer = ippsMalloc_8u(bufSize1 * (numThreads - 1) + bufSize2);
        }


        #pragma omp barrier
        {
            if (pBuffer)
            {
                Ipp32u  i;
                Ipp8u  *pDstT; Ipp8u  *pOneBuf;
                IppiPoint srcOffset = {0, 0};
                IppiPoint dstOffset = {0, 0};
                IppiSize  srcSizeT = srcSize; IppiSize  dstSizeT = dstTileSize;


                i = omp_get_thread_num();


                dstSizeT.height = slice; dstOffset.y += i * slice;


                if (i == numThreads - 1) dstSizeT = dstLastTileSize;
```

```
                pDstT = (Ipp8u*)((char*)pDst + dstOffset.y * dstStep);

                pOneBuf = pBuffer + i * bufSize1;

                pStatus[i] = ippiWarpAffineLinear_8u_C3R (pSrc, srcStep, pDstT, dstStep,
dstOffset, dstSizeT, pSpec, pOneBuf);
            }
        }
    }

    ippsFree(pSpec);

    if (pBuffer == NULL) return ippStsNoMemErr;

    ippsFree(pBuffer);

    for (Ipp32u i = 0; i < numThreads; ++i)
    {
        /* Return bad status */
        if(pStatus[i] != ippStsNoErr) return pStatus[i];
    }

    return status;
}
```

See Also

User-defined Border Types

WarpAffineGetSize Computes the size of the specification structure and the size of the external work buffer for the warp affine transform.

WarpGetBufferSize Computes the size of the work buffer for the warp transform.

Edge Smoothing

The *Smooth Edge* feature is an artificial method to reduce aliasing artifacts at the transformed source image edges. Aliasing artifacts may appear because the transformation algorithms skip a destination pixel if its source origin is out of the source image ROI. Thus, borders of the transformed source image can look stepped:



a) SMOOTH_EDGE option turned off



b) SMOOTH_EDGE option turned on

If the `smoothEdge` flag is set, destination pixels that are closest to the transformed source image edges are mixed with sampled source pixels by the following formula:

`dstRes = srcSampled *(1-a)+ dstExist * a`

where

- `srcSampled` is the intensity of the source pixel after transformation.
- `dstExist` is the intensity of the destination pixel before transformation.
- `a` is the weight of the outer pixel; set by the function.
- `dstRes` is the intensity of the resulting destination pixel.

The edge smoothing method is not universal: in some cases it can improve the image, but in other cases it can be inefficient. For example, edge smoothing does not increase the quality of images with high contrast borders, and it is not recommended to apply edge smoothing to such images.

NOTE Edge smoothing is a post-processing operation: it is performed after transformation. When warping a tiled image, artifacts may appear on tile borders. In this case, edges are not smoothed.

GetAffineQuad

Computes vertex coordinates of the quadrangle, to which the source ROI rectangle is mapped by the affine transform.

Syntax

```
IppStatus ippiGetAffineQuad (IppiRect srcRoi, double quad[4][2], const double coeffs[2][3]);
```

```
IppStatus ippiGetAffineQuad_L(IppiRectL srcRoi , double quad[4][2], const double coeffs[2][3]);
```

Include Files

`ippi.h`

Flavors with the `_L` suffix: `ippi_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

srcRoi	Region of interest in the source image (of the IppiRect type).
quad	Output array. Contains vertex coordinates of the quadrangle, to which the source ROI is mapped by the affine transform function.
coeffs	The given affine transform coefficients.

Description

This function operates with ROI (see ROI Processing in Geometric Transforms).

This function is used as a support function for WarpAffineNearest, WarpAffineLinear, and WarpAffineCubic functions. It computes vertex coordinates of the quadrangle, to which the source rectangular ROI is mapped by the affine transform function using the given coefficients coeffs.

The first dimension [4] of the array quad[4][2] is equal to the number of vertices, and the second dimension [2] means x and y coordinates of the vertex. Quadrangle vertices have the following meaning:

quad[0] corresponds to the transformed top-left corner of the source ROI,

quad[1] corresponds to the transformed top-right corner of the source ROI,

quad[2] corresponds to the transformed bottom-right corner of the source ROI,

quad[3] corresponds to the transformed bottom-left corner of the source ROI.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsCoeffErr	Indicates an error condition if $c_{00} * c_{11} - c_{01} * c_{10} = 0$.
ippStsSizeErr	Indicates an error condition if srcRoi has a size field with zero or negative value.

GetAffineBound

Computes the bounding rectangle for the source ROI transformed by the ippiWarpAffine function.

Syntax

```
IppStatus ippiGetAffineBound (IppiRect srcRoi, double bound[2][2], const double coeffs[2][3]);
```

Include Files

ippi.h

Flavors with the _L suffix: ippi_l.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

srcRoi	Region of interest in the source image (of the IppiRect type).
bound	Output array. Contains vertex coordinates of the bounding rectangle for the transformed source ROI.
coeffs	The given affine transform coefficients.

Description

This functionis used as a support function for WarpAffineNearest, WarpAffineLinear, and WarpAffineCubic functions. It computes vertex coordinates of the smallest bounding rectangle for the quadrangle quad, to which the source ROI is mapped by the affine transform function using coefficients coeffs.

bound[0] specifies x, y coordinates of the top-left corner, bound[1] specifies x, y coordinates of the bottom-right corner.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsCoeffErr	Indicates an error condition if $c_{00} * c_{11} - c_{01} * c_{10} = 0$.
ippStsSizeErr	Indicates an error condition if srcRoi has a size field with zero or negative value.

GetAffineSrcRoi

Computes ROI of an image for affine transform.

Syntax

```
IppStatus ippiGetAffineSrcRoi (IppiSize srcSize, const double coeffs[2][3],
IppiWarpDirection direction, IppiPoint dstRoiOffset, IppiSize dstRoiSize, IppiRect
*srcRoi);
```

Include Files

ippi.h

Flavors with the _L suffix: ippi_l.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

srcSize	Size of the source image, in pixels.
coeffs	Coefficients for affine transform.
direction	Transformation direction. Supported values: • ippWarpForward • Forward transformation. • ippWarpBackward • Backward transformation.
dstRoiSize	Size of the ROI of destination image.
dstRoiOffset	Offset of the destination image ROI.
srcRoi	Pointer to the computed region of interest in the source image.

Description

This function operates with ROI (see ROI Processing in Geometric Transforms).

This function is used as a support function for the ippiWarpAffineLinear, WarpAffineNearest, and WarpAffineCubic functions. It computes ROI of the source image to perform affine transformation for a given destination ROI. To process the given destination ROI, the computed source ROI with borders must be accessible in memory. If the source ROI outside pixels are out of the source image origin, the border pixels are processed according to the border flag that is passed to the ippiWarpAffineLinear, WarpAffineNearest, and WarpAffineCubic functions.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or warning.
ippStsRectErr	Indicates an error condition if width or height of the srcRoi is less than or equal to 1.
ippStsOutOfRangeErr	Indicates an error if the destination image offset has a field with a negative value.
ippStsSizeErr	Indicates an error condition if width or height of the source or destination image is less than, or equal to zero.
ippStsWrongIntersetQuad	Indicates a warning that no operation is performed because the transformed source image has no intersection with the destination ROI.
ippStsNullPtrErr	Indicates an error if one of the specified pointers is NULL.
ippStsWarpDirectionErr	Indicates an error when the direction value is illegal.
ippStsCoefErr	Indicates an error condition, if affine transformation is singular.

GetAffineTransform

Computes affine transform coefficients to map the source ROI to the quadrangle with the specified vertex coordinates

Syntax

```
IppStatus ippiGetAffineTransform (IppiRect srcRoi, const double quad[4][2], double coeffs[2][3]);
```

Include Files

`ippi.h`

Flavors with the `_L` suffix: `ippi_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`srcRoi`	Region of interest in the source image (of the `IppiRect` type).
`quad`	Vertex coordinates of the quadrangle, to which the source ROI is mapped by the affine transform function.
`coeffs`	Output array. Contains the target affine transform coefficients.

Description

This function operates with ROI (see ROI Processing in Geometric Transforms).

This function is used as a support function for the ippiWarpAffineLinear, WarpAffineNearest, and WarpAffineCubic functions. It computes the coefficients `coeffs` of the affine transform that must be used by the warping function to map the source rectangular ROI to the quadrangle with the specified vertex coordinates `quad`. The first dimension [4] of the array `quad` [4][2] is equal to the number of vertices, and the second dimension [2] means `x` and `y` coordinates of the vertex. Quadrangle vertices have the following meaning:

`quad` [0] corresponds to the transformed top-left corner of the source ROI, `quad` [1] corresponds to the transformed top-right corner of the source ROI, `quad` [2] corresponds to the transformed bottom-right corner of the source ROI, `quad` [3] corresponds to the transformed bottom-left corner of the source ROI.

The function computes the coordinates of the 4th vertex of the destination quadrangle that uniquely depends on the three other vertices. If the computed coordinates are not equal to the ones specified in `quad`, the function returns the warning message and continues operation with the computed values.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or warning.

ippStsRectErr	Indicates an error condition if width or height of the srcRoi is less than or equal to 1.
ippStsCoeffErr	Indicates an error condition if $c_{00} * c_{11} - c_{01} * c_{10} = 0$.
ippStsSizeErr	Indicates an error condition if srcRoi has a size field with zero or negative value.
ippStsAffineQuadChanged	Indicates a warning that coordinates of the 4th vertex of the specified quadrangle quad are not correct.

GetRotateTransform

Computes the affine coefficients for the rotation transform.

Syntax

```
IppStatus ippiGetRotateTransform(double angle, double xShift, double yShift, double coeffs[2][3]);
```

Include Files

ippi.h

Flavors with the _L suffix: ippi_l.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

angle	Angle of rotation, in degrees. The source image is rotated counterclockwise around the origin (0, 0).
xShift, yShift	Shift along horizontal (x) or vertical (y) axis that is performed after rotation.
coeffs	Computed affine transform coefficients for the given rotation parameters.

Description

This function computes the coefficients for the affine transform that rotates an image by the specified angle around the origin (0, 0) and shifts the image after rotation. The result is stored in the coeffs parameter.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.

ippStsSizeErr	Indicates an error when one of the coeffs values is NULL.
ippStsOutOfRangeErr	Indicates an error when angle is not-a-number (NaN) or infinity.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

// A simple example of computing the affine coefficients for the transform
// that rotates an image using Intel(R) Integrated Performance Primitives (Intel(R) IPP)
functions:
//     ippiGetRotateTransform
//     ippiWarpAffineLinearInit
//     ippiWarpGetBufferSize
//     ippiWarpAffineLinear_8u_C3R

#include <stdio.h>
#include "ipp.h"

#define WIDTH   64 /* source image width */
#define HEIGHT  64 /* source image height */
#define NUM_CHN 3

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define check_sts(st)        \
    if ((st) != ippStsNoErr) \
        goto exitLine; /* Go to Exit if Intel(R) IPP function returned status different from
ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main(void)
{
    IppStatus status = ippStsNoErr;
    IppiWarpSpec *pSpec = NULL;                                  /* Pointer to the
specification structure */
    IppiSize srcSize = {WIDTH, HEIGHT}, dstSize = {WIDTH, HEIGHT}; /* Size of source/destination
images */
```

```
    int srcStep, dstStep;                                          /* Steps, in bytes, through
the source/destination images */
    Ipp8u *pSrc = NULL, *pDst = NULL;                              /* Pointers to source/
destination images */
    double angle = 30., xShift = 2., yShift = 1.;
    double coeffs[2][3] = {0};
    IppiPoint dstOffset = {0, 0};               /* Offset of the destination image ROI with
respect to the destination image origin */
    int specSize = 0, initSize = 0, bufSize = 0; /* Work buffer size */
    Ipp8u *pBuffer = NULL;
    IppiBorderType borderType = ippBorderConst;
    IppiWarpDirection direction = ippWarpForward; /* Transformation direction */
    Ipp64f pBorderValue[NUM_CHN] = {0};

    pSrc = ippiMalloc_8u_C3(srcSize.width, srcSize.height, &srcStep);
    pDst = ippiMalloc_8u_C3(dstSize.width, dstSize.height, &dstStep);

    check_sts(status = ippiGetRotateTransform(angle, xShift, yShift, (double(*)[3])coeffs));

    /* Spec and init buffer sizes */
    check_sts(status = ippiWarpAffineGetSize(srcSize, dstSize, ipp8u, coeffs, ippLinear,
direction, borderType, &specSize, &initSize));

    pSpec = (IppiWarpSpec *)ippsMalloc_8u(specSize);

    /* Filter initialization */
    check_sts(status = ippiWarpAffineLinearInit(srcSize, dstSize, ipp8u, coeffs, direction,
NUM_CHN, borderType, pBorderValue, 0, pSpec));

    /* Get work buffer size */
    check_sts(status = ippiWarpGetBufferSize(pSpec, dstSize, &bufSize));

    pBuffer = ippsMalloc_8u(bufSize);

    /* WarpAffine processing */
    check_sts(status = ippiWarpAffineLinear_8u_C3R(pSrc, srcStep, pDst, dstStep, dstOffset,
dstSize, pSpec, pBuffer));

    EXIT_MAIN
    ippiFree(pSrc);
    ippiFree(pDst);
    ippsFree(pBuffer);
    ippsFree(pSpec);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

GetRotateShift

Computes shift values for rotation of an image around the specified center.

Syntax

```
IppStatus ippiGetRotateShift (double xCenter, double yCenter, double angle, double*
xShift, double* yShift);
```

Include Files

ippi.h

Flavors with the _L suffix: ippi_l.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

xCenter, yCenter	Coordinates of the required center of rotation.
angle	The angle in degrees to rotate the image clockwise around the point with coordinates (xCenter, yCenter).
xShift, yShift	Pointers to computed shift values along horizontal and vertical axes. These shift values should be passed to ippiRotate function to bring about the desired rotation around (xCenter, yCenter).

Description

Use this function if you need to rotate an image about an arbitrary center (xCenter, yCenter) rather than the origin (0,0). The function helps compute shift values xShift, yShift that should be passed to the warping function for the rotation around (xCenter, yCenter) to take place.

Example shows how to use the function ippiGetRotateShift.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if xShift or yShift pointer is NULL.

WarpAffineGetSize

Computes the size of the specification structure and the size of the external work buffer for the warp affine transform.

Syntax

```
IppStatus ippiWarpAffineGetSize(IppiSize srcSize, IppiSize dstSize, IppDataType
dataType, const double coeffs[2][3], IppiInterpolationType interpolation,
IppiWarpDirection direction, IppiBorderType borderType, int* pSpecSize, int*
pInitBufSize);
```

Include Files

ippi.h

Flavors with the _L suffix: ippi_l.h

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`srcSize`	Size of the source image, in pixels.
`dstSize`	Size of the destination image, in pixels.
`dataType`	Data type of the source and destination images. Supported values: `ipp8u`, `ipp16u`, `ipp16s`, `ipp32f`, and `ipp64f`.
`coeffs`	Coefficients for the affine transform.
`interpolation`	Interpolation method. Supported values: `ippNearest`, `ippLinear`, and `ippCubic`.
`direction`	Transformation direction. Supported values: • `ippWarpForward` • Forward transformation • `ippWarpBackward` • Backward transformation
`borderType`	Type of border. Possible values are: • `ippBorderConst` • Values of all border pixels are set to constant. • `ippBorderRepl` • Border is replicated from the edge pixels. • `ippBorderTransp` • Outer pixels are not processed. • `ippBorderInMem` • Border is obtained from the source image pixels in memory.
`pSpecSize`	Pointer to the size, in bytes, of the specification structure.
`pInitBufSize`	Pointer to the size, in bytes, of the temporary buffer.

Description

This function computes the size of the specification structure and the external work buffer for the following functions, depending on the `interpolation` parameter: WarpAffineNearestInit, WarpAffineLinearInit, or WarpAffineCubicInit.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when one of the specified pointers is `NULL`.

ippStsNoOperation	Indicates a warning if width or height of any image is zero.
ippStsSizeErr	Indicates an error If the width or height of the source or destination image is less than, or equal to one.
ippStsDataTypeErr	Indicates an error when dataType has an illegal value.
ippStsWarpDirectionErr	Indicates an error when direction has an illegal value.
ippStsInterpolationErr	Indicates an error when interpolation has an illegal value.
ippStsNotSupportedModeErr	Indicates an error if the requested mode is not supported.
ippStsCoeffErr	Indicates an error when affine transformation is singular.
ippStsBorderErr	Indicates an error when borderType has an illegal value.
ippStsWrongIntersectQuad	Indicates a warning that no operation is performed if the transformed source image extended with borders has no intersection with the destination image.
ippStsExceededSizeErr	Indicates an error in the following cases:

See Also

WarpAffineNearestInit Initializes the specification structure for image affine warping with the nearest neighbor interpolation method.

WarpAffineLinearInit Initializes the specification structure for image affine warping with the linear interpolation method.

WarpAffineCubicInit Initializes the specification structure for image affine warping with the cubic interpolation method.

WarpQuadGetSize

Computes the size of the specification structure and the size of the temporary buffer for warping an arbitrary quadrangle in the source image to the quadrangle in the destination image.

Syntax

```
IppStatus ippiWarpQuadGetSize(IppiSize srcSize, const double srcQuad[4][2], IppiSize dstSize, const double dstQuad[4][2], IppiWarpTransformType transform, IppDataType dataType, IppiInterpolationType interpolation, IppiBorderType borderType, int* pSpecSize, int* pInitBufSize);
```

Include Files

ippi.h

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`srcSize`	Size of the source image, in pixels.
`srcQuad`	Quadrangle in the source image.
`dstSize`	Size of the destination image, in pixels.
`dstQuad`	Quadrangle in the destination image.
`transform`	Type of the warp tranform. Supported values: • `ippWarpAffine` • Affine warping • `ippWarpPerspective` • Perspective warping
`dataType`	Data type of the source and destination images. Supported values: `ipp8u`, `ipp16u`, `ipp16s`, and `ipp32f`.
`interpolation`	Interpolation method. Supported values: `ippNearest`, `ippLinear`, and `ippCubic`.
`borderType`	Type of border. Supported values: • `ippBorderTransp` • Outer pixels are not processed. • `ippBorderInMem` • Border is obtained from the source image pixels in memory.
`pSpecSize`	Pointer to the size, in bytes, of the specification structure.
`pInitBufSize`	Pointer to the size, in bytes, of the temporary buffer.

Description

This function computes the size of the specification structure and the temporary buffer for the following functions, depending on the `interpolation` parameter: ippiWarpQuadNearestInit, ippiWarpQuadLinearInit, or ippiWarpQuadCubicInit.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when one of the specified pointers is `NULL`.

ippStsSizeErr	Indicates an error in the following cases: • If the width or height of the source or destination image is less than, or equal to one. • If one of the calculated sizes exceeds the maximum positive 32-bit signed integer value. The size of the one of the processed images is too large.
ippStsDataTypeErr	Indicates an error when `dataType` has an illegal value.
ippStsWarpTransformErr	Indicates an error when `transform` has an illegal value.
ippStsInterpolationErr	Indicates an error when `interpolation` has an illegal value.
ippStsQuadErr	Indicates an error when any of the given quadrangles is non-convex or degenerates into a triangle, line, or point.
ippStsWrongIntersectQuad	Indicates a warning that no operation is performed if the transformed source image extended with borders has no intersection with the destination image.
ippStsAffineQuadChanged	Indicates a warning when coordinates of the fourth vertex of `dstQuad` are changed by the function, if `transform` is set to `ippWarpAffine`.
ippStsBorderErr	Indicates an error when `borderType` has an illegal value.

See Also

WarpQuadNearestInit Initializes the specification structure for warping an arbitrary quadrangle in the source image to the quadrangle in the destination image with the nearest neighbor interpolation method.

WarpQuadLinearInit Initializes the specification structure for warping an arbitrary quadrangle in the source image to the quadrangle in the destination image with the linear interpolation method.

WarpQuadCubicInit Initializes the specification structure for warping an arbitrary quadrangle in the source image to the quadrangle in the destination image with the cubic interpolation method.

WarpGetBufferSize

Computes the size of the work buffer for the warp transform.

Syntax

```
IppStatus ippiWarpGetBufferSize(const IppiWarpSpec* pSpec, IppiSize dstRoiSize, int* pBufSize);
```

Include Files

`ippi.h`

Flavors with the _L suffix: ippi_l.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSpec	Pointer to the specification structure for the warp operation.
dstRoiSize	Size of the processed destination image ROI, in pixels.
pBufSize	Pointer to the size of the external buffer, in bytes.

Description

This function computes the size of the external buffer for the warp transform. The specification structure pointed by pSpec defines the warp algorithm parameters.

Before using this function, you need to initialize the specification structure using one of the following functions: WarpAffineNearestInit, WarpAffineLinearInit, or WarpAffineCubicInit.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsContextMatchErr	Indicates an error when the specification structure is invalid.
ippStsSizeErr	Indicates an error in the following cases: • If width or height of the destination image is negative, or equal to zero. • If the calculated buffer size exceeds the maximum positive pBufSize data type. The size of the processed image ROI is too large.
ippStsSizeWrn	Indicates a warning when the size of the destination image is more than the size of the destination image origin.

See Also

WarpAffineNearestInit Initializes the specification structure for image affine warping with the nearest neighbor interpolation method.

WarpAffineLinearInit Initializes the specification structure for image affine warping with the linear interpolation method.

WarpAffineCubicInit Initializes the specification structure for image affine warping with the cubic interpolation method.

WarpAffineNearestInit

Initializes the specification structure for image affine warping with the nearest neighbor interpolation method.

Syntax

```
IppStatus ippiWarpAffineNearestInit(IppiSize srcSize, IppiSize dstSize, IppDataType dataType, const double coeffs[2][3], IppiWarpDirection direction, int numChannels, IppiBorderType borderType, const Ipp64f* pBorderValue, int smoothEdge, IppiWarpSpec* pSpec);
```

Include Files

`ippi.h`

Flavors with the `_L` suffix: `ippi_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`srcSize`	Size of the source image, in pixels.
`dstSize`	Size of the destination image, in pixels.
`dataType`	Data type of the source and destination images. Supported values: `ipp8u`, `ipp16u`, `ipp16s`, `ipp32f`, and `ipp64f`.
`coeffs`	Coefficients for the affine transform.
`direction`	Transformation direction. Supported values: * `ippWarpForward` - forward transformation * `ippWarpBackward` - backward transformation
`numChannels`	Number of channels in the image. Supported values: 1, 3, or 4.
`borderType`	Type of border. Supported values: * `ippBorderConst` - values of all border pixels are set to a constant; * `ippBorderRepl` - border is replicated from the edge pixels; * `ippBorderTransp` - outer pixels are not processed; * `ippBorderInMem` - border is obtained from the source image pixels in memory.
`pBorderValue`	Pointer to the constant value to assign to pixels of the constant border. This parameter is applicable only to the `ippBorderConst` border type.

smoothEdge	Flag for edge smoothing. Supported values: • 0 - transformation without edge smoothing. • 1 - transformation with edge smoothing. This feature is supported only for the ippBorderTransp and ippBorderInMem border types.
pSpec	Pointer to the specification structure.

Description

This function initializes the IppiWarpSpec structure for the ippiWarpAffineNearest function that performs warp affine transformation with the nearest neighbor interpolation method. To compute the size of the specification structure, use the WarpAffineGetSize function.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when: • one of the specified pointers is NULL, excepting pBorderValue • pBorderValue is NULL when border type is set to ippBorderConst
ippStsNoOperation	Indicates a warning if width or height of any image is zero.
ippStsSizeErr	Indicates an error when width or height of the source or destination image is less than, or equal to one.
ippStsDataTypeErr	Indicates an error when dataType has an illegal value.
ippStsWarpDirectionErr	Indicates an error when direction has an illegal value.
ippStsCoeffErr	Indicates an error when affine transformation is singular.
ippStsWrongIntersectQuad	Indicates a warning that no operation is performed if the transformed source image extended with borders has no intersection with the destination image. The edge smoothing feature is not supported for the ippBorderRepl and ippBorderConst border types.
ippStsNotSupportedModeErr	Indicates an error when the requested mode is not supported.
ippStsBorderErr	Indicates an error when borderType has an illegal value.

ippStsNumChannelsErr	Indicates an error when numChannels has an illegal value.
ippStsExceededSizeErr	Indicates an error if width or height of the destination image or the source image exceeds 33554431 (0x1FFFFF).

See Also

WarpAffineGetSize Computes the size of the specification structure and the size of the external work buffer for the warp affine transform.

WarpQuadNearestInit

Initializes the specification structure for warping an arbitrary quadrangle in the source image to the quadrangle in the destination image with the nearest neighbor interpolation method.

Syntax

```
IppStatus ippiWarpQuadNearestInit(IppiSize srcSize, const double srcQuad[4][2],
IppiSize dstSize, const double dstQuad[4][2], IppiWarpTransformType transform,
IppDataType dataType, int numChannels, IppiBorderType borderType, const Ipp64f*
pBorderValue, int smoothEdge, IppiWarpSpec* pSpec);
```

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

srcSize	Size of the source image, in pixels.
srcQuad	Quadrangle in the source image.
dstSize	Size of the destination image, in pixels.
dstQuad	Quadrangle in the destination image.
transform	Type of the warp tranform. Supported values: • ippWarpAffine • Affine warping • ippWarpPerspective • Perspective warping
dataType	Data type of the source and destination images. Supported values: ipp8u, ipp16u, ipp16s, and ipp32f.
numChannels	Number of channels in the image. Supported values: 1, 3, or 4.

borderType	Type of border. Supported values: • ippBorderTransp • Outer pixels are not processed. • ippBorderInMem • Border is obtained from the source image pixels in memory.
pBorderValue	Pointer to the constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.
smoothEdge	Flag for edge smoothing. Supported values: • 0 - transformation without edge smoothing. • 1 - transformation with edge smoothing.
pSpec	Pointer to the specification structure.

Description

This function initializes the IppiWarpSpec structure for warping an arbitrary quadrangle in the source image to quadrangle in the destination image with the nearest neighbor interpolation method. To compute the size of the specification structure, use the WarpQuadGetSize function.

Transformation coefficients are computed internally, based on the mapping of the source quadrangle to the specified destination quadrangle dstQuad and transform type transform. In case of affine transform, the function computes the coordinates of the fourth vertex of the destination quadrangle that uniquely depends on the three other vertices. If the computed coordinates are not equal to the corresponding values specified in dstQuad, the function returns the warning message and continues initialization with the computed values.

The first dimension [4] of the array specifying the quadrangle srcQuad[4][2] or dstQuad[4][2] is equal to the number of vertices, and the second dimension [2] contains the x and y coordinates of the vertex.

You can apply the edge smoothing feature only if the source quadrangle entirely lies in the source image.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when: • One of the specified pointers is NULL, excepting pBorderValue • The value of pBorderValue is NULL when the border type is set to ippBorderConst
ippStsSizeErr	Indicates an error when width or height of the source or destination image is less than, or equal to one.
ippStsDataTypeErr	Indicates an error when dataType has an illegal value.
ippStsWarpTransformErr	Indicates an error when transform has an illegal value.

ippStsQuadErr	Indicates an error when any of the given quadrangles is non-convex or degenerates into a triangle, line, or point.
ippStsWrongIntersectQuad	Indicates a warning that no operation is performed if the transformed source image extended with borders has no intersection with the destination image.
ippStsAffineQuadChanged	Indicates a warning when coordinates of the fourth vertex of `dstQuad` are changed by the function, if `transform` is set to `ippWarpAffine`.
ippStsBorderErr	Indicates an error when `borderType` has an illegal value.
ippStsNumChannelsErr	Indicates an error when `numChannels` has an illegal value.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

// A simple example of the Spec structure initialization for warping of
// an arbitrary quadrangle in the source image to the quadrangle
// in the destination image using Intel(R) Integrated Performance Primitives (Intel(R) IPP)
functions:
//     ippiWarpQuadGetSize
//     ippiWarpQuadLinearInit
//     ippiWarpGetBufferSize
//     ippiWarpPerspectiveLinear_8u_C3R

#include <stdio.h>
#include "ipp.h"

#define WIDTH  640 /* Source image width */
#define HEIGHT 480 /* Destination image width */

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define check_sts(st)        \
    if ((st) != ippStsNoErr) \
```

```
        goto exitLine; /* Go to Exit if Intel(R) IPP function returned status different from
ippStsNoErr */

static const double srcQuad[4][2] = /* Vertex coordinates of the quadrangle */
    {{0.0, 0.0}, {8.0, 0.0}, {6.0, 8.0}, {2.0, 8.0}};

static const double dstQuad[4][2] = /* Vertex coordinates of the quadrangle */
    {{0.0, 0.0}, {8.0, 0.0}, {6.0, 8.0}, {2.0, 8.0}};

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main(void)
{
    IppStatus status = ippStsNoErr;
    Ipp8u *pSrc = NULL, *pDst = NULL;                              /* Pointers to source/
destination images */
    int srcStep = 0, dstStep = 0;                                  /* Steps, in bytes, through
the source/destination images */
    IppiSize srcSize = {WIDTH, HEIGHT}, dstSize = {WIDTH, HEIGHT}; /* Size of source/destination
ROI in pixels */
    IppiInterpolationType interpolation = ippLinear;
    IppiBorderType borderType =
        ippBorderTransp;           /* Transparent border : destination image pixels mapped to
the outer source image pixels are not changed. */
    int specSize = 0, bufSize = 0; /* Size, in bytes, of the specification/temporary buffer
structure */
    IppiWarpTransformType warpTransformType = ippWarpPerspective;
    IppiWarpSpec *pSpec = NULL; /* Pointer to the specification structure/work buffer */
    Ipp8u *pBuffer = NULL;
    IppiPoint dstOffset = {0, 0}; /* Offset of the destination image ROI with respect to the
destination image origin */
    int numChannels = 3;

    pSrc = ippiMalloc_8u_C3(srcSize.width, srcSize.height, &srcStep);
    pDst = ippiMalloc_8u_C3(dstSize.width, dstSize.height, &dstStep);

    /* Spec and init buffer sizes */
    check_sts(status =
                  ippiWarpQuadGetSize(srcSize, srcQuad, dstSize, dstQuad, warpTransformType,
ipp8u, interpolation, borderType, &specSize, &bufSize));

    pSpec = (IppiWarpSpec *)ippsMalloc_8u(specSize);

    /* Filter initialization */
    check_sts(status = ippiWarpQuadLinearInit(srcSize, srcQuad, dstSize, dstQuad,
warpTransformType, ipp8u, numChannels, borderType, 0, 0, pSpec));

    check_sts(status = ippiWarpGetBufferSize(pSpec, dstSize, &bufSize));

    pBuffer = ippsMalloc_8u(bufSize);
    /* Perform perspective transform of an image */
    check_sts(status = ippiWarpPerspectiveLinear_8u_C3R(pSrc, srcStep, pDst, dstStep, dstOffset,
dstSize, pSpec, pBuffer));

    EXIT_MAIN
    ippsFree(pSpec);
    ippsFree(pBuffer);
```

```
    ippiFree(pSrc);
    ippiFree(pDst);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

See Also

WarpQuadGetSize Computes the size of the specification structure and the size of the temporary buffer for warping an arbitrary quadrangle in the source image to the quadrangle in the destination image.

WarpAffineNearest

Performs warp affine transformation of an image using the nearest neighbor interpolation method.

Syntax

```
IppStatus ippiWarpAffineNearest_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype> pDst, int dstStep, IppiPoint dstRoiOffset, IppiSize dstRoiSize, const
IppiWarpSpec* pSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

8u_C1R	16u_C1R	16s_C1R	32f_C1R	64f_C1R
8u_C3R	16u_C3R	16s_C3R	32f_C3R	64f_C3R
8u_C4R	16u_C4R	16s_C4R	32f_C4R	64f_C4R

Include Files

`ippi.h`

Flavors with the `_L` suffix: `ippi_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image.
`srcStep`	Distance, in bytes, between the starting points of consecutive lines in the source image buffer.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance, in bytes, between the starting points of consecutive lines in the destination image buffer.
`dstRoiOffset`	Offset of the destination image ROI with respect to the destination image origin.
`dstRoiSize`	Size of the destination image ROI, in pixels.

pSpec	Pointer to the specification structure for the warp operation.
pBuffer	Pointer to the work buffer.

Description

This function transforms the source image pixel coordinates (x, y) according to the following formulas:

$x' = c_{00} * x + c_{01} * y + c_{02}$

$y' = c_{10} * x + c_{11} * y + c_{12}$

where

- x' and y' are the pixel coordinates in the transformed image
- c_{ij} are the affine transform coefficients passed to the coeffs array during initialization

The affine warping is a general linear transform that incorporates such elementary transformations as scaling, rotation, translation, stretching, and shearing. It always transforms parallel lines into parallel lines and preserves equal distances between points on a line.

The WarpAffineNearest function operates with ROI. The transformed part of the image is resampled with the nearest neighbor interpolation method and stored in the destination image ROI. You need to define the destination image ROI origin by the following parameters: the offset of the destination ROI with respect to the destination image origin and the destination image ROI size. The parameter pSrc must point to the source image origin. The parameter pDst must point to the processed destination image ROI.

If you initialize the specification structure using the WarpQuadNearestInit function, the operations take place only inside the specified source quadrangle srcQuad that is set in WarpQuadNearestInit.

To specify the algorithm for borders processing, set the borderType and pBorderValue parameters when initializing the IppiWarpSpec structure. The data type of pBorderValue is automatically converted from Ipp64f to the data type of the processed images. The function supports the following algorithms for borders processing:

- If the border type is equal to ippBorderRepl, the source image outer pixels are replicated from the edge pixels.
- If the border type is equal to ippBorderConst, the outer pixels are set to the constant value specified in pBorderValue.
- If the border type is equal to ippBorderTransp or ippBorderInMem, destination image pixels mapped to the outer source image pixels are not changed.

Before using the WarpAffineNearest function, you need to initialize the IppiWarpSpec structure using the WarpAffineNearestInit function and compute the size of the external buffer pBuffer using the WarpGetBufferSize function.

To compute the affine transform parameters, use the GetAffineQuad, GetAffineBound, and GetAffineTransform functions.

For an example on how to use this function, refer to the WarpQuadNearestInit function description.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.

ippStsNoOperation	Indicates a warning when width or height of the destination image is equal to zero.
ippStsBorderErr	Indicates an error if border type has an illegal value.
ippStsContextMatchErr	Indicates an error when context data is invalid.
ippStsNotSupportedModeErr	Indicates an error when the requested mode is not supported.
ippStsSizeErr	Indicates an error when width or height of the source or destination image ROI is negative.
ippStsStepErr	Indicates an error when the step value is not a multiple of data type.
ippStsOutOfRangeErr	Indicates an error when the destination image offset point is outside the destination image origin.
ippStsSizeWrn	Indicates a warning when the destination image ROI size is more than the destination image origin size.
ippStsWrongIntersectQuad	Indicates a warning that no operation is performed if the destination ROI has no intersection with the transformed source image origin.

See Also

ROI Processing in Geometric Transforms

WarpAffineNearestInit Initializes the specification structure for image affine warping with the nearest neighbor interpolation method.

WarpQuadNearestInit Initializes the specification structure for warping an arbitrary quadrangle in the source image to the quadrangle in the destination image with the nearest neighbor interpolation method.

WarpGetBufferSize Computes the size of the work buffer for the warp transform.

GetAffineBound Computes the bounding rectangle for the source ROI transformed by the ippiWarpAffine function.

GetAffineQuad Computes vertex coordinates of the quadrangle, to which the source ROI rectangle is mapped by the affine transform.

GetAffineTransform Computes affine transform coefficients to map the source ROI to the quadrangle with the specified vertex coordinates

WarpAffineLinearInit

Initializes the specification structure for image affine warping with the linear interpolation method.

Syntax

```
IppStatus ippiWarpAffineLinearInit(IppiSize srcSize, IppiSize dstSize, IppDataType
dataType, const double coeffs[2][3], IppiWarpDirection direction, int numChannels,
IppiBorderType borderType, const Ipp64f* pBorderValue, int smoothEdge, IppiWarpSpec*
pSpec);
```

Include Files

`ippi.h`

Flavors with the `_L` suffix: `ippi_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`srcSize`	Size of the source image, in pixels.
`dstSize`	Size of the destination image, in pixels.
`dataType`	Data type of the source and destination images. Supported values: `ipp8u`, `ipp16u`, `ipp16s`, `ipp32f`, and `ipp64f`.
`coeffs`	Coefficients for the affine transform.
`direction`	Transformation direction. Supported values: * `ippWarpForward` - forward transformation * `ippWarpBackward` - backward transformation
`numChannels`	Number of channels in the image. Supported values: 1, 3, or 4.
`borderType`	Type of border. Supported values: * `ippBorderConst` - the outer pixels of the source image are set to the constant value; * `ippBorderRepl` - border is replicated from the edge pixels; * `ippBorderTransp` - outer pixels are not processed; * `ippBorderInMem` - border is obtained from the source image pixels in memory.
`pBorderValue`	Pointer to the constant value to assign to pixels of the constant border. This parameter is applicable only to the `ippBorderConst` border type.
`smoothEdge`	Flag for edge smoothing. Supported values: • `0` - transformation without edge smoothing. • `1` - transformation with edge smoothing. This feature is supported only for the `ippBorderTransp` and `ippBorderInMem` border types.
`pSpec`	Pointer to the specification structure.

Description

This function initializes the `IppiWarpSpec` structure for the ippiWarpAffineLinear function that performs warp affine transformation with the linear interpolation method. To compute the size of the specification structure, use the WarpAffineGetSize function.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when: • one of the specified pointers is NULL, excepting pBorderValue • pBorderValue is NULL when border type is set to ippBorderConst
ippStsNoOperation	Indicates a warning if width or height of any image is zero.
ippStsSizeErr	Indicates an error when width or height of the source or destination image is less than, or equal to one.
ippStsDataTypeErr	Indicates an error when dataType has an illegal value.
ippStsWarpDirectionErr	Indicates an error when direction has an illegal value.
ippStsCoeffErr	Indicates an error when affine transformation is singular.
ippStsWrongIntersectQuad	Indicates a warning that no operation is performed if the transformed source image extended with borders has no intersection with the destination image.
ippStsNotSupportedModeErr	Indicates an error when the requested mode is not supported. The edge smoothing feature is not supported for the ippBorderRepl and ippBorderConst border types.
ippStsBorderErr	Indicates an error when borderType has an illegal value.
ippStsNumChannelsErr	Indicates an error when numChannels has an illegal value.
ippStsExceededSizeErr	Indicates an error if width or height of the destination image or the source image exceeds 33554431 (0x1FFFFFF).

See Also

WarpAffineGetSize Computes the size of the specification structure and the size of the external work buffer for the warp affine transform.

WarpQuadLinearInit

Initializes the specification structure for warping an arbitrary quadrangle in the source image to the quadrangle in the destination image with the linear interpolation method.

Syntax

```
IppStatus ippiWarpQuadLinearInit(IppiSize srcSize, const double srcQuad[4][2], IppiSize dstSize, const double dstQuad[4][2], IppiWarpTransformType transform, IppDataType dataType, int numChannels, IppiBorderType borderType, const Ipp64f* pBorderValue, int smoothEdge, IppiWarpSpec* pSpec);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`srcSize`	Size of the source image, in pixels.
`srcQuad`	Quadrangle in the source image.
`dstSize`	Size of the destination image, in pixels.
`dstQuad`	Quadrangle in the destination image.
`transform`	Type of the warp tranform. Supported values: • `ippWarpAffine` • Affine warping • `ippWarpPerspective` • Perspective warping
`dataType`	Data type of the source and destination images. Supported values: `ipp8u`, `ipp16u`, `ipp16s`, and `ipp32f`.
`numChannels`	Number of channels in the image. Supported values: 1, 3, or 4.
`borderType`	Type of border. Supported values: • `ippBorderTransp` • Outer pixels are not processed. • `ippBorderInMem` • Border is obtained from the source image pixels in memory.
`pBorderValue`	Pointer to the constant value to assign to pixels of the constant border. This parameter is applicable only to the `ippBorderConst` border type.
`smoothEdge`	Flag for edge smoothing. Supported values: • `0` - transformation without edge smoothing. • `1` - transformation with edge smoothing.

pSpec	Pointer to the specification structure.

Description

This function initializes the IppiWarpSpec structure for warping an arbitrary quadrangle in the source image to quadrangle in the destination image with the linear interpolation method. To compute the size of the specification structure, use the WarpQuadGetSize function.

Transformation coefficients are computed internally, based on the mapping of the source quadrangle to the specified destination quadrangle dstQuad and transform type transform. In case of affine transform, the function computes the coordinates of the fourth vertex of the destination quadrangle that uniquely depends on the three other vertices. If the computed coordinates are not equal to the corresponding values specified in dstQuad, the function returns the warning message and continues initialization with the computed values.

The first dimension [4] of the array specifying the quadrangle srcQuad[4][2] or dstQuad[4][2] is equal to the number of vertices, and the second dimension [2] contains the x and y coordinates of the vertex.

For an example on how to use this function, refer to the example provided with the WarpQuadNearestInit function description.

You can apply the edge smoothing feature only if the source quadrangle entirely lies in the source image.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when: • One of the specified pointers is NULL, excepting pBorderValue • The value of pBorderValue is NULL when the border type is set to ippBorderConst
ippStsSizeErr	Indicates an error when width or height of the source or destination image is less than, or equal to one.
ippStsDataTypeErr	Indicates an error when dataType has an illegal value.
ippStsWarpTransformErr	Indicates an error when transform has an illegal value.
ippStsQuadErr	Indicates an error when any of the given quadrangles is non-convex or degenerates into a triangle, line, or point.
ippStsWrongIntersectQuad	Indicates a warning that no operation is performed if the transformed source image extended with borders has no intersection with the destination image.
ippStsAffineQuadChanged	Indicates a warning when coordinates of the fourth vertex of dstQuad are changed by the function, if transform is set to ippWarpAffine.
ippStsBorderErr	Indicates an error when borderType has an illegal value.

ippStsNumChannelsErr	Indicates an error when numChannels has an illegal value.

See Also

WarpQuadGetSize Computes the size of the specification structure and the size of the temporary buffer for warping an arbitrary quadrangle in the source image to the quadrangle in the destination image.

WarpQuadNearestInit Initializes the specification structure for warping an arbitrary quadrangle in the source image to the quadrangle in the destination image with the nearest neighbor interpolation method.

WarpAffineLinear

Performs warp affine transformation of an image using the linear interpolation method.

Syntax

```
IppStatus ippiWarpAffineLinear_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiPoint dstRoiOffset, IppiSize dstRoiSize, const
IppiWarpSpec* pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32f_C1R	64f_C1R
8u_C3R	16u_C3R	16s_C3R	32f_C3R	64f_C3R
8u_C4R	16u_C4R	16s_C4R	32f_C4R	64f_C4R

Include Files

ippi.h

Flavors with the _L suffix: ippi_l.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc	Pointer to the source image.
srcStep	Distance, in bytes, between the starting points of consecutive lines in the source image buffer.
pDst	Pointer to the destination image ROI.
dstStep	Distance, in bytes, between the starting points of consecutive lines in the destination image buffer.
dstRoiOffset	Offset of the destination image ROI with respect to the destination image origin.
dstRoiSize	Size of the destination image ROI, in pixels.

pSpec	Pointer to the specification structure for the warp operation.
pBuffer	Pointer to the work buffer.

Description

This function transforms the source image pixel coordinates (x, y) according to the following formulas:

$x' = c_{00} * x + c_{01} * y + c_{02}$

$y' = c_{10} * x + c_{11} * y + c_{12}$

where

- x' and y' are the pixel coordinates in the transformed image
- c_{ij} are the affine transform coefficients passed to the coeffs array during initialization

The affine warping is a general linear transform that incorporates such elementary transformations as scaling, rotation, translation, stretching, and shearing. It always transforms parallel lines into parallel lines and preserves equal distances between points on a line.

The WarpAffineLinear function operates with ROI. The transformed part of the image is resampled with the linear interpolation method and stored in the destination image ROI. You need to define the destination image ROI origin by the following parameters: the offset of the destination ROI with respect to the destination image origin and the destination image ROI size. The parameter pSrc must point to the source image origin. The parameter pDst must point to the processed destination image ROI.

If you initialize the specification structure using the ippiWarpQuadLinearInit function, the operations take place only inside the specified source quadrangle srcQuad that is set in ippiWarpQuadLinearInit.

To specify the algorithm for borders processing, set the borderType and pBorderValue parameters when initializing the IppiWarpSpec structure. The data type of pBorderValue is automatically converted from Ipp64f to the data type of the processed images. The function supports the following algorithms for borders processing:

- If the border type is equal to ippBorderRepl, the source image outer pixels are replicated from the edge pixels.
- If the border type is equal to ippBorderConst, the outer pixels are set to the constant value specified in pBorderValue.
- If the border type is equal to ippBorderTransp or ippBorderInMem, destination image pixels mapped to the outer source image pixels are not changed.

Before using the ippiWarpAffineLinear function, you need to initialize the IppiWarpSpec structure using the ippiWarpAffineLinearInit function and compute the size of the external buffer pBuffer using the WarpGetBufferSize function.

To compute the affine transform parameters, use the GetAffineQuad, GetAffineBound, and GetAffineTransform functions.

For an example on how to use this function, refer to the WarpQuadNearestInit function description.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.

ippStsNoOperation	Indicates a warning when width or height of the destination image is equal to zero.
ippStsBorderErr	Indicates an error if border type has an illegal value.
ippStsContextMatchErr	Indicates an error when context data is invalid.
ippStsNotSupportedModeErr	Indicates an error when the requested mode is not supported.
ippStsSizeErr	Indicates an error when width or height of the source or destination image ROI is negative.
ippStsStepErr	Indicates an error when the step value is not a multiple of data type.
ippStsOutOfRangeErr	Indicates an error when the destination image offset point is outside the destination image origin.
ippStsSizeWrn	Indicates a warning when the destination image ROI size is more than the destination image origin size.
ippStsWrongIntersectQuad	Indicates a warning that no operation is performed if the destination ROI has no intersection with the transformed source image origin.

See Also

ROI Processing in Geometric Transforms

WarpAffineLinearInit Initializes the specification structure for image affine warping with the linear interpolation method.

WarpQuadLinearInit Initializes the specification structure for warping an arbitrary quadrangle in the source image to the quadrangle in the destination image with the linear interpolation method.

WarpGetBufferSize Computes the size of the work buffer for the warp transform.

GetAffineBound Computes the bounding rectangle for the source ROI transformed by the ippiWarpAffine function.

GetAffineQuad Computes vertex coordinates of the quadrangle, to which the source ROI rectangle is mapped by the affine transform.

GetAffineTransform Computes affine transform coefficients to map the source ROI to the quadrangle with the specified vertex coordinates

WarpQuadNearestInit Initializes the specification structure for warping an arbitrary quadrangle in the source image to the quadrangle in the destination image with the nearest neighbor interpolation method.

WarpAffineCubicInit

Initializes the specification structure for image affine warping with the cubic interpolation method.

Syntax

```
IppStatus ippiWarpAffineCubicInit(IppiSize srcSize, IppiSize dstSize, IppDataType
dataType, const double coeffs[2][3], IppiWarpDirection direction, int numChannels,
Ipp64f valueB, Ipp64f valueC, IppiBorderType borderType, const Ipp64f* pBorderValue,
int smoothEdge, IppiWarpSpec* pSpec, Ipp8u* pInitBuf);
```

Include Files

`ippi.h`

Flavors with the `_L` suffix: `ippi_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`srcSize`	Size of the source image, in pixels.
`dstSize`	Size of the destination image, in pixels.
`dataType`	Data type of the source and destination images. Supported values: `ipp8u`, `ipp16u`, `ipp16s`, `ipp32f`, and `ipp64f`.
`coeffs`	Coefficients for the affine transform.
`direction`	Transformation direction. Supported values: * `ippWarpForward` - forward transformation * `ippWarpBackward` - backward transformation
`numChannels`	Number of channels in the image. Supported values: 1, 3, or 4.
`valueB`, `valueC`	The first (B) and second (C) parameter for the cubic filter.
`borderType`	Type of border. Supported values: * `ippBorderConst` - values of all border pixels are set to a constant; * `ippBorderRepl` - border is replicated from the edge pixels; * `ippBorderTransp` - outer pixels are not processed; * `ippBorderInMem` - border is obtained from the source image pixels in memory.
`pBorderValue`	Pointer to the constant value to assign to pixels of the constant border. This parameter is applicable only to the `ippBorderConst` border type.
`smoothEdge`	The smooth edge flag. The following values are supported: 0 - transform without edge smoothing, 1 - transform with edge smoothing. This feature is supported only for the `ippBorderTransp` and `ippBorderInMem` border types.

pSpec	Pointer to the specification structure.
pInitBuf	Pointer to the temporary buffer for the cubic filter initialization.

Description

This function initializes the IppiWarpSpec structure for the ippiWarpAffineCubic function that performs warp affine transformation with the cubic interpolation method. Before using this function, compute the size of the specification structure and the size of the external buffer pInitBuf using the WarpAffineGetSize function.

Application Notes

Intel IPP warping functions do not support the IPPI_INTER_CUBIC mode. You can use interpolation with two-parameter cubic filters instead. This approach provides the interpolation quality that is comparable with IPPI_INTER_CUBIC. For interpolation formulas refer to Interpolation with Two-Parameter Cubic Filters. You can vary B and C values to get a result that fits the required task.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when: • one of the specified pointers is NULL, excepting pBorderValue • pBorderValue is NULL when border type is set to ippBorderConst
ippStsNoOperation	ndicates a warning if width or height of any image is zero.
ippStsSizeErr	Indicates an error when width or height of the source or destination image is less than, or equal to one.
ippStsDataTypeErr	Indicates an error when dataType has an illegal value.
ippStsWarpDirectionErr	Indicates an error when direction has an illegal value.
ippStsCoeffErr	Indicates an error when affine transformation is singular.
ippStsWrongIntersectQuad	Indicates a warning that no operation is performed if the transformed source image extended with borders has no intersection with the destination image.
ippStsNotSupportedModeErr	Indicates an error when the requested mode is not supported. The edge smoothing feature is not supported for the ippBorderRepl and ippBorderConst border types.
ippStsBorderErr	Indicates an error when borderType has an illegal value.

ippStsNumChannelsErr	Indicates an error when numChannels has an illegal value.
ippStsExceededSizeErr	Indicates an error if width or height of the destination image or the source image exceeds 33554431 (0x1FFFFF).

See Also

WarpAffineGetSize Computes the size of the specification structure and the size of the external work buffer for the warp affine transform.

Interpolation with Two-Parameter Cubic Filters

WarpQuadCubicInit

Initializes the specification structure for warping an arbitrary quadrangle in the source image to the quadrangle in the destination image with the cubic interpolation method.

Syntax

```
IppStatus ippiWarpQuadCubicInit(IppiSize srcSize, const double srcQuad[4][2], IppiSize dstSize, const double dstQuad[4][2], IppiWarpTransformType transform, IppDataType dataType, int numChannels, Ipp64f valueB, Ipp64f valueC, IppiBorderType borderType, const Ipp64f* pBorderValue, int smoothEdge, IppiWarpSpec* pSpec, Ipp8u* pInitBuf);
```

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

srcSize	Size of the source image, in pixels.
srcQuad	Quadrangle in the source image.
dstSize	Size of the destination image, in pixels.
dstQuad	Quadrangle in the destination image.
transform	Type of the warp tranform. Supported values: • ippWarpAffine • Affine warping • ippWarpPerspective • Perspective warping
dataType	Data type of the source and destination images. Supported values: ipp8u, ipp16u, ipp16s, and ipp32f.

numChannels	Number of channels in the image. Supported values: 1, 3, or 4.
valueB, valueC	The first (B) and second (C) parameter for the cubic filter.
borderType	Type of border. Supported values: • ippBorderTransp • Outer pixels are not processed. • ippBorderInMem • Border is obtained from the source image pixels in memory.
pBorderValue	Pointer to the constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.
smoothEdge	Flag for edge smoothing. Supported values: • 0 - transformation without edge smoothing. • 1 - transformation with edge smoothing.
pSpec	Pointer to the specification structure.
pInitBuf	Pointer to the temporary buffer for the cubic filter initialization.

Description

This function initializes the IppiWarpSpec structure for warping an arbitrary quadrangle in the source image to quadrangle in the destination image with the cubic interpolation method. Before using this function, compute the size of the specification structure and the size of the external buffer pInitBuf using the WarpQuadGetSize function.

Transformation coefficients are computed internally, based on the mapping of the source quadrangle to the specified destination quadrangle dstQuad and transform type transform. In case of affine transform, the function computes the coordinates of the fourth vertex of the destination quadrangle that uniquely depends on the three other vertices. If the computed coordinates are not equal to the corresponding values specified in dstQuad, the function returns the warning message and continues initialization with the computed values.

The first dimension [4] of the array specifying the quadrangle srcQuad[4][2] or dstQuad[4][2] is equal to the number of vertices, and the second dimension [2] contains the x and y coordinates of the vertex.

For an example on how to use this function, refer to the example provided with the WarpQuadNearestInit function description.

You can apply the edge smoothing feature only if the source quadrangle entirely lies in the source image.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when: • One of the specified pointers is NULL, excepting pBorderValue

	• The value of `pBorderValue` is `NULL` when the border type is set to `ippBorderConst`
`ippStsSizeErr`	Indicates an error when width or height of the source or destination image is less than, or equal to one.
`ippStsDataTypeErr`	Indicates an error when `dataType` has an illegal value.
`ippStsWarpTransformErr`	Indicates an error when `transform` has an illegal value.
`ippStsQuadErr`	Indicates an error when any of the given quadrangles is non-convex or degenerates into a triangle, line, or point.
`ippStsWrongIntersectQuad`	Indicates a warning that no operation is performed if the transformed source image extended with borders has no intersection with the destination image.
`ippStsAffineQuadChanged`	Indicates a warning when coordinates of the fourth vertex of `dstQuad` are changed by the function, if `transform` is set to `ippWarpAffine`.
`ippStsBorderErr`	Indicates an error when `borderType` has an illegal value.
`ippStsNumChannelsErr`	Indicates an error when `numChannels` has an illegal value.

See Also

WarpQuadGetSize Computes the size of the specification structure and the size of the temporary buffer for warping an arbitrary quadrangle in the source image to the quadrangle in the destination image.

WarpQuadNearestInit Initializes the specification structure for warping an arbitrary quadrangle in the source image to the quadrangle in the destination image with the nearest neighbor interpolation method.

WarpAffineCubic

Performs warp affine transformation of an image using the cubic interpolation method.

Syntax

```
IppStatus ippiWarpAffineCubic_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiPoint dstRoiOffset, IppiSize dstRoiSize, const
IppiWarpSpec* pSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

`8u_C1R`	`16u_C1R`	`16s_C1R`	`32f_C1R`	`64f_C1R`
`8u_C3R`	`16u_C3R`	`16s_C3R`	`32f_C3R`	`64f_C3R`

8u_C4R	16u_C4R	16s_C4R	32f_C4R	64f_C4R

Include Files

ippi.h

Flavors with the _L suffix: ippi_l.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc	Pointer to the source image.
srcStep	Distance, in bytes, between the starting points of consecutive lines in the source image buffer.
pDst	Pointer to the destination image ROI.
dstStep	Distance, in bytes, between the starting points of consecutive lines in the destination image buffer.
dstRoiOffset	Offset of the destination image ROI with respect to the destination image origin.
dstRoiSize	Size of the destination image ROI, in pixels.
pSpec	Pointer to the specification structure for the warp operation.
pBuffer	Pointer to the work buffer.

Description

This function transforms the source image pixel coordinates (x, y) according to the following formulas:

$x' = c_{00} * x + c_{01} * y + c_{02}$

$y' = c_{10} * x + c_{11} * y + c_{12}$

where

- x′ and y′ are the pixel coordinates in the transformed image
- c_{ij} are the affine transform coefficients passed to the coeffs array during initialization

The affine warping is a general linear transform that incorporates such elementary transformations as scaling, rotation, translation, stretching, and shearing. It always transforms parallel lines into parallel lines and preserves equal distances between points on a line.

The WarpAffineCubic function operates with ROI. The transformed part of the image is resampled with the cubic interpolation method and stored in the destination image ROI. You need to define the destination image ROI origin by the following parameters: the offset of the destination ROI with respect to the destination image origin and the destination image ROI size. The parameter pSrc must point to the source image origin. The parameter pDst must point to the processed destination image ROI.

If you initialize the specification structure using the ippiWarpQuadCubicInit function, the operations take place only inside the specified source quadrangle srcQuad that is set in ippiWarpQuadCubicInit.

To specify the algorithm for borders processing, set the `borderType` and `pBorderValue` parameters when initializing the `IppiWarpSpec` structure. The data type of `pBorderValue` is automatically converted from `Ipp64f` to the data type of the processed images. The function supports the following algorithms for borders processing:

- If the border type is equal to `ippBorderRepl`, the source image outer pixels are replicated from the edge pixels.
- If the border type is equal to `ippBorderConst`, the outer pixels are set to the constant value specified in `pBorderValue`.
- If the border type is equal to `ippBorderTransp`, destination image pixels mapped to the outer source image pixels are not changed. The outer pixels required for cubic interpolation are replicated from the edge pixels.
- If the border type is equal to `ippBorderInMem`, destination image pixels mapped to the outer source image pixels are not changed. The outer pixels required for cubic interpolation are obtained from the out of the source image origin space. Cubic interpolation requires additional one-pixel edge from each source image side.
- The mixed border types can be obtained by the bitwise operation `OR` between `ippBorderTransp` and the `ippBorderInMemTop`, `ippBorderInMemBottom`, `ippBorderInMemLeft`, `ippBorderInMemRight` values

Before using the `ippiWarpAffineCubic` function, you need to initialize the `IppiWarpSpec` structure using the ippiWarpAffineCubicInit function and compute the size of the external buffer `pBuffer` using the WarpGetBufferSize function.

To compute the affine transform parameters, use the GetAffineQuad, GetAffineBound, and GetAffineTransform functions.

For an example on how to use this function, refer to the WarpQuadNearestInit function description.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when one of the specified pointers is `NULL`.
`ippStsNoOperation`	Indicates a warning when width or height of the destination image is equal to zero.
`ippStsBorderErr`	Indicates an error if border type has an illegal value.
`ippStsContextMatchErr`	Indicates an error when context data is invalid.
`ippStsNotSupportedModeErr`	Indicates an error when the requested mode is not supported.
`ippStsSizeErr`	Indicates an error when width or height of the source or destination image ROI is negative.
`ippStsStepErr`	Indicates an error when the step value is not a multiple of data type.
`ippStsOutOfRangeErr`	Indicates an error when the destination image offset point is outside the destination image origin.
`ippStsSizeWrn`	Indicates a warning when the destination image ROI size is more than the destination image origin size.

ippStsWrongIntersectQuad	Indicates a warning that no operation is performed, if the transformed source image has no intersection with the destination image.

See Also

ROI Processing in Geometric Transforms

WarpAffineCubicInit Initializes the specification structure for image affine warping with the cubic interpolation method.

WarpQuadCubicInit Initializes the specification structure for warping an arbitrary quadrangle in the source image to the quadrangle in the destination image with the cubic interpolation method.

WarpGetBufferSize Computes the size of the work buffer for the warp transform.

GetAffineBound Computes the bounding rectangle for the source ROI transformed by the ippiWarpAffine function.

GetAffineQuad Computes vertex coordinates of the quadrangle, to which the source ROI rectangle is mapped by the affine transform.

GetAffineTransform Computes affine transform coefficients to map the source ROI to the quadrangle with the specified vertex coordinates

WarpQuadNearestInit Initializes the specification structure for warping an arbitrary quadrangle in the source image to the quadrangle in the destination image with the nearest neighbor interpolation method.

GetPerspectiveQuad

Computes vertex coordinates of the quadrangle, to which the source ROI rectangle is mapped by the perspective transform.

Syntax

```
IppStatus ippiGetPerspectiveQuad(IppiRect srcRoi, double quad[4][2], const double coeffs[3][3]);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`srcRoi`	Region of interest in the source image (of the `IppiRect` type).
`quad`	Output array. Contains vertex coordinates of the quadrangle, to which the source ROI is mapped by the perspective transform function.
`coeffs`	The given perspective transform coefficients.

Description

This function operates with ROI (see ROI Processing in Geometric Transforms).

This function is used as a support function for the WarpPerspectiveNearest, WarpPerspectiveLinear, and WarpPerspectiveCubic functions. It computes vertex coordinates of the quadrangle, to which the source rectangular ROI is mapped by the perspective transform function using the given coefficients `coeffs`.

The first dimension [4] of the array `quad` [4][2] is equal to the number of vertices, and the second dimension [2] means `x` and `y` coordinates of the vertex. Quadrangle vertices have the following meaning:

`quad` [0] corresponds to the transformed top-left corner of the source ROI,

`quad` [1] corresponds to the transformed top-right corner of the source ROI,

`quad` [2] corresponds to the transformed bottom-right corner of the source ROI,

`quad` [3] corresponds to the transformed bottom-left corner of the source ROI.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsSizeErr`	Indicates an error condition if `srcRoi` has a size field with zero or negative value.
`ippStsCoeffErr`	Indicates an error condition if coefficient values are invalid.

GetPerspectiveBound

Computes the bounding rectangle for the source ROI transformed by the `ippiWarpPerspective` function.

Syntax

```
IppStatus ippiGetPerspectiveBound(IppiRect srcRoi, double bound[2][2], const double coeffs[3][3]);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`srcRoi`	Region of interest in the source image (of the `IppiRect` type).
`bound`	Output array. Contains vertex coordinates of the bounding rectangle for the transformed source ROI.
`coeffs`	The given perspective transform coefficients.

Description

This function operates with ROI (see ROI Processing in Geometric Transforms).

This function is used as a support function for the WarpPerspectiveNearest, WarpPerspectiveLinear, and WarpPerspectiveCubic functions. It computes vertex coordinates of the smallest bounding rectangle for the quadrangle `quad`, to which the source ROI is mapped by the perspective transform function using the given coefficients `coeffs`.

`bound` [0] specifies `x`, `y` coordinates of the top-left corner, `bound` [1] specifies `x`, `y` coordinates of the bottom-right corner.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsSizeErr`	Indicates an error condition if `srcRoi` has a size field with zero or negative value.
`ippStsCoeffErr`	Indicates an error condition if coefficient values are invalid.

GetPerspectiveTransform

Computes the perspective transform coefficients to map the source ROI to the quadrangle with the specified vertex coordinates.

Syntax

```
IppStatus ippiGetPerspectiveTransform(IppiRect srcRoi, const double quad[4][2], double coeffs[3][3]);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`srcRoi`	Region of interest in the source image (of the `IppiRect` type).
`quad`	Vertex coordinates of the quadrangle, to which the source ROI is mapped by the perspective transform function.
`coeffs`	Output array. Contains the target perspective transform coefficients.

Description

This function operates with ROI (see ROI Processing in Geometric Transforms).

This function is used as a support function for the WarpPerspectiveNearest, WarpPerspectiveLinear, and WarpPerspectiveCubic functions. It computes the coefficients `coeffs` that should be used by the function to map the source rectangular ROI to the quadrangle with the given vertex coordinates `quad`.

The first dimension [4] of the array `quad` [4][2] is equal to the number of vertices, and the second dimension [2] means `x` and `y` coordinates of the vertex. Quadrangle vertices have the following meaning:

`quad` [0] corresponds to the transformed top-left corner of the source ROI,

`quad` [1] corresponds to the transformed top-right corner of the source ROI,

`quad` [2] corresponds to the transformed bottom-right corner of the source ROI,

`quad` [3] corresponds to the transformed bottom-left corner of the source ROI.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsSizeErr`	Indicates an error condition if `srcRoi` has a size field with zero or negative value.
`ippStsCoeffErr`	Indicates an error condition if coefficient values are invalid.
`ippStsRectErr`	Indicates an error condition if width or height of the `srcRoi` is less than or equal to 1.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

// A simple example of perspective warping of an image
// using Intel(R) Integrated Performance Primitives (Intel(R) IPP) functions:
//     ippiGetPerspectiveTransform
//     ippiWarpPerspectiveSize
//     ippiWarpPerspectiveNearestInit
//     ippiWarpGetBufferSize
//     ippiWarpPerspectiveNearest_32f_C1R

#include <stdio.h>
#include "ipp.h"

#define WIDTH  8 /* source image width */
#define HEIGHT 8 /* source image height */
```

```
/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define check_sts(st)        \
    if ((st) != ippStsNoErr) \
        goto exitLine; /* Go to Exit if Intel(R) IPP function returned status different from
ippStsNoErr */

static Ipp32f pSrc[WIDTH * HEIGHT] = /* Pointers to source images */
    {1.f, 1.f, 1.f, 1.f, 1.f, 1.f, 1.f, 1.f, 1.f, 1.f, 1.f, 5.f, 5.f, 1.f, 1.f, 1.f, 1.f, 1.f,
5.f, 5.f, 5.f, 5.f,
     1.f, 1.f, 1.f, 5.f, 5.f, 5.f, 5.f, 5.f, 5.f, 1.f, 1.f, 5.f, 5.f, 5.f, 5.f, 5.f, 5.f, 1.f,
1.f, 1.f, 1.f, 5.f, 5.f,
     5.f, 5.f, 1.f, 1.f, 1.f, 1.f, 1.f, 5.f, 5.f, 1.f, 1.f, 1.f, 1.f, 1.f, 1.f, 1.f, 1.f, 1.f,
1.f, 1.f};

static const double quad[4][2] = /* Vertex coordinates of the quadrangle */
    {{0.0, 0.0}, {8.0, 0.0}, {6.0, 8.0}, {0.0, 8.0}};

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main(void)
{
    IppStatus status = ippStsNoErr;
    IppiWarpSpec *pSpec = NULL;              /* Pointer to the specification structure for the
forward transform */
    IppiWarpSpec *pSpecB = NULL;             /* Pointer to the specification structure for the
backward transform */
    IppiSize srcSize = {WIDTH, HEIGHT};      /* Size of source ROI in pixels */
    IppiRect srcRoi = {0, 0, WIDTH, HEIGHT}; /* Region of interest in the source image */
    IppiRect dstRoi = {0, 0, WIDTH, HEIGHT}; /* Region of interest in the destination image */
    IppiPoint dstOffset = {0, 0};            /* Offset of the destination image ROI with respect
to the destination image origin */
    IppiPoint dstbOffset = {0, 0};           /* Offset of the back destination image ROI with
respect to the back destination image origin */
    int srcStep = WIDTH * sizeof(Ipp32f), dstStep = srcStep,
        dstbStep = srcStep;                            /* Steps, in bytes, through the source/
destination/back destination images */
    Ipp32f pDst[WIDTH * HEIGHT], pDstB[WIDTH * HEIGHT]; /* Pointers to destination/back
destination images */
    IppiSize dstSize = {WIDTH, HEIGHT};                /* Size of the destination image */
    IppiSize dstbSize = {WIDTH, HEIGHT};               /* Size of the back destination image */
    double coeffs[3][3] = {0};
    int specSize = 0, initSize = 0, bufSize = 0;    /* Buffer sizes for the forward transform */
    int specSizeB = 0, initSizeB = 0, bufSizeB = 0; /* Buffer sizes for the backward transform */
    Ipp8u *pBuffer = NULL, *pBufferB = NULL;        /* Work buffers */
    IppiBorderType borderType = ippBorderTransp;
    IppiWarpDirection direction = ippWarpForward;   /* Forward transformation direction */
    IppiWarpDirection directionB = ippWarpBackward; /* Backward transformation direction */

    check_sts(status = ippiSet_32f_C1R(2, pDst, dstStep, dstSize));

    check_sts(status = ippiSet_32f_C1R(3, pDstB, dstbStep, dstbSize));

    /* Computes perspective transform matrix */
    check_sts(status = ippiGetPerspectiveTransform(srcRoi, quad, (double(*)[3])coeffs));
```

```
    /* Spec and init buffer sizes for the forward transform*/
    check_sts(status = ippiWarpPerspectiveGetSize(srcSize, srcRoi, dstSize, ipp32f, coeffs,
ippNearest, direction, borderType, &specSize, &initSize));

    /* Spec and init buffer sizes for the backward transform*/
    check_sts(status =
                  ippiWarpPerspectiveGetSize(dstSize, dstRoi, dstbSize, ipp32f, coeffs,
ippNearest, directionB, borderType, &specSizeB, &initSizeB));

    pSpec = (IppiWarpSpec *)ippsMalloc_8u(specSize);

    pSpecB = (IppiWarpSpec *)ippsMalloc_8u(specSizeB);

    /* Filter initialization for the forward transform */
    check_sts(status = ippiWarpPerspectiveNearestInit(srcSize, srcRoi, dstSize, ipp32f, coeffs,
direction, 1, borderType, NULL, 0, pSpec));

    /* Filter initialization for the backward transform */
    check_sts(status = ippiWarpPerspectiveNearestInit(dstSize, dstRoi, dstbSize, ipp32f, coeffs,
directionB, 1, borderType, NULL, 0, pSpecB));

    /* Get work buffer size for the forward transform */
    check_sts(status = ippiWarpGetBufferSize(pSpec, dstSize, &bufSize));

    /* Get work buffer size for the backward transform */
    check_sts(status = ippiWarpGetBufferSize(pSpecB, dstSize, &bufSizeB));

    pBuffer = ippsMalloc_8u(bufSize);

    pBufferB = ippsMalloc_8u(bufSizeB);

    /* Warp perspective processing for the forward transform */
    check_sts(status = ippiWarpPerspectiveNearest_32f_C1R(pSrc, srcStep, pDst, dstStep,
dstOffset, dstSize, pSpec, pBuffer));

    /* Warp perspective processing for the backward transform */
    check_sts(status = ippiWarpPerspectiveNearest_32f_C1R(pDst, dstStep, pDstB, dstbStep,
dstbOffset, dstbSize, pSpecB, pBufferB));

    EXIT_MAIN
    ippsFree(pSpec);
    ippsFree(pSpecB);
    ippsFree(pBuffer);
    ippsFree(pBufferB);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}

/*
pDst
1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0
1.0  1.0  1.0  5.0  5.0  1.0  1.0  1.0
1.0  1.0  1.0  5.0  5.0  1.0  1.0  1.0
1.0  1.0  5.0  5.0  5.0  5.0  1.0  1.0
1.0  5.0  5.0  5.0  5.0  5.0  5.0  1.0
1.0  5.0  5.0  5.0  5.0  5.0  5.0  2.0
1.0  1.0  5.0  5.0  5.0  5.0  1.0  2.0
```

```
1.0  1.0  1.0  5.0  5.0  1.0  1.0  2.0

pDstB
1.0  1.0  1.0  1.0  1.0  1.0  1.0  3.0
1.0  1.0  1.0  5.0  5.0  1.0  1.0  3.0
1.0  1.0  5.0  5.0  5.0  5.0  1.0  3.0
1.0  5.0  5.0  5.0  5.0  5.0  5.0  1.0
1.0  5.0  5.0  5.0  5.0  5.0  5.0  2.0
1.0  1.0  5.0  5.0  5.0  5.0  1.0  1.0
3.0  3.0  3.0  3.0  3.0  3.0  3.0  3.0
3.0  3.0  3.0  3.0  3.0  3.0  3.0  3.0
*/
```

WarpGetRectInfinite

Returns an infinite rectangle.

Syntax

```
IppiRect ippiWarpGetRectInfinite(void);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Description

The function returns a constant rectangle that is considered as an infinite rectangle by Intel IPP `WarpPerspective` functions. Use this rectangle in the following functions: `WarpPerspectiveGetSize`, `WarpPerspectiveNearestInit`, `WarpPerspectiveLinearInit`, and `WarpPerspectiveCubicInit`.

NOTE The macro definition is: `#define ippRectInfinite ippiWarpGetRectInfinite()`.

WarpPerspectiveGetSize

Computes the size of the specification structure and the size of the external work buffer for the warp perspective transform.

Syntax

```
IppStatus ippiWarpPerspectiveGetSize(IppiSize srcSize, IppiRect srcRoi, IppiSize
dstSize, IppDataType dataType, const double coeffs[3][3], IppiInterpolationType
interpolation, IppiWarpDirection direction, IppiBorderType borderType, int* pSpecSize,
int* pInitBufSize);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`srcSize`	Size of the source image, in pixels.
`srcRoi`	Source image ROI (of the IppiRect type). Set this parameter to `ippRectInfinite` to use the entire source image extended by outer pixels according to the border type.
`dstSize`	Size of the destination image, in pixels.
`dataType`	Data type of the source and destination images. Supported values: `ipp8u`, `ipp16u`, `ipp16s`, and `ipp32f`.
`coeffs`	Coefficients for the perspective transform.
`interpolation`	Interpolation method. Supported values: `ippNearest`, `ippLinear`, and `ippCubic`.
`direction`	Transformation direction. Supported values: • `ippWarpForward` • Forward transformation • `ippWarpBackward` • Backward transformation
`borderType`	Type of border. Supported values: • `ippBorderConst` • Values of all border pixels are set to a constant. • `ippBorderRepl` • Border is replicated from the edge pixels. • `ippBorderTransp` • Outer pixels are not processed. • `ippBorderInMem` • Border is obtained from the source image pixels in memory.
`pSpecSize`	Pointer to the size, in bytes, of the specification structure.
`pInitBufSize`	Pointer to the size, in bytes, of the temporary buffer.

Description

This function computes the size of the specification structure and the external work buffer for the following functions, depending on the `interpolation` parameter: ippiWarpPerspectiveNearestInit, ippiWarpPerspectiveLinearInit, or ippiWarpPerspectiveCubicInit.

You can set the value of the `srcRoi` parameter to `ippRectInfinite`, which means that the ROI is not specified.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error in the following cases: • If the width or height of the source or destination image is less than, or equal to one. • If one of the calculated sizes exceeds the maximum positive 32-bit signed integer value. The size of the one of the processed images is too large.
ippStsRectErr	Indicates an error in the following cases, if the source image ROI is not ippRectInfinite: • If the source image ROI has no intersection with the image. • Either x or y component of the source image ROI is negative. • Width or height of the source image ROI is less than, or equal to zero.
ippStsDataTypeErr	Indicates an error when dataType has an illegal value.
ippStsWarpDirectionErr	Indicates an error when direction has an illegal value.
ippStsInterpolationErr	Indicates an error when interpolation has an illegal value.
ippStsCoeffErr	Indicates an error when perspective transformation is singular.
ippStsWrongIntersectQuad	Indicates a warning that no operation is performed if the transformed source image extended with borders has no intersection with the destination image.
ippStsBorderErr	Indicates an error when borderType has an illegal value.
ippStsSizeWrn	Indicates a warning when srcRoi exceeds the source image.

See Also

WarpPerspectiveNearestInit Initializes the specification structure for image perspective warping with the nearest neighbor interpolation method.

WarpPerspectiveLinearInit Initializes the specification structure for image perspective warping with the linear interpolation method.

WarpPerspectiveCubicInit Initializes the specification structure for image perspective warping with the cubic interpolation method.

WarpPerspectiveNearestInit

Initializes the specification structure for image perspective warping with the nearest neighbor interpolation method.

Syntax

```
IppStatus ippiWarpPerspectiveNearestInit(IppiSize srcSize, IppiRect srcRoi, IppiSize dstSize, IppDataType dataType, const double coeffs[3][3], IppiWarpDirection direction, int numChannels, IppiBorderType borderType, const Ipp64f* pBorderValue, int smoothEdge, IppiWarpSpec* pSpec);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`srcSize`	Size of the source image, in pixels.
`srcRoi`	Source image ROI (of the IppiRect type). Set this parameter to `ippRectInfinite` to use the entire source image extended by outer pixels according to the border type.
`dstSize`	Size of the destination image, in pixels.
`dataType`	Data type of the source and destination images. Supported values: `ipp8u`, `ipp16u`, `ipp16s`, and `ipp32f`.
`coeffs`	Coefficients for the perspective transform.
`direction`	Transformation direction. Supported values: * `ippWarpForward` - forward transformation * `ippWarpBackward` - backward transformation
`numChannels`	Number of channels in the image. Supported values: 1, 3, or 4.
`borderType`	Type of border. Supported values: * `ippBorderConst` - values of all border pixels are set to a constant; * `ippBorderRepl` - border is replicated from the edge pixels; * `ippBorderTransp` - outer pixels are not processed; * `ippBorderInMem` - border is obtained from the source image pixels in memory.

pBorderValue	Pointer to the constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.
smoothEdge	Flag for edge smoothing. Supported values: • 0 - transformation without edge smoothing. • 1 - transformation`` with edge smoothing. This feature is supported only for the ippBorderTransp and ippBorderInMem border types.
pSpec	Pointer to the specification structure.

Description

This function initializes the IppiWarpSpec structure for the WarpPerspectiveNearest function that performs warp perspective transformation with the nearest neighbor interpolation method. To compute the size of the specification structure, use the WarpPerspectiveGetSize function.

You can set the value of the srcRoi parameter to ippRectInfinite, which means that the ROI is not specified.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when: • pSpec is NULL • pBorderValue is NULL when the border type is set to ippBorderConst
ippStsSizeErr	Indicates an error when width or height of the source or destination image is less than, or equal to one.
ippStsRectErr	Indicates an error in the following cases, if the source image ROI is not ippRectInfinite: • If the source image ROI has no intersection with the image. • Either x or y component of the source image ROI is negative. • Width or height of the source image ROI is less than, or equal to zero.
ippStsDataTypeErr	Indicates an error when dataType has an illegal value.
ippStsWarpDirectionErr	Indicates an error when direction has an illegal value.

ippStsCoeffErr	Indicates an error when perspective transformation is singular.
ippStsWrongIntersectQuad	Indicates a warning that no operation is performed if the transformed source image extended with borders has no intersection with the destination image.
ippStsNotSupportedModeErr	Indicates an error when the requested mode is not supported. The edge smoothing feature is not supported for the ippBorderRepl and ippBorderConst border types.
ippStsBorderErr	Indicates an error when borderType has an illegal value.
ippStsNumChannelsErr	Indicates an error when numChannels has an illegal value.
ippStsSizeWrn	Indicates a warning when srcRoi exceeds the source image.

See Also

WarpPerspectiveGetSize Computes the size of the specification structure and the size of the external work buffer for the warp perspective transform.

WarpPerspectiveNearest Performs warp perspective transformation of an image using the nearest neighbor interpolation method.

WarpPerspectiveNearest

Performs warp perspective transformation of an image using the nearest neighbor interpolation method.

Syntax

```
IppStatus ippiWarpPerspectiveNearest_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype> pDst, int dstStep, IppiPoint dstRoiOffset, IppiSize dstRoiSize, const
IppiWarpSpec* pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_C4R	16u_C4R	16s_C4R	32f_C4R

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc	Pointer to the source image.
srcStep	Distance, in bytes, between the starting points of consecutive lines in the source image buffer.
pDst	Pointer to the destination image ROI.
dstStep	Distance, in bytes, between the starting points of consecutive lines in the destination image buffer.
dstRoiOffset	Offset of the destination image ROI with respect to the destination image origin.
dstRoiSize	Size of the destination image ROI, in pixels.
pSpec	Pointer to the specification structure for the warp operation.
pBuffer	Pointer to the work buffer.

Description

This function transforms the source image pixel coordinates (x, y) according to the following formulas:

$x' = (c_{00} * x + c_{01} * y + c_{02}) / (c_{20} * x + c_{21} * y + c_{22})$

$y' = (c_{10} * x + c_{11} * y + c_{12}) / (c_{20} * x + c_{21} * y + c_{22})$

where

- x' and y' are the pixel coordinates in the transformed image
- c_{ij} are the affine transform coefficients passed to the coeffs array during initialization

The ippiWarpPerspectiveNearest function operates with ROI (see Regions of Interest in Intel IPP). The transformed part of the source image is resampled with the nearest neighbor interpolation method and stored in the destination image ROI. You need to define the destination image ROI origin by the following parameters: the offset of the destination ROI with respect to the destination image origin and the destination image ROI size. The parameter pDst must point to the processed destination image ROI.

If you initialize the warp specification structure using the WarpPerspectiveNearestInit function, you can specify the source image ROI in the following ways:

- Set the srcRoi value to ippRectInfinite, which means that the ROI is not specified. In this case, pSrc must point to the processed source image. Pixels that are outside the source image boundaries are computed according to the specified border type.
- Set the srcRoi value to the part of the processed source image. In this case, pSrc must point to the processed source image ROI. The operations take place only inside the specified region of interest srcRoi. It means that the destination image pixels mapped to the outer pixels of the specified source image region are not changed.

If you initialize the warp specification structure using the WarpQuadNearestInit function, set the pSrc value to the processed source image. The operations take place only inside the source quadrangle srcQuad that is specified in the WarpQuadNearestInit function.

To specify the algorithm for borders processing, set the borderType and pBorderValue parameters when initializing the IppiWarpSpec structure. The data type of borderValue is automatically converted from Ipp64f to the data type of the processed images. The function supports the following algorithms for borders processing:

- If the border type is equal to `ippBorderRepl`, the source image outer pixels are replicated from the edge pixels.
- If the border type is equal to `ippBorderConst`, the outer pixels are set to the constant value specified in `pBorderValue`.
- If the border type is equal to `ippBorderTransp` or `ippBorderInMem`, destination image pixels mapped to the outer source image pixels are not changed.

Before using the `ippiWarpPerspectiveNearest` function, you need to initialize the `IppiWarpSpec` structure using the WarpPerspectiveNearestInit function and compute the size of the external buffer `pBuffer` using the WarpGetBufferSize function.

To compute the perspective transform parameters, use the GetPerspectiveQuad, GetPerspectiveBound, and GetPerspectiveTransform functions.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when one of the specified pointers is `NULL`.
`ippStsNoOperation`	Indicates a warning when width or height of the destination image is equal to zero.
`ippStsContextMatchErr`	Indicates an error when context data is invalid.
`ippStsNotSupportedModeErr`	Indicates an error when the requested mode is not supported.
`ippStsSizeErr`	Indicates an error when width or height of the source or destination image ROI is negative.
`ippStsStepErr`	Indicates an error when the step value is not a multiple of data type.
`ippStsOutOfRangeErr`	Indicates an error when the destination image offset point is outside the destination image origin.
`ippStsSizeWrn`	Indicates a warning when the destination image ROI size is more than the destination image origin size.
`ippStsWrongIntersectQuad`	Indicates a warning that no operation is performed if the transformed source image extended with borders has no intersection with the destination image.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
```

```
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *****************************************************************************/

// A simple example of the warp perspective transform
// using Intel(R) Integrated Performance Primitives (Intel(R) IPP) functions:
//     ippiWarpPerspectiveGetSize
//     ippiWarpPerspectiveCubicInit
//     ippiWarpGetBufferSize
//     ippiWarpPerspectiveCubic_8u_C3R

#include <stdio.h>
#include "ipp.h"

#define WIDTH  640 /* Source image width */
#define HEIGHT 480 /* Destination image width */

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define check_sts(st)        \
    if ((st) != ippStsNoErr) \
        goto exitLine; /* Go to Exit if Intel(R) IPP function returned status different from
ippStsNoErr */

static const double coeffs[3][3] = {/* Coefficients for the perspective transform */
                                    {
                                        2.3439242437790240,
                                        0.0025160437043002296,
                                        -499.09988916944815,
                                    },
                                    {
                                        0.0058031616363677600,
                                        1.4997057893818542,
                                        -181.04366815436885,
                                    },
                                    {1.1527998857985334e-005, 1.9607666631981373e-006,
0.99730924424366529}};

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main(void)
{
    IppStatus status = ippStsNoErr;
    Ipp8u *pSrc = NULL, *pDst = NULL;                              /* Pointers to source/
destination images */
    int srcStep = 0, dstStep = 0;                                  /* Steps, in bytes, through
the source/destination images */
    IppiSize srcSize = {WIDTH, HEIGHT}, dstSize = {WIDTH, HEIGHT}; /* Size of source/destination
ROI in pixels */
    IppiWarpSpec *pSpec = NULL;                                    /* Pointer to the
specification structure */
    Ipp8u *pInitBuf = NULL;                                        /* Pointer to temporary
buffer for transform initialization*/
    Ipp8u *pBuffer = NULL;                                         /* Pointer to the work buffer
```

```
*/

    pSrc = ippiMalloc_8u_C3(srcSize.width, srcSize.height, &srcStep);
    pDst = ippiMalloc_8u_C3(dstSize.width, dstSize.height, &dstStep);

    /* Start perspective transform */
    {
        int specSize = 0; /* Size, in bytes, of the specification structure */
        int initSize = 0; /* Size, in bytes, of the temporary buffer */
        int bufSize = 0;  /* Size, in bytes, of the work buffer */
        const Ipp32s numChannels = 3;
        IppiPoint dstOffset = {0, 0}; /* Offset of the destination image ROI with respect to the
destination image origin */
        IppStatus status = ippStsNoErr;
        IppiBorderType borderType = ippBorderConst;
        IppiWarpDirection direction = ippWarpForward; /* Transformation direction */
        Ipp64f pBorderValue[3];
        Ipp64f valB = 0.0, valC = 0.5;     /* Catmull-Rom filter coefficients for cubic
interpolation */
        IppiRect srcRoi = ippRectInfinite; /* Region of interest is not specified */
        int smoothEdge = 0;                /* Flag for edge smoothing */
        IppiInterpolationType interpolation = ippCubic;
        int i = 0;

        for (i = 0; i < numChannels; ++i)
            pBorderValue[i] = 255.0;

        /* Computes the size of the specification structure and the size of the external work
buffer for the warp perspective transform */
        check_sts(
            status = ippiWarpPerspectiveGetSize(srcSize, srcRoi, dstSize, ipp8u, coeffs,
interpolation, direction, borderType, &specSize, &initSize));

        pSpec = (IppiWarpSpec *)ippsMalloc_8u(specSize);
        pInitBuf = ippsMalloc_8u(initSize);

        /* Initializes the specification structure for image perspective warping with the cubic
interpolation method */
        check_sts(status = ippiWarpPerspectiveCubicInit(srcSize, srcRoi, dstSize, ipp8u, coeffs,
direction, numChannels, valB, valC, borderType,
                                                        pBorderValue, smoothEdge, pSpec,
pInitBuf));

        /* Work buffer size */
        check_sts(status = ippiWarpGetBufferSize(pSpec, dstSize, &bufSize));

        pBuffer = ippsMalloc_8u(bufSize);

        /* Performs warp perspective transformation of an image using the cubic interpolation
method */
        check_sts(status = ippiWarpPerspectiveCubic_8u_C3R(pSrc, srcStep, pDst, dstStep,
dstOffset, dstSize, pSpec, pBuffer));
    }

    EXIT_MAIN
    ippsFree(pInitBuf);
    ippsFree(pSpec);
    ippsFree(pBuffer);
```

```
    ippiFree(pSrc);
    ippiFree(pDst);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

See Also

ROI Processing in Geometric Transforms

WarpPerspectiveNearestInit Initializes the specification structure for image perspective warping with the nearest neighbor interpolation method.

WarpQuadNearestInit Initializes the specification structure for warping an arbitrary quadrangle in the source image to the quadrangle in the destination image with the nearest neighbor interpolation method.

WarpGetBufferSize Computes the size of the work buffer for the warp transform.

GetPerspectiveBound Computes the bounding rectangle for the source ROI transformed by the ippiWarpPerspective function.

GetPerspectiveQuad Computes vertex coordinates of the quadrangle, to which the source ROI rectangle is mapped by the perspective transform.

GetPerspectiveTransform Computes the perspective transform coefficients to map the source ROI to the quadrangle with the specified vertex coordinates.

WarpPerspectiveLinearInit

Initializes the specification structure for image perspective warping with the linear interpolation method.

Syntax

```
IppStatus ippiWarpPerspectiveLinearInit(IppiSize srcSize, IppiRect srcRoi, IppiSize
dstSize, IppDataType dataType, const double coeffs[3][3], IppiWarpDirection direction,
int numChannels, IppiBorderType borderType, const Ipp64f* pBorderValue, int smoothEdge,
IppiWarpSpec* pSpec);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`srcSize`	Size of the source image, in pixels.
`srcRoi`	Source image ROI (of the IppiRect type). Set this parameter to `ippRectInfinite` to use the entire source image extended by outer pixels according to the border type.
`dstSize`	Size of the destination image, in pixels.

dataType	Data type of the source and destination images. Supported values: ipp8u, ipp16u, ipp16s, and ipp32f.
coeffs	Coefficients for the perspective transform.
direction	Transformation direction. Supported values: • ippWarpForward • Forward transformation • ippWarpBackward • Backward transformation
numChannels	Number of channels in the image. Supported values: 1, 3, or 4.
borderType	Type of border. Supported values: * ippBorderConst - values of all border pixels are set to a constant; * ippBorderRepl - border is replicated from the edge pixels; * ippBorderTransp - outer pixels are not processed; * ippBorderInMem - border is obtained from the source image pixels in memory.
pBorderValue	Pointer to the constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.
smoothEdge	Flag for edge smoothing. Supported values: • 0 - transformation without edge smoothing. • 1 - transformation`` with edge smoothing. This feature is supported only for the ippBorderTransp and ippBorderInMem border types.
pSpec	Pointer to the specification structure.

Description

This function initializes the IppiWarpSpec structure for the ippiWarpPerspectiveLinear function that performs warp perspective transformation with the linear interpolation method. To compute the size of the specification structure, use the WarpPerspectiveGetSize function.

You can set the value of the srcRoi parameter to ippRectInfinite, which means that the ROI is not specified.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when: • pSpec is NULL • pBorderValue is NULL when the border type is set to ippBorderConst

ippStsSizeErr	Indicates an error when width or height of the source or destination image is less than, or equal to one.
ippStsRectErr	Indicates an error in the following cases, if the source image ROI is not ippRectInfinite: • If the source image ROI has no intersection with the image. • Either x or y component of the source image ROI is negative. • Width or height of the source image ROI is less than, or equal to zero.
ippStsDataTypeErr	Indicates an error when dataType has an illegal value.
ippStsWarpDirectionErr	Indicates an error when direction has an illegal value.
ippStsCoeffErr	Indicates an error when perspective transformation is singular.
ippStsWrongIntersectQuad	Indicates a warning that no operation is performed if the transformed source image extended with borders has no intersection with the destination image.
ippStsNotSupportedModeErr	Indicates an error when the requested mode is not supported. The edge smoothing feature is not supported for the ippBorderRepl and ippBorderConst border types.
ippStsBorderErr	Indicates an error when borderType has an illegal value.
ippStsNumChannelsErr	Indicates an error when numChannels has an illegal value.
ippStsSizeWrn	Indicates a warning when srcRoi exceeds the source image.

See Also

WarpPerspectiveGetSize Computes the size of the specification structure and the size of the external work buffer for the warp perspective transform.

WarpPerspectiveLinear Performs warp perspective transformation of an image using the linear interpolation method.

WarpPerspectiveLinear

Performs warp perspective transformation of an image using the linear interpolation method.

Syntax

```
IppStatus ippiWarpPerspectiveLinear_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype> pDst, int dstStep, IppiPoint dstRoiOffset, IppiSize dstRoiSize, const
IppiWarpSpec* pSpec, Ipp8u* pBuffer);
```

Supported values for `mod`:

`8u_C1R`	`16u_C1R`	`16s_C1R`	`32f_C1R`
`8u_C3R`	`16u_C3R`	`16s_C3R`	`32f_C3R`
`8u_C4R`	`16u_C4R`	`16s_C4R`	`32f_C4R`

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image.
`srcStep`	Distance, in bytes, between the starting points of consecutive lines in the source image buffer.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance, in bytes, between the starting points of consecutive lines in the destination image buffer.
`dstRoiOffset`	Offset of the destination image ROI with respect to the destination image origin.
`dstRoiSize`	Size of the destination image ROI, in pixels.
`pSpec`	Pointer to the specification structure for the warp operation.
`pBuffer`	Pointer to the work buffer.

Description

This function transforms the source image pixel coordinates (`x`, `y`) according to the following formulas:

$x' = (c_{00} * x + c_{01} * y + c_{02}) / (c_{20} * x + c_{21} * y + c_{22})$

$y' = (c_{10} * x + c_{11} * y + c_{12}) / (c_{20} * x + c_{21} * y + c_{22})$

where

- x' and y' are the pixel coordinates in the transformed image
- c_{ij} are the affine transform coefficients passed to the `coeffs` array during initialization

The ippiWarpPerspectiveLinear function operates with ROI (see Regions of Interest in Intel IPP). The transformed part of the source image is resampled with the linear interpolation method and stored in the destination image ROI. You need to define the destination image ROI origin by the following parameters: the offset of the destination ROI with respect to the destination image origin and the destination image ROI size. The parameter pDst must point to the processed destination image ROI.

If you initialize the warp specification structure using the ippiWarpPerspectiveLinearInit function, you can specify the source image ROI in the following ways:

- Set the srcRoi value to ippRectInfinite, which means that the ROI is not specified. In this case, pSrc must point to the processed source image. Pixels that are outside the source image boundaries are computed according to the specified border type.
- Set the srcRoi value to the part of the processed source image. In this case, pSrc must point to the processed source image ROI. The operations take place only inside the specified region of interest srcRoi. It means that the destination image pixels mapped to the outer pixels of the specified source image region are not changed.

If you initialize the warp specification structure using the ippiWarpQuadLinearInit function, set the pSrc value to the processed source image. The operations take place only inside the source quadrangle srcQuad that is specified in the ippiWarpQuadLinearInit function.

To specify the algorithm for borders processing, set the borderType and pBorderValue parameters when initializing the IppiWarpSpec structure. The data type of pBorderValue is automatically converted from Ipp64f to the data type of the processed images. The function supports the following algorithms for borders processing:

- If the border type is equal to ippBorderRepl, the source image outer pixels are replicated from the edge pixels.
- If the border type is equal to ippBorderConst, the outer pixels are set to the constant value specified in pBorderValue.
- If the border type is equal to ippBorderTransp or ippBorderInMem, destination image pixels mapped to the outer source image pixels are not changed.

Before using the ippiWarpPerspectiveLinear function, you need to initialize the IppiWarpSpec structure using the ippiWarpPerspectiveLinearInit function and compute the size of the external buffer pBuffer using the ippiWarpGetBufferSize function.

To compute the perspective transform parameters, use the ippiGetPerspectiveQuad, ippiGetPerspectiveBound, and ippiGetPerspectiveTransform functions.

For an example on how to use this functionality, refer to the example provided with the ippiWarpPerspectiveNearest function description.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsNoOperation	Indicates a warning when width or height of the destination image is equal to zero.
ippStsContextMatchErr	Indicates an error when context data is invalid.
ippStsNotSupportedModeErr	Indicates an error when the requested mode is not supported.

ippStsSizeErr	Indicates an error when width or height of the source or destination image ROI is negative.
ippStsStepErr	Indicates an error when the step value is not a multiple of data type.
ippStsOutOfRangeErr	Indicates an error when the destination image offset point is outside the destination image origin.
ippStsSizeWrn	Indicates a warning when the destination image ROI size is more than the destination image origin size.
ippStsWrongIntersectQuad	Indicates a warning that no operation is performed if the transformed source image extended with borders has no intersection with the destination image.

See Also

ROI Processing in Geometric Transforms

WarpPerspectiveLinearInit Initializes the specification structure for image perspective warping with the linear interpolation method.

WarpQuadLinearInit Initializes the specification structure for warping an arbitrary quadrangle in the source image to the quadrangle in the destination image with the linear interpolation method.

WarpGetBufferSize Computes the size of the work buffer for the warp transform.

GetPerspectiveBound Computes the bounding rectangle for the source ROI transformed by the ippiWarpPerspective function.

GetPerspectiveQuad Computes vertex coordinates of the quadrangle, to which the source ROI rectangle is mapped by the perspective transform.

GetPerspectiveTransform Computes the perspective transform coefficients to map the source ROI to the quadrangle with the specified vertex coordinates.

WarpPerspectiveNearest Performs warp perspective transformation of an image using the nearest neighbor interpolation method.

WarpPerspectiveCubicInit

Initializes the specification structure for image perspective warping with the cubic interpolation method.

Syntax

```
IppStatus ippiWarpPerspectiveCubicInit(IppiSize srcSize, IppiRect srcRoi, IppiSize dstSize, IppDataType dataType, const double coeffs[3][3], IppiWarpDirection direction, int numChannels, Ipp64f valueB, Ipp64f valueC, IppiBorderType borderType, const Ipp64f* pBorderValue, int smoothEdge, IppiWarpSpec* pSpec, Ipp8u* pInitBuf);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`srcSize`	Size of the source image, in pixels.
`srcRoi`	Source image ROI (of the IppiRect type). Set this parameter to `ippRectInfinite` to use the entire source image extended by outer pixels according to the border type.
`dstSize`	Size of the destination image, in pixels.
`dataType`	Data type of the source and destination images. Supported values: `ipp8u`, `ipp16u`, `ipp16s`, and `ipp32f`.
`coeffs`	Coefficients for the perspective transform.
`direction`	Transformation direction. Supported values: • `ippWarpForward` • Forward transformation • `ippWarpBackward` • Backward transformation
`numChannels`	Number of channels in the image. Supported values: 1, 3, or 4.
`valueB`, `valueC`	The first (B) and second (C) parameter for the cubic filter.
`borderType`	Type of border. Supported values: • `ippBorderConst` • Values of all border pixels are set to a constant. • `ippBorderRepl` • Border is replicated from the edge pixels. • `ippBorderTransp` • Outer pixels are not processed. • `ippBorderInMem` • Border is obtained from the source image pixels in memory.
`pBorderValue`	Pointer to the constant value to assign to pixels of the constant border. This parameter is applicable only to the `ippBorderConst` border type.
`smoothEdge`	Flag to enable/disable edge smoothing. Possible values: 0 - transform without edge smoothing, 1 - transform with edge smoothing. This feature is supported only for the `ippBorderTransp` and `ippBorderInMem` border types.
`pSpec`	Pointer to the specification structure.

pInitBuf	Pointer to the temporary buffer for the cubic filter initialization.

Description

This function initializes the IppiWarpSpec structure for the ippiWarpPerspectiveCubic function that performs that performs warp perspective transformation with the cubic interpolation method. Before using this function, compute the size of the specification structure and the size of the external buffer pInitBuf using the WarpPerspectiveGetSize function.

You can set the value of the srcRoi parameter to ippRectInfinite, which means that the ROI is not specified.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when: • pSpec is NULL • pBorderValue is NULL when the border type is set to ippBorderConst
ippStsSizeErr	Indicates an error when width or height of the source or destination image is less than, or equal to one.
ippStsRectErr	Indicates an error in the following cases, if the source image ROI is not ippRectInfinite: • If the source image ROI has no intersection with the image. • Either x or y component of the source image ROI is negative. • Width or height of the source image ROI is less than, or equal to zero.
ippStsDataTypeErr	Indicates an error when dataType has an illegal value.
ippStsWarpDirectionErr	Indicates an error when direction has an illegal value.
ippStsCoeffErr	Indicates an error when perspective transformation is singular.
ippStsWrongIntersectQuad	Indicates a warning that no operation is performed if the transformed source image extended with borders has no intersection with the destination image.
ippStsNotSupportedModeErr	Indicates an error when the requested mode is not supported. The edge smoothing feature is not supported for the ippBorderRepl and ippBorderConst border types.

ippStsBorderErr	Indicates an error when borderType has an illegal value.
ippStsNumChannelsErr	Indicates an error when numChannels has an illegal value.
ippStsSizeWrn	Indicates a warning when srcRoi exceeds the source image.

See Also

WarpPerspectiveGetSize Computes the size of the specification structure and the size of the external work buffer for the warp perspective transform.

WarpPerspectiveCubic Performs warp perspective transformation of an image using the cubic interpolation method.

WarpPerspectiveCubic

Performs warp perspective transformation of an image using the cubic interpolation method.

Syntax

```
IppStatus ippiWarpPerspectiveCubic_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype> pDst, int dstStep, IppiPoint dstRoiOffset, IppiSize dstRoiSize, const
IppiWarpSpec* pSpec, Ipp8u* pBuffer);
```

Supported values for mod:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_C4R	16u_C4R	16s_C4R	32f_C4R

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc	Pointer to the source image.
srcStep	Distance, in bytes, between the starting points of consecutive lines in the source image buffer.
pDst	Pointer to the destination image ROI.
dstStep	Distance, in bytes, between the starting points of consecutive lines in the destination image buffer.

dstRoiOffset	Offset of the destination image ROI with respect to the destination image origin.
dstRoiSize	Size of the destination image ROI, in pixels.
pSpec	Pointer to the specification structure for the warp operation.
pBuffer	Pointer to the work buffer.

Description

This function transforms the source image pixel coordinates (x, y) according to the following formulas:

$x' = (c_{00} * x + c_{01} * y + c_{02}) / (c_{20} * x + c_{21} * y + c_{22})$

$y' = (c_{10} * x + c_{11} * y + c_{12}) / (c_{20} * x + c_{21} * y + c_{22})$

where

- x' and y' are the pixel coordinates in the transformed image
- c_{ij} are the affine transform coefficients passed to the coeffs array during initialization

The ippiWarpPerspectiveCubic function operates with ROI (see Regions of Interest in Intel IPP). The transformed part of the source image is resampled with the cubic interpolation method and stored in the destination image ROI. You need to define the destination image ROI origin by the following parameters: the offset of the destination ROI with respect to the destination image origin and the destination image ROI size. The parameter pDst must point to the processed destination image ROI.

If you initialize the warp specification structure using the ippiWarpPerspectiveCubicInit function, you can specify the source image ROI in the following ways:

- Set the srcRoi value to ippRectInfinite, which means that the ROI is not specified. In this case, pSrc must point to the processed source image. Pixels that are outside the source image boundaries are computed according to the specified border type.
- Set the srcRoi value to the part of the processed source image. In this case, pSrc must point to the processed source image ROI. The operations take place only inside the specified region of interest srcRoi. It means that the destination image pixels mapped to the outer pixels of the specified source image region are not changed.

If you initialize the warp specification structure using the ippiWarpQuadCubicInit function, set the pSrc value to the processed source image. The operations take place only inside the source quadrangle srcQuad that is specified in the ippiWarpQuadCubicInit function.

To specify the algorithm for borders processing, set the borderType and pBorderValue parameters when initializing the IppiWarpSpec structure. The data type of pBorderValue is automatically converted from Ipp64f to the data type of the processed images. The function supports the following algorithms for borders processing:

- If the border type is equal to ippBorderRepl, the source image outer pixels are replicated from the edge pixels.
- If the border type is equal to ippBorderConst, the outer pixels are set to the constant value specified in pBorderValue.
- If the border type is equal to ippBorderTransp, destination image pixels mapped to the outer source image pixels are not changed. The outer pixels required for cubic interpolation are replicated from the edge pixels.

- If the border type is equal to `ippBorderInMem`, destination image pixels mapped to the outer source image pixels are not changed. The outer pixels required for cubic interpolation are obtained from the out of the source image origin space. Cubic interpolation requires additional one-pixel edge from each source image side.
- The mixed border types can be obtained by the bitwise operation `OR` between `ippBorderTransp` and the `ippBorderInMemTop`, `ippBorderInMemBottom`, `ippBorderInMemLeft`, `ippBorderInMemRight` values

Before using the `ippiWarpPerspectiveCubic` function, you need to initialize the `IppiWarpSpec` structure using the ippiWarpPerspectiveCubicInit function and compute the size of the external buffer `pBuffer` using the ippiWarpGetBufferSize function.

To compute the perspective transform parameters, use the ippiGetPerspectiveQuad, ippiGetPerspectiveBound, and ippiGetPerspectiveTransform functions.

For an example on how to use this functionality, refer to the example provided with the ippiWarpPerspectiveNearest function description.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when one of the specified pointers is `NULL`.
`ippStsNoOperation`	Indicates a warning when width or height of the destination image is equal to zero.
`ippStsContextMatchErr`	Indicates an error when context data is invalid.
`ippStsNotSupportedModeErr`	Indicates an error when the requested mode is not supported.
`ippStsSizeErr`	Indicates an error when width or height of the source or destination image ROI is negative.
`ippStsStepErr`	Indicates an error when the step value is not a multiple of data type.
`ippStsOutOfRangeErr`	Indicates an error when the destination image offset point is outside the destination image origin.
`ippStsSizeWrn`	Indicates a warning when the destination image ROI size is more than the destination image origin size.
`ippStsWrongIntersectQuad`	Indicates a warning that no operation is performed if the transformed source image extended with borders has no intersection with the destination image.

See Also

ROI Processing in Geometric Transforms

WarpPerspectiveCubicInit Initializes the specification structure for image perspective warping with the cubic interpolation method.

WarpQuadCubicInit Initializes the specification structure for warping an arbitrary quadrangle in the source image to the quadrangle in the destination image with the cubic interpolation method.

WarpGetBufferSize Computes the size of the work buffer for the warp transform.

GetPerspectiveBound Computes the bounding rectangle for the source ROI transformed by the ippiWarpPerspective function.

GetPerspectiveQuad Computes vertex coordinates of the quadrangle, to which the source ROI rectangle is mapped by the perspective transform.

GetPerspectiveTransform Computes the perspective transform coefficients to map the source ROI to the quadrangle with the specified vertex coordinates.

WarpPerspectiveNearest Performs warp perspective transformation of an image using the nearest neighbor interpolation method.

GetBilinearQuad

Computes the vertex coordinates of the quadrangle, to which the source rectangular ROI is mapped by the bilinear transform.

Syntax

```
IppStatus ippiGetBilinearQuad(IppiRect srcRoi, double quad[4][2], const double coeffs[2][4]);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`srcRoi`	Region of interest in the source image (of the `IppiRect` type).
`quad`	Output array. Contains vertex coordinates of the quadrangle, to which the source ROI is mapped by the bilinear transform function.
`coeffs`	The given bilinear transform coefficients.

Description

This function operates with ROI (see ROI Processing in Geometric Transforms).

This function is used as a support function for ippiWarpBilinear. It computes vertex coordinates of the quadrangle, to which the source rectangular ROI is mapped by the bilinear transform function ippiWarpBilinear using coefficients `coeffs`.

The first dimension [4] of the array `quad` [4][2] is equal to the number of vertices, and the second dimension [2] means `x` and `y` coordinates of the vertex. Quadrangle vertices have the following meaning:

`quad` [0] corresponds to the transformed top-left corner of the source ROI,

`quad` [1] corresponds to the transformed top-right corner of the source ROI,

`quad` [2] corresponds to the transformed bottom-right corner of the source ROI,

`quad` [3] corresponds to the transformed bottom-left corner of the source ROI.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsSizeErr	Indicates an error condition if srcRoi has a size field with zero or negative value.
ippStsCoeffErr	Indicates an error condition if coefficient values are invalid.

GetBilinearBound

Computes the bounding rectangle for the source ROI transformed by the ippiWarpBilinear function.

Syntax

```
IppStatus ippiGetBilinearBound(IppiRect srcRoi, double bound[2][2], const double coeffs[2][4]);
```

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

srcRoi	Region of interest in the source image (of the IppiRect type).
bound	Output array. Contains vertex coordinates of the bounding rectangle for the transformed source ROI.
coeffs	The given bilinear transform coefficients.

Description

This function operates with ROI (see ROI Processing in Geometric Transforms).

This functionis used as a support function for ippiWarpBilinear. It computes vertex coordinates of the smallest bounding rectangle for the quadrangle quad, to which the source ROI is mapped by the bilinear transform function ippiWarpBilinear using coefficients coeffs.

bound [0] specifies x, y coordinates of the top-left corner, bound [1] specifies x, y coordinates of the bottom-right corner.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.

ippStsSizeErr	Indicates an error condition if srcRoi has a size field with zero or negative value.
ippStsCoeffErr	Indicates an error condition if coefficient values are invalid.

GetBilinearTransform

Computes bilinear transform coefficients to map the source ROI to the quadrangle with the specified vertex coordinates.

Syntax

```
IppStatus ippiGetBilinearTransform(IppiRect srcRoi, const double quad[4][2], double coeffs[2][4]);
```

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

srcRoi	Region of interest in the source image (of the IppiRect type).
quad	Vertex coordinates of the quadrangle, to which the source ROI is mapped by the bilinear transform function.
coeffs	Output array. Contains the target bilinear transform coefficients.

Description

This function operates with ROI (see ROI Processing in Geometric Transforms).

This function is used as a support function for ippiWarpBilinear. It computes the coefficients coeffs of thebilinear transform that maps the source rectangular ROI to the quadrangle with the specified vertex coordinates quad.

The first dimension [4] of the array quad [4][2] is equal to the number of vertices, and the second dimension [2] means x and y coordinates of the vertex. Quadrangle vertices have the following meaning:

quad [0] corresponds to the transformed top-left corner of the source ROI,

quad [1] corresponds to the transformed top-right corner of the source ROI,

quad [2] corresponds to the transformed bottom-right corner of the source ROI,

quad [3] corresponds to the transformed bottom-left corner of the source ROI.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsSizeErr	Indicates an error condition if srcRoi has a size field with zero or negative value.
ippStsCoeffErr	Indicates an error condition if coefficient values are invalid.
ippStsRectErr	Indicates an error condition if width or height of the srcRoi is less than or equal to 1.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

// A simple example of performing bilinear warping of an image
// using Intel(R) Integrated Performance Primitives (Intel(R) IPP) functions:
//     ippiWarpBilinearGetBufferSize
//     ippiGetBilinearTransform
//     ippiWarpBilinear_32f_C1R
//     ippiWarpBilinearBack_32f_C1R

#include <stdio.h>
#include "ipp.h"

#define WIDTH  8 /* source image width */
#define HEIGHT 8 /* source image height */

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define check_sts(st)        \
    if ((st) != ippStsNoErr) \
        goto exitLine; /* Go to Exit if Intel(R) IPP function returned status different from
ippStsNoErr */

static Ipp32f pSrc[WIDTH * HEIGHT] = /* Pointers to source images */
    {1.f, 1.f, 1.f, 1.f, 1.f, 1.f, 1.f, 1.f, 1.f, 1.f, 1.f, 1.f, 5.f, 5.f, 1.f, 1.f, 1.f, 1.f, 1.f,
5.f, 5.f, 5.f, 5.f,
     1.f, 1.f, 1.f, 5.f, 5.f, 5.f, 5.f, 5.f, 5.f, 1.f, 1.f, 5.f, 5.f, 5.f, 5.f, 5.f, 5.f, 1.f,
1.f, 1.f, 1.f, 5.f, 5.f,
     5.f, 5.f, 1.f, 1.f, 1.f, 1.f, 1.f, 5.f, 5.f, 1.f, 1.f, 1.f, 1.f, 1.f, 1.f, 1.f, 1.f, 1.f,
```

```
1.f, 1.f};
static const double quad[4][2] = /* Vertex coordinates of the quadrangle */
    {{1.0, 1.0}, {8.0, 0.0}, {6.0, 8.0}, {2.0, 6.0}};

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main(void)
{
    IppStatus status = ippStsNoErr;
    IppiSize srcSize = {WIDTH, HEIGHT};      /* Size of source ROI in pixels */
    IppiRect srcRoi = {0, 0, WIDTH, HEIGHT}; /* Region of interest in the source image */
    int srcStep = WIDTH * sizeof(Ipp32f), dstStep = srcStep,
        dstbStep = srcStep;                          /* Steps, in bytes, through the source/
destination/back destination images */
    Ipp32f pDst[WIDTH * HEIGHT], pDstB[WIDTH * HEIGHT]; /* Pointers to destination/back
destination images */
    IppiSize dstSize = {WIDTH, HEIGHT};
    IppiRect dstbRoi = {0, 0, WIDTH, HEIGHT}; /* Region of interest in the back destination
image */
    IppiRect dstRoi = {0, 0, WIDTH, HEIGHT};  /* Region of interest in the destination image */
    double coeffs[2][4] = {0};
    int bufSize = 0, bufSizeB = 0; /* Work buffers size */
    Ipp8u *pBuffer = NULL, *pBufferB = NULL;

    check_sts(status = ippiSet_32f_C1R(2, pDst, srcStep, srcSize));

    check_sts(status = ippiSet_32f_C1R(3, pDstB, srcStep, srcSize));

    /* Computes bilinear transform matrix */
    check_sts(status = ippiGetBilinearTransform(srcRoi, quad, (double(*)[4])coeffs));

    /* Computes the size of external buffer for bilinear transform */
    check_sts(status = ippiWarpBilinearGetBufferSize(srcSize, srcRoi, dstRoi, ippWarpForward,
coeffs, IPPI_INTER_NN, &bufSize));

    /* Computes the size of external buffer for Back bilinear transform */
    check_sts(status = ippiWarpBilinearGetBufferSize(dstSize, dstRoi, dstbRoi, ippWarpBackward,
coeffs, IPPI_INTER_NN, &bufSizeB));

    pBuffer = ippsMalloc_8u(bufSize);
    pBufferB = ippsMalloc_8u(bufSizeB);

    /* Performs bilinear warping of an image */
    check_sts(status = ippiWarpBilinear_32f_C1R(pSrc, srcSize, srcStep, srcRoi, pDst, dstStep,
dstRoi, coeffs, IPPI_INTER_NN, pBuffer));

    /* Performs an inverse bilinear warping of an image */
    check_sts(status = ippiWarpBilinearBack_32f_C1R(pDst, dstSize, dstStep, dstRoi, pDstB,
dstbStep, dstbRoi, coeffs, IPPI_INTER_NN, pBuffer));

    EXIT_MAIN
    ippsFree(pBuffer);
    ippsFree(pBufferB);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

WarpBilinearGetBufferSize

Computes the size of the work buffer for bilinear warping.

Syntax

```
IppStatus ippiWarpBilinearGetBufferSize(IppiSize srcSize, IppiRect srcRoi, IppiRect dstRoi, IppiWarpDirection direction, const double coeffs[2][4], int interpolation, int* pBufSize);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`srcSize`	Size of the source image, in pixels.
`srcRoi`	Source image ROI (of the IppiRect type).
`dstRoi`	Destination image ROI (of the IppiRect type).
`direction`	Transformation direction. Supported values: • `ippWarpForward` • Forward transformation • `ippWarpBackward` • Backward transformation
`coeffs`	Coefficients for the perspective transform.
`interpolation`	Interpolation mode. Supported values: • `IPPI_INTER_NN` • Nearest neighbor interpolation • `IPPI_INTER_LINEAR` • Linear interpolation • `IPPI_INTER_CUBIC` • Cubic interpolation • `IPPI_INTER_EDGE` • Use edge smoothing in addition to one of the above
`pBufSize`	Pointer to the size, in bytes, of the external buffer.

Description

This function computes the size of the external work buffer required for bilinear warping of the source image ROI. The result is stored in the `pBufSize` parameter.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pBufSize is NULL.
ippStsSizeErr	Indicates an error when one of the image dimensions is less than, or equal to zero.
ippStsWarpDirectionErr	Indicates an error when direction has an illegal value.
ippStsCoeffErr	Indicates an error when bilinear transformation is singular.
ippStsInterpolationErr	Indicates an error when interpolation has an illegal value.
ippStsWrongIntersectQuad	Indicates a warning that no operation is performed if the transformed source image extended with borders has no intersection with the destination image.

See Also

WarpBilinear MODIFIED API. Performs bilinear warping of the source image using the specified transform coefficients.

WarpBilinear

MODIFIED API. Performs bilinear warping of the source image using the specified transform coefficients.

Syntax

```
IppStatus ippiWarpBilinear_<mod>(const Ipp<datatype>* pSrc, IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* pDst, int dstStep, IppiRect dstRoi, const double coeffs[2][4], int interpolation, Ipp8u* pBuffer);
```

Supported values for mod:

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R
8u_C4R	16u_C4R	32f_C4R

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc	Pointer to the source image origin.An array of separate pointers to each plane in case of data in planar format.
srcSize	Size in pixels of the source image.
srcStep	Distance in bytes between starts of consecutive lines in the source image buffer.
srcRoi	Region of interest in the source image (of the IppiRect type).
pDst	Pointer to the destination image origin. An array of separate pointers to each plane in case of data in planar format.
dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.
dstRoi	Region of interest in the destination image (of the IppiRect type).
coeffs	The bilinear transform coefficients.
interpolation	Specifies the interpolation mode. Use one of the following values: • IPPI_INTER_NN • Nearest neighbor interpolation • IPPI_INTER_LINEAR • Linear interpolation • IPPI_INTER_CUBIC • Cubic interpolation • IPPI_SMOOTH_EDGE • Use edge smoothing in addition to one of the above modes.
pBuffer	Pointer to the external work buffer.

Description

NOTE
Important
The API of this function has been modified in Intel IPP 9.0 release.

This function operates with ROI (see ROI Processing in Geometric Transforms).

This bilinear warp function transforms the source image pixel coordinates (x, y) according to the following formulas:

$x' = c_{00} * x*y + c_{01} * x + c_{02} * y + c_{03}$

$y' = c_{10} * x*y + c_{11} * x + c_{12} * y + c_{13}$

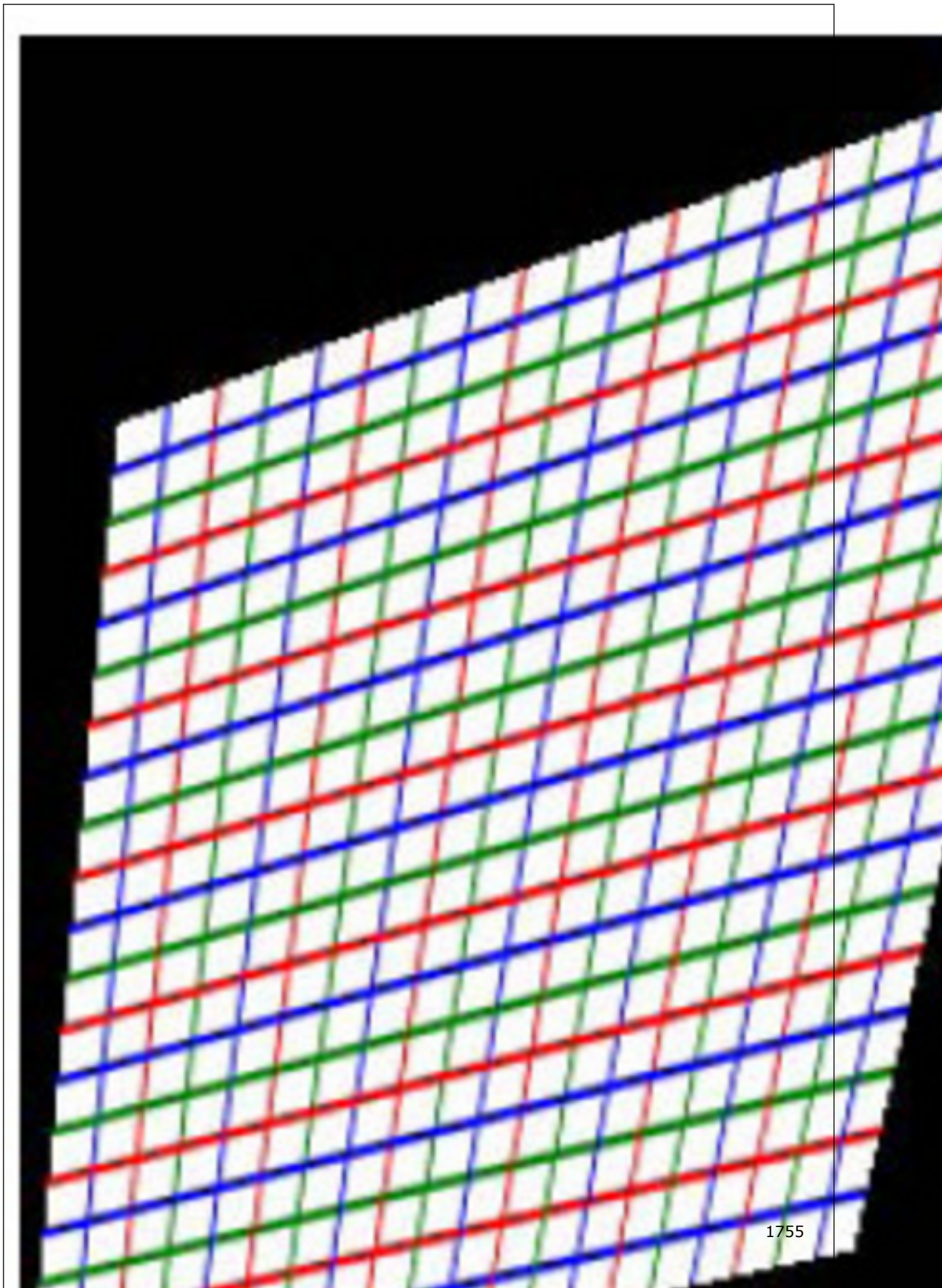
where x′ and y′ denote the pixel coordinates in the transformed image, and c_{ij} are the bilinear transform coefficients passed in the array coeffs.

The bilinear transform preserves equal distances between points on a line.

The transformed part of the source image is resampled using the interpolation mode specified by the `interpolation` parameter, and written to the destination image ROI.

Figure Bilinear Transform of an Image gives an example of applying the bilinear warping function `ippiWarpBilinear` to a sample image.

Bilinear Transform of an Image



To estimate how the source image ROI will be transformed by the `ippiWarpBilinear` function, use functions ippiWarpBilinearQuad and ippiGetBilinearBound. To calculate coefficients of the bilinear transform which maps source ROI to a given quadrangle, use ippiGetBilinearTransform function .

Before using this function, compute the size of the external work buffer `pBuffer` using the WarpBilinearGetBufferSize function.

Example shows how to use the `ippiWarpBilinear_32f_C1R` function.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if any image dimension has zero or negative value.
`ippStsStepErr`	Indicates an error condition if `srcStep` or `dstStep` has a zero or negative value.
`ippStsInterpolationErr`	Indicates an error condition if `interpolation` has an illegal value.
`ippStsRectErr`	Indicates an error condition if width or height of the intersection of the `srcRoi` and source image is less than or equal to 1.
`ippStsCoeffErr`	Indicates an error condition if coefficient values are invalid.
`ippStsWrongIntersectROIErr`	Indicates an error condition if `srcRoi` has no intersection with the source image.
`ippStsWrongIntersectQuad`	Indicates a warning that no operation is performed if the transformed source ROI has no intersection with the destination ROI.

See Also

Regions of Interest in Intel IPP

WarpBilinearQuad MODIFIED API. Performs bilinear warping of the source image that transforms the given source quadrangle to the specified destination quadrangle.

GetBilinearBound Computes the bounding rectangle for the source ROI transformed by the ippiWarpBilinear function.

GetBilinearTransform Computes bilinear transform coefficients to map the source ROI to the quadrangle with the specified vertex coordinates.

WarpBilinearGetBufferSize Computes the size of the work buffer for bilinear warping.

WarpBilinearBack

MODIFIED API. Performs an inverse bilinear warping of the source image.

Syntax

```
IppStatus ippiWarpBilinearBack_<mod>(const Ipp<datatype>* pSrc, IppiSize srcSize, int srcStep, IppiRect srcRoi, Ipp<datatype>* pDst, int dstStep, IppiRect dstRoi, const double coeffs[2][4], int interpolation, Ipp8u* pBuffer);
```

Supported values for `mod`:

`8u_C1R`	`16u_C1R`	`32f_C1R`
`8u_C3R`	`16u_C3R`	`32f_C3R`
`8u_C4R`	`16u_C4R`	`32f_C4R`

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image origin.An array of separate pointers to each plane in case of data in planar format.
`srcSize`	Size in pixels of the source image.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image buffer.
`srcRoi`	Region of interest in the source image (of the `IppiRect` type).
`pDst`	Pointer to the destination image origin. An array of separate pointers to each plane in case of data in planar format.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image buffer.
`dstRoi`	Region of interest in the destination image (of the `IppiRect` type).
`coeffs`	The bilinear transform coefficients.
`interpolation`	Specifies the interpolation mode. Use one of the following values: • `IPPI_INTER_NN` • Nearest neighbor interpolation • `IPPI_INTER_LINEAR` • Linear interpolation • `IPPI_INTER_CUBIC` • Cubic interpolation.

pBuffer	Pointer to the external work buffer.

Description

NOTE
Important
The API of this function has been modified in Intel IPP 9.0 release.

This function operates with ROI (see ROI Processing in Geometric Transforms).

This function performs the inverse transform to that defined by ippiWarpBilinear function. Pixel coordinates x' and y' in the transformed image are obtained from the following equations

$c_{00} * x' * y' + c_{01} * x' + c_{02} * y' + c_{03} = x$

$c_{10} * x' * y' + c_{11} * x' + c_{12} * y' + c_{13} = y$

where x and y denote the pixel coordinates in the source image, and coefficients c_{ij} are given in the array coeffs. Thus, you do not need to invert transform coefficients in your application program before calling ippiWarpBilinearBack.

Note that inverse transform functions handle source and destination ROI in a different way than other geometric transform functions. The implementation of the inverse transform functions has the following logic:

- Backward transform is applied to coordinates of each pixel in the destination ROI. The result is coordinates of some pixel in the source image.
- If the obtained source pixel is inside the source ROI, the corresponding pixel in the destination ROI is modified accordingly; otherwise, no changes are made.

Before using this function, compute the size of the external work buffer pBuffer using the WarpBilinearGetBufferSize function.

Example shows how to use the function ippiWarpBilinearBack_32f_C1R.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if any image dimension has zero or negative value.
ippStsStepErr	Indicates an error condition if srcStep or dstStep has a zero or negative value.
ippStsInterpolationErr	Indicates an error condition if interpolation has an illegal value.
ippStsCoeffErr	Indicates an error condition if coefficient values are invalid.
ippStsWrongIntersectROIErr	Indicates an error condition if srcRoi has no intersection with the source image.

ippStsWrongIntersectQuad	Indicates a warning that no operation is performed if the transformed source ROI has no intersection with the destination ROI.

See Also

Regions of Interest in Intel IPP

WarpBilinear MODIFIED API. Performs bilinear warping of the source image using the specified transform coefficients.

WarpBilinearGetBufferSize Computes the size of the work buffer for bilinear warping.

WarpBilinearQuadGetBufferSize

Computes the size of the work buffer for bilinear warping of an arbitrary quadrangle in the source image ROI to the quadrangle in the destination image.

Syntax

```
IppStatus ippiWarpBilinearQuadGetBufferSize(IppiSize srcSize, IppiRect srcRoi, const double srcQuad[4][2], IppiRect dstRoi, const double dstQuad[4][2], int interpolation, int* pBufSize);
```

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

srcSize	Size of the source image, in pixels.
srcRoi	Source image ROI (of the IppiRect type).
srcQuad	Quadrangle in the source image.
dstRoi	Destination image ROI (of the IppiRect type).
dstQuad	Quadrangle in the destination image.
interpolation	Interpolation mode. Supported values: • IPPI_INTER_NN • Nearest neighbor interpolation • IPPI_INTER_LINEAR • Linear interpolation • IPPI_INTER_CUBIC • Cubic interpolation • IPPI_INTER_EDGE • Use edge smoothing in addition to one of the above

pBufSize	Pointer to the size, in bytes, of the external buffer.

Description

This function computes the size of the external work buffer required for bilinear warping of an arbitrary quadrangle in the source image ROI to the quadrangle in the destination image. The result is stored in the pBufSize parameter.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pBufSize is NULL.
ippStsSizeErr	Indicates an error: • If one of the ROI coordinates has a negative value. • If one of the ROI dimensions is less than, or equal to zero.
ippStsCoeffErr	Indicates an error when bilinear transformation is singular.
ippStsInterpolationErr	Indicates an error when interpolation has an illegal value.

See Also

WarpBilinearQuad MODIFIED API. Performs bilinear warping of the source image that transforms the given source quadrangle to the specified destination quadrangle.

WarpBilinearQuad

MODIFIED API. Performs bilinear warping of the source image that transforms the given source quadrangle to the specified destination quadrangle.

Syntax

```
IppStatus ippiWarpBilinearQuad_<mod>(const Ipp<datatype>* pSrc, IppiSize srcSize, int srcStep, IppiRect srcRoi, const double srcQuad[4][2], Ipp<datatype>* pDst, int dstStep, IppiRect dstRoi, const double dstQuad[4][2], int interpolation, Ipp8u* pBuffer);
```

Supported values for mod:

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R
8u_C4R	16u_C4R	32f_C4R

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc	Pointer to the source image origin. An array of separate pointers to each plane in case of data in planar format.
srcSize	Size in pixels of the source image.
srcStep	Distance in bytes between starts of consecutive lines in the source image buffer.
srcRoi	Region of interest in the source image (of the IppiRect type).
srcQuad	A given quadrangle in the source image.
pDst	Pointer to the destination image origin. An array of separate pointers to each plane in case of data in planar format.
dstStep	Distance in bytes between starts of consecutive lines in the destination image buffer.
dstRoi	Region of interest in the destination image (of the IppiRect type).
dstQuad	A given quadrangle in the destination image.
interpolation	Specifies the interpolation mode. Use one of the following values: • IPPI_INTER_NN • Nearest neighbor interpolation • IPPI_INTER_LINEAR • Linear interpolation • IPPI_INTER_CUBIC • Cubic interpolation • IPPI_SMOOTH_EDGE • Use edge smoothing in addition to one of the above modes.
pBuffer	Pointer to the external work buffer.

Description

NOTE
Important
The API of this function has been modified in Intel IPP 9.0 release.

This function operates with ROI (see ROI Processing in Geometric Transforms).

This functionapplies a bilinear transform to an arbitrary quadrangle srcQuad in the source image pSrc. Theoperations take place only in the intersection of the source image ROI srcRoi and the source quadrangle srcQuad. The function ippiWarpBilinearQuad uses the same formulas for pixel mapping as in the case of

the ippiWarpBilinear function. Transform coefficients are computed internally, based on the mapping of the source quadrangle to the quadrangle `dstQuad` specified in the destination image `pDst`. The `dstQuad` should have a non-empty intersection with the destination image ROI `dstRoi`.

The first dimension [4] of the array specifying the quadrangle `srcQuad` [4][2] or `dstQuad` [4][2] is equal to the number of vertices, and the second dimension [2] holds `x` and `y` coordinates of the vertex.

Edge smoothing interpolation is applicable only if the source quadrangle lies in the source image ROI.

Before using this function, compute the size of the external work buffer `pBuffer` using the WarpBilinearGetBufferSize function.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if any image dimension has zero or negative value.
`ippStsStepErr`	Indicates an error condition if `srcStep` or `dstStep` has a zero or negative value.
`ippStsInterpolationErr`	Indicates an error condition if `interpolation` has an illegal value.
`ippStsQuadErr`	Indicates an error condition if `srcQuad` or `dstQuad` degenerates into triangle.
`ippStsWrongIntersectROIErr`	Indicates an error condition if `srcRoi` has no intersection with the source image.
`ippStsWrongIntersectQuad`	Indicates a warning that no operation is performed if the `srcRoi` has no intersection with the `srcQuad`, or `dstRoi` has no intersection with the `dstQuad`.

See Also

Regions of Interest in Intel IPP

WarpBilinear MODIFIED API. Performs bilinear warping of the source image using the specified transform coefficients.

WarpBilinearGetBufferSize Computes the size of the work buffer for bilinear warping.

Mirror

Mirrors an image about the specified axis (axes).

Syntax

Case 1: Not-in-place operation

```
IppStatus ippiMirror_<mod>(const Ipp<datatype>* pSrc, int srcStep, Ipp<datatype>* pDst,
int dstStep, IppiSize roiSize, IppiAxis flip);
```

Supported values for `mod`:

8u_C1R	16u_C1R	16s_C1R	32s_C1R	32f_C1R
8u_C3R	16u_C3R	16s_C3R	32s_C3R	32f_C3R
8u_C4R	16u_C4R	16s_C4R	32s_C4R	32f_C4R
8u_AC4R	16u_AC4R	16s_AC4R	32s_AC4R	32f_AC4R

Case 2: In-place operation

```
IppStatus ippiMirror_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize roiSize,
IppiAxis flip);
```

Supported values for `mod`:

8u_C1IR	16u_C1IR	16s_C1IR	32s_C1IR	32f_C1IR
8u_C3IR	16u_C3IR	16s_C3IR	32s_C3IR	32f_C3IR
8u_C4IR	16u_C4IR	16s_C4IR	32s_C4IR	32f_C4IR
8u_AC4IR	16u_AC4IR	16s_AC4IR	32s_AC4IR	32f_AC4IR

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source buffer.
`srcStep`	Distance, in bytes, between the starting points of consecutive lines in the source image buffer.
`pDst`	Pointer to the destination buffer.
`dstStep`	Distance, in bytes, between the starting points of consecutive lines in the destination image buffer.
`pSrcDst`	Pointer to the source and destination buffer for the in-place operation.
`srcDstStep`	Distance, in bytes, between the starting points of consecutive lines in the source and destination image buffer for the in-place operation.
`roiSize`	Size of the destination ROI in pixels.
`flip`	Specifies the axis to mirror the image about. Use the following values to specify the axes: • `ippAxsHorizontal` • for the horizontal axis.

	• ippAxsVertical • for the vertical axis. • ippAxsBoth • for both horizontal and vertical axes. • ippAxs45 • for the 45-degree rotated axis. • ippAxs135 • for the 135-degree rotated axis.

Description

The ippiMirror function operates with ROI (see Regions of Interest in Intel IPP). This function mirrors the source image pSrc about the axis (axes) specified by the value of the flip parameter and writes the result to the destination image pDst. Each function flavor can mirror an image about the horizontal or vertical axis or both.

The ippiMirror_8u_C1R, ippiMirror_16u_C1R, ippiMirror_16s_C1R, and ippiMirror_32f_C1R function flavors can also use the ippAxs45 or ippAxs135 value of the flip parameter to mirror the source image about an axis rotated counterclockwise by 45 degrees or 135 degrees, respectively. For mirroring with each of these values of the flip parameter, the sizes of the source and destination ROI are different, and

roiSize.height = srcRoiSize.width

roiSize.width = srcRoiSize.height

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value, or when one of the dimensions is equal to 1.
ippStsMirrorFlipErr	Indicates an error condition if flip has an illegal value.
ippStsNotSupportedModeErr	Indicates an error condition if intersection of the source and destination ROI is detected.
ippStsStepErr	Indicates an error condition if srcStep or dstStep has a zero or negative value or is not a multiple of the image data size (4 for floating-point images or 2 for short-integer images)

Examples

Mirror1 bout horizontal and vertical axes:

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
```

```
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *****************************************************************************/

// A simple example of mirroring an image about
// a horizontal and vertical axis using Intel(R) Integrated Performance Primitives (Intel(R)
IPP) functions:
//     ippiMirror_8u_C1R

#include <stdio.h>
#include "ipp.h"

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define check_sts(st)        \
    if ((st) != ippStsNoErr) \
        goto exitLine; /* Go to Exit if Intel(R) IPP function returned status different from
ippStsNoErr */

int main(void)
{
    IppStatus status = ippStsNoErr;
    Ipp8u pSrc[8 * 4] = {/* Pointer to source images */
                         1, 2, 3, 4, 8, 8, 8, 8, 1, 2, 3, 4, 8, 8, 8, 8, 1, 2, 3, 4, 8, 8, 8, 8,
1, 2, 3, 4, 8, 8, 8, 8};
    Ipp8u pDst[4 * 4];            /* Pointer to destination images */
    int srcStep = 8, dstStep = 4; /* Steps, in bytes, through the source/destination images */
    IppiSize roiSize = {4, 4};    /* Size of source/destination ROI in pixels */
    IppiAxis flip = ippAxsBoth;   /* Specifies the axis to mirror the image about */

    check_sts(status = ippiMirror_8u_C1R(pSrc, srcStep, pDst, dstStep, roiSize, flip));

    EXIT_MAIN
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

Mirror about diagonal axes:

```
/*****************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
```

```
warranties, other than
 * those that are expressly stated in the License.
 *****************************************************************************/

// A simple example of mirroring an image about
// the ippAxs45 and ippAxs135 axis using Intel(R) Integrated Performance Primitives (Intel(R)
IPP) functions:
//     ippiMirror_8u_C1R

#include <stdio.h>
#include "ipp.h"

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define check_sts(st)        \
    if ((st) != ippStsNoErr) \
        goto exitLine; /* Go to Exit if Intel(R) IPP function returned status different from
ippStsNoErr */

int main(void)
{
    IppStatus status = ippStsNoErr;
    Ipp8u pSrc[8 * 4] = {/* Pointer to source images */
                         1, 2, 3, 4, 8, 8, 8, 8, 1, 2, 3, 4, 8, 8, 8, 8, 1, 2, 3, 4, 8, 8, 8, 8,
1, 2, 3, 4, 8, 8, 8, 8};
    Ipp8u pDst[8 * 8] = {/* Pointer to destination images */
                         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
                         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0};
    int srcStep = 8, dstStep = 8; /* Steps, in bytes, through the source/destination images */
    IppiSize roiSize = {3, 4};    /* Size of source/destination ROI in pixels */
    IppiAxis flip1 = ippAxs45;    /* The axis to mirror the image about */
    IppiAxis flip2 = ippAxs135;   /* The axis to mirror the image about */

    check_sts(status = ippiMirror_8u_C1R(pSrc, srcStep, pDst, dstStep, roiSize, flip1));
    check_sts(status = ippiMirror_8u_C1R(pSrc, srcStep, pDst, dstStep, roiSize, flip2));

    EXIT_MAIN
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

Remap

Performs the look-up coordinate mapping of pixels of the source image.

Syntax

Case 1: Operation on pixel-order data

```
IppStatus ippiRemap_<mod>(const Ipp<datatype>* pSrc, IppiSize srcSize, int srcStep,
IppiRect srcRoi, const Ipp32f* pxMap, int xMapStep, const Ipp32f* pyMap, int yMapStep,
Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, int interpolation);
```

Supported values for `mod`:

8u_C1R	16u_C1R	16s_C1R	32f_C1R
8u_C3R	16u_C3R	16s_C3R	32f_C3R
8u_C4R	16u_C4R	16s_C4R	32f_C4R
	16u_AC4R	16s_AC4R	32f_AC4R

```
IppStatus ippiRemap_<mod>(const Ipp<datatype>* pSrc, IppiSize srcSize, int srcStep,
IppiRect srcRoi, const Ipp64f* pxMap, int xMapStep, const Ipp64f* pyMap, int yMapStep,
Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, int interpolation);
```

Supported values for `mod`:

64f_C1R
64f_C3R
64f_C4R
64f_AC4R

```
IppStatus ippiRemap_8u_AC4R(const Ipp<datatype>* pSrc, IppiSize srcSize, int srcStep,
IppiRect srcRoi, const Ipp32f* pxMap, int xMapStep, const Ipp32f* pyMap, int yMapStep,
Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, int interpolation);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image origin. An array of separate pointers to each plane in case of data in planar format.
`srcSize`	Size, in pixels, of the source image.
`srcStep`	Distance, in bytes, between the starting points of consecutive lines in the source image buffer.
`srcRoi`	Region of interest in the source image (of the `IppiRect` type).
`pxMap, pyMap`	Pointers to the starts of 2D buffers, containing tables of the `x - and y - coordinates`.
`xMapStep, yMapStep`	Steps, in bytes, through the buffers containing tables of the `x - and y - coordinates`.
`pDst`	Pointer to the destination image ROI. An array of separate pointers to ROI in each plane for planar image.

dstStep	Distance, in bytes, between the starting points of consecutive lines in the destination image buffer.
dstRoiSize	Size of the destination ROI in pixels.
interpolation	Specifies the interpolation mode. Possible values are: • IPPI_INTER_NN - nearest neighbor interpolation • IPPI_INTER_LINEAR - linear interpolation • IPPI_INTER_CUBIC - cubic interpolation • IPPI_INTER_LANCZOS - interpolation with Lanczos window • IPPI_INTER_CUBIC2P_CATMULLROM – Catmull - Rom spline the following flag is used additionally to the above modes: • IPPI_SMOOTH_EDGE - edge smoothing

Description

This function operates with ROI (see ROI Processing in Geometric Transforms).

This function transforms the source image by remapping its pixels. Pixel remapping is performed using pxMap and pyMap buffers to look-up the coordinates of the source image pixel that is written to the target destination image pixel. The application has to supply these look-up tables. The remapping of the source pixels to the destination pixels is made according to the following formula:

dst_pixel[i, j]=src_pixel[pxMap[i, j],pyMap[i, j]]

where i,j are the x – and y – coordinates of the target destination image pixel dst_pixel;

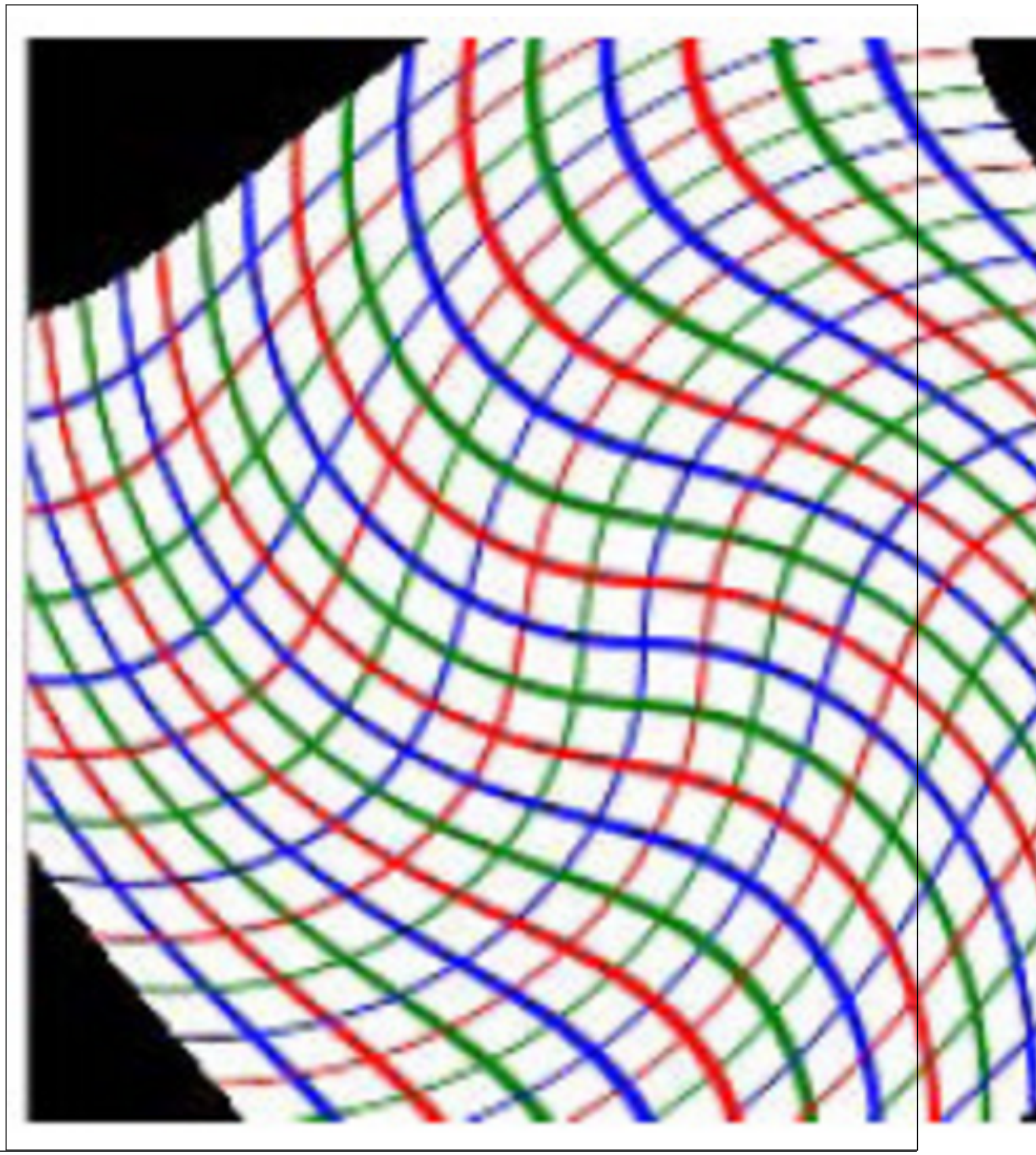
pxMap[i, j] contains the x – coordinates of the source image pixels src_pixel that are written to dst_pixel;

pyMap[i, j] contains the y – coordinates of the source image pixels src_pixel that are written to dst_pixel.

If the referenced coordinates correspond to a pixel outside of the source ROI, and the flag IPPI_SMOOTH_EDGE is not set, then no mapping of the source pixel is performed.

Figure Remapping the Sample Image gives an example of applying the function `ippiRemap` to a sample image that is a square grid of alternating blue, red, and green lines.

Remapping the Sample Image



The transformed part of the image is resampled using the interpolation method specified by the `interpolation` parameter, and is written to the destination image ROI. The function can be used with or without edge smoothing. The pseudo code below shows how it works.

The function works without edge smoothing - the flag `IPPI_SMOOTH_EDGE` is not set:

```
if (xMap < srcRoi.x || xMap > srcRoi.x + srcRoi.width - 1 || yMap < srcRoi.y || yMap > srcRoi.y
+ srcRoi.height - 1)
  not fill dst   /* do not remap */
else
  fill dst with Interpolate(Src, xMap, yMap)   /* remap */
```

The function works with edge smoothing - the flag `IPPI_SMOOTH_EDGE` is set:

```
if (xMap < srcRoi.x - 1 || xMap > srcRoi.x+srcRoi.width || yMap < srcRoi.y - 1 || yMap > srcRoi.y
+srcRoi.height)
  not fill dst   /* do not remap */
else if (xMap < srcRoi.x || xMap > srcRoi.x+srcRoi.width-1 || yMap < srcRoi.y || yMap > srcRoi.y
+srcRoi.height-1)
  fill dst with Interpolate(Src, fillvalue, xMap, yMap)    /* smoothing */
else
  fill dst with Interpolate(Src, xMap, yMap)   /* remap */
```

If the width or height of the input image `pSrc` is insufficient for the interpolation method specified by the parameter `interpolation`, the function uses interpolation with a smaller number of pixels for both width and height dimensions. See the following pseudocode:

```
if (width < 6 || height < 6) {
    inter== lanczos ? inter = cubic;
    if ( (width < 4 || height < 4) && ( inter == cubic || inter == catmullrom) )
          inter = linear;
    if (width == 1 || height == 1)
          inter = nearest;
}
```

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `srcSize` or `dstRoiSize` has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if one of the `srcStep`, `dstStep`, `xMapStep`, or `yMapStep` has a zero or negative value.
`ippStsInterpolationErr`	Indicates an error condition if `interpolation` has an illegal value.
`ippStsWrongIntersectROIErr`	Indicates an error condition if `srcRoi` has no intersection with the source image.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
```

```
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *****************************************************************************/

// A simple example of the source image transformation by remapping
// of its pixels using Intel(R) Integrated Performance Primitives (Intel(R) IPP) function:
//     ippiRemap_8u_C1R

#include <stdio.h>
#include "ipp.h"

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define check_sts(st)        \
    if ((st) != ippStsNoErr) \
        goto exitLine; /* Go to Exit if Intel(R) IPP function returned status different from
ippStsNoErr */

int main(void)
{
    IppStatus status = ippStsNoErr;
    Ipp8u pSrc[4 * 4] = /* Pointers to source images */
        {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4, 5};
    Ipp8u pDst[4 * 4];            /* Pointers to destination images */
    int srcStep = 4, dstStep = 4; /* Steps, in bytes, through the source/destination images */
    IppiSize roiSize = {4, 4};    /* Size of source/destination ROI in pixels */
    Ipp32f pxMap[4 * 4] = {       /* Pointer to x map */
                           0, 1, 2, 3, 0, 1, 2, 0, 0, 1, 0, 0, 0, 1, 2, 3};
    Ipp32f pyMap[4 * 4] = {/* Pointer to y map */
                           0, 0, 0, 0, 1, 1, 1, 0, 2, 2, 0, 0, 3, 3, 3, 3};
    int xStep = 16, yStep = 16;   /* Steps, in bytes, through the x and y maps */
    IppiRect rect = {0, 0, 4, 4}; /* Region of interest in the source image */

    check_sts(status = ippiRemap_8u_C1R(pSrc, roiSize, srcStep, rect, pxMap, xStep, pyMap,
yStep, pDst, dstStep, roiSize, IPPI_INTER_LINEAR));

    EXIT_MAIN
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

Miscellaneous Image Transforms

This section describes the Intel® IPP image processing functions that perform adaptive thresholding of the image ROI.

- ThresholdAdaptiveBoxGetBufferSize
- ThresholdAdaptiveBox
- ThresholdAdaptiveGaussGetBufferSize

- ThresholdAdaptiveGaussInit
- ThresholdAdaptiveGauss

ThresholdAdaptiveBoxGetBufferSize

Computes the size of the work buffer for adaptive thresholding with the Box method.

Syntax

```
IppStatus ippiThresholdAdaptiveBoxGetBufferSize(IppiSize roiSize, IppiSize maskSize,
IppDataType dataType, int numChannels, int* pBufferSize);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`roiSize`	Size of the destination image ROI, in pixels.
`maskSize`	Size of the kernel that is used to calculate a threshold level. Width and height of `maskSize` must be equal and odd.
`dataType`	Data type of the source and destination images. Possible value is `ipp8u`.
`numChannels`	Number of channels in the images. Possible value is 1.
`pBufferSize`	Pointer to the size, in bytes, of the external work buffer.

Description

This function computes the size, in bytes, of the external work buffer needed for the ThresholdAdaptiveBox function. The result is stored in `pBufferSize`.

For an example on how to use this function, refer to the example provided with the ThresholdAdaptiveBox function description.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error when `pBufferSize` is `NULL`.
`ippStsSizeErr`	Indicates an error when `roiSize` has a field with a zero or negative value.
`ippStsMaskSizeErr`	Indicates an error when `maskSize` has a field with a zero or negative value, or fields of `maskSize` are not equal.
`ippStsDataTypeErr`	Indicates an error when `dataType` has an illegal value.
`ippStsNumChannelsErr`	Indicates an error when `numChannels` has an illegal value.

See Also

ThresholdAdaptiveBox Performs adaptive thresholding with the Box method.

ThresholdAdaptiveBox

Performs adaptive thresholding with the Box method.

Syntax

```
IppStatus ippiThresholdAdaptiveBox_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, IppiSize maskSize, Ipp32f delta, Ipp8u valGT, Ipp8u valLE, IppiBorderType borderType, Ipp8u borderValue, Ipp8u* pBuffer);
```

```
IppStatus ippiThresholdAdaptiveBox_8u_C1IR(Ipp8u* pSrcDst, int srcDstStep, IppiSize roiSize, IppiSize maskSize, Ipp32f delta, Ipp8u valGT, Ipp8u valLE, IppiBorderType borderType, Ipp8u borderValue, Ipp8u* pBuffer);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance, in bytes, between the starting points of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance, in bytes, between the starting points of consecutive lines in the destination image.
`pSrcDst`	Pointer to the source and destination image ROI (for the in-place function).
`srcDstStep`	Distance, in bytes, between the starting points of consecutive lines in the source and destination image (for the in-place function).
`roiSize`	Size of the destination image ROI, in pixels.
`maskSize`	Size of the kernel that is used to calculate a threshold level. Width and height of `maskSize` must be equal and odd.
`delta`	Value for threshold calculation.
`valGT`	Output pixel if the source pixel value is more than threshold.
`valLE`	Output pixel if the source pixel value is less than, or equal to threshold.
`borderType`	Type of border. Possible values are:

`ippBorderConst`	Values of all border pixels are set to a constant.

ippBorderRepl	Border is replicated from the edge pixels.
ippBorderInMem	Border is obtained from the source image pixels in memory.

Mixed borders are also supported. They can be obtained by the bitwise operation OR between ippBorderRepl or ippBorderConst and the following flags:

- ippBorderInMemTop
- ippBorderInMemBottom
- ippBorderInMemLeft
- ippBorderInMemRight

Each of these flags means that for the corresponding border the outside pixels of the source image are in memory.

borderValue — Constant value(s) to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.

pBuffer — Pointer to the work buffer. To calculate the size of the temporary buffer, use the ThresholdAdaptiveBoxGetBufferSize function.

Description

This function performs adaptive thresholding of the source image ROI using the Box method. Output pixels are calculated according to the following formulas:

pDst(x,y) = valGT, if pSrc(x,y) > T(x,y)

pDst(x,y) = valLE, if pSrc(x,y) <= T(x,y)

where

T(x,y) is a mean of the maskSize.width * maskSize.height neighborhood of a (x, y) pixel minus delta.

Before using this function, compute the size of the external work buffer using the ThresholdAdaptiveBoxGetBufferSize function.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pSrc, pDst, pSrcDst, or pBuffer is NULL.
ippStsSizeErr	Indicates an error when roiSize has a field with a zero or negative value.
ippStsMaskSizeErr	Indicates an error when maskSize has a field with a zero or negative value, or fields of maskSize are not equal.
ippStsBorderErr	Indicates an error when borderType has an illegal value.

Example

The code example below demonstrates how to use the `ippiThresholdAdaptiveBox_8u_C1R` and ThresholdAdaptiveBoxGetBufferSize functions.

```
IppStatus threshold_adaptive_box_8u_c1( void ) {
    Ipp8u pSrc[8*8] =
    {
          0, 255,   1, 254,   2, 253,   3, 252,
        251,   4, 250,   5, 249,   6, 248,   7,
          8, 247,   9, 246,  10, 245,  11, 244,
        243,  12 , 242, 13, 241,  14, 240,  15,
         16, 239,  17, 238,  18, 237,  19, 236,
        235,  20, 234,  21, 233,  22, 232,  23,
         24, 231,  25, 230,  26, 229,  26, 228,
        227,  27, 226,  28, 225,  29, 224,  30
    };
    Ipp8u    pDst[8*8];
    IppiSize roiSize = {8, 8};
    IppiSize maskSize = {3, 3};
    IppiBorderType borderType = ippBorderConst;
    int     srcStep = 8 * sizeof(Ipp8u);
    int     dstStep = 8 * sizeof(Ipp8u);
    int     bufferSize;
    IppStatus status;
    Ipp32f  delta = 0.5f;
    Ipp8u   valGT = 254;
    Ipp8u   valLE = 1;
    Ipp8u   *pBuffer;


    ippiThresholdAdaptiveBoxGetBufferSize(roiSize, maskSize, ipp8u, 1, &bufferSize);
    pBuffer = ippsMalloc_8u(bufferSize);
    ippiThresholdAdaptiveBox_8u_C1R(pSrc, srcStep, pDst, dstStep, roiSize, maskSize, delta,
valGT, valLE, borderType, 33, pBuffer);
    ippsFree(pBuffer);
    return status;
}
```

`pDst` after function execution:

```
  1 254   1 254   1 254   1 254
254   1 254   1 254   1 254   1
  1 254   1 254   1 254   1 254
254   1 254   1 254   1 254   1
  1 254   1 254   1 254   1 254
254   1 254   1 254   1 254   1
  1 254   1 254   1 254   1 254
254   1 254   1 254   1 254   1
```

See Also

ThresholdAdaptiveBoxGetBufferSize Computes the size of the work buffer for adaptive thresholding with the Box method.

ThresholdAdaptiveGaussGetBufferSize

Computes the size of the adaptive threshold specification structure and the size of the work buffer for adaptive thresholding with the Gaussian method.

Syntax

```
IppStatus ippiThresholdAdaptiveGaussGetBufferSize(IppiSize roiSize, IppiSize maskSize,
IppDataType dataType, int numChannels, int* pSpecSize, int* pBufferSize);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`roiSize`	Size of the destination image ROI, in pixels.
`maskSize`	Size of the kernel that is used to calculate a threshold level. Width and height of `maskSize` must be equal and odd.
`dataType`	Data type of the source and destination images. Possible value is `ipp8u`.
`numChannels`	Number of channels in the images. Possible value is 1.
`pSpecSize`	Pointer to the size of the adaptive threshold specification structure.
`pBufferSize`	Pointer to the size (in bytes) of the external work buffer.

Description

This function computes the size, in bytes, of the adaptive threshold specification structure and the size of the external work buffer needed for the ThresholdAdaptiveGauss function. The results are stored in `pSpecSize` and `pBufferSize`.

For an example on how to use this function, refer to the example provided with the ThresholdAdaptiveGauss function description.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error when `pSpecSize` or `pBufferSize` is `NULL`.
`ippStsSizeErr`	Indicates an error when `roiSize` has a field with a zero or negative value.
`ippStsMaskSizeErr`	Indicates an error when `maskSize` has a field with a zero or negative value, or fields of `maskSize` are not equal.
`ippStsDataTypeErr`	Indicates an error when `dataType` has an illegal value.
`ippStsNumChannelsErr`	Indicates an error when `numChannels` has an illegal value.

See Also

ThresholdAdaptiveGauss Performs adaptive thresholding with the Gaussian method.

ThresholdAdaptiveGaussInit

Initializes the threshold adaptive specification structure for adaptive thresholding with the Gaussian method.

Syntax

```
IppStatus ippiThresholdAdaptiveGaussInit(IppiSize roiSize, IppiSize maskSize,
IppDataType dataType, int numChannels, Ipp32f sigma, IppiThresholdAdaptiveSpec* pSpec);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`roiSize`	Size of the destination image ROI, in pixels.
`maskSize`	Size of the kernel that is used to calculate a threshold value. Width and height of `maskSize` must be equal and odd.
`dataType`	Data type of the source and destination images. Possible value is `ipp8u`.
`numChannels`	Number of channels in the images. Possible value is 1.
`sigma`	Value of sigma that is used to calculate a threshold value for the Gaussian method. If sigma value is less than, or equal to zero, `sigma` is set automatically in compliance with the kernel size.
`pSpec`	Pointer to the adaptive threshold specification structure.

Description

This function initializes the adaptive threshold specification structure `pSpec` for adaptive thresholding with the Gaussian method. Before using this function, compute the size of the specification structure using the ThresholdAdaptiveGaussGetBufferSize function.

If sigma is less than, or equal to zero, it is set according to the following formula:

```
sigma=0.3 * ((maskSize.width - 1) * 0.5 - 1) + 0.8
```

For an example on how to use this function, refer to the example provided with the ThresholdAdaptiveGauss function description.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error when `pSpecSize` is `NULL`.
`ippStsSizeErr`	Indicates an error when `dstRoiSize` has a field with a zero or negative value.
`ippStsMaskSizeErr`	Indicates an error when `maskSize` has a field with a zero or negative value, or fields of `maskSize` are not equal.
`ippStsDataTypeErr`	Indicates an error when `dataType` has an illegal value.

ippStsNumChannelsErr	Indicates an error when numChannels has an illegal value.

See Also

ThresholdAdaptiveGaussGetBufferSize Computes the size of the adaptive threshold specification structure and the size of the work buffer for adaptive thresholding with the Gaussian method.

ThresholdAdaptiveGauss Performs adaptive thresholding with the Gaussian method.

ThresholdAdaptiveGauss

Performs adaptive thresholding with the Gaussian method.

Syntax

```
IppStatus ippiThresholdAdaptiveGauss_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u*
pDst, int dstStep, IppiSize roiSize, Ipp32f delta, Ipp8u valGT, Ipp8u valLE,
IppiBorderType borderType, Ipp8u borderValue, IppiThresholdAdaptiveSpec* pSpec, Ipp8u*
pBuffer);
```

```
IppStatus ippiThresholdAdaptiveGauss_8u_C1IR(Ipp8u* pSrcDst, int srcDstStep, IppiSize
roiSize, Ipp32f delta, Ipp8u valGT, Ipp8u valLE, IppiBorderType borderType, Ipp8u
borderValue, IppiThresholdAdaptiveSpec* pSpec, Ipp8u* pBuffer);
```

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance, in bytes, between the starting points of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance, in bytes, between the starting points of consecutive lines in the destination image.
pSrcDst	Pointer to the source and destination image ROI (for the in-place function).
srcDstStep	Distance, in bytes, between the starting points of consecutive lines in the source and destination image (for the in-place function).
roiSize	Size of the destination image ROI, in pixels.
delta	Value for threshold calculation.
valGT	Output pixel if the source pixel value is more than threshold.
valLE	Output pixel if the source pixel value is less than, or equal to threshold.

`borderType` Type of border. Possible values are:

`ippBorderConst`	Values of all border pixels are set to a constant.
`ippBorderRepl`	Border is replicated from the edge pixels.
`ippBorderInMem`	Border is obtained from the source image pixels in memory.

Mixed borders are also supported. They can be obtained by the bitwise operation OR between `ippBorderRepl` or `ippBorderConst` and the following flags:

- `ippBorderInMemTop`
- `ippBorderInMemBottom`
- `ippBorderInMemLeft`
- `ippBorderInMemRight`

Each of these flags means that for the corresponding border the outside pixels of the source image are in memory.

`borderValue` Constant value(s) to assign to pixels of the constant border. This parameter is applicable only to the `ippBorderConst` border type.

`pSpec` Pointer to the adaptive threshold specification structure.

`pBuffer` Pointer to the work buffer. To calculate the size of the temporary buffer, use the ThresholdAdaptiveGaussGetBufferSize function.

Description

This function performs adaptive thresholding of the source image ROI using the Gaussian method. Output pixels are calculated according to the following formulas:

`pDst(x,y) = valGT` if `pSrc(x,y) > T(x,y)`

`pDst(x,y) = valLE` if `pSrc(x,y) <= T(x,y)`

where

`T(x,y)` is a weighted sum (cross-correlation with a Gaussian window) of the `maskSize.width * maskSize.height` neighborhood of a (`x`, `y`) pixel minus `delta`.

The function uses a separable Gaussian filter. Filter coefficients are computed according to the following formula:

$G_i = A * \exp(- (i-(maskSize.width - 1)/2)^2)/(0.5 * sigma^2)$

where

`A` is a scale factor for $\sum_i G_i = 1$ (`i = 0`, …, `maskSize.width - 1`)

Before using this function, compute the size of the external work buffer and specification structure using the ThresholdAdaptiveGaussGetBufferSize function, and initialize the structure using the ThresholdAdaptiveGaussInit function.

Return Values

`ippStsNoErr`	Indicates no error.

ippStsNullPtrErr	Indicates an error when pSrc, pDst, pSrcDst, pSpec, or pBuffer is NULL.
ippStsSizeErr	Indicates an error when roiSize has a field with a zero or negative value.
ippStsContextMatchErr	Indicates an error when pSpec does not match.
ippStsBorderErr	Indicates an error when borderType has an illegal value.

Example

The code example below demonstrates how to use the ippiThresholdAdaptiveGauss_8u_C1R, ThresholdAdaptiveGaussGetBufferSize, and ThresholdAdaptiveGaussInit functions.

```
IppStatus threshold_adaptive_gauss_8u_c1( void ) {
    Ipp8u pSrc[8*8] =
    {
          0, 255,   1, 254,   2, 253,   3, 252,
        251,   4, 250,   5, 249,   6, 248,   7,
          8, 247,   9, 246,  10, 245,  11, 244,
        243,  12 , 242, 13, 241,  14, 240,  15,
         16, 239,  17, 238,  18, 237,  19, 236,
        235,  20, 234,  21, 233,  22, 232,  23,
         24, 231,  25, 230,  26, 229,  26, 228,
        227,  27, 226,  28, 225,  29, 224,  30
    };
    Ipp8u    pDst[8*8];
    IppiSize roiSize = {8, 8};
    IppiSize maskSize = {3, 3};
    IppiBorderType borderType = ippBorderConst;
    int     srcStep = 8 * sizeof(Ipp8u);
    int     dstStep = 8 * sizeof(Ipp8u);
    int     bufferSize;
    int     specSize;
    IppStatus status;
    Ipp32f  sigma = 10.0f;
    Ipp32f  delta = 0.5f;
    Ipp8u   valGT = 254;
    Ipp8u   valLE = 1;
    IppiThresholdAdaptiveSpec *pSpec;
    Ipp8u   *pBuffer;


    ippiThresholdAdaptiveGaussGetBufferSize(roiSize, maskSize, ipp8u, 1, &specSize, &bufferSize);
    pSpec = (IppiThresholdAdaptiveSpec *)ippsMalloc_8u(specSize);
    pBuffer = ippsMalloc_8u(bufferSize);
    ippiThresholdAdaptiveGaussInit(roiSize, maskSize, ipp8u, 1, sigma, pSpec);
    ippiThresholdAdaptiveGauss_8u_C1R(pSrc, srcStep, pDst, dstStep, roiSize, delta, valGT,
valLE, borderType, 33, pSpec, pBuffer);
    ippsFree(pBuffer);
    ippsFree(pSpec);


    return status;
}
```

`pDst` after function execution:

```
  1 254   1 254   1 254   1 254
254   1 254   1 254   1 254   1
  1 254   1 254   1 254   1 254
254   1 254   1 254   1 254   1
  1 254   1 254   1 254   1 254
254   1 254   1 254   1 254   1
  1 254   1 254   1 254   1 254
254   1 254   1 254   1 254   1
```

See Also

ThresholdAdaptiveGaussGetBufferSize Computes the size of the adaptive threshold specification structure and the size of the work buffer for adaptive thresholding with the Gaussian method.

ThresholdAdaptiveGaussInit Initializes the threshold adaptive specification structure for adaptive thresholding with the Gaussian method.

Wavelet Transforms

Intel® IPP implements image processing functions that perform two-dimensional discrete wavelet transform (DWT).

In many applications the multiresolution analysis by discrete wavelet transforms is a better alternative to windowing and discrete Fourier analysis techniques. On the one hand, the forward two-dimensional wavelet transform may be considered as a decomposition of an image on the base of functions bounded or localized in space; and on the other, the wavelet transforms are related to subband filtering and resampling.

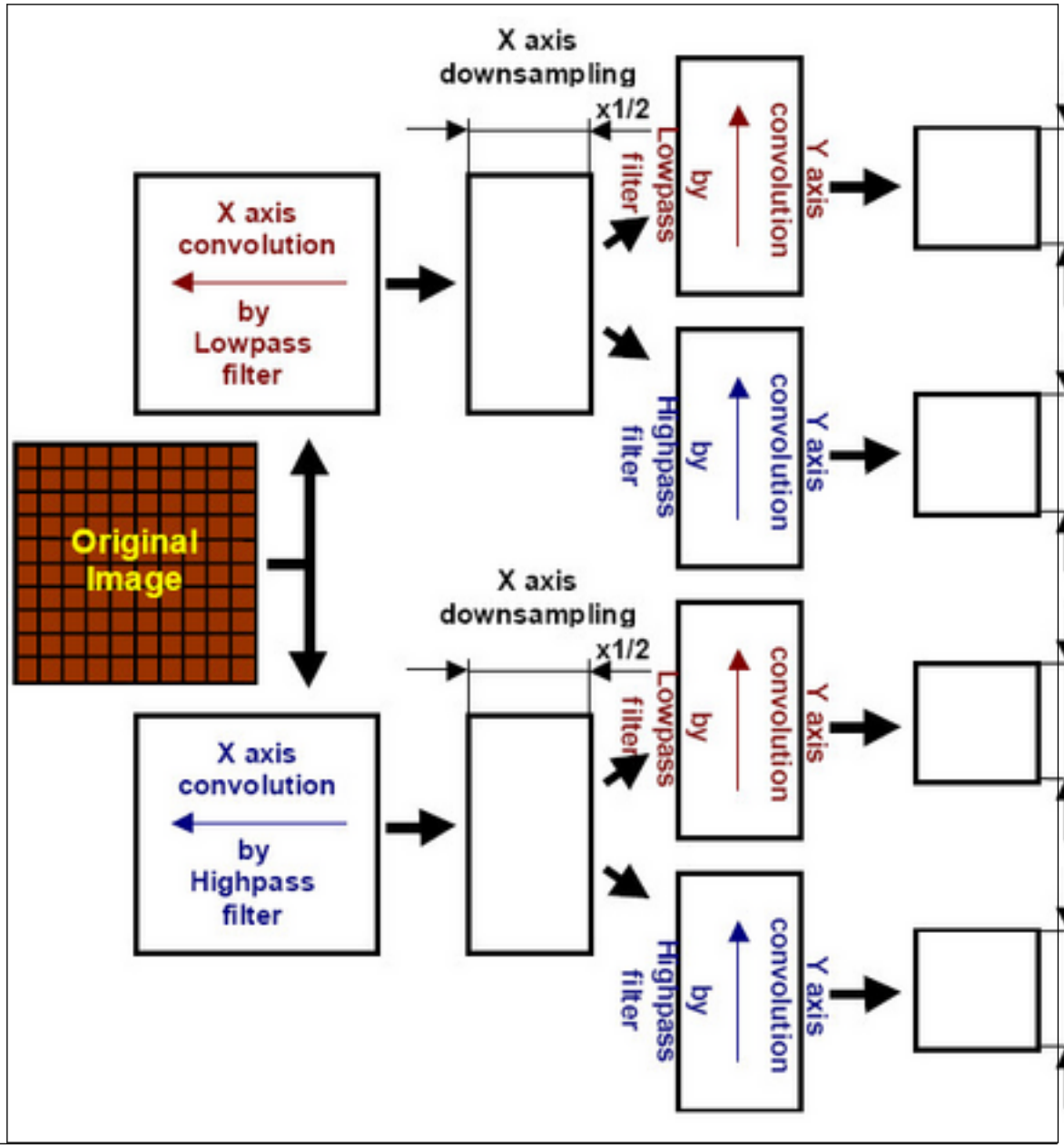
Intel IPP for image processing contains one-level discrete wavelet decomposition and reconstruction functions. It also provides the necessary interface for initialization and deallocation of the transform context structure.

The wavelet transform type can be set by specifying the appropriate filter taps in the initialization function.

Note that Intel IPP supports only one-dimensional finite impulse response filters for separable convolution.

The Intel IPP functions for wavelet decomposition and reconstruction use fast polyphase algorithm, which is equivalent to traditional application of separable convolution and dyadic resampling in different order. Figure Equivalent Scheme of Wavelet Decomposition Algorithm shows the equivalent algorithm of wavelet-based image decomposition:

Equivalent Scheme of Wavelet Decomposition Algorithm



Decomposition operation applied to a source image produces four output images of equal size: approximation image, horizontal detail image, vertical detail image, and diagonal detail image.

These decomposition components have the following meaning:

- The 'approximation' image is obtained by vertical and horizontal lowpass filtering.
- The 'horizontal detail' image is obtained by vertical highpass and horizontal lowpass filtering.
- The 'vertical detail' image is obtained by vertical lowpass and horizontal highpass filtering.
- The 'diagonal detail' image is obtained by vertical and horizontal highpass filtering.

The above image names are used in this document for identification convenience only.

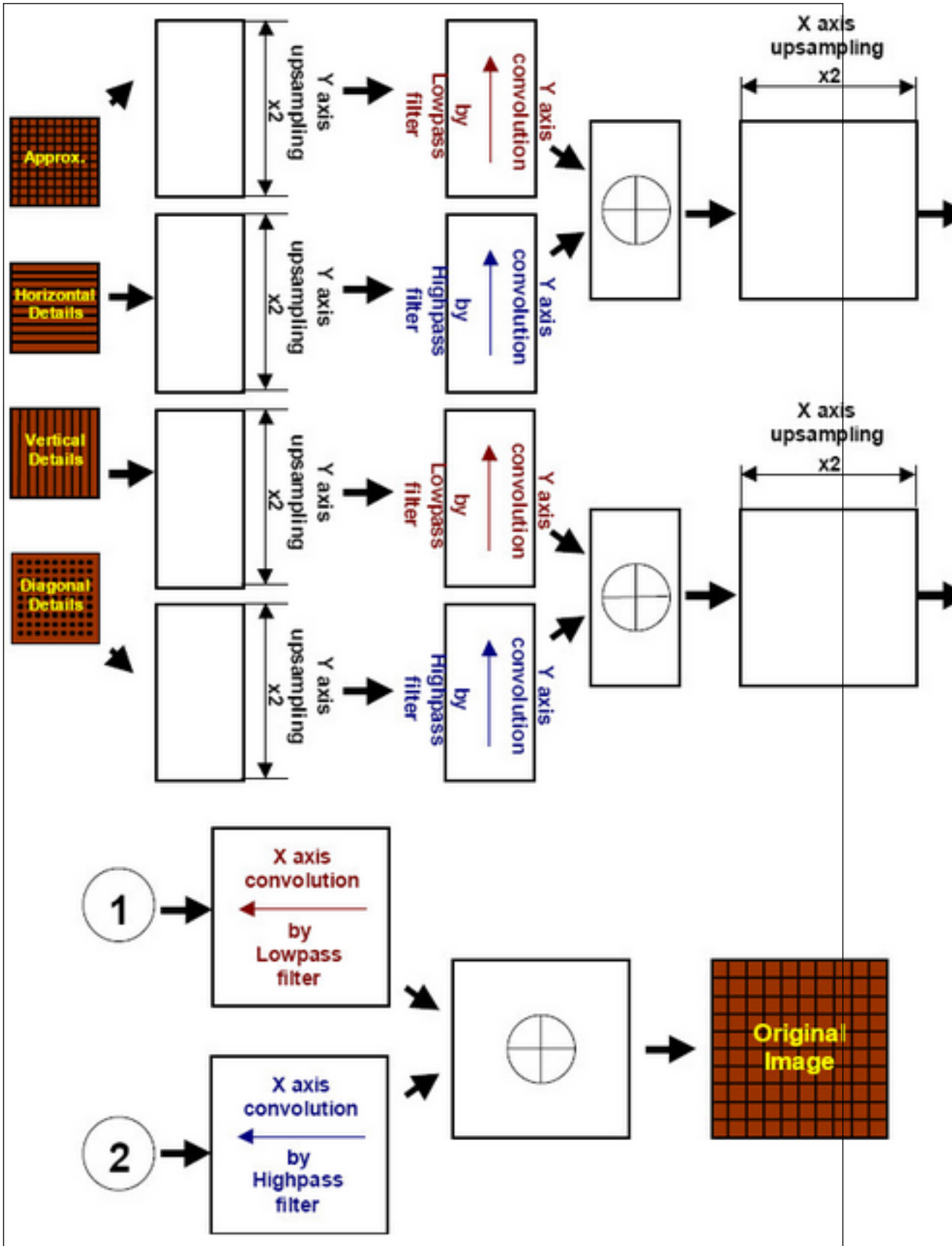
The wavelet-based image reconstruction can be represented by a sequence of separate convolution and dyadic upsampling.

The reconstruction function uses four input images that are the same as those resulting from the decomposition operation.

Figure Equivalent Scheme of Wavelet Reconstruction Algorithm shows the equivalent algorithm of wavelet reconstruction of an image.

Wavelet transform functions support regions of interest (ROI, see Regions of Interest in Intel IPP) in the images. However, these functions do not perform internally any border extensions of image ROI data. It means that source images must already contain all border data that are necessary for convolution operations. See descriptions of the functions `ippiWTFwd` and `ippiWTInv` for detailed information on how to calculate extended image border sizes.

Equivalent Scheme of Wavelet Reconstruction Algorithm



- WTFwdGetSize
- WTFwdInit
- WTFwd
- WTFwdGetSize
- WTInvInit
- WTInv

WTFwdGetSize

Calculates the size of the specification structure and work buffer for a forward wavelet transform.

Syntax

```
IppStatus ippiWTFwdGetSize_32f(int numChannels, int lenLow, int anchorLow, int lenHigh,
int anchorHigh, int* pSpecSize, int* pBufSize);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`numChannels`	Number of channels in the image. Possible values are 1 or 3.
`lenLow`	Length of the lowpass filter.
`anchorLow`	Anchor position of the lowpass filter.
`lenHigh`	Length of the highpass filter.
`anchorHigh`	Anchor position of the highpass filter.
`pSpecSize`	Pointer to the computed size of the `ippiWTFwd` specification structure, in bytes.
`pBufSize`	Pointer to the computed size of the work buffer, in bytes.

Description

This function computes the size, in bytes, of the specification structure and work buffer required for the forward wavelet transform function ippiWTFwd.

For an example on how to use this function, refer to the example provided with the ippiWTInv function description.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error when any of the specified pointers is `NULL`.

ippStsNumChannlesErr	Indicates an error when numChannels has an illegal value.
ippStsNumChannlesErr	Indicates an error when lenLow or lenHigh is less than 2.
ippStsAnchorErr	Indicates an error when anchorLow or anchorHigh is less than zero.

See Also

WTFwd Performs one-level wavelet decomposition of an image.

WTInv Performs one-level wavelet reconstruction of an image.

WTFwdInit

Initializes the forward wavelet transform context structure.

Syntax

```
IppStatus ippiWTFwdInit_32f_C1R(IppiWTFwdSpec_32f_C1R* pSpec, const Ipp32f* pTapsLow,
int lenLow, int anchorLow, const Ipp32f* pTapsHigh, int lenHigh, int anchorHigh);
```

```
IppStatus ippiWTFwdInit_32f_C3R(IppiWTFwdSpec_32f_C3R* pSpec, const Ipp32f* pTapsLow,
int lenLow, int anchorLow, const Ipp32f* pTapsHigh, int lenHigh, int anchorHigh);
```

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSpec	Double pointer to the forward wavelet transform specification structure.
pTapsLow	Pointer to lowpass filter taps.
lenLow	Length of the lowpass filter.
anchorLow	Anchor position of the lowpass filter.
pTapsHigh	Pointer to highpass filter taps.
lenHigh	Length of the highpass filter.
anchorHigh	Anchor position of the highpass filter.

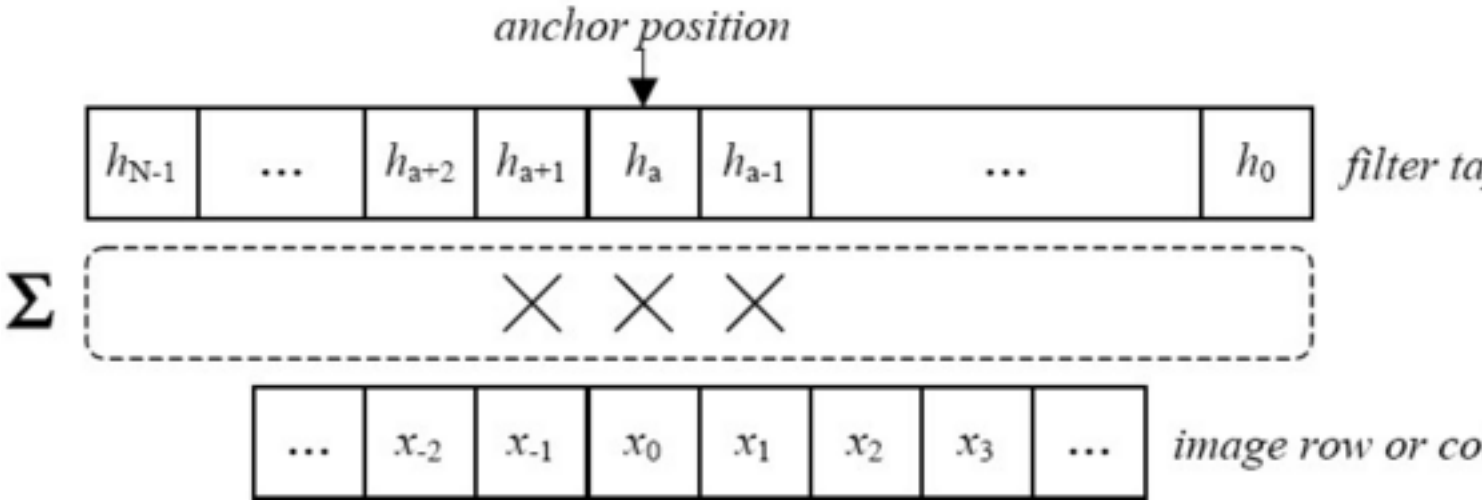
Description

This function initializes the specification structure pSpec for a one-level wavelet decomposition.

The forward wavelet transform specification structure contains parameters of a wavelet filter bank used for image decomposition. The filter bank consists of two analysis filters and includes the lowpass decomposition filter (or *coarse* filter) and the highpass decomposition filter (or *detail* filter).

The parameters `pTapsLow` and `pTapsHigh` specify coefficients, and `anchorLow` and `anchorHigh – anchor` positions for two synthesis filters. The anchor value sets the initial leftmost filter position relative to image row or column as shown in the figure below:

Anchor Value and Initial Filter Position for Wavelet Decomposition



Here `a` stands for anchor value, `N` is filter length, x `0` is the starting pixel of the processed row or column, and x `–1`, x `–2` ,… are the additional border pixels that are needed for calculations. The anchor value and filter length completely determine right, left, top, and bottom border sizes for the source image used in decomposition. The corresponding C-language expressions to calculate border sizes are given in the description of ippiWTFwd function.

For an example on how to use this function, refer to the example provided with the ippiWTInv function description.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error when any of the specified pointers is `NULL`.
`ippStsNumChannelsErr`	Indicates an error when `numChannels` has an illegal value.
`ippStsSizeErr`	Indicates an error when filter length `lenLow` or `lenHigh` is less than 2.
`ippStsAnchorErr`	Indicates an error when anchor position `anchorLow` or `anchorHigh` is less than zero.

See Also

WTFwdGetSize Calculates the size of the specification structure and work buffer for a forward wavelet transform.

WTFwd Performs one-level wavelet decomposition of an image.

WTInv Performs one-level wavelet reconstruction of an image.

WTFwd

Performs one-level wavelet decomposition of an image.

Syntax

```
IppStatus ippiWTFwd_32f_C1R (const Ipp32f* pSrc, int srcStep, Ipp32f* pApproxDst, int approxStep, Ipp32f* pDetailXDst, int detailXStep, Ipp32f* pDetailYDst, int detailYStep, Ipp32f* pDetailXYDst, int detailXYStep, IppiSize dstRoiSize, const IppiWTFwdSpec_32f_C1R* pSpec, Ipp8u* pBuffer);
```

```
IppStatus ippiWTFwd_32f_C3R (const Ipp32f* pSrc, int srcStep, Ipp32f* pApproxDst, int approxStep, Ipp32f* pDetailXDst, int detailXStep, Ipp32f* pDetailYDst, int detailYStep, Ipp32f* pDetailXYDst, int detailXYStep, IppiSize dstRoiSize, const IppiWTFwdSpec_32f_C3R* pSpec, Ipp8u* pBuffer);
```

Include Files

`ippi.h`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance, in bytes, between the starting points of consecutive lines in the source image buffer.
`pApproxDst`	Pointer to ROI of the destination approximation image.
`approxStep`	Distance, in bytes, between the starting points of consecutive lines in the approximation image buffer.
`pDetailXDst`	Pointer to ROI of the destination horizontal detail image.
`detailXStep`	Distance, in bytes, between the starting points of consecutive lines in the horizontal detail image buffer.
`pDetailYDst`	Pointer to ROI of the destination vertical detail image.
`detailYStep`	Distance, in bytes, between the starting points of consecutive lines in the vertical detail image buffer.
`pDetailXYDst`	Pointer to ROI of the destination diagonal detail image.
`detailXYStep`	Distance, in bytes, between the starting points of consecutive lines in the diagonal detail image buffer.
`dstRoiSize`	Size of the ROI in pixels for all destination images.
`pSpec`	Pointer to the allocated and initialized forward DWT specification structure.
`pBuffer`	Pointer to the allocated buffer for intermediate operations

Description

This function operates with ROI (see Regions of Interest in Intel IPP). This function performs one-level wavelet decomposition of a source image pointed to by `pSrc` into four destination subimages pointed to by `pApproxDst`, `pDetailXDst`, `pDetailYDst`, and `pDetailXYDst`. See Figure Equivalent Scheme of Wavelet Decomposition Algorithm for the equivalent algorithm of `ippiWTFwd` function operation.

Wavelet parameters are contained in the forward transform specification structure `pSpec`. Before using this function, compute the size of the structure and work buffer using the WTFwdGetSize function and initialize the structure using WTFwdInit.

The pointer `pSrc` points to memory location of the source image rectangular ROI of size `srcWidth` by `srcHeight` which is uniquely determined by the size of destination ROI as:

srcWidth = 2 * `dstRoiSize`.width

srcHeight = 2 * `dstRoiSize`.height

The source image ROI size always has even dimensions, as it is computed from the `dstRoiSize` parameter as follows:

srcRoiSize.width = 2 * `dstRoiSize`.width

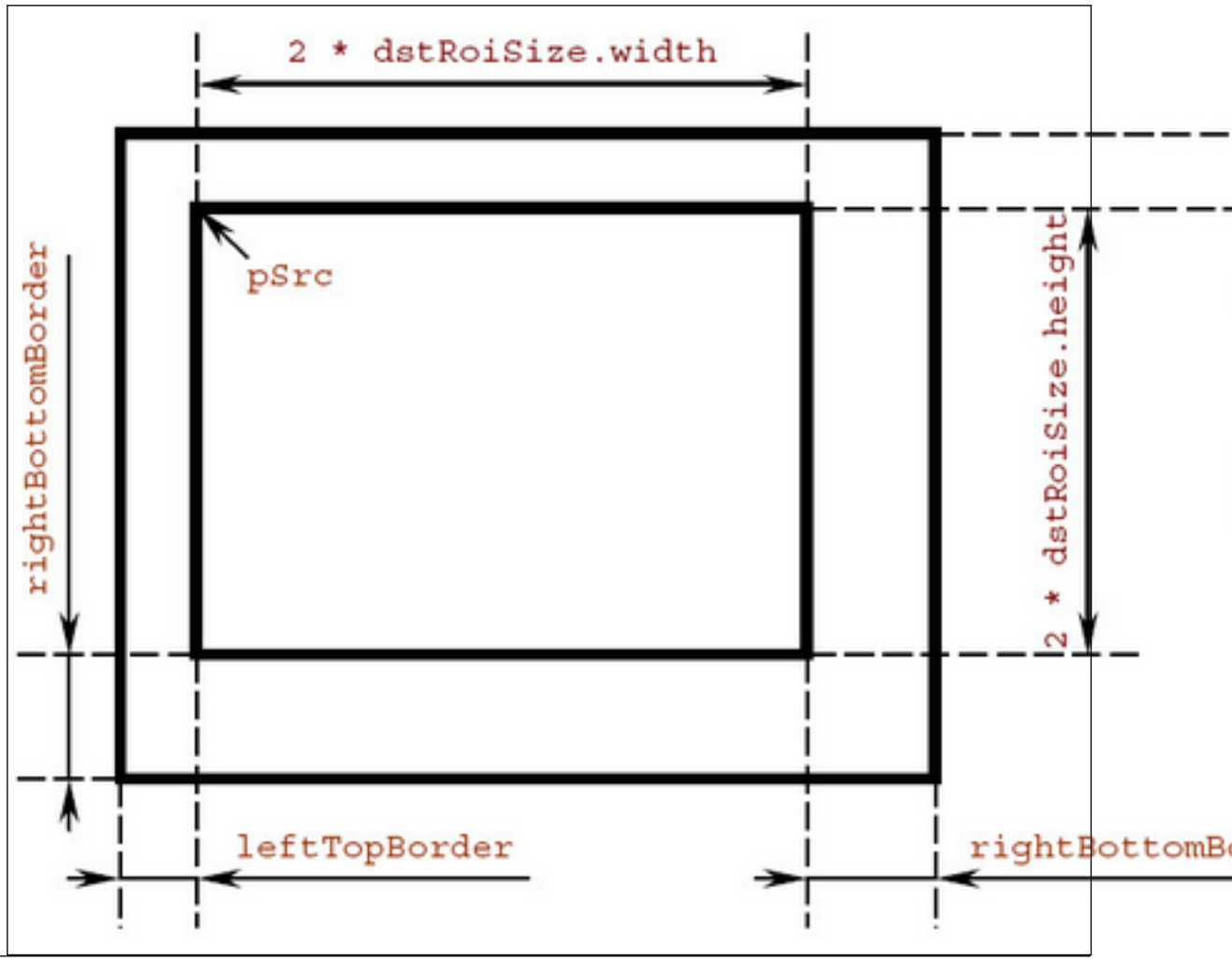
srcRoiSize.height = 2 * `dstRoiSize`.height

To use this function for images with uneven width or height, you should truncate the last column/row or extend an image to even dimensions.

NOTE The source image ROI does not include border pixels necessary to compute some destination pixels. It means that prior to using `ippiWTFwd` function the application program must apply some border extension model (symmetrical, wraparound or another) to the source image ROI through filling of neighboring memory locations. As a result, the size of memory block allocated for the source image must be extended to accommodate for added border pixels outside ROI formal boundaries.

Figure Extended Source Image for Wavelet Decomposition schematically shows the source image ROI and extended image area.

Extended Source Image for Wavelet Decomposition



Use the following C-language expressions to calculate extended image border sizes:

```
int leftBorderLow   = lenLow   - 1 - anchorLow;
int leftBorderHigh  = lenHigh  - 1 - anchorHigh;
int rightBorderLow  = anchorLow;
int rightBorderHigh = anchorHigh;
int leftTopBorder     = IPP_MAX(leftBorderLow, leftBorderHigh);
int rightBottomBorder = IPP_MAX(rightBorderLow, rightBorderHigh);
```

See the description of the function WTFwdInit for the explanation of the parameters.

Note that the left and top borders have equal size. The same holds for the right and bottom borders which have equal size too.

The size of the source image area extended by border pixels can be defined as

```
srcWidthWithBorders = srcWidth  + leftTopBorder + rightBottomBorder;
srcHeightWithBorders = srcHeight + leftTopBorder + rightBottomBorder;
```

All destination images have equal size specified by the parameter `dstRoiSize`.

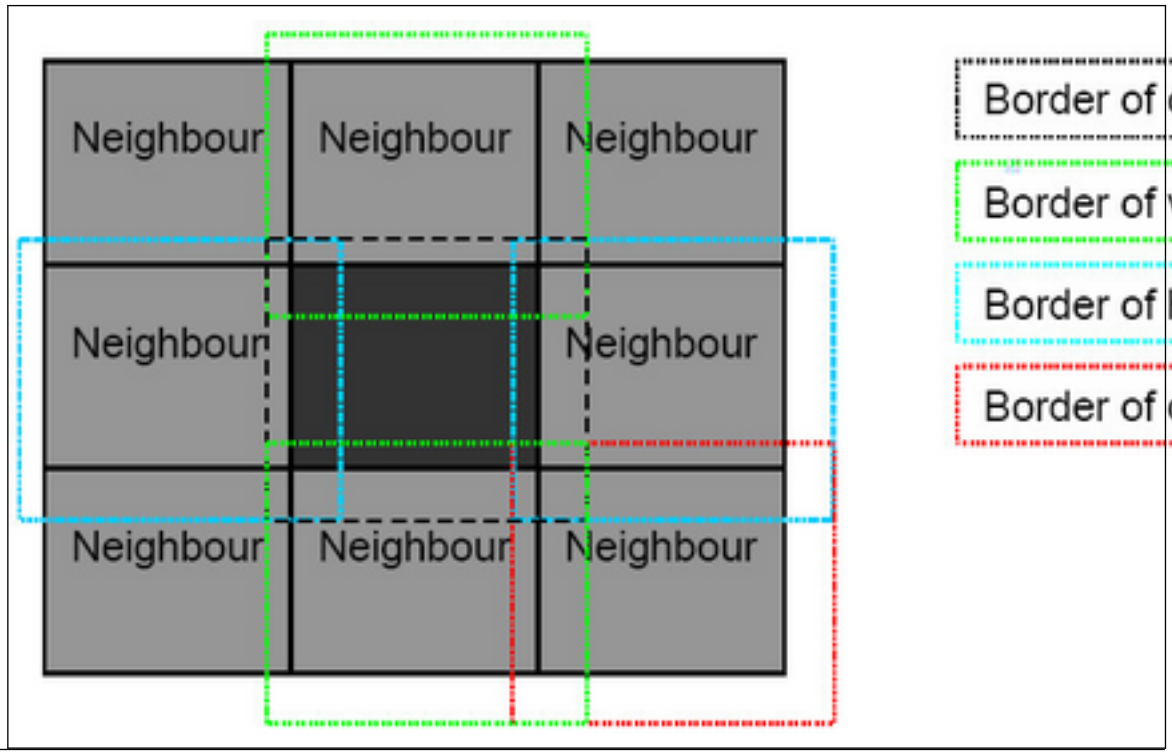
Conversely, to perform a wavelet reconstruction of the full size source image from the component images obtained by decomposition, use extended component images for the reconstruction pass. See the description of the function ippiWTInv for more details.

The ROI concept used in wavelet transform functions can be applied to processing large images by blocks, or 'tiles'. To accomplish this, the source image should be subdivided into overlapping blocks in the following way:

- Main part (ROI) of each block is adjacent to neighboring blocks and has no intersection with neighbor's ROIs;
- Extended borders of each block overlap with ROIs of neighboring blocks.

This subdivision scheme is illustrated in Figure Image Division into Blocks with Overlapping Borders .

Image Division into Blocks with Overlapping Borders



For an example on how to use this function, refer to the example provided with the WTInv function description.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `dstRoiSize` has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if step through any buffer is less than or equal to zero.

ippStsContextMatchErr	Indicates an error condition if a pointer to an invalid specification structure is passed.

See Also

WTFwdGetSize Calculates the size of the specification structure and work buffer for a forward wavelet transform.

WTFwdInit Initializes the forward wavelet transform context structure.

WTInv Performs one-level wavelet reconstruction of an image.

WTFwdGetSize

Calculates the size of the specification structure and work buffer for an inverse wavelet transform.

Syntax

```
IppStatus ippiWTInvGetSize_32f(int numChannels, int lenLow, int anchorLow, int lenHigh,
int anchorHigh, int* pSpecSize, int* pBufSize);
```

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

numChannels	Number of channels in the image. Possible values are 1 or 3.
lenLow	Length of the lowpass filter.
anchorLow	Anchor position of the lowpass filter.
lenHigh	Length of the highpass filter.
anchorHigh	Anchor position of the highpass filter.
pSpecSize	Pointer to the computed size of the ippiWTInv specification structure, in bytes.
pBufSize	Pointer to the computed size of the work buffer, in bytes.

Description

This function computes the size, in bytes, of the specification structure and work buffer required for the inverse wavelet transform function ippiWTInv.

For an example on how to use this function, refer to the example provided with the ippiWTInv function description.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.
ippStsNumChannlesErr	Indicates an error when numChannels has an illegal value.
ippStsNumChannlesErr	Indicates an error when lenLow or lenHigh is less than 2.
ippStsAnchorErr	Indicates an error when anchorLow or anchorHigh is less than zero.

See Also

WTInv Performs one-level wavelet reconstruction of an image.

WTInvInit

Initializes the inverse wavelet transform specification structure.

Syntax

```
IppStatus ippiWTInvInit_32f_C1R (IppiWTInvSpec_32f_C1R* pSpec,const Ipp32f* pTapsLow,
int lenLow, int anchorLow,const Ipp32f* pTapsHigh, int lenHigh, int anchorHigh);
```

```
IppStatus ippiWTInvInit_32f_C3R (IppiWTInvSpec_32f_C3R* pSpec,const Ipp32f* pTapsLow,
int lenLow, int anchorLow,const Ipp32f* pTapsHigh, int lenHigh, int anchorHigh);
```

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSpec	Double pointer to a new allocated and initialized inverse DWT context structure.
pTapsLow	Pointer to lowpass filter taps.
lenLow	Length of the lowpass filter.
anchorLow	Anchor position of the lowpass filter.
pTapsHigh	Pointer to highpass filter taps.
lenHigh	Length of the highpass filter.
anchorHigh	Anchor position of the highpass filter.

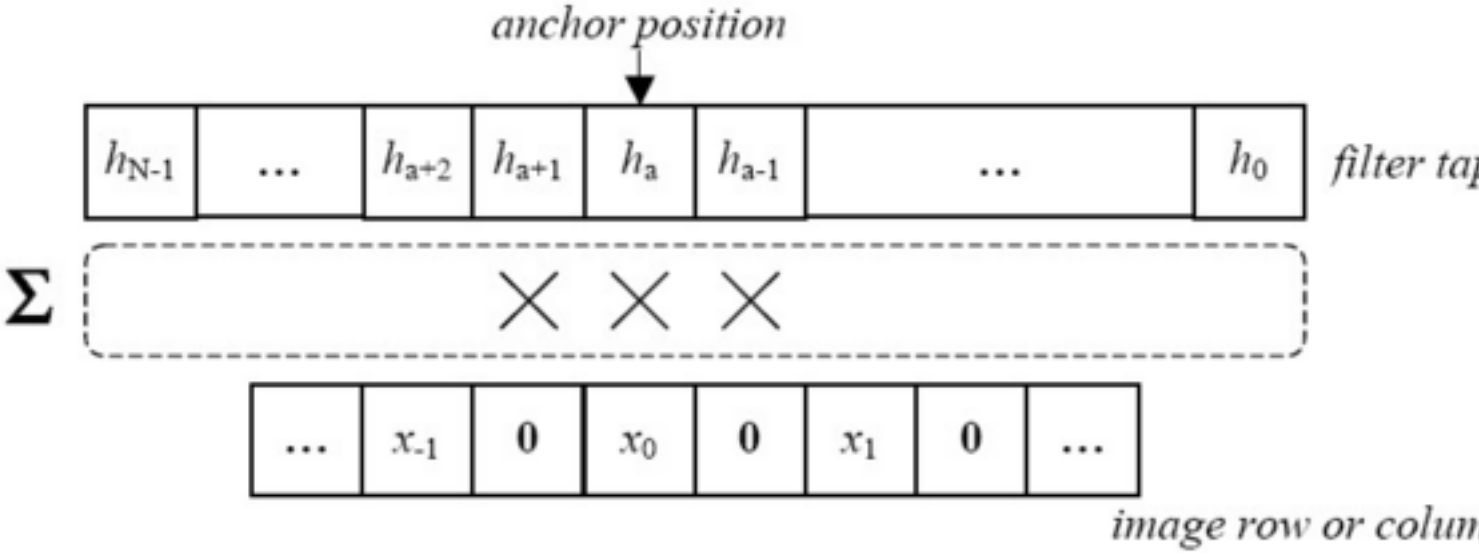
Description

This function initializes the specification structure pSpec for a one-level wavelet reconstruction.

The inverse wavelet transform specification structure contains parameters of a wavelet filter bank used for image reconstruction. The filter bank consists of two synthesis filters and includes the lowpass reconstruction filter (or *coarse* filter) and the highpass reconstruction filter (or *detail* filter).

The parameters `pTapsLow` and `pTapsHigh` specify coefficients, and `anchorLow` and `anchorHigh` - anchor positions for two synthesis filters. The anchor value sets the initial leftmost filter position relative to image row or column as shown in the figure below:

Anchor Value and Initial Filter Position for Wavelet Reconstruction



Here `a` stands for anchor value, `N` is filter length, `x0` is the starting pixel of the processed row or column, and `x-1`, `x-2` ,… are the additional border pixels that are needed for calculations. As seen from this figure, anchor position is specified relative to upsampled source data. The anchor value and filter length completely determine right, left, top, and bottom border sizes for source images used in reconstruction. The corresponding C-language expressions to calculate border sizes are given in the description of the ippWTInv function.

For an example on how to use this function, refer to the example provided with the ippiWTInv function description.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error when any of the specified pointers is `NULL`.
`ippStsNumChannelsErr`	Indicates an error when `numChannels` has an illegal value.
`ippStsSizeErr`	Indicates an error when filter length `lenLow` or `lenHigh` is less than 2.
`ippStsAnchorErr`	Indicates an error when anchor position `anchorLow` or `anchorHigh` is less than zero.

See Also

WTInvGetSize Calculates the size of the specification structure and work buffer for an inverse wavelet transform.

WTInv Performs one-level wavelet reconstruction of an image.

WTInv

Performs one-level wavelet reconstruction of an image.

Syntax

```
IppStatus ippiWTInv_32f_C1R(const Ipp32f* pApproxSrc, int approxStep, const Ipp32f*
pDetailXSrc, int detailXStep, const Ipp32f* pDetailYSrc, int detailYStep, const Ipp32f*
pDetailXYSrc, int detailXYStep, IppiSize srcRoiSize, Ipp32f* pDst, int dstStep, const
IppiWTInvSpec_32f_C1R* pSpec,Ipp8u* pBuffer);
```

```
IppStatus ippiWTInv_32f_C3R(const Ipp32f* pApproxSrc, int approxStep, const Ipp32f*
pDetailXSrc, int detailXStep, const Ipp32f* pDetailYSrc, int detailYStep, const Ipp32f*
pDetailXYSrc, int detailXYStep, IppiSize srcRoiSize, Ipp32f* pDst, int dstStep, const
IppiWTInvSpec_32f_C3R* pSpec,Ipp8u* pBuffer);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pApproxSrc`	Pointer to ROI of the source approximation image.
`approxStep`	Distance, in bytes, between the starting points of consecutive lines in the approximation image buffer.
`pDetailXSrc`	Pointer to ROI of the source horizontal detail image.
`detailXStep`	Distance, in bytes, between the starting points of consecutive lines in the horizontal detail image buffer.
`pDetailYSrc`	Pointer to ROI of the source vertical detail image.
`detailYStep`	Distance, in bytes, between the starting points of consecutive lines in the vertical detail image buffer.
`pDetailXYSrc`	Pointer to ROI of the source diagonal detail image.
`detailXYStep`	Distance, in bytes, between the starting points of consecutive lines in the diagonal detail image buffer.
`srcRoiSize`	Size of ROI in pixels for all source images.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance, in bytes, between the starting points of consecutive lines in the destination image buffer.
`pSpec`	Pointer to the allocated and initialized inverse DWT specification structure.

pBuffer	Pointer to the allocated buffer for intermediate operations.

Description

This function operates with ROI (see Regions of Interest in Intel IPP). This function performs wavelet reconstruction of the output image pDst from the four component images. See Figure "Image Division into Blocks with Overlapping Borders" for the equivalent algorithm of ippiWTInv function operation. Wavelet parameters are contained in the inverse transform specification structure pSpec. Before using this function, compute the size of the structure and work buffer using the ippiWTInvGetSize function and initialize the structure using ippiWTInvInit.

The pointers pApproxSrc, pDetailXSrc, pDetailYSrc, and pDetailXYSrc point to ROIs of source images excluding borders. All source ROIs have the same size srcRoiSize, while the destination image size is uniquely determined from the following relations:

```
dstWidth = 2 * srcRoiSize.width; dstHeight = 2 *srcRoiSize.height;
```

As source ROIs do not include border pixels required to computations, the application program have to apply a border extension model (symmetrical, wraparound or another) to ROIs of all source images filling the neighboring memory locations. Note the border sizes may be different for different source images. The following C-language expressions can be used to calculate extended image border sizes:

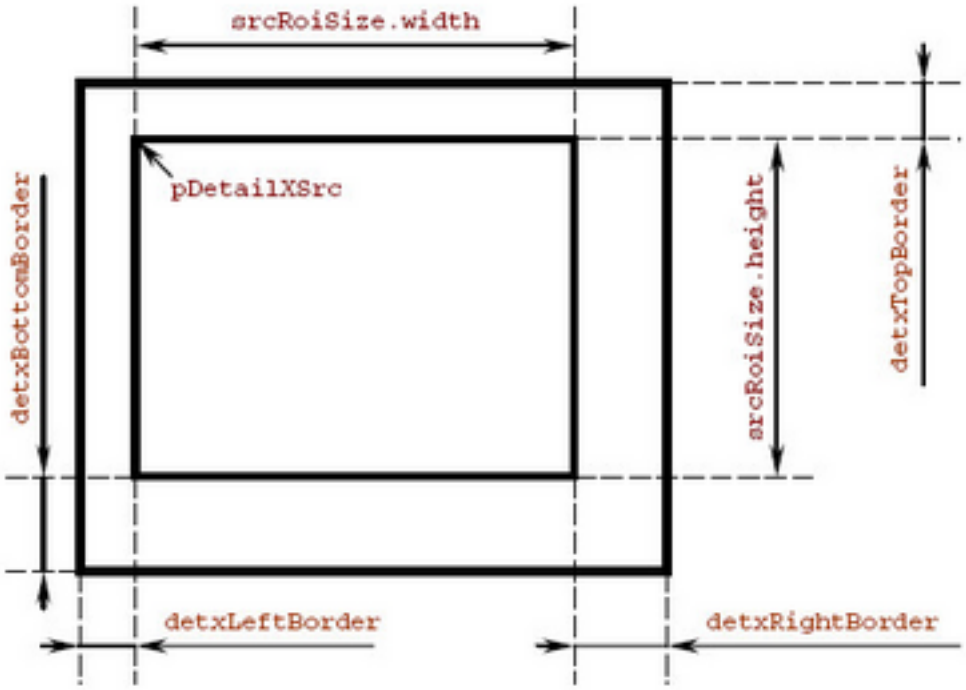
```
int leftBorderLow    = (lenLow - 1 - anchorLow) / 2;
int leftBorderHigh   = (lenHigh - 1 - anchorHigh) / 2;
int rightBorderLow   = (anchorLow + 1) / 2;
int rightBorderHigh  = (anchorHigh + 1) / 2;
int apprLeftBorder   = leftBorderLow;
int apprRightBorder  = rightBorderLow;
int apprTopBorder    = leftBorderLow;
int apprBottomBorder = rightBorderLow;
int detxLeftBorder   = leftBorderLow;
int detxRightBorder  = rightBorderLow;
int detxTopBorder    = leftBorderHigh;
int detxBottomBorder = rightBorderHigh;
int detyLeftBorder   = leftBorderHigh;
int detyRightBorder  = rightBorderHigh;
int detyTopBorder    = leftBorderLow;
int detyBottomBorder = rightBorderLow;
int detxyLeftBorder   = leftBorderHigh;
int detxyRightBorder  = rightBorderHigh;
int detxyTopBorder    = leftBorderHigh;
int detxyBottomBorder = rightBorderHigh;
```

See the description of the function ippiWTInvInit for the explanation of the used parameters.

The above relations show that left and top borders always have equal size only for approximation and diagonal detail images. Right and bottom borders also have equal size only for approximation and diagonal detail images. Thus, the size of memory block allocated for each source image must be extended to accommodate for added border pixels outside ROI.

Figure "Extended Horizontal Detail Source Image for Wavelet Reconstruction" shows ROI and extended image area for the horizontal detail source image.

Extended Horizontal Detail Source Image for Wavelet Reconstruction



Sizes of source images extended by border pixels can be calculated as follows:

```
apprWidthWithBorders  = srcWidth  + apprLeftBorder + apprRightBorder;
apprHeightWithBorders = srcHeight + apprTopBorder  + apprBottomBorder;
detxWidthWithBorders  = srcWidth  + detxLeftBorder + detxRightBorder;
detxHeightWithBorders = srcHeight + detxTopBorder  + detxBottomBorder;
detyWidthWithBorders  = srcWidth  + detyLeftBorder + detyRightBorder;
detyHeightWithBorders = srcHeight + detyTopBorder  + detyBottomBorder;
detxyWidthWithBorders  = srcWidth  + detxyLeftBorder + detxyRightBorder;
detxyHeightWithBorders = srcHeight + detxyTopBorder  + detxyBottomBorder;
```

The ROI concept can be used to reconstruct large images by blocks or ‘tiles’.

To accomplish this, each the source images into blocks with overlapping borders, similar to what is considered in the description of the function ippiWTFwd. Each component must be subdivided into the same pattern of rectangular blocks.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `srcRoiSize` has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if step through any buffer is less than or equal to zero.
`ippStsContextMatchErr`	Indicates an error condition if a pointer to an invalid specification structure is passed.

Example

The example below shows how to use the function `ippiWTInv_32f_C1R`.

```
void func_wavelet()
{
    IppiWTFwdSpec_32f_C1R* pSpecFwd;
    IppiWTInvSpec_32f_C1R* pSpecInv;
    int specSizeFwd, specSizeInv;
    Ipp32f pTapsLow[3] = {0.25, 0.5, 0.25};
    int lenLow = 3;
    int anchorLow = 1;
    Ipp32f pTapsHigh[3] = { 0.75, -0.25, -0.125};
    int lenHigh = 3;
    int anchorHigh = 1;
    Ipp32f pSrc[8*8] = { 0.0,  0.0,  0.0, 11.1, 11.1,  0.0,  0.0,  0.0,
                         0.0,  0.0,  0.0, 11.1, 11.1,  0.0,  0.0,  0.0,
                         0.0,  0.0,  0.0, 11.1, 11.1,  0.0,  0.0,  0.0,
                        11.1, 11.1, 11.1, 11.1, 11.1, 11.1, 11.1, 11.1,
                        11.1, 11.1, 11.1, 11.1, 11.1, 11.1, 11.1, 11.1,
                         0.0,  0.0,  0.0, 11.1, 11.1,  0.0,  0.0,  0.0,
                         0.0,  0.0,  0.0, 11.1, 11.1,  0.0,  0.0,  0.0,
                         0.0,  0.0,  0.0, 11.1, 11.1,  0.0,  0.0,  0.0};
    Ipp32f pSrcB[9*9];
    int srcStepB = 9*sizeof(Ipp32f);
    IppiSize roiSizeB = {9, 9};
    int srcStep = 8*sizeof(Ipp32f);
    IppiSize roiSize = {8, 8};
    Ipp32f pDetailXDst[4*4];
    Ipp32f pDetailYDst[4*4];
    Ipp32f pDetailXYDst[4*4];
    Ipp32f pApproxDst[4*4];
    IppiSize dstRoiSize = {4, 4};
    int bufSizeFwd, bufSizeInv;
    Ipp8u* pBufferFwd;
    Ipp8u* pBufferInv;
    Ipp32f pDstInv[8*8];
    Ipp32f pAppB[5*5];
    Ipp32f pXB[5*5];
    Ipp32f pYB[5*5];
    Ipp32f pXYB[5*5];
    int StepB =  5*sizeof(Ipp32f);
    IppiSize roiInvSize = {4, 4};
    IppiSize roiInvSizeB = {5, 5};
    int stepDstInv = 8*sizeof(Ipp32f);
    int approxStep, detailXStep, detailYStep, detailXYStep;
    approxStep = detailXStep = detailYStep = detailXYStep = 4*sizeof(Ipp32f);
    //adds border to the source image
    ippiCopyWrapBorder_32s_C1R((Ipp32s*)pSrc, srcStep, roiSize, (Ipp32s*)pSrcB, srcStepB,
roiSizeB, 1, 1);
    //performs forward  wavelet transform
    ippiWTFwdGetSize_32f(1, lenLow, anchorLow, lenHigh, anchorHigh, &specSizeFwd, &bufSizeFwd);
    pSpecFwd = (IppiWTFwdSpec_32f_C1R*)ippMalloc(specSizeFwd);
    pBufferFwd = (Ipp8u*)ippMalloc(bufSizeFwd);
    ippiWTFwdInit_32f_C1R( pSpecFwd, pTapsLow, lenLow, anchorLow, pTapsHigh, lenHigh,
anchorHigh);
    ippiWTFwd_32f_C1R (pSrcB + roiSizeB.width + 1, srcStepB, pApproxDst, approxStep,
       pDetailXDst, detailXStep, pDetailYDst, detailYStep, pDetailXYDst, detailXYStep,
dstRoiSize, pSpecFwd, pBufferFwd);
```

```
    printf_32f_2D("After WTFwd ->\n pApproxDst", pApproxDst, dstRoiSize, approxStep,
ippStsNoErr);
    printf_32f_2D("pDetailXDst", pDetailXDst, dstRoiSize, detailXStep, ippStsNoErr);
    printf_32f_2D("pDetailYDst", pDetailYDst, dstRoiSize, detailYStep, ippStsNoErr);
    printf_32f_2D("pDetailXYDst", pDetailXYDst, dstRoiSize, detailXYStep, ippStsNoErr);


    //-----------------------------------
    ippiWTInvGetSize_32f(1, lenLow, anchorLow, lenHigh, anchorHigh, &specSizeInv, &bufSizeInv);
    pSpecInv = (IppiWTInvSpec_32f_C1R*)ippMalloc(specSizeInv);
    pBufferInv = (Ipp8u*)ippMalloc(bufSizeInv);
    ippiWTInvInit_32f_C1R(pSpecInv, pTapsLow, lenLow, anchorLow, pTapsHigh, lenHigh, anchorHigh);
    //adds border to four images obtained after ippiWTFwd
    ippiCopyWrapBorder_32s_C1R((Ipp32s*)pApproxDst, approxStep, dstRoiSize, (Ipp32s*)pAppB,
StepB, roiInvSizeB, 0, 0);
    ippiCopyWrapBorder_32s_C1R((Ipp32s*)pDetailXDst, detailXStep, dstRoiSize, (Ipp32s*)pXB,
StepB, roiInvSizeB, 0, 0);
    ippiCopyWrapBorder_32s_C1R((Ipp32s*)pDetailYDst, detailYStep, dstRoiSize, (Ipp32s*)pYB,
StepB, roiInvSizeB, 0, 0);
    ippiCopyWrapBorder_32s_C1R((Ipp32s*)pDetailXYDst, detailXYStep, dstRoiSize, (Ipp32s*)pXYB,
StepB, roiInvSizeB, 0, 0);
    //performs inverse  wavelet transform
    ippiWTInv_32f_C1R( pAppB, StepB, pXB, StepB, pYB, StepB, pXYB, StepB, roiInvSize, pDstInv,
stepDstInv, pSpecInv,  pBufferInv);


    printf_32f_2D("After WTFinv ->\n pDstInv", pDstInv, roiSize, stepDstInv, ippStsNoErr);


    ippFree(pSpecFwd);
    ippFree(pSpecInv);
    ippFree(pBufferFwd);
    ippFree(pBufferInv);
}
```

Result:

```
After WTFwd ->
pApproxDst
0.0  2.8   8.3  0.0
2.8  4.9   9.0  2.8
8.3  9.0  10.4 8.3
0.0  2.8   8.3  0.0
pDetailXDst
  0.0   1.0  3.1   0.0
  8.3   7.3  5.2   8.3
 -4.2  -2.1  2.1  -4.2
  0.0   1.0  3.1   0.0
pDetailYDst
  0.0   8.3   -4.2   0.0
  1.0   7.3   -2.1   1.0
  3.1   5.2    2.1   3.1
  0.0   8.3   -4.2   0.0
pDetailXYDst
  0.0   3.1   -1.6   0.0
  3.1   0.0    4.7   3.1
 -1.6   4.7   -4.7  -1.6
```

```
  0.0   3.1   -1.6   0.0
After WTFinv ->
pDstInv
 0.0    2.8    -0.3   -0.6    2.1    1.1    0.0    0.0
 2.8    5.9     2.7    4.9    3.4    5.4    2.8    5.1
-0.3    2.7    -0.6   -1.2    2.2    0.7   -0.3   -0.5
-0.6    4.9    -1.2   -2.2    3.9    1.2   -0.6   -1.0
 2.1    3.4     2.2    3.9    1.8    3.7    2.1    3.8
 1.1    5.4     0.7    1.2    3.7    3.2    1.1    1.9
 0.0    2.8    -0.3   -0.6    2.1    1.1    0.0    0.0
 0.0    5.1    -0.5   -1.0    3.8    1.9    0.0    0.0
```

See Also

WTInvGetSize Calculates the size of the specification structure and work buffer for an inverse wavelet transform.

WTInvInit Initializes the inverse wavelet transform specification structure.

WTFwd Performs one-level wavelet decomposition of an image.

Computer Vision

This chapter provides some background for the computer vision concepts used in the Intel® IPP library as well as detailed descriptions of the Intel IPP image processing functions for computer vision. These functions are combined in groups by their functionality.

- Using ippiAdd for Background Differencing
- Feature Detection Functions
- Distance Transform Functions
- Image Gradients
- Flood Fill Functions
- Motion Analysis and Object Tracking
- Universal Pyramids
- Image Inpainting
- Image Segmentation
- Pattern Recognition
- Camera Calibration and 3D Reconstruction

Using ippiAdd for Background Differencing

This section describes functions that help build a statistical model of a background. This model can be used to subtract the background from an image.

Here, the term “background” stands for a set of motionless image pixels – that is, pixels that do not belong to any object, moving in front of the camera. The definition of background can vary if considered in other techniques of object extraction. For example, if the depth map of the scene can be obtained, for example, with the help of stereo, background can be determined as static parts of the scene that are located far enough from the camera.

The simplest background model assumes that every background pixel brightness varies independently, according to normal distribution. To calculate the characteristics of the background, several dozens of frames, as well as their squares, can be accumulated. That is, for every pixel location we find the sum of pixel values in this location `S(x, y)`, using the function ippiAdd, and the sum of squares of the values `Sq(x, y)`, using the function ippiAddSquare. Then mean is calculated as

$$m(x, y) = \frac{S(x, y)}{N}$$

where N is the number of collected frames, and standard deviation as

$$stddev(x, y) = sqrt\left(\frac{Sq(x, y)}{N} - \left(\frac{S(x, y)}{N}\right.\right.$$

After that, the pixel in a certain pixel location within a certain frame is considered as belonging to a moving object, if the condition `abs(p(x, y) - m(x, . y)) < C * stddev(x, y)`, where C is a constant, is met. If C is equal to 3, it satisfies the "three sigmas" rule. To obtain such background model, objects should be put away from the camera for a few seconds, so that the whole image from the camera represents the subsequent background observation.

Adapting the background differencing model to changes in lighting conditions and background scenes, for example, when the camera moves or an object passes behind the front object, can improve the described technique.

The mean brightness can be calculated through replacing the simple average with the running average found by using the function ippiAddWeighted. Also, several techniques can be used to identify moving scene parts and exclude them while accumulating background information. These techniques include change detection (see the functions ippiAbsDiff and ippiThreshold), optical flow, and some other operations.

Relevant addition functions used for background differencing include:

`Add_8u32f_C1IR, Add_8s32f_C1IR, Add_32f_C1IR,`

`Add_8u32f_C1IMR, Add_8s32f_C1IMR, Add_32f_C1IMR` (see Add),

and also all flavors of AddSquare, AddProduct, and AddWeighted.

Feature Detection Functions

This section describes feature detection functions.

The set of the Sobel derivative filters is generally used to find edges, ridges, and blobs, especially in case of scale-space images, for example, pyramids.

The following naming conventions are used in the equations described below:

- D_x and D_y are the first x and y derivatives, respectively.
- D_{xx} are D_{yy} are the second x and y derivatives, respectively.
- D_{xy} is the partial x and y derivative.

- D_{xxx} and D_{yyy} are the third x and y derivatives, respectively.
- D_{xxy} and D_{xyy} are the third partial x and y derivatives.

- Corner Detection
- FastNGetSize
- FastNGetBufferSize
- FastNInit
- FastN
- FastN2DToVec
- HarrisCornerGetBufferSize
- HarrisCorner
- Canny Edge Detector
- Histogram of Oriented Gradients (HOG) Descriptor
- Hough Transform
- LineSuppressionGetBufferSize
- LineSuppression

Corner Detection

The Sobel and Scharr first derivative operators are to be used to take the x and y derivatives of an image. Then a small region of interest (ROI) is to be defined to detect corners in.

A 2x2 matrix of sums of the x and y derivatives is created as follows:

$$A = \sum_u \sum_v w(u, v) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} = \begin{bmatrix} \langle I_x^2 \rangle & \langle I_x I_y \rangle \\ \langle I_x I_y \rangle & \langle I_y^2 \rangle \end{bmatrix}$$

Solving

$$det(C - \lambda I) = 0$$

where λ is a column vector of the eigen values and I is the identity matrix, gives the eigen values. For the 2x2 matrix of the equation above, the solutions may be written in a closed form:

$$\lambda = \frac{\Sigma D_x^2 + \Sigma D_y^2 \pm \sqrt{(\Sigma D_x^2 + \Sigma D_y^2)^2 - 4 \cdot (\Sigma D_x^2 \Sigma D_y^2 - (\Sigma D_x D_y)^2)}}{2}$$

If $\lambda_1, \lambda_2 > t$, where t is some threshold, then a corner is considered to be found at that location. This can be very useful for object or shape recognition.

FastNGetSize

Computes the size of the FastN context structure.

Syntax

```
IppStatus ippiFastNGetSize(IppiSize srcSize, int circleRadius, int N, int
orientationBins, int option, IppDataType dataType, int numChannels, int* pSpecSize);
```

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`srcSize` — Size of the source ROI, in pixels.

`circleRadius` — Radius of the pixel circle. The radius value equal to 1 corresponds to the distance between the closest pixels from common row or column. Supported values are 1, 2, 3, 5, 7, 9.

`N` — Critical number of serial pixels on circle for defining a corner. The ranges are as follows:

circleRadius	N
1	5 ≤ N ≤ 8
2	7 ≤ N ≤ 12
3	9 ≤ N ≤ 16
5	15 ≤ N ≤ 28
7	21 ≤ N ≤ 40
9	27 ≤ N ≤ 52

`orientationBins` — The number of bins to define direction. Supported values are from 2 to 64.

`option` — Mode of processing. Supported values:

- `IPP_FASTN_CIRCLE`
- `(IPP_FASTN_CIRCLE | IPP_FASTN_ORIENTATION)`
- `(IPP_FASTN_CIRCLE | IPP_FASTN_SCORE_MODE0)`
- `(IPP_FASTN_CIRCLE | IPP_FASTN_ORIENTATION | IPP_FASTN_SCORE_MODE0)`
- `(IPP_FASTN_CIRCLE | IPP_FASTN_SCORE_MODE0 | IPP_FASTN_NMS)`
- `(IPP_FASTN_CIRCLE | IPP_FASTN_ORIENTATION | IPP_FASTN_SCORE_MODE0 | IPP_FASTN_NMS)`

`dataType` — Data type of the source and destination images. Supported value is `ipp8u`.

numChannels	Number of channels in the images. Supported value is 1.
pSpecSize	Pointer to the computed size of the specification structure.

Description

This function computes the size of the FastN context structure for the FastN function. The result is stored in pSpecSize.

Use the computed pSpecSize value to allocate memory using the ippMalloc or ippsMalloc functions. The allocated memory can be freed only by the ippFree or ippsFree functions, respectively. For more information about the memory allocation functions, refer to the *Support Functions* section of the *Intel IPP Developer Reference, vol. 1*.

For an example on how to use this function, refer to the example provided with the FastN function description.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error when pSpecSize is NULL.
ippStsSizeErr	Indicates an error when srcSize is less than, or equal to zero.
ippDataTypeErr	Indicates an error when dataType has an illegal value.
ippNumChannelsErr	Indicates an error when numChannels has an illegal value.
ippOutOfRangeErr	Indicates an error when orientationBins or N has an illegal value.
ippBadArgErr	Indicates an error when option or circleRadius has an illegal value.

See Also

FastN Detects corners in an image using the FastN algorithm.

FastNGetBufferSize

Computes the size of the work buffer for the FastN function.

Syntax

```
IppStatus ippiFastNGetBufferSize(IppiFastNSpec* pSpec, IppiSize dstRoiSize, int* pBufSize);
```

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSpec	Pointer to the FastN specification structure.
dstRoiSize	Size of the destination ROI, in pixels.
pBufSize	Pointer to the computed size of the work buffer.

Description

This function computes the size of the work buffer for the FastN function. The result is stored in pBufSize.

Use the computed pBufSize value to allocate memory using the ippMalloc or ippsMalloc functions. The allocated memory can be freed only by the ippFree or ippsFree functions, respectively. For more information about the memory allocation functions, refer to the *Support Functions* section of the *Intel IPP Developer Reference, vol. 1*.

For an example on how to use this function, refer to the example provided with the FastN function description.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error when pSpecSize or pBufSize is NULL.
ippStsSizeErr	Indicates an error when dstRoiSize is less than, or equal to zero.

See Also

FastN Detects corners in an image using the FastN algorithm.

FastNInit

Initializes the FastN context structure.

Syntax

```
IppStatus ippiFastNInit(IppiSize srcSize, int circleRadius, int N, int orientationBins,
int option, Ipp32f threshold, IppDataType dataType, int numChannels, IppiFastNSpec*
pSpec);
```

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

srcSize	Size of the source ROI, in pixels.

`circleRadius` Radius of the pixel circle. The radius value equal to 1 corresponds to the distance between the closest pixels from common row or column. Supported values are 1, 2, 3, 5, 7, 9.

`N` Critical number of serial pixels on circle for defining a corner. The ranges are as follows:

`circleRadius`	`N`
1	5 ≤ N ≤ 8
2	7 ≤ N ≤ 12
3	9 ≤ N ≤ 16
5	15 ≤ N ≤ 28
7	21 ≤ N ≤ 40
9	27 ≤ N ≤ 52

`orientationBins` The number of bins to define direction. Supported values are from 2 to 64.

`option` Mode of processing. Supported values:

- `IPP_FASTN_CIRCLE`
- `(IPP_FASTN_CIRCLE | IPP_FASTN_ORIENTATION)`
- `(IPP_FASTN_CIRCLE | IPP_FASTN_SCORE_MODE0)`
- `(IPP_FASTN_CIRCLE | IPP_FASTN_ORIENTATION | IPP_FASTN_SCORE_MODE0)`
- `(IPP_FASTN_CIRCLE | IPP_FASTN_SCORE_MODE0 | IPP_FASTN_NMS)`
- `(IPP_FASTN_CIRCLE | IPP_FASTN_ORIENTATION | IPP_FASTN_SCORE_MODE0 | IPP_FASTN_NMS)`

`threshold` Threshold value to detect critical pixels. The value must be more than, or equal to zero.

`dataType` Data type of the source and destination images. Supported value is `ipp8u`.

`numChannels` Number of channels in the images. Supported value is 1.

`pSpec` Pointer to the specification structure.

Description

This function initializes the FastN context structure for the FastN function.

For an example on how to use this function, refer to the example provided with the FastN function description.

Return Values

`ippStsNoErr` Indicates no error. Any other value indicates an error.

`ippStsNullPtrErr` Indicates an error when `pSpec` is `NULL`.

`ippStsSizeErr` Indicates an error when `srcSize` is less than, or equal to zero.

`ippDataTypeErr` Indicates an error when `dataType` has an illegal value.

ippNumChannelsErr	Indicates an error when numChannels has an illegal value.
ippOutOfRangeErr	Indicates an error when orientationBins or N has an illegal value.
ippBadArgErr	Indicates an error when option or circleRadius has an illegal value.
ippThresholdErr	Indicates an error when threshold is negative.

See Also

FastN Detects corners in an image using the FastN algorithm.

FastN

Detects corners in an image using the FastN algorithm.

Syntax

```
IppStatus ippiFastN_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDstCorner, int dstCornerStep, Ipp8u* pDstScore, int dstScoreStep, int* pNumCorner, IppiPoint srcRoiOffset, IppiSize dstRoiSize, IppiFastNSpec* pSpec, Ipp8u* pBuffer);
```

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Pointer to the source image.
srcStep	Distance, in bytes, between the starting points of consecutive lines in the source image.
pDstCorner	Pointer to the destination image with corners.
dstCornerStep	Distance, in bytes, between the starting points of consecutive lines in the destination image with corners.
pDstScore	Pointer to the destination image with scores.
dstScoreStep	Distance, in bytes, between the starting points of consecutive lines in the destination image with scores.
pNumCorner	Pointer to the calculated number of corners.
srcRoiOffset	Offset in the source image.
dstRoiSize	Size of the destination ROI, in pixels.
pSpec	Pointer to the specification structure.
pBuffer	Pointer to the work buffer.

Description

The `ippiFastN` function implements the FastN corner detection algorithm. This function detects corners in the source image, calculates orientation and score of corners.

The figures below show pixels location for different radius values.

Radius = 1

Px3	Px4	Px5
Px2	pxc	Px6
Px1	Px0	Px7

Radius = 2

	Px5	Px6	Px7	
Px4				Px8
Px3		pxc		Px9
Px2				Px10
	Px1	Px0	Px11	

Radius = 3

		Px7	Px8	Px9		
	Px6				Px10	
Px5						Px11
Px4			pxc			Px12
Px3						Px13
	Px2				Px14	
		Px1	Px0	Px15		

Radius = 5

			Px12	Px13	Px14	Px15	Px16			
		Px11						Px17		
	Px10								Px18	
Px9										Px19
Px8										Px20
Px7					pxc					Px21
Px6										Px22
Px5										Px23
	Px4								Px24	
		Px3						Px25		
			Px2	Px1	Px0	Px27	Px26			

Radius = 7

					Px18	Px19	Px20	Px21	Px22					
			Px16	Px17						Px23	Px24			
		Px15										Px25		
	Px14												Px26	
	Px13												Px27	
Px12														Px28
Px11														Px29
Px10							pxc							Px30
Px9														Px31
Px8														Px32
	Px7												Px33	
	Px6												Px34	
		Px5										Px35		
			Px4	Px3							Px36			
					Px2	Px1	Px0	Px39	Px38	Px37				

Radius = 9

						Px23	Px24	Px25	Px26	Px27	Px28	Px29						
				Px21	Px22								Px30	Px31				
			Px20												Px32			
		Px19														Px33		
	Px18																Px34	
	Px17																Px35	
Px16																		Px36
Px15																		Px37
Px14																		Px38
Px13									pxc									Px39
Px12																		Px40
Px11																		Px41
Px10																		Px42
	Px9																Px43	
	Px8																Px44	
		Px7														Px45		
			Px6												Px46			
				Px5	Px4								Px48	Px47				
						Px3	Px2	Px1	Px0	Px51	Px50	Px49						

The function defines a pixel as a corner if the value of N consecutive pixels located on the circle with the center at the current pixel is greater (less) than the value of the current - central pixel. Differences between the values of pixels on the circle and the central pixel value must be greater than the threshold value. In this case, the corresponding pixel of the pDstCorner image is set to the following values in binary format:

- If greater - 01xxxxxx
- If less - 10xxxxxx

If the IPP_FASTN_ORIENTATION mode is not set, xxxxxx = 000000, otherwise, xxxxxx = the number of sector (bin) to which the corner directs. The orientationBin parameter defines the number of sectors that is computed from the bottom clockwise. If the source pixel is not a corner, the corresponding pixel of pDstCorner is set to zero.

If the IPP_FASTN_SCORE_MODE0 mode is set, the corresponding pixel of pDstCorner is set to the corner score. Score is a maximum among minimal differences with threshold calculated for every N consecutive pixels. If the source pixel is not a corner, the corresponding pixel of pDstScore is set to zero.

If the IPP_FASTN_NMS mode is set, the corners that have a corner with the greater score among neighboring pixels are cancelled.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error when: • pSrc, pDstCorner, pNumCorner, pSpec, or pBuffer is NULL • pDstScore is NULL if option is not equal to IPP_FASTN_SCORE_MODE0
ippStsSizeErr	Indicates an error when dstRoiSize is less than, or equal to zero.
ippDataTypeErr	Indicates an error when dataType has an illegal value.
ippNumChannelsErr	Indicates an error when numChannels has an illegal value.
ippOutOfRangeErr	Indicates an error when orientationBins or N has an illegal value.
ippBadArgsErr	Indicates an error when option or circleRadius has an illegal value.
ippThresholdErr	Indicates an error when threshold is negative.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

// A simple example of the corner finding FastN algorithm using Intel(R) Integrated Performance
```

```
Primitives (Intel(R) IPP) functions:
//     ippiFastNGetSize
//     ippiFastNInit
//     ippiFastNGetBufferSize
//     ippiFastN_8u_C1R
//     ippiFastN2DToVec_8u

#include <stdio.h>
#include "ipp.h"

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define check_sts(st)        \
    if ((st) != ippStsNoErr) \
        goto exitLine; /* Go to Exit if Intel(R) IPP function returned status different from
ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

Ipp8u pSrc[16 * 16] = {/* Pointer to source images */
                       0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   70,  70,  70,  70,
70,  70,  0,   0,   0,   0,   0,   0,   0,   0,
                       0,   0,   70,  70,  70,  70,  70,  70,  0,   0,   0,   0,   0,   0,
0,   0,   0,   0,   70,  70,  70,  70,  70,  70,
                       0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   70,  70,  70,  70,
70,  70,  0,   0,   0,   0,   0,   0,   0,   0,
                       0,   0,   70,  70,  70,  70,  70,  70,  0,   0,   0,   0,   0,   0,
0,   0,   0,   0,   70,  70,  70,  70,  70,  70,
                       0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
0,   0,   255, 255, 255, 255, 255, 255, 255, 255,
                       255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
                       255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255,
255, 255, 10,  10,  10,  10,  10,  10,  255, 255,
                       255, 255, 255, 100, 100, 100, 100, 100, 10,  10,  10,  10,  10,  10,
255, 255, 255, 255, 100, 100, 100, 100, 100, 100,
                       10,  10,  10,  10,  10,  10,  255, 255, 255, 255, 100, 100, 100, 100,
100, 100, 10,  10,  10,  10,  10,  10,  255, 255,
                       255, 255, 100, 100, 100, 100, 100, 100, 10,  10,  10,  10,  10,  10,
255, 255, 255, 255, 100, 100, 100, 100, 100, 100,
                       10,  10,  10,  10,  10,  10,  255, 255, 255, 255, 100, 100, 100, 100,
100, 100};
IppiSize srcSize = {16, 16}; /* Size of source/destination ROI in pixels */
IppiSize dstSize = {16, 16};
IppStatus status = ippStsNoErr;
Ipp8u pDstCorner[16 * 16]; /* Pointer to destination images */
IppiCornerFastN *pDstStruct = NULL;
int srcStep = 16, dstCornerStep = 16; /* Steps, in bytes, through the source/destination images
*/
int circleRadius = 3;                /* Radius for corner finding */
int N = 9;                           /*  Critical number of pixels that have different value */
int orientationBins = 64;            /* Number bins for defining direction */
int option = IPP_FASTN_CIRCLE;       /* Defines the mode of processing */
const IppDataType dataType = ipp8u;
int nChannels = 1;
int specSize = 0;               /* Size, in bytes, of the spec */
```

```
IppiFastNSpec *pSpec = NULL;     /* Pointer to Spec */
Ipp32f threshold = 50.0f;        /* Level for definition of critical pixels */
int bufSize = 0;                 /* Work buffer size */
Ipp8u *pBuffer = NULL;           /* Pointer to the work buffer */
int maxLen = 0;                  /* Length of array of structures */
IppiPoint srcRoiOffset = {0, 0}; /* Offset in source image */
int numRealCorners;              /* Number of corners */

int main(void)
{
    /* Initialization of Spec for FastN */
    check_sts(status = ippiFastNGetSize(srcSize, circleRadius, N, orientationBins, option,
dataType, nChannels, &specSize));

    pSpec = (IppiFastNSpec *)ippsMalloc_8u(specSize);

    /* Define spec size for FastN */
    check_sts(status = ippiFastNInit(srcSize, circleRadius, N, orientationBins, option,
threshold, dataType, nChannels, pSpec));

    check_sts(status = ippiFastNGetBufferSize(pSpec, dstSize, &bufSize));

    pBuffer = ippsMalloc_8u(bufSize);

    /* Define size of working buffer for FastN */
    check_sts(status = ippiFastN_8u_C1R(pSrc, srcStep, pDstCorner, dstCornerStep, NULL, 0,
&maxLen, srcRoiOffset, dstSize, pSpec, pBuffer));

    pDstStruct = (IppiCornerFastN *)ippsMalloc_8u(sizeof(IppiCornerFastN) * maxLen);

    /* Corner finding with FastN */
    check_sts(status = ippiFastN2DToVec_8u(pDstCorner, dstCornerStep, NULL, 0, pDstStruct,
dstSize, maxLen, &numRealCorners, pSpec));

    EXIT_MAIN
    ippsFree(pDstStruct);
    ippsFree(pBuffer);
    ippsFree(pSpec);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

See Also

FastNGetSize Computes the size of the FastN context structure.

FastNInit Initializes the FastN context structure.

FastNGetBufferSize Computes the size of the work buffer for the FastN function.

FastN2DToVec Converts corners from two-dimensional image to an array of structures.

FastN2DToVec

Converts corners from two-dimensional image to an array of structures.

Syntax

```
IppStatus ippiFastN2DToVec_8u(const Ipp8u* pSrcCorner, int srcCornerStep, const Ipp8u*
pSrcScore, int srcScoreStep, IppiCornerFastN* pDst, IppiSize srcRoiSize, int maxLen,
int* pNumCorners, IppiFastNSpec* pSpec);
```

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrcCorner`	Pointer to the source image with corners.
`srcCornerStep`	Distance, in bytes, between the starting points of consecutive lines in the source image with corners.
`pSrcScore`	Pointer to the source image with scores.
`srcScoreStep`	Distance, in bytes, between the starting points of consecutive lines in the source image with score.
`pDst`	Pointer to the destination vector of structures.
`srcRoiSize`	Size of the source ROI, in pixels.
`maxLen`	Length of the array of structures.
`pNumCorners`	Pointer to the number of corners in the source image.
`pSpec`	Pointer to the specification structure.

Description

This function converts two-dimensional image with corners to an array of structures. The result is stored in `pDst`.

For an example on how to use this function, refer to the example provided with the FastN function description.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error when: • `pSrcCorner`, `pDst`, `pNumCorners`, or `pSpec` is `NULL` • `pSrcScore` is `NULL` if `option` is not equal to `IPP_FASTN_SCORE_MODE0`
`ippStsSizeErr`	Indicates an error when `srcRoiSize` is less than, or equal to zero.
`ippContextMatchErr`	Indicates an error when the specification structure does not match the operation.
`ippDataTypeErr`	Indicates an error when `dataType` has an illegal value.

ippNumChannelsErr	Indicates an error when numChannels has an illegal value.
ippOutOfRangeErr	Indicates an error when orientationBins or N has an illegal value.
ippBadArgsErr	Indicates an error when option or circleRadius has an illegal value.
ippThresholdErr	Indicates an error when threshold is negative.

See Also

FastN Detects corners in an image using the FastN algorithm.

HarrisCornerGetBufferSize

Calculates the size of the temporary buffer for the ippiHarrisCorner function.

Syntax

```
IppStatus ippiHarrisCornerGetBufferSize(IppiSize roiSize, IppiMaskSize filterMask,
Ipp32u avgWndSize, IppDataType dataType, int numChannels, int* pBufferSize);
```

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

roiSize	Size of the source and destination image ROI in pixels.
filterMask	Size of the derivative filter aperture. Possible values are ippMskSize3x3 and ippMskSize5x5.
avgWndSize	Linear size of a neighborhood block for averaging.
dataType	Data type of the source image. Possible values are ipp8u and ipp32f.
numChannels	Number of channels in the images. Possible value is 1.
pBufferSize	Pointer to the size (in bytes) of the external work buffer.

Description

This function calculates the size of the temporary buffer needed for the HarrisCorner function.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when pBufferSize is NULL.
ippStsSizeErr	Indicates an error in the following cases:

	• when `roiSize` is less than, or equal to zero • when `avgWndSize` is equal to zero
`ippStsMaskSizeErr`	Indicates an error when `filterMask` has an illegal value.
`ippStsDataTypeErr`	Indicates an error when `dataType` has an illegal value.
`ippStsNumChannelsErr`	Indicates an error when `numChannels` has an illegal value.

See Also

HarrisCorner Implements Harris corner detection algorithm.

HarrisCorner

Implements Harris corner detection algorithm.

Syntax

```
IppStatus ippiHarrisCorner_8u32f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiDifferentialKernel filterType, IppiMaskSize filterMask, Ipp32u avgWndSize, float k, float scale, IppiBorderType borderType, Ipp8u borderValue, Ipp8u* pBuffer);
```

```
IppStatus ippiHarrisCorner_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, IppiDifferentialKernel filterType, IppiMaskSize filterMask, Ipp32u avgWndSize, float k, float scale, IppiBorderType borderType, Ipp32f borderValue, Ipp8u* pBuffer);
```

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the source image.
`srcStep`	Distance, in bytes, between the starting points of consecutive lines in the source image.
`pDst`	Pointer to the destination image.
`dstStep`	Distance, in bytes, between the starting points of consecutive lines in the destination image.
`roiSize`	Size of the source and destination image ROI in pixels.
`filterType`	Type of the derivative operator. Possible values are:

`ippKernelSobel`	Sobel filter
`ippKernelScharr`	Scharr filter

ippKernelCentralDiff	Central differences operator

filterMask — Size of the derivative filter aperture. The list of possible values depends on the filterType value:

Filter Type	Filter Mask
ippKernelSobel	ippMskSize3x3, ippMskSize5x5
ippKernelScharr	ippMskSize3x3
ippKernelCentralDiff	ippMskSize3x3

avgWndSize — Linear size of a neighborhood block for averaging.

k — Harris detector free coefficient.

scale — Destination image scale factor.

borderType — Type of border. Possible values are:

ippBorderConst	Values of all border pixels are set to constant.
ippBorderRepl	Border is replicated from the edge pixels.
ippBorderInMem	Border is obtained from the source image pixels in memory.
ippBorderMirror	Border pixels are mirrored from the source image boundary pixels.

Mixed borders are also supported. They can be obtained by the bitwise operation OR between ippBorderRepl or ippBorderConst and the following flags:

- ippBorderInMemTop
- ippBorderInMemBottom
- ippBorderInMemLeft
- ippBorderInMemRight

Each of these flags means that for the corresponding border the outside pixels of the source image are in memory.

borderValue — Constant value(s) to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type. The ippiHarrisCorner function uses the specified constant value only at the first stage of the algorithm. At the third stage, the function uses zero constant border.

pBuffer — Pointer to the pre-allocated temporary buffer. To calculate the size of the temporary buffer, use the HarrisCornerGetBufferSize function.

Description

This function goes through the following stages to implement the Harris corner detection algorithm:

1. Computes $I(\grave{x}, y)'_x$ and $I(\grave{x}, y)'_y$ gradients for each (x, y) pixel of the image. The function computes gradients using the derivative operator specified by the filterType and filterMask parameters.

2. Computes products of the gradients at each (`x`, `y`) pixel of the image:

$$I_{xx}^{(x,y)} = I_x^{(x,y)} * I_x^{(x,y)}, \quad I_{yx}^{(x,y)} = I_{xy}^{(x,y)} = I_x^{(x,y)} * I_y^{(x,y)}, \quad I_{yy}^{(x,y)} = I_y^{(x,y)} * I_y^{(x,y)}$$

3. Performs averaging of the products of gradients over a rectangular neighborhood block at each pixel of the image. The block size is specified by the `avgWndSize` value.

$$S_{xx}^{(x,y)} = \sum_{y'} \sum_{x'} I_{xx}^{(x',y')}, \; S_{xy}^{(x,y)} = \sum_{y'} \sum_{x'} I_{xy}^{(x',y')}, \; S_{yy}^{(x,y)} = \sum_{y'} \sum_{x'} I_{yy}^{(x',y')}$$

4. Defines 2x2 gradient covariance matrix `H(``x`, y)` over a rectangular neighborhood block for each (`x`, `y`) pixel of the image.

$$H^{(x,y)} = \begin{pmatrix} S_{xx}^{(x,y)} & S_{xy}^{(x,y)} \\ S_{yx}^{(x,y)} & S_{yy}^{(x,y)} \end{pmatrix}$$

5. Computes the corner response at each pixel of the image:

$$dst(x,y) = \det H^{(x,y)} - k * (\mathrm{tr} H^{(x,y)})^2$$

where

`k` is the Harris detector free parameter

The first and third stages of the function algorithm are filtering operations; they use border processing approach that is specified by the `borderType` parameter.

The `scale` parameter is applied to the output image.

Before using this function, compute the size of the temporary work buffer using the HarrisCornerGetBufferSize function.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error when `pSrc`, `pDst`, or `pBufferSize` is `NULL`.

ippStsSizeErr	Indicates an error in the following cases: • when roiSize is less than, or equal to zero • when avgWndSize is equal to zero
ippStsNotEvenStepErr	Indicates an error when one of the step values is not divisible by 4 for floating point images.
ippStsFilterTypeErr	Indicates an error when filterType has an illegal value.
ippStsMaskSizeErr	Indicates an error when filterMask has an illegal value.
ippStsBorderErr	Indicates an error when borderType has an illegal value.
ippStsStepErr	Indicates an error when srcStep or dstStep has a negative value.
ippStsInplaceModeNotSupportedErr	Indicates an error when pSrc and pDst point to the same image.

Example

The code example below demonstrates how to use the ippiHarrisCorner_8u32f_C1R and ippiHarrisCornerGetBufferSize functions.

```
...
int    bufSize = 0;
Ipp8u* pBuffer  = 0;
Ipp32u numChannels = 1;
IppStatus status = ippStsNoErr;
IppiBorderType borderType = ippBorderRepl;
IppiDifferentialKernel filterType = ippFilterSobel;
IppiMaskSize filterMask = ippMskSize5x5;
Ipp32u avgWndSize = 3;
Ipp32f scale = 1.0f;


/* Computes the temporary work buffer size */
status = ippiHarrisCornerGetBufferSize(roiSize, filterMask, avgWndSize, ipp8u, numChannels,
&bufSize);


/* Memory allocation */
if (status != ippStsNoErr) pBuffer = ippsMalloc_8u(bufSize);


if (pBuffer != NULL)
{
    /* Harris Corner processing */
    status = ippiHarrisCorner_8u32f_C1R(pSrc, srcStep, pDst, dstStep, roiSize, filterType,
filterMask, avgWndSize, 0.04f,
        scale, borderType, 0, pBuffer);
```

```
    ippsFree(pBuffer);
}
...
```

See Also

HarrisCornerGetBufferSize Calculates the size of the temporary buffer for the ippiHarrisCorner function.

Canny Edge Detector

This subsection describes a classic edge detector proposed by J.Canny, see [Canny86]. The detector uses a grayscale image as an input and outputs a black-and-white image, where non-zero pixels mark detected edges. The algorithm consists of three stages described below.

Stage 1: Differentiation

The image data is differentiated in `x` and `y` directions. From the computed `x` and `y` gradient components, Canny edge detector functions calculate the magnitude and angle of the gradient vector.

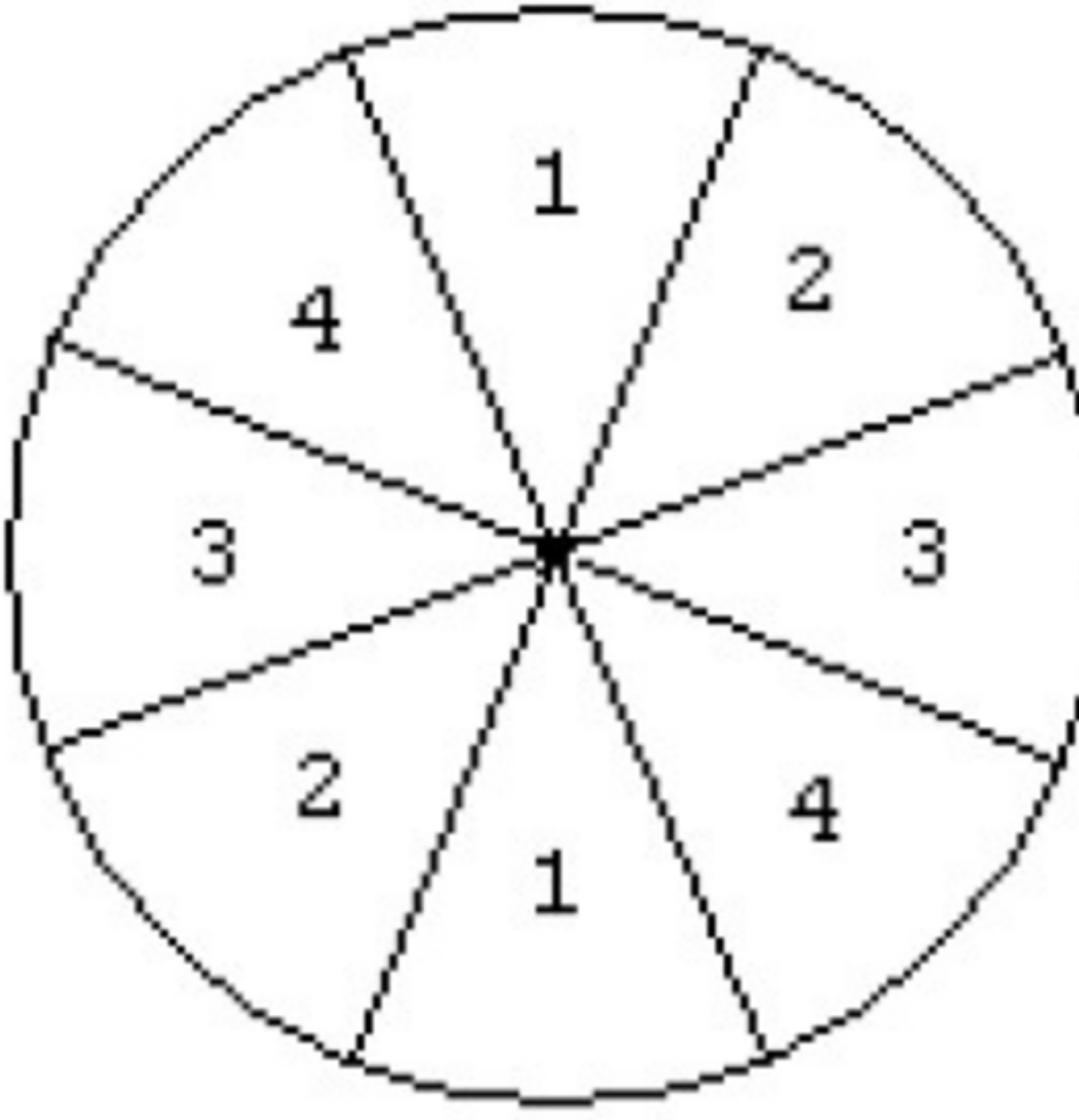
NOTE The `ippiSobel` functions perform the first stage and Canny edge detector functions use their output.

Stage 2: Non-Maximum Suppression

With the rate of intensity change found at each point in the image, edges must be placed at the points of maximum values of gradient magnitude. It is preferable to suppress non-maximum points that are perpendicular to the edge direction, rather than parallel to the edge direction, as the edge is strong along an extended contour.

The algorithm starts off with sorting the direction of gradient to one of four sectors shown in the figure below.

Gradient Sectors



The algorithm passes a 3 × 3 neighborhood across the magnitude array. At each point, the center element of the neighborhood is compared with its two neighbors along the line of the gradient given by the sector value. If the central value is not greater than the neighbors, it is suppressed.

Stage 3: Edge Thresholding

The Canny operator uses the so-called “hysteresis” thresholding. Most thresholders use a single threshold limit, which means that if the edge values fluctuate above and below this value, the line appears broken. This phenomenon is commonly referred to as “streaking”. Hysteresis counters streaking by setting an upper and lower edge value limit. Considering a line segment, if a value lies above the upper threshold limit, it is immediately accepted. If the value lies below the low threshold, it is rejected. Points which lie between the two limits are accepted if they are connected to pixels which are also accepted. The likelihood of streaking is small, since the line segment points must fluctuate above the upper limit and below the lower limit for streaking to occur.

J.Canny recommends the ratio of high to low limit be in the range two or three to one, based on predicted signal-to-noise ratios.

Example shows how to use the Intel IPP functions for the Canny edge detector.

- CannyBorderGetSize
- CannyBorder
- CannyGetSize
- Canny
- EigenValsVecsGetBufferSize
- EigenValsVecsBorder
- EigenValsVecs
- MinEigenValGetBufferSize
- MinEigenValBorder
- MinEigenVal

CannyBorderGetSize

Calculates the size of the temporary buffer for the `ippiCannyBorder` function.

Syntax

```
IppStatus ippiCannyBorderGetSize(IppiSize roiSize, IppiDifferentialKernel filterType,
IppiMaskSize mask, IppDataType dataType, int* pBufferSize);
```

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`roiSize`	Size of the image ROI in pixels.
`filterType`	Type of the filter to be applied. Possible values are `ippFilterSobel` and `ippFilterScharr`.
`mask`	The size of the mask. Possible values are `ippMskSize3x3` and `ippMskSize5x5`.
`dataType`	Data type of the image. Possible value is `ipp8u`.
`pBufferSize`	Pointer to the variable that returns the size of the temporary buffer.

Description

This function calculates the size of the temporary buffer needed for the CannyBorder function.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when the pBufferSize pointer is NULL.
ippStsMaskSizeErr	Indicates an error when mask has an illegal value.
ippStsDataTypeErr	Indicates an error when dataType has an illegal value.
ippStsSizeErr	Indicates an error when roiSize has a field with a zero or negative value.

See Also

CannyBorder Implements Canny algorithm for edge detection.

CannyBorder

Implements Canny algorithm for edge detection.

Syntax

```
IppStatus ippiCannyBorder_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roiSize, IppiDifferentialKernel filterType, IppiMaskSize mask, IppiBorderType borderType, Ipp8u borderValue, Ipp32f lowThresh, Ipp32f highThresh, IppNormType norm, Ipp8u* pBuffer);
```

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Pointer to the source image.
srcStep	Distance, in bytes, between the starting points of consecutive lines in the source image.
pDst	Pointer to the destination image.
dstStep	Distance, in bytes, between the starting points of consecutive lines in the destination image.
roiSize	Size of the source image ROI in pixels.
filterType	Type of the filter to be applied. Possible values are ippFilterSobel and ippFilterScharr.

mask	The size of the mask. Possible values are ippMskSize3x3 and ippMskSize5x5 for ippFilterSobel, and ippMskSize3x3 for ippFilterScharr.
borderType	Type of border. Possible values are:

ippBorderConst	Values of all border pixels are set to constant.
ippBorderRepl	Border is replicated from the edge pixels.
ippBorderInMem	Border is obtained from the source image pixels in memory.

Mixed borders are also supported. They can be obtained by the bitwise operation OR between ippBorderRepl and ippBorderInMemTop, ippBorderInMemBottom, ippBorderInMemLeft, ippBorderInMemRight.

borderValue	Constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.
lowThresh	Lower threshold for edges detection.
highThresh	Upper threshold for edges detection.
norm	Normalization mode. Possible values are ippNormL1 and ippNormL2.
pBuffer	Pointer to the pre-allocated temporary buffer. To calculate the size of the temporary buffer, use the CannyBorderGetSize function.

Description

This function operates with ROI (see Regions of Interest in Intel IPP). This function finds edges in the ROI of the source image with the user-defined border types using the Canny edge detector algorithm. The output image is stored in pDst.

Before using this function, compute the size of the temporary work buffer using the CannyBorderGetSize function.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsMaskSizeErr	Indicates an error when mask has an illegal value.
ippStsSizeErr	Indicates an error when roiSize has a field with a zero or negative value.
ippStsStepErr	Indicates an error when srcStep or dstStep is less than roi.width*<pixelSize>.
ippStsBadArgErr	Indicates an error when lowThresh is negative, or highThresh is less than lowThresh.

ippStsNotEvenStepErr	Indicates an error when one of the step values is not divisible by 2 for 16s images, and by 4 for 32f images.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

// A simple example of edge detection algorithm implemented with Intel(R) Integrated Performance
Primitives (Intel(R) IPP) functions:
//     ippiCannyBorderGetSize
//     ippiCannyBorder_8u_C1R

#include <stdio.h>
#include "ipp.h"

#define WIDTH        128   /* image width */
#define HEIGHT       64    /* image height */
#define THRESH_LOW   50.f  /* Low threshold for edges detection */
#define THRESH_HIGHT 150.f /* Upper threshold for edges detection */
#define BORDER_VAL   0

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define check_sts(st)        \
    if ((st) != ippStsNoErr) \
        goto exitLine; /* Go to Exit if Intel(R) IPP function returned status different from
ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main(void)
{
    IppStatus status = ippStsNoErr;
    Ipp8u *pSrc = NULL, *pDst = NULL;   /* Pointers to source/destination images */
    int srcStep = 0, dstStep = 0;       /* Steps, in bytes, through the source/destination
images */
    IppiSize roiSize = {WIDTH, HEIGHT}; /* Size of source/destination ROI in pixels */
    Ipp8u *pBuffer = NULL;              /* Pointer to the work buffer */
    int iTmpBufSize = 0;                /* Common work buffer size */
    IppiDifferentialKernel filterType = ippFilterScharr;
```

```
    pSrc = ippiMalloc_8u_C1(roiSize.width, roiSize.height, &srcStep);
    pDst = ippiMalloc_8u_C1(roiSize.width, roiSize.height, &dstStep);

    /* Start Edge Detection algorithm */
    check_sts(status = ippiCannyBorderGetSize(roiSize, filterType, ippMskSize3x3, ipp8u,
&iTmpBufSize));

    pBuffer = ippsMalloc_8u(iTmpBufSize);

    check_sts(status = ippiCannyBorder_8u_C1R(pSrc, srcStep, pDst, dstStep, roiSize, filterType,
ippMskSize3x3, ippBorderRepl, BORDER_VAL, THRESH_LOW,
                                              THRESH_HIGHT, ippNormL2, pBuffer));

    EXIT_MAIN
    ippsFree(pBuffer);
    ippiFree(pSrc);
    ippiFree(pDst);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

See Also

CannyBorderGetSize Calculates the size of the temporary buffer for the ippiCannyBorder function.

CannyGetSize

Calculates size of temporary buffer for the `ippiCanny` function.

Syntax

```
IppStatus ippiCannyGetSize(IppiSize roiSize, int* pBufferSize);
```

```
IppStatus ippiCannyGetSize_L(IppiSizeL roi, IppSizeL* bufferSize);
```

Include Files

`ippcv.h`

Flavors with the `_L` suffix: `ippcv_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`roiSize`, `roi`	Size of the image ROI, in pixels.
`pBufferSize`, `bufferSize`	Pointer to the computed size of the temporary buffer.

Description

This function calculates the size of a temporary buffer for the ippiCanny function.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if pBufferSize pointer is NULL.
ippStsSizeErr	Indicates an error condition if pRoiSize has a field with zero or negative value.

Canny

Implements Canny algorithm for edge detection.

Syntax

```
IppStatus ippiCanny_16s8u_C1R(Ipp16s* pSrcDx, int srcDxStep, Ipp16s* pSrcDy, int srcDyStep, Ipp8u* pDstEdges, int dstEdgeStep, IppiSize roiSize, Ipp32f lowThreshold, Ipp32f highThreshold, Ipp8u* pBuffer);
```

```
IppStatus ippiCanny_32f8u_C1R(Ipp32f* pSrcDx, int srcDxStep, Ipp32f* pSrcDy, int srcDyStep, Ipp8u* pDstEdges, int dstEdgeStep, IppiSize roiSize, Ipp32f lowThreshold, Ipp32f highThreshold, Ipp8u* pBuffer);
```

Platform-aware functions

```
IppStatus ippiCanny_16s8u_C1R_L(Ipp16s* pSrcDx, IppSizeL srcDxStep, Ipp16s* pSrcDy, IppSizeL srcDyStep, Ipp8u* pDstEdges, IppSizeL dstEdgeStep, IppiSizeL roiSize, Ipp32f lowThreshold, Ipp32f highThreshold, IppNormType norm, Ipp8u* pBuffer);
```

```
IppStatus ippiCanny_32f8u_C1R_L(Ipp32f* pSrcDx, IppSizeL srcDxStep, Ipp32f* pSrcDy, IppSizeL srcDyStep, Ipp8u* pDstEdges, IppSizeL dstEdgeStep, IppiSizeL roiSize, Ipp32f lowThreshold, Ipp32f highThreshold, IppNormType norm, Ipp8u* pBuffer);
```

Include Files

ippcv.h

Flavors with the _L suffix: ippcv_l.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrcDx	Pointer to the source image ROI x - derivative.
srcDxStep	Distance in bytes between starts of consecutive lines in the source image pSrcDx.
pSrcDy	Pointer to the source image ROI y - derivative.
srcDyStep	Distance, in bytes, between the starting points of consecutive lines in the source image pSrcDy.
pDstEdges	Pointer to the output array of the detected edges.

dstEdgeStep	Distance, in bytes, between the starting points of consecutive lines in the output image.
roiSize	Size of the source image ROI in pixels.
lowThreshold	Lower threshold for edges detection.
highThreshold	Upper threshold for edges detection.
norm	Normalization type; supported values: ippNormL1 and ippNormL2.
pBuffer	Pointer to the pre-allocated temporary buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP). This function finds edges in the source image ROI and stores them into the output image pDstEdges using the Canny algorithm. The function requires a temporary working buffer; its size should be computed previously by calling the function ippiCannyGetSize.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if pRoiSize has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if srcDxStep, srcDyStep or dstEdgeStep is less than roi.width*<pixelSize>.
ippStsBadArgErr	Indicates an error when lowThresh is negative or highThresh is less than lowThresh.
ippStsNotEvenStepErr	Indicates an error condition if one of the step values is not divisible by 2 for 16s images, and by 4 for 32f images.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
```

```
 *****************************************************************************/

// A simple example of the edge detection algorithm implemented with Intel(R) Integrated
Performance Primitives (Intel(R) IPP) functions:
//     ippiFilterSobelNegVertBorder_8u16s_C1R
//     ippiFilterSobelHorizBorder_8u16s_C1R
//     ippiCanny_16s8u_C1R

#include <stdio.h>
#include "ipp.h"

#define WIDTH        128   /* image width */
#define HEIGHT       64    /* image height */
#define THRESH_LOW   50.f  /* Low   threshold for edges detection */
#define THRESH_HIGHT 150.f /* Upper threshold for edges detection */
#define BORDER_VAL   0

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define check_sts(st)        \
    if ((st) != ippStsNoErr) \
        goto exitLine; /* Go to Exit if Intel(R) IPP function returned status different from
ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main(void)
{
    IppStatus status = ippStsNoErr;
    Ipp8u *pSrc = NULL, *pDst = NULL;   /* Pointers to source/destination images */
    int srcStep = 0, dstStep = 0;       /* Steps, in bytes, through the source/destination
images */
    IppiSize roiSize = {WIDTH, HEIGHT}; /* Size of source/destination ROI in pixels */

    int iTmpBufSizeSobV = 0;
    int iTmpBufSizeSobH = 0;
    int iTmpBufSizeCanny = 0;
    int iTmpBufSize = 0;                        /* Common work buffer size */
    int dxStep = 0, dyStep = 0;                 /* Steps, in bytes, through buffer_dx/buffer_dy
*/
    Ipp8u *buffer = NULL;                       /* Pointer to the common work buffer */
    Ipp16s *buffer_dx = NULL, *buffer_dy = NULL; /* Pointer to the buffer for first derivatives
with respect to X / Y */

    pSrc = ippiMalloc_8u_C1(roiSize.width, roiSize.height, &srcStep);
    pDst = ippiMalloc_8u_C1(roiSize.width, roiSize.height, &dstStep);

    /* Start Edge Detection algorithm */
    {
        buffer_dx = ippiMalloc_16s_C1(roiSize.width, roiSize.height, &dxStep);
        buffer_dy = ippiMalloc_16s_C1(roiSize.width, roiSize.height, &dyStep);

        check_sts(status = ippiFilterSobelVertBorderGetBufferSize(roiSize, ippMskSize3x3, ipp8u,
ipp16s, 1, &iTmpBufSizeSobV));
        check_sts(status = ippiFilterSobelHorizBorderGetBufferSize(roiSize, ippMskSize3x3,
ipp8u, ipp16s, 1, &iTmpBufSizeSobH));
```

```
        check_sts(ippiCannyGetSize(roiSize, &iTmpBufSizeCanny));

        /* Find maximum buffer size */
        iTmpBufSize = (iTmpBufSizeSobV > iTmpBufSizeSobH) ? iTmpBufSizeSobV : iTmpBufSizeSobH;
        iTmpBufSize = (iTmpBufSize > iTmpBufSizeCanny) ? iTmpBufSize : iTmpBufSizeCanny;
        buffer = ippsMalloc_8u(iTmpBufSize);

        check_sts(status = ippiFilterSobelNegVertBorder_8u16s_C1R(pSrc, srcStep, buffer_dx,
dxStep, roiSize, ippMskSize3x3, ippBorderRepl, BORDER_VAL,
                                                                  buffer));

        check_sts(status = ippiFilterSobelHorizBorder_8u16s_C1R(pSrc, srcStep, buffer_dy,
dyStep, roiSize, ippMskSize3x3, ippBorderRepl, BORDER_VAL,
                                                                buffer));

        check_sts(status = ippiCanny_16s8u_C1R(buffer_dx, dxStep, buffer_dy, dyStep, pDst,
dstStep, roiSize, THRESH_LOW, THRESH_HIGHT, buffer));
    }

    EXIT_MAIN
    ippsFree(buffer);
    ippiFree(buffer_dx);
    ippiFree(buffer_dy);
    ippiFree(pSrc);
    ippiFree(pDst);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

EigenValsVecsGetBufferSize

Calculates size of temporary buffer for the function `ippiEigenValsVecs`.

Syntax

```
IppStatus ippiEigenValsVecsGetBufferSize_32f_C1R(IppiSize roiSize,int apertureSize, int
avgWindow, int* pBufferSize);
```

```
IppStatus ippiEigenValsVecsGetBufferSize_8u32f_C1R(IppiSize roiSize, int apertureSize,
int avgWindow, int* pBufferSize);
```

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`roiSize`	Size of the source image ROI in pixels.

apertureSize	Size (pixels) of the derivative operator used by the function, possible values are 3 or 5.
avgWindow	Size of the blurring window in pixels, possible values are 3 or 5.
pBufferSize	Pointer to the variable that returns the size of the temporary buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function calculates the size of a temporary buffer to be used by the function ippiEigenValsVecs.

NOTE
CAUTION
The parameters apertureSize and avgWindow must be the same for both functions ippiEigenValsVecsGetBufferSize and ippiEigenValsVecs.

Example "Using the function ippiEigenValsVecs" shows how to use the function ippiEigenValsVecsGetBufferSizee.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiWidth has zero or negative value, or if apertureSize or avgWindow has an illegal value.

EigenValsVecsBorder

Calculates eigen values and eigen vectors of image blocks for corner detection.

Syntax

```
IppStatus ippiEigenValsVecsBorder_8u32f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32f* pEigenVV, int eigStep, IppiSize roiSize, IppiKernelType kernType, int apertureSize, int avgWindow, IppiBorderType borderType, Ipp8u borderValue, Ipp8u* pBuffer);
```

```
IppStatus ippiEigenValsVecsBorder_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pEigenVV, int eigStep, IppiSize roiSize, IppiKernelType kernType, int apertureSize, int avgWindow, IppiBorderType borderType, Ipp32f borderValue, Ipp8u* pBuffer);
```

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance, in bytes, between the starting points of consecutive lines in the source image.
pEigenVV	Image to store the results.
eigStep	Distance, in bytes, between the starting points of consecutive lines in the output image.
roiSize	Size of the source image ROI, in pixels.
kernType	Specifies the type of kernel used to compute derivatives, possible values are:

ippKernelSobel	Sobel kernel 3x3 or 5x5
ippKernelSobelNeg	Negative Sobel kernel 3x3 or 5x5
ippKernelScharr	Scharr kernel 3x3

apertureSize	Size of the derivative operator in pixels, possible values are 3 or 5.
border	Type of image border. Possible values:

ippBorderConst	Values of all border pixels are set to a constant.
ippBorderRepl	Border is replicated from the edge pixels.

borderValue	Constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.
avgWindow	Size of the blurring window in pixels, possible values are 3 or 5.
pBuffer	Pointer to the temporary buffer.

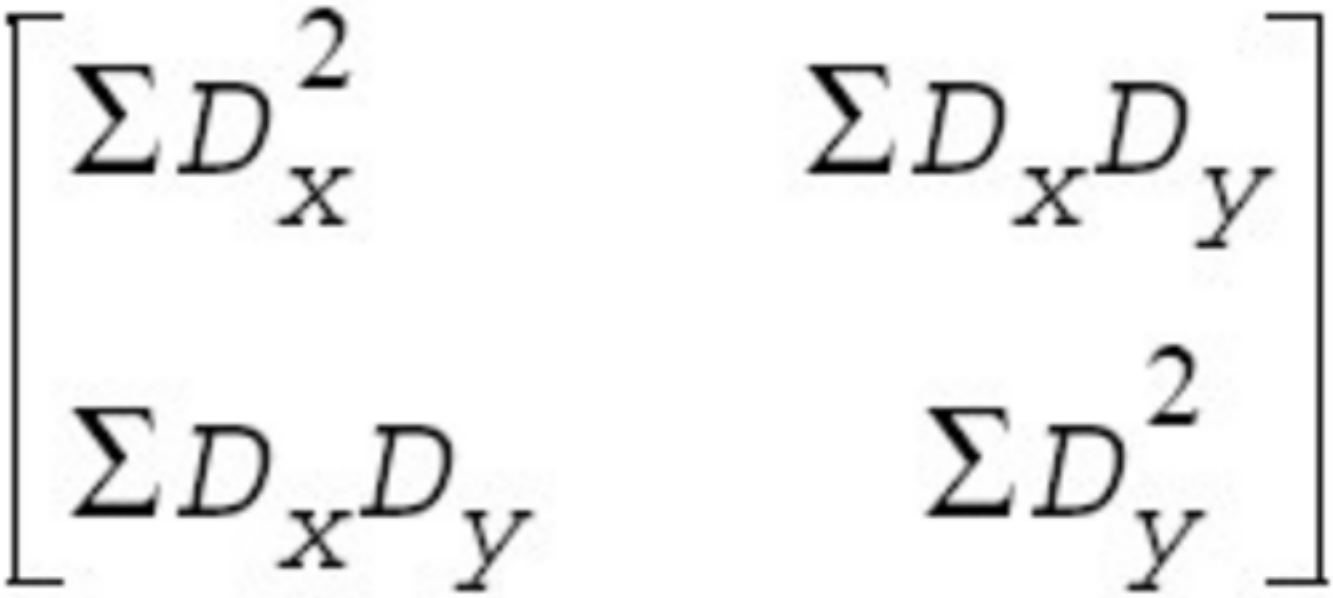
Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function takes a block around the pixel and computes the first derivatives D_x and D_y. This operation is performed for every pixel of the image using either Sobel or Scharr kernel in accordance with the kernType parameter. The apertureSize parameter specifies the size of the Sobel kernel. If this parameter is set to 3, the function used 3x3 kernel, if it is set to 5, the function uses 5x5 kernel. Only 3x3 size is available for the Scharr kernel, therefore the parameter apertureSize must be set to 3 if the Scharr kernel is used.

NOTE
CAUTION
If the parameter apertureSize is set to 5 for operation with the Scharr kernel, the function returns error status.

The function computes eigen values and vectors of the following matrix:



The summation is performed over the full block with averaging over the blurring window with size `avgWindow`.

The image `eigenVV` has the following format. For every pixel of the source image it contains six floating-point values - λ1, λ2 , x1, y1, x2, y2. These values are defined as follows:

λ1, λ2	Eigen values of the above matrix (λ1 ≥λ2 ≥ 0).
x1, y1	Coordinates of the normalized eigen vector corresponding to λ1.
x2, y2	Coordinates of the normalized eigen vector corresponding to λ2.

In case of a singular matrix or when one eigen value is much smaller than the second one, all these six values are set to 0.

The function requires a temporary work buffer. Before using this function, compute the size of the buffer using the ippiEigenValsVecsGetBufferSize function.

NOTE
CAUTION
The parameters `apertureSize` and `avgWindow` must be the same for both functions `ippiEigenValsVecsGetBufferSize` and `ippiEigenValsVecsBorder`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.

ippStsSizeErr	Indicates an error condition if pRoiSize has a field with zero or negative value, or if apertureSize or avgWindow has an illegal value; or if kernType has a wrong value.
ippStsStepErr	Indicates an error condition if srcStep is less than roiSize.width*<``pixelSize>, or ``eigStep is less than roiSize.width*sizeof(Ipp32f)*6.
ippStsNotEvenStepErr	Indicates an error condition if steps for floating-point images are not divisible by 4.
ippStsBorderErr	Indicates an error if borderType has an illegal value.

See Also

Regions of Interest in Intel IPP

EigenValsVecsGetBufferSize Calculates size of temporary buffer for the function ippiEigenValsVecs.

EigenValsVecs

Calculates eigen values and eigen vectors of image blocks for corner detection.

Syntax

```
IppStatus ippiEigenValsVecs_8u32f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32f* pEigenVV, int eigStep, IppiSize roiSize, IppiKernelType kernType, int apertureSize, int avgWindow, Ipp8u* pBuffer);
```

```
IppStatus ippiEigenValsVecs_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f* pEigenVV, int eigStep, IppiSize roiSize, IppiKernelType kernType, int apertureSize, int avgWindow, Ipp8u* pBuffer);
```

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance, in bytes, between the starting points of consecutive lines in the source image.
pEigenVV	Image to store the results.
eigStep	Distance, in bytes, between the starting points of consecutive lines in the output image.
roiSize	Size of the source image ROI, in pixels.

kernType	Specifies the type of kernel used to compute derivatives, possible values are:

ippKernelSobel	Sobel kernel 3x3 or 5x5
ippKernelSobelNeg	Negative Sobel kernel 3x3 or 5x5
ippKernelScharr	Scharr kernel 3x3

apertureSize	Size of the derivative operator in pixels, possible values are 3 or 5.
avgWindow	Size of the blurring window in pixels, possible values are 3 or 5.
pBuffer	Pointer to the temporary buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function takes a block around the pixel and computes the first derivatives D_x and D_y. This operation is performed for every pixel of the image using either Sobel or Scharr kernel in accordance with the kernType parameter. The apertureSize parameter specifies the size of the Sobel kernel. If this parameter is set to 3, the function used 3x3 kernel, if it is set to 5, the function uses 5x5 kernel. Only 3x3 size is available for the Scharr kernel, therefore the parameter apertureSize must be set to 3 if the Scharr kernel is used.

NOTE
CAUTION
If the parameter apertureSize is set to 5 for operation with the Scharr kernel, the function returns error status.

The function computes eigen values and vectors of the following matrix:

$$\begin{bmatrix} \sum D_x^2 & \sum D_x D_y \\ \sum D_x D_y & \sum D_y^2 \end{bmatrix}$$

The summation is performed over the full block with averaging over the blurring window with size `avgWindow`.

The image `eigenVV` has the following format. For every pixel of the source image it contains six floating-point values - λ1, λ2 , x1, y1, x2, y2. These values are defined as follows:

λ1, λ2	Eigen values of the above matrix (λ1 ≥λ2 ≥ 0).
x1, y1	Coordinates of the normalized eigen vector corresponding to λ1.
x2, y2	Coordinates of the normalized eigen vector corresponding to λ2.

In case of a singular matrix or when one eigen value is much smaller than the second one, all these six values are set to 0.

The function requires a temporary working buffer; its size should be computed previously by calling the function ippiEigenValsVecsGetBufferSize.

NOTE
CAUTION
The parameters `apertureSize` and `avgWindow` must be the same for both functions `ippiEigenValsVecsGetBufferSize` and `ippiEigenValsVecs`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `pRoiSize` has a field with zero or negative value, or if `apertureSize` or `avgWindow` has an illegal value; or if `kernType` has wrong value.
`ippStsStepErr`	Indicates an error condition if `srcStep` is less than `roiSize.width*<``pixelSize>, or ``eigStep` is less than `roiSize.width*sizeof(Ipp32f)*6`.
`ippStsNotEvenStepErr`	Indicates an error condition if steps for floating-point images are not divisible by 4.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
```

```
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *****************************************************************************/

// A simple example of calculating eigen values and eigen vectors
// using Intel(R) Integrated Performance Primitives (Intel(R) IPP) functions:
//     ippiEigenValsVecsGetBufferSize_8u32f_C1R
//     ippiEigenValsVecs_8u32f_C1R

#include <stdio.h>
#include "ipp.h"

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define check_sts(st)        \
    if ((st) != ippStsNoErr) \
        goto exitLine; /* Go to Exit if Intel(R) IPP function returned status different from
ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main(void)
{
    IppStatus status = ippStsNoErr;
    Ipp8u pSrc[4 * 3] = /* Pointers to source images */
        {2, 17, 2, 21, 9, 4, 11, 5, 2, 32, 7, 2};
    Ipp32f pEigenVV[24 * 3];   /* Image of eigen values and eigen vectors */
    IppiSize roiSize = {4, 3}; /* Size of source ROI in pixels */
    int apertureSize = 3;      /* Linear size of derivative filter aperture */
    int avgWindow = 3;         /* Linear size of averaging window */
    int pBufferSize = 0;       /* Pointer to work buffer */
    Ipp8u *pBuffer = NULL;

    /* Calculate size of temporary buffer */
    check_sts(status = ippiEigenValsVecsGetBufferSize_8u32f_C1R(roiSize, apertureSize,
avgWindow, &pBufferSize));

    pBuffer = ippsMalloc_8u(pBufferSize);

    /*  Calculate both eigen values and eigen vectors of 2x2 autocorrelation */
    /*  Gradient matrix for every pixel. */
    check_sts(status =
                  ippiEigenValsVecs_8u32f_C1R(pSrc, 4, pEigenVV, 24 * sizeof(Ipp32f), roiSize,
ippKernelSobel, apertureSize, avgWindow, pBuffer));

    EXIT_MAIN
    ippsFree(pBuffer);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

MinEigenValGetBufferSize

Calculates size of temporary buffer for the function `ippiMinEigenVal`.

Syntax

```
IppStatus ippiMinEigenValGetBufferSize_32f_C1R(IppiSize roiSize, int apertureSize, int avgWindow, int* pBufferSize);
```

```
IppStatus ippiMinEigenValGetBufferSize_8u32f_C1R(IppiSize roiSize, int apertureSize, int avgWindow, int* pBufferSize);
```

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`roiSize`	Size of the source image ROI in pixels.
`apertureSize`	Size (in pixels) of the derivative operator used by the function, possible values are 3 or 5.
`avgWindow`	Size of the blurring window in pixels, possible values are 3 or 5.
`pBufferSize`	Pointer to the variable that returns the size of the temporary buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP). This function calculates the size of a temporary buffer to be used by the function ippiMinEigenVal.

NOTE
CAUTION
The parameters `apertureSize` and `avgWindow` must be the same for both functions `ippiMinEigenValGetBufferSize` and `ippiMinEigenVal`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiWidth` has a zero or negative value, or if `apertureSize` or `avgWindow` has an illegal value.

MinEigenValBorder

Calculates the minimal eigen value of image blocks for corner detection.

Syntax

```
IppStatus ippiMinEigenValBorder_8u32f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32f*
pMinEigenVal, int minValStep, IppiSize roiSize, IppiKernelType kernType, int
apertureSize, int avgWindow, IppiBorderType borderType, Ipp8u borderValue, Ipp8u*
pBuffer);
```

```
IppStatus ippiMinEigenValBorder_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f*
pMinEigenVal, int minValStep, IppiSize roiSize, IppiKernelType kernType, int
apertureSize, int avgWindow, IppiBorderType borderType, Ipp32f borderValue, Ipp8u*
pBuffer);
```

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance, in bytes, between the starting points of consecutive lines in the source image.
`pMinEigenVal`	Pointer to the image that is filled with the minimal eigen values.
`minValStep`	Distance, in bytes, between the starting points of consecutive lines in the output image.
`roiSize`	Size of the source image ROI in pixels.
`kernType`	Specifies the type of kernel used to compute derivatives, possible values are:

ippKernelSobel	Sobel kernel 3x3 or 5x5
ippKernelSobelNeg	Negative Sobel kernel 3x3 or 5x5
`ippKernelScharr`	Scharr kernel 3x3

`apertureSize`	Size of the derivative operator, in pixels; possible values are 3 or 5.
`avgWindow`	Size of the averaging window, in pixels; possible values are 3 or 5.
`borderType`	Type of image border. Possible values:

ippBorderConst	Values of all border pixels are set to a constant.
ippBorderRepl	Border is replicated from the edge pixels.

ippBorderMirror	Border pixels are mirrored from the source image boundary pixels.

borderValue	Constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.
pBuffer	Pointer to the temporary buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP). This function takes a block around the pixel and computes the first derivatives D_x and D_y. This operation is performed for every pixel of the image using either Sobel or Scharr kernel in accordance with the kernType parameter. The apertureSize parameter specifies the size of the Sobel kernel. If apertureSize is set to 3, the function uses 3x3 kernel, if it is set to 5, the function uses 5x5 kernel. Only 3x3 size is available for the Scharr kernel, therefore the parameter apertureSize must be set to 3 if the Scharr kernel is used.

NOTE
CAUTION
If the parameter apertureSize is set to 5 for operation with the Scharr kernel, the function returns error status.

The function computes the minimal eigen value λ (λ≥ 0) of the following matrix:

$$\begin{bmatrix} \Sigma D_x^2 & \Sigma D_x D_y \\ \Sigma D_x D_y & \Sigma D_y^2 \end{bmatrix}$$

The summation is performed over the full block with averaging over the blurring window with size avgWindow.

The function requires a temporary work buffer. Before using this function, compute the size of the work buffer using the ippiMinEigenValGetBufferSize function.

NOTE
CAUTION
The parameters apertureSize and avgWindow must be the same for both functions ippiMinEigenValGetBufferSize and ippiMinEigenValBorder.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.

ippStsSizeErr	Indicates an error condition if pRoiSize has a field with zero or negative value, or if apertureSize or avgWindow has an illegal value; or if kernType has wrong value.
ippStsStepErr	Indicates an error condition if srcStep is less than roiSize.width*<``pixelSize>, or ``eigenvvStep is less than roiSize.width*sizeof(Ipp32f).
ippStsNotEvenStepErr	Indicates an error condition if steps for floating-point images are not divisible by 4.
ippStsBorderErr	Indicates an error if borderType has an illegal value.

See Also

Regions of Interest in Intel IPP

MinEigenValGetBufferSize Calculates size of temporary buffer for the function ippiMinEigenVal.

MinEigenVal

Calculates the minimal eigen value of image blocks for corner detection.

Syntax

```
IppStatus ippiMinEigenVal_8u32f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32f*
pMinEigenVal, int minValStep, IppiSize roiSize, IppiKernelType kernType, int
apertureSize, int avgWindow, Ipp8u* pBuffer);
```

```
IppStatus ippiMinEigenVal_32f_C1R(const Ipp32f* pSrc, int srcStep, Ipp32f*
pMinEigenVal, int minValStep, IppiSize roiSize, IppiKernelType kernType, int
apertureSize, int avgWindow, Ipp8u* pBuffer);
```

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pMinEigenVal	Pointer to the image that is filled with the minimal eigen values.
minValStep	Distance in bytes between starts of consecutive lines in the output image.
roiSize	Size of the source image ROI in pixels.

kernType	Specifies the type of kernel used to compute derivatives, possible values are:

ippKernelSobel	Sobel kernel 3x3 or 5x5
ippKernelSobelNeg	Negative Sobel kernel 3x3 or 5x5
ippKernelScharr	Scharr kernel 3x3

apertureSize	Size of the derivative operator in pixels, possible values are 3 or 5.
avgWindow	Size of the averaging window in pixels, possible values are 3 or 5.
pBuffer	Pointer to the temporary buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP). This function takes a block around the pixel and computes the first derivatives D_x and D_y. When using the MinEigenVal function, you need to verify that the processing pixels beyond the ROI are available in memory. This operation is performed for every pixel of the image using either Sobel or Scharr kernel in accordance with the kernType parameter. The size of the Sobel kernel may be specified the parameter apertureSize. If this parameter is set to 3, the function used 3x3 kernel, if it is set to 5, the function uses 5x5 kernel. Only 3x3 size is available for the Scharr kernel, therefore the parameter apertureSize must be set to 3 if the Scharr kernel is used.

NOTE
CAUTION
If the parameter apertureSize is set to 5 for operation with the Scharr kernel, the function returns error status.

Then, the function computes the minimal eigen value λ (λ≥ 0) of the following matrix:

$$\begin{bmatrix} \Sigma D_x^2 & \Sigma D_x D_y \\ \Sigma D_x D_y & \Sigma D_y^2 \end{bmatrix}$$

The summation is performed over the full block with averaging over the blurring window with size avgWindow.

The function requires a temporary working buffer; its size should be computed previously by calling the function ippiMinEigenValGetBufferSize.

NOTE
CAUTION
The parameters apertureSize and avgWindow must be the same for both functions ippiMinEigenValGetBufferSize and ippiMinEigenVal.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if pRoiSize has a field with zero or negative value, or if apertureSize or avgWindow has an illegal value; or if kernType has wrong value.
ippStsStepErr	Indicates an error condition if srcStep is less than roiSize.width*<``pixelSize>, or ``eigenvvStep is less than roiSize.width*sizeof(Ipp32f).
ippStsNotEvenStepErr	Indicates an error condition if steps for floating-point images are not divisible by 4.

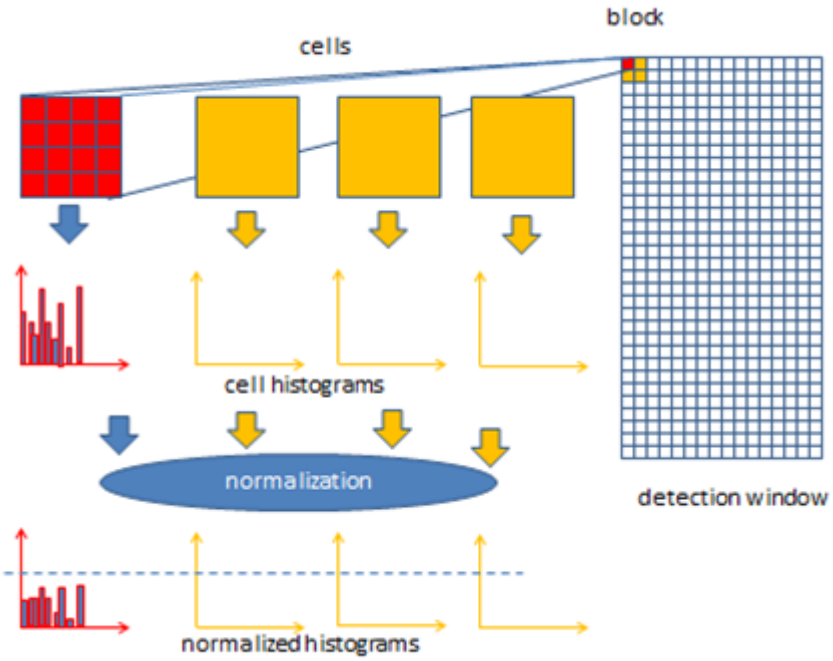
Histogram of Oriented Gradients (HOG) Descriptor

Histogram of oriented gradients (HOG) is a feature descriptor used to detect objects in computer vision and image processing. The HOG descriptor technique counts occurrences of gradient orientation in localized portions of an image - detection window, or region of interest (ROI).

Implementation of the HOG descriptor algorithm is as follows:

1. Divide the image into small connected regions called cells, and for each cell compute a histogram of gradient directions or edge orientations for the pixels within the cell.
2. Discretize each cell into angular bins according to the gradient orientation.
3. Each cell's pixel contributes weighted gradient to its corresponding angular bin.
4. Groups of adjacent cells are considered as spatial regions called blocks. The grouping of cells into a block is the basis for grouping and normalization of histograms.
5. Normalized group of histograms represents the block histogram. The set of these block histograms represents the descriptor.

The following figure demonstrates the algorithm implementation scheme:



Computation of the HOG descriptor requires the following basic configuration parameters:

- Masks to compute derivatives and gradients
- Geometry of splitting an image into cells and grouping cells into a block
- Block overlapping
- Normalization parameters

According to [Dalal05] the recommended values for the HOG parameters are:

- 1D centered derivative mask [-1, 0, +1]
- Detection window size is 64x128
- Cell size is 8x8
- Block size is 16x16 (2x2 cells)

Intel® IPP implementation does not assume any default fixed set of parameters values. The IppiHOGConfig structure defines HOG parameters used in Intel IPP functions.

There are some limitations to the values of basic configuration parameters:

```
#define IPP_HOG_MAX_CELL   (16)  /* max size of cell */
#define IPP_HOG_MAX_BLOCK  (64)  /* max size of block */
#define IPP_HOG_MAX_BINS   (16)  /* max number of bins */
```

See Also

Structures and Enumerators

- HOGGetSize
- HOGInit
- HOGGetBufferSize
- HOGGetDescriptorSize
- HOG

HOGGetSize

Computes the size of the HOG context structure.

Syntax

```
IppStatus ippiHOGGetSize(const IppiHOGConfig* pConfig, int* pHOGSpecSize);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pConfig`	Pointer to the HOG context structure.
`pHOGSpecSize`	Pointer to the size of the HOG context structure, in bytes.

Description

This function checks the parameters of the HOG configuration and computes the size, in bytes, of the HOG context structure `pHOGSpecSize`.

For an example on how to use this function, refer to the example provided with the HOG function description.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error when one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error when one of the `winSize` fields in the `pConfig` parameter has a zero or negative value.
`ippStsNotSupportedModeErr`	Indicates an error in HOG configuration: • `cellSize` is less than 2, or more than `IPP_HOG_MAX_CELL` • `cellSize` is more than `blockSize`, or `blockSize` is more than `IPP_HOG_MAX_BLOCK` • `blockSize` is not a multiple of `cellSize` • Block does not have 2x2 cell geometry • `blockStride` is not a multiple of `cellSize` • Detection window size is not a multiple of `blockStride` • `nbins` is less than 2, or more than `IPP_HOG_MAX_BINS` • `sigma` or `threshold` value is less than, or equal to zero

See Also

HOG Computes the HOG descriptor.

HOGInit

Initializes the HOG context structure.

Syntax

```
IppStatus ippiHOGInit(const IppiHOGConfig* pConfig, IppiHOGSpec* pHOGSpec);
```

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pConfig	Pointer to the HOG context structure.
pHOGSpec	Pointer to the HOG context structure.

Description

This function checks the parameters of the HOG configuration and initializes the HOG context structure.

For an example on how to use this function, refer to the example provided with the HOG function description.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when one of the winSize fields in the pConfig parameter has a zero or negative value.
ippStsNotSupportedModeErr	Indicates an error in HOG configuration: • cellSize is less than 2, or more than IPP_HOG_MAX_CELL • cellSize is more than blockSize, or blockSize is more than IPP_HOG_MAX_BLOCK • blockSize is not a multiple of cellSize • Block does not have 2x2 cell geometry • blockStride is not a multiple of cellSize • Detection window size is not a multiple of blockStride • nbins is less than 2, or more than IPP_HOG_MAX_BINS • sigma or threshold value is less than, or equal to zero

See Also

HOG Computes the HOG descriptor.

HOGGetBufferSize

Computes the size of the work buffer for the ippiHOG function.

Syntax

```
IppStatus ippiHOGGetBufferSize(const IppiHOGSpec* pHOGSpec, IppiSize roiSize, int*
pBufferSize);
```

Include Files

ippi.h

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pHOGSpec`	Pointer to the HOG context structure.
`roiSize`	Maximum size of the source image ROI, in pixels.
`pBufferSize`	Pointer to the size of the work buffer, in bytes.

Description

This function computes the size of the work buffer for the HOG function.

For an example on how to use this function, refer to the example provided with the HOG function description.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error when one of the specified pointers is `NULL`.
`ippStsContextmatchErr`	Indicates an error when the context parameter does not match the operation.

See Also

HOG Computes the HOG descriptor.

HOGGetDescriptorSize

Computes the size of the HOG descriptor.

Syntax

```
IppStatus ippiHOGGetDescriptorSize(const IppiHOGSpec* pHOGSpec, int*
pWinDescriptorSize);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pHOGSpec`	Pointer to the HOG context structure.
`pWinDescriptorSize`	Pointer to the size of the descriptor window, in bytes.

Description

This function computes the size of the buffer for a single detection window. If the subsequent call(s) of the HOG function target processing of multiple detection windows, the size of the buffer must be increased proportionally.

For an example on how to use this function, refer to the example provided with the HOG function description.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error.
`ippStsNullPtrErr`	Indicates an error when one of the specified pointers is `NULL`.
`ippStsContextmatchErr`	Indicates an error when the context parameter does not match the operation.

See Also

HOG Computes the HOG descriptor.

HOG

Computes the HOG descriptor.

Syntax

```
IppStatus ippiHOG_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, IppiSize roiSize,
const IppiPoint* pLocation, int nLocations, Ipp32f* pDst, const IppiHOGSpec* pHOGSpec,
IppiBorderType borderID, Ipp<srcDatatype> borderValue, Ipp8u* pBuffer);
```

Supported values for `mod`:

`8u32f_C1R`	`16u32f_C1R`	`16s32f_C1R`	`32f_C1R`

```
IppStatus ippiHOG_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep, IppiSize roiSize,
const IppiPoint* pLocation, int nLocations, Ipp32f* pDst, const IppiHOGSpec* pHOGCtx,
IppiBorderType borderID, Ipp<srcDatatype> borderValue[3], Ipp8u* pBuffer);
```

Supported values for `mod`:

`8u32f_C3R`	`16u32f_C3R`	`16s32f_C3R`	`32f_C3R`

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance, in bytes, between the starting points of consecutive lines in the source image.

roiSize	Size of the source image ROI, in pixels.
pLocation	Pointer to the array with detection window locations.
nLocations	Number of locations.
pDst	Pointer to the HOG descriptor.
pHOGCtx, pHOGSpec	Pointer to the HOG context/specification structure.
borderID	Type of border. Possible values are:

ippBorderConst	Values of all border pixels are set to a constant.
ippBorderRepl	Border is replicated from the edge pixels.
ippBorderInMem	Border is obtained from the source image pixels in memory.
ippBorderMirror	Border pixels are mirrored from the source image boundary pixels.

Mixed borders are also supported. They can be obtained by the bitwise operation OR between ippBorderRepl and ippBorderInMemTop, ippBorderInMemBottom, ippBorderInMemLeft, ippBorderInMemRight.

borderValue	Constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.
pBuffer	Pointer to the work buffer.

Description

This function computes the HOG descriptor over defined locations of the detection window. Flavors with the C1 suffix operate on one-channel (gray) images, and C3 flavors operate on color images.

Before using this function, compute the size of the context structure, work buffer, and descriptor using the HOGGetSize, HOGGetBufferSize, and HOGGetDescriptorSize functions, respectively. To initialize the HOG context structure, use the HOGInit function.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsContextmatchErr	Indicates an error when the context parameter does not match the operation.
ippStsStepErr	Indicates an error when srcStep is less than, or equal to zero.
ippStsNotEvenStepErr	Indicates an error when srcStep is not divisible by input data type size.

ippStsBorderErr	Indicates an error when borderID has an illegal value.
ippStsSizeErr	Indicates an error when roiSize is less than, or equal to zero.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

// A simple example of computing HOG descriptor using Intel(R) Integrated Performance Primitives
(Intel(R) IPP) functions:
//     ippiHOGGetSize
//     ippiHOGInit
//     ippiHOGGetBufferSize
//     ippiHOGGetDescriptorSize
//     ippiHOG_8u32f_C1R

#include <stdio.h>
#include "ipp.h"

#define WIND_WIDTH  64  /* detection window width  */
#define WIND_HEIGHT 128 /* detection window image height */

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define check_sts(st)        \
    if ((st) != ippStsNoErr) \
        goto exitLine; /* Go to Exit if Intel(R) IPP function returned status different from
ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

static IppiHOGConfig config = {
    /* HOG configure structure */
    1,                          /* Use OpenCV compatible output format */
    8,                          /* Cell size (pixels) */
    16,                         /* Block size (pixels) */
    8,                          /* Block stride size (pixels) */
    9,                          /* Number of bins (orientations) in the histogram */
    1.0f,                       /* Sigma value (in the applied Gaussian) */
    0.2f,                       /* Threshold value (applied for normalization) */
```

```
    {WIND_WIDTH, WIND_HEIGHT}, /* Size (width and height in pixels) of detection window */
};

int main(void)
{
    IppStatus status = ippStsNoErr;
    Ipp8u *pSrc = NULL;                          /* Pointers to source/destination/feature
images */
    int srcStep = 0;                             /* Steps, in bytes, through the source/
destination/feature images */
    IppiSize roiSize = {WIND_WIDTH, WIND_HEIGHT}; /* Size of source ROI in pixels */
    const Ipp8u borderValue = 0;
    IppiHOGSpec *pHOGctx = NULL;                          /* Pointer to the HOG context  */
    int hogCtxSize = 0, hogBuffSize = 0, winBuffSize = 0; /* Sizes of buffers */
    Ipp8u *pBuffer = NULL;                                /* Pointer to the work buffer */
    IppiPoint loc[] = {{0, 0}, {2, 1}, {4, 2}, {6, 3}};   /* Locations of detection window */
    int numLocs = sizeof(loc) / sizeof(IppiPoint);
    Ipp32f *pDescriptor = NULL; /* HOG descriptor */

    check_sts(status = ippiHOGGetSize(&config, &hogCtxSize)); /* Get size of HOG context */

    pHOGctx = (IppiHOGSpec *)ippsMalloc_8u(hogCtxSize);
    check_sts(status = ippiHOGInit(&config, pHOGctx)); /* Initialize HOG context */

    pSrc = ippiMalloc_8u_C1(roiSize.width, roiSize.height, &srcStep);

    /* Compute the temporary work buffer size */
    check_sts(status = ippiHOGGetBufferSize(pHOGctx, roiSize, &hogBuffSize));

    pBuffer = ippsMalloc_8u(hogBuffSize);

    /* Compute size of HOG descriptor */
    check_sts(status = ippiHOGGetDescriptorSize(pHOGctx, &winBuffSize));

    pDescriptor = (Ipp32f *)ippsMalloc_8u(numLocs * winBuffSize);

    check_sts(status = ippiHOG_8u32f_C1R(pSrc, srcStep, roiSize, /* Compute HOG descriptor */
                                         loc, numLocs, pDescriptor, pHOGctx, ippBorderConst,
borderValue, pBuffer));

    EXIT_MAIN
    ippiFree(pSrc);
    ippsFree(pHOGctx);
    ippsFree(pBuffer);
    ippsFree(pDescriptor);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

See Also

HOGGetSize Computes the size of the HOG context structure.

HOGGetBufferSize Computes the size of the work buffer for the ippiHOG function.

HOGGetDescriptorSize Computes the size of the HOG descriptor.

HOGInit Initializes the HOG context structure.

Hough Transform

The Hough transform is a general technique that allows to detect the flat curves in binary images [Gon93]. The current version of Intel IPP implements the following:

- Detection of the straight lines that are defined by the parametric equation:

  `r=x*cos(*Θ* ) +y*sin(*Θ* )`, where `r` and *Θ* are the length and angle from the origin of a normal to the line, respectively.
- Detection of lines using the probabilistic Hough transform algorithm Matas00.

- HoughLineGetSize
- HoughLine
- HoughLine_Region
- HoughProbLineGetSize
- HoughProbLineInit
- HoughProbLine

HoughLineGetSize

Computes the size of the working buffer for the straight lines detection.

Syntax

```
IppStatus ippiHoughLineGetSize_8u_C1R(IppiSize roiSize, IppPointPolar delta, int maxLineCount, int* pBufSize);
```

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`roiSize`	Size of the source image ROI, in pixels.
`delta`	Step of discretization (`delta.rho` – radial increment, `delta.theta` – angular increment).
`maxLineCount`	Number of elements of the destination buffer.
`pBufSize`	Pointer to the size of the working buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function computes the size of the temporary working buffer that is required for the function ippiHoughLine.

Return Values

`ippStsNoErr`	Indicates no error.

ippStsNullPtrErr	Indicates an error if pBufSize is NULL.
ippStsSizeErr	Indicates an error condition if roiSize or delta has a field with zero or negative value.

HoughLine

Detects straight lines in the source image.

Syntax

```
IppStatus ippiHoughLine_8u32f_C1R(const Ipp8u* pSrc, int srcStep, IppiSize roiSize,
IppPointPolar delta, int threshold, IppPointPolar* pLine, int maxLineCount, int*
pLineCount, Ipp8u* pBuffer);
```

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
roiSize	Size of source image ROI in pixels.
delta	Step of discretization (delta.rho – radial increment, delta.theta – angular increment).
threshold	Minimum number of points that are required to detect the line.
pLine	Pointer to the destination buffer for lines.
pLineCount	Number of detected lines. If the value is more than maxLineCount, the function returns the ippStsDstSizeLessExpected status.
maxLineCount	Number of elements of the destination buffer.
pBuffer	Pointer to the working buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

In the binarised source image pSrc, the function performs detection of the straight line defined by the equation given at the beginning of section Hough Transform. The level of discretization delta is specified as the input parameters. The performance and effectiveness of the function is strongly depends on this parameter. The function requires the external working buffer pBuffer, which size should be computed previously using the function ippiHoughLineGetSize.

NOTE
CAUTION
The value of the parameter `delta` must not be greater than the value of the parameter `delta` set when the size of the working buffer is computed.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize` has a field with zero or negative value; or if `maxLineCount` is less than or equal to 0.
`ippStsStepErr`	Indicates an error condition if `srcStep` has an illegal value.
`ippStsBadArgErr`	Indicates an error condition if `threshold` is less than or equal to 0; or `delta.rho` is less than 0, or greater than sum of the width and height of the ROI; or `delta.theta` is less than 0, or greater than `p`.
`ippStsDstSizeLessExpected`	Indicates a warning if number of the detected lines is greater than the size of the destination buffer `maxLineCount`.

HoughLine_Region

Detects straight lines with the specified range of parameters in the source image.

Syntax

```
IppStatus ippiHoughLine_Region_8u32f_C1R(const Ipp8u* pSrc, int srcStep, IppiSize roiSize, IppPointPolar* pLine, IppPointPolar dstRoi[2], int maxLineCount, int* pLineCount, IppPointPolar delta, int threshold, Ipp8u* pBuffer);
```

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.

roiSize	Size of source image ROI, in pixels.
pLine	Pointer to the destination array of detected lines.
dstRoi	Specifies the range of parameters of straight lines to be detected.
pLineCount	Pointer to the number of detected lines.
delta	Step of discretization (delta.rho – radial increment, delta.theta – angular increment).
threshold	Minimum number of points that are required to detect the line.
maxLineCount	Maximum number of lines in the destination buffer.
pBuffer	Pointer to the working buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

In the binary source image pSrc, this function performs detection of the straight line defined by the equation given at the beginning of section Hough Transform. Only lines line[n] with the parameters satisfying the following conditions are detected:

dstRoi[0].rho ≤ line [n]. rho ≤ dstRoi [1]. rho;

dstRoi[0].theta ≤ line [n]. theta ≤ dstRoi [1]. theta ;

where n = 0. pLineCount.

The level of discretization delta is specified as the input parameters. The performance and effectiveness of the function is strongly depends on this parameter. The function requires the external working buffer pBuffer, which size should be computed previously using the function ippiHoughLineGetSize.

NOTE
CAUTION
The value of the parameter delta must not be greater than the value of the parameter delta set when the size of the working buffer is computed.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value; or if maxLineCount is less than or equal to 0.
ippStsStepErr	Indicates an error condition if srcStep has an illegal value.
ippStsBadArgErr	Indicates an error condition if threshold is less than or equal to 0; or delta.rho is less than 0, or greater than sum of the width and height of the

	ROI; or `delta.theta` is less than 0, or greater than `p`; or some filed of `dstRoi[0]` is greater than the corresponding filed of `dstRoi[1]`.
`ippStsDstSizeLessExpected`	Indicates a warning if number of the detected lines is greater than the size of the destination buffer `maxLineCount`.

HoughProbLineGetSize

Computes the size of the working buffer and spec structure for line detection with the probabilistic Hough transform algorithm.

Syntax

```
IppStatus ippiHoughProbLineGetSize_8u_C1R(IppiSize roiSize, IppPointPolar delta, int* pSpecSize, int* pBufSize);
```

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`roiSize`	Size of the source image ROI, in pixels.
`delta`	Step of discretization (`delta.rho - radial` increment, `delta.theta - angular` increment).
`pSpecSize`	Size of the `IppiHoughProbSpec` structure.
`pBufSize`	Pointer to the size of the working buffer.

Description

This function operates with ROI.

This function computes the size of the temporary working buffer and the `IppiHoughProbSpec` specification structure for the ippiHoughProbLine function.

For an example on how to use this function, see the example provided with the ippiHoughProbLine function description.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if `pBufSize` is `NULL`.

ippStsSizeErr	Indicates an error condition if roiSize or delta has a field with a zero or negative value.

See Also

Regions of Interest in Intel IPP

HoughProbLine Detects lines in the source image using the probabilistic Hough transform.

HoughProbLineInit

Initializes the specification structure for line detection with the probabilistic Hough transform algorithm.

Syntax

```
IppStatus ippiHoughProbLineInit_8u32f_C1R(IppiSize roiSize, IppPointPolar delta,
IppHintAlgorithm hint, IppiHoughProbSpec* pSpec);
```

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

roiSize	Size of the source image ROI, in pixels.
delta	Step of discretization (delta.rho – radial increment, delta.theta – angular increment).
hint	Suggests using specific code for calculations. Possible values for the hint parameter are described in IppHintAlgorithm enum Type.
pSpec	Pointer to the IppiHoughProbSpec structure.

Description

This function operates with ROI.

This function initializes the IppiHoughProbSpec specification structure that is required for the ippiHoughProbLine function.

For an example on how to use this function, see the example provided with the ippiHoughProbLine function description.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if pBufSize is NULL.
ippStsSizeErr	Indicates an error condition if roiSize or delta has a field with zero or negative value.

See Also

Regions of Interest in Intel IPP

HoughProbLine Detects lines in the source image using the probabilistic Hough transform.

HoughProbLine

Detects lines in the source image using the probabilistic Hough transform.

Syntax

```
IppStatus ippiHoughProbLine_8u32f_C1R(const Ipp8u* pSrc, int srcStep, IppiSize roiSize,
int threshold, int lineLength, int lineGap, IppiPoint* pLine, int maxLineCount, int*
pLineCount, Ipp8u* pBuffer, const IppiHoughProbSpec* pSpec);
```

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance, in bytes, between the starting points of consecutive lines in the source image.
`roiSize`	Size of the source image ROI in pixels.
`threshold`	Minimum number of points that are required to detect the line.
`lineLength`	Minimum length of the line.
`lineGap`	Maximum length of the gap between lines.
`pLine`	Pointer to the detected line: `pLine[2*n]` indicates beginning of the line, `pLine[2*n+1]` indicates end of the line.
`pLineCount`	Number of detected lines.
`maxLineCount`	Number of elements in the destination buffer.
`pBuffer`	Pointer to the working buffer.
`pSpec`	Pointer to the specification structure.

Description

This function operates with ROI.

This function detects line segments of the binary `pSrc` image using the probabilistic Hough transform. Before using this function, compute the size of the working buffer and specification structure using the ippiHoughProbGetSize function and initialize the structure using the ippiHoughProbLineInit function.

This function implements the probabilistic Hough transform algorithm for line detection, described in Matas00.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value; or if maxLineCount is less than or equal to 0.
ippStsStepErr	Indicates an error condition if srcStep has an illegal value.
ippStsBadArgErr	Indicates an error condition if threshold is less than or equal to 0.
ippStsDstSizeLessExpected	Indicates a warning if number of the detected lines is greater than the size of the destination buffer maxLineCount.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

// A simple example of detecting lines in the source image using
// the probabilistic Hough transform using Intel(R) Integrated Performance Primitives (Intel(R)
IPP) functions:
//     ippiHoughProbLineGetSize_8u_C1R
//     ippiHoughProbLineInit_8u32f_C1R
//     ippiHoughProbLine_8u32f_C1R

#include <stdio.h>
#include "ipp.h"

#define WIDTH  800 /* image width  */
#define HEIGHT 600 /* image height */

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
```

```
#define check_sts(st)        \
    if ((st) != ippStsNoErr) \
        goto exitLine; /* Go to Exit if Intel(R) IPP function returned status different from
ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main(void)
{
    int i = 0;
    IppStatus status = ippStsNoErr;
    Ipp8u *pSrc = NULL;                /* Pointers to source images */
    int srcStep = 0;                   /* Steps, in bytes, through the source images */
    IppiSize roiSize = {WIDTH, HEIGHT}; /* Size of source ROI in pixels */
    int bufferSize = 0, specSize = 0;   /* Size of the working buffer and spec structure */
    Ipp8u *pBuffer = NULL;
    IppiHoughProbSpec *pSpec = NULL;
    IppiPoint pLineP[128]; /* Pointer to output array of detected lines */
    int threshold = 20, lineLength = 40, ineGap = 4, maxLineCount = 128;
    IppPointPolar delta = {1, (Ipp32f)IPP_PI / 180.f / 3.f}; /* Step of discretization */
    int lineCount = 0;                                       /* Number of detected lines. */

    pSrc = ippiMalloc_8u_C1(roiSize.width, roiSize.height, &srcStep);

    check_sts(status = ippiSet_8u_C1R(0, pSrc, srcStep, roiSize)); /* Zero image */

    for (i = 0; i < 70; i++) { /* Draw line (100,100)-(170,170) */
        pSrc[(i + 100) * srcStep + i + 100] = 255;
    }

    /* Compute the size of the working buffer and spec structure */
    check_sts(status = ippiHoughProbLineGetSize_8u_C1R(roiSize, delta, &specSize, &bufferSize));

    pBuffer = (Ipp8u *)ippsMalloc_8u(bufferSize);
    pSpec = (IppiHoughProbSpec *)ippsMalloc_8u(specSize);

    /* Initialize the specification structure */
    check_sts(status = ippiHoughProbLineInit_8u32f_C1R(roiSize, delta, ippAlgHintNone, pSpec));

    /* Detect lines in the source image using the probabilistic Hough transform. */
    check_sts(status = ippiHoughProbLine_8u32f_C1R(pSrc, srcStep, roiSize, threshold,
lineLength, ineGap, pLineP, maxLineCount, &lineCount, pBuffer,
                                                   pSpec));

    EXIT_MAIN
    ippsFree(pBuffer);
    ippiFree(pSrc);
    ippsFree(pSpec);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

See Also

Regions of Interest in Intel IPP

HoughProbLineGetSize Computes the size of the working buffer and spec structure for line detection with the probabilistic Hough transform algorithm.

HoughProbLineInit Initializes the specification structure for line detection with the probabilistic Hough transform algorithm.

LineSuppressionGetBufferSize

Computes the size of the external buffer for the `ippiLineSuppression` function.

Syntax

```
IppStatus ippiLineSuppressionGetBufferSize(IppiSize roiSize, IppiMaskSize filterMask,
Ipp32u avgWndSize, IppDataType dataType, int numChannels, int* pBufferSize);
```

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`roiSize`	Size of the source and destination image ROI, in pixels.
`filterMask`	Size of the derivative filter aperture. Possible values are `ippMskSize3x3` and `ippMskSize5x5`.
`avgWndSize`	Linear size of a neighborhood block for averaging.
`dataType`	Data type of the source image.
`numChannels`	Number of channels in the images. Possible value is 1.
`pBufferSize`	Pointer to the size, in bytes, of the external work buffer.

Description

This function calculates the size of the temporary buffer for the ippiLineSuppression function.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error when `pBufferSize` is `NULL`.
`ippStsSizeErr`	Indicates an error in the following cases: • when `roiSize` is less than, or equal to zero • when `avgWndSize` is equal to zero
`ippStsMaskSizeErr`	Indicates an error when `filterMask` has an illegal value.
`ippStsDataTypeErr`	Indicates an error when `dataType` has an illegal value.
`ippStsNumChannelsErr`	Indicates an error when `numChannels` has an illegal value.

See Also

LineSuppression Implements the line suppression algorithm.

LineSuppression

Implements the line suppression algorithm.

Syntax

```
IppStatus ippiLineSuppression_8u_C1R(const Ipp8u* pSrc, int srcStep, const Ipp8u*
pFeature, int featureStep, Ipp8u* pDst, int dstStep, IppiSize roiSize,
IppiDifferentialKernel filterType, IppiMaskSize filterMask, Ipp32u avgWndSize, float
threshold, IppiBorderType borderType, Ipp8u borderValue, Ipp8u* pBuffer);
```

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the source image.
`srcStep`	Distance, in bytes, between the starting points of consecutive lines in the source image.
`pFeature`	Pointer to the feature points image.
`featureStep`	Distance, in bytes, between the starting points of consecutive lines in the feature points image.
`pDst`	Pointer to the destination image.
`dstStep`	Distance, in bytes, between the starting points of consecutive lines in the destination image.
`roiSize`	Size of the source and destination image ROI, in pixels.
`filterType`	Type of the derivative operator. Possible values are:

`ippKernelSobel`	Sobel filter
`ippKernelScharr`	Scharr filter
`ippKernelCentralDiff`	Central differences operator

`filterMask` Size of the derivative filter aperture. The list of possible values depends on the `filterType` value:

Filter Type	Filter Mask
`ippKernelSobel`	`ippMskSize3x3`, `ippMskSize5x5`
`ippKernelScharr`	`ippMskSize3x3`

ippKernelCentralDiff	ippMskSize3x3

`avgWndSize` Linear size of a neighborhood block for averaging.

`threshold` Line suppression threshold.

`borderType` Type of border. Possible values are:

ippBorderConst	Values of all border pixels are set to a constant.
ippBorderRepl	Border is replicated from the edge pixels.
ippBorderInMem	Border is obtained from the source image pixels in memory.

Mixed borders are also supported. They can be obtained by the bitwise operation OR between `ippBorderRepl` or `ippBorderConst` and the following flags:

- `ippBorderInMemTop`
- `ippBorderInMemBottom`
- `ippBorderInMemLeft`
- `ippBorderInMemRight`

Each of these flags means that for the corresponding border the outside pixels of the source image are in memory.

`borderValue` Constant value(s) to assign to pixels of the constant border. This parameter is applicable only to the `ippBorderConst` border type.

`pBuffer` Pointer to the work buffer. To calculate the size of the temporary buffer, use the LineSuppressionGetBufferSize function.

Description

The `ippiLineSuppression` function implements the line suppression algorithm. This function uses two input images: the original image and feature points image containing the lines (edges) and corners. This function performs the following steps:

1. Computes $I(x, y)_x$ and $I(x, y)_y$ gradients for each (`x`, `y`) pixel of the image. The function computes gradients using the derivative operator specified by the `filterType` and `filterMask` parameters.
2. Computes products of the gradients at each (`x`, `y`) pixel of the image:

$$I_{xx}^{(x,y)} = I_x^{(x,y)} * I_x^{(x,y)}, \quad I_{yx}^{(x,y)} = I_{xy}^{(x,y)} = I_x^{(x,y)} * I_y^{(x,y)}, \quad I_{yy}^{(x,y)} = I_y^{(x,y)} * I_y^{(x,y)}$$

3. Performs averaging of the products of gradients over a rectangular neighborhood block at each pixel of the image. The block size is specified by the `avgWndSize` value.

$$S_{xx}^{(x,y)} = \sum_{y'} \sum_{x'} I_{xx}^{(x',y')}, \quad S_{xy}^{(x,y)} = S_{yx}^{(x,y)} = \sum_{y'} \sum_{x'} I_{xy}^{(x',y')}, \quad S_{yy}^{(x,y)} = \sum_{y'} \sum_{x'} I_{yy}^{(x',y')}$$

4. Defines 2x2 gradient covariance matrix $H(x, y)$ over a rectangular neighborhood block for each (`x`, `y`) pixel of the image.

$$H^{(x,y)} = \begin{pmatrix} S_{xx}^{(x,y)} & S_{xy}^{(x,y)} \\ S_{yx}^{(x,y)} & S_{yy}^{(x,y)} \end{pmatrix}$$

5. For each (`x`, `y`) pixel of the image checks the condition below. If the condition is true, the considered point is not a feature point.

$$\frac{(\lambda_1 + \lambda_2)^2}{\lambda_1 * \lambda_2} = \frac{(trH^{(x,y)})^2}{\det H^{(x,y)}} > threshold$$

where

- `threshold` is the line suppression threshold value passed to the function
- $\lambda_1 * \lambda_2$ are eigenvalues of the matrix `H(``x`, y)`. If both eigenvalues have large positive values, the point belongs to a corner. If λ_1 is much bigger than λ_2, the function clears out the candidate point.

The first and third stages of the function algorithm are filtering operations; they use border processing approach that is specified by the `borderType` parameter.

Before using this function, compute the size of the external work buffer using the LineSuppressionGetBufferSize function.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when `pSrc`, `pFeature`, `pDst`, or `pBufferSize` is `NULL`.
`ippStsSizeErr`	Indicates an error in the following cases: • when `roiSize` is less than, or equal to zero • when `avgWndSize` is equal to zero
`ippStsNotEvenStepErr`	Indicates an error when one of the step values is not divisible by 4 for floating point images.
`ippStsFilterTypeErr`	Indicates an error when `filterType` has an illegal value.
`ippStsMaskSizeErr`	Indicates an error when `filterMask` has an illegal value.

ippStsBorderErr	Indicates an error when borderType has an illegal value.
ippStsStepErr	Indicates an error when srcStep or dstStep has a negative value.
ippStsInplaceModeNotSupportedErr	Indicates an error when pFeature and pDst point to the same image.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

// A simple example of Line Suppression
// using Intel(R) Integrated Performance Primitives (Intel(R) IPP) functions:
//     ippiLineSuppressionGetBufferSize
//     ippiLineSuppression_8u_C1R

#include <stdio.h>
#include "ipp.h"

#define WIDTH  128 /* image width  */
#define HEIGHT 64  /* image height */

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define check_sts(st)        \
    if ((st) != ippStsNoErr) \
        goto exitLine; /* Go to Exit if Intel(R) IPP function returned status different from
ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main(void)
{
    IppStatus status = ippStsNoErr;
    Ipp8u *pSrc = NULL, *pDst = NULL, *pFeature = NULL; /* Pointers to source/destination/
feature images */
    int srcStep = 0, dstStep = 0, featureStep = 0;      /* Steps, in bytes, through the source/
destination/feature images */
    IppiSize roiSize = {WIDTH, HEIGHT};                 /* Size of source/destination ROI in
pixels */
```

```
    IppiMaskSize filterMask = ippMskSize3x3;            /* Linear size of derivative filter
aperture */
    Ipp32u avgWndSize = 1;                              /* Neighbourhood block size for
smoothing */
    Ipp32u numChannels = 1;
    Ipp32f threshold = 5.0f;
    const Ipp8u borderValue = 0;
    Ipp8u *pBuffer = NULL; /* Pointer to the work buffer */
    int bufSize = 0;
    IppiDifferentialKernel filterType = ippFilterSobel; /* Type of derivative operator */

    pSrc = ippiMalloc_8u_C1(roiSize.width, roiSize.height, &srcStep);
    pDst = ippiMalloc_8u_C1(roiSize.width, roiSize.height, &dstStep);
    pFeature = ippiMalloc_8u_C1(roiSize.width, roiSize.height, &featureStep);

    /* Computes the temporary work buffer size */
    check_sts(status = ippiLineSuppressionGetBufferSize(roiSize, filterMask, avgWndSize, ipp8u,
numChannels, &bufSize));
    pBuffer = ippsMalloc_8u(bufSize);

    /* Line Suppression processing */
    check_sts(status = ippiLineSuppression_8u_C1R(pSrc, srcStep, pFeature, featureStep, pDst,
dstStep, roiSize, filterType, filterMask, avgWndSize,
                                                  threshold, ippBorderRepl, borderValue,
pBuffer));

    EXIT_MAIN
    ippiFree(pSrc);
    ippiFree(pDst);
    ippiFree(pFeature);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

See Also

LineSuppressionGetBufferSize Computes the size of the external buffer for the ippiLineSuppression function.

Distance Transform Functions

This section describes the distance transform functions.

Distance transform is used for calculating the distance to an object. The input is an image with feature and non-feature pixels. The function labels every non-feature pixel in the output image with a distance to the closest feature pixel. Feature pixels are marked with zero.

Distance transform is used for a wide variety of subjects including skeleton finding and shape analysis.

- DistanceTransform
- GetDistanceTransformMask
- FastMarchingGetBufferSize
- FastMarching
- TrueDistanceTransformGetBufSize
- TrueDistanceTransform

DistanceTransform

Calculates distance to the closest zero pixel for all non-zero pixels of source image.

Syntax

Case 1: Not-in-place operations

```
IppStatus ippiDistanceTransform_3x3_<mod>(const Ipp8u* pSrc, int srcStep,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize, Ipp32s* pMetrics);
```

```
IppStatus ippiDistanceTransform_5x5_<mod>(const Ipp8u* pSrc, int srcStep,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize roiSize, Ipp32s* pMetrics);
```

Supported values for `mod`:

`8u_C1R`	`8u16u_C1R`

```
IppStatus ippiDistanceTransform_3x3_8u32f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32f*
pDst, int dstStep, IppiSize roiSize, Ipp32f* pMetrics);
```

```
IppStatus ippiDistanceTransform_5x5_8u32f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32f*
pDst, int dstStep, IppiSize roiSize, Ipp32f* pMetrics);
```

Case 2: In-place operations

```
IppStatus ippiDistanceTransform_3x3_8u_C1IR(Ipp8u* pSrcDst, int srcDstStep, IppiSize
roiSize, Ipp32s* pMetrics);
```

```
IppStatus ippiDistanceTransform_5x5_8u_C1IR(Ipp8u* pSrcDst, int srcDstStep, IppiSize
roiSize, Ipp32s* pMetrics);
```

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`pDst`	Pointer to the ROI in the destination distance image.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image.
`pSrcDst`	Pointer to the source and destination image ROI for in-place operation.
`srcDstStep`	Distance in bytes between starts of consecutive lines in the source and destination image for in-place operation.
`roiSize`	Size of the image ROI in pixels.
`pMetrics`	Pointer to the array that specifies used metrics.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function approximates the actual distance from the closest zero pixel to each certain pixel with the sum of atomic distances from the fixed set. The set consists of two values for a 3x3 mask and three values for a 5x5 mask.

Figure 3x3 Mask shows the result of the distance transform of a 7x7 image with zero point in the center. This example corresponds to a 3 x 3 mask. Two numbers specify metrics in case of the 3 x 3 mask:

- distance between two pixels that share an edge,
- distance between the pixels that share a corner.

In this case the values are 1 and 1.5 correspondingly.

3x3 Mask

4.5	4	3.5	3	3.5	4	4.5
4	3	2.5	2	2.5	3	4
3.5	2.5	1.5	1	1.5	2.5	3.5
3	2	1	0	1	2	3
3.5	2.5	1.5	1	1.5	2.5	3.5
4	3	2.5	2	2.5	3	4
4.5	4	3.5	3	3.5	4	4.5

Figure 5x5 Mask shows the distance transform for the same image, but for a 5 x 5 mask.

For this mask yet another number is added to specify metrics - the additional distance, i.e., the distance between pixels corresponding to the chess knight move.

In this example, the additional distance is equal to 2.

5x5 Mask

4	3.5	3	3	3	3.5	4
3.5	3	2	2	2	3	3.5
3	2	1.5	1	1.5	2	3
3	2	1	0	1	2	3
3	2	1.5	1	1.5	2	3
3.5	3	2	2	2	3	3.5
4	3.5	3	3	3	3.5	4

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize` has a field with zero or negative value.

ippStsStepErr	Indicates an error condition if srcStep or dstStep is less than roiSize.width*<pixelSize>.
ippStsNotEvenStepErr	Indicates an error condition if step value is not divisible by 2 for 16u images, and by 4 for 32f images.
ippStsCoeffErr	Indicates an error condition if at least one element of pMetrics array has zero or negative value.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

// A simple example of calculating distance between pixels
// using Intel(R) Integrated Performance Primitives (Intel(R) IPP) function:
//     ippiDistanceTransform_3x3_8u_C1R

#include <stdio.h>
#include "ipp.h"

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define check_sts(st)        \
    if ((st) != ippStsNoErr) \
        goto exitLine; /* Go to Exit if Intel(R) IPP function returned status different from
ippStsNoErr */

int main(void)
{
    IppStatus status = ippStsNoErr;
    Ipp8u pSrc[7 * 7] = {/* Pointer to source image */
                         1, 2, 3, 4, 5, 6, 7, 1, 0, 3, 4, 5, 6, 7, 1, 2, 3, 4, 5, 6, 7, 1, 2, 3,
0,
                         5, 6, 7, 1, 2, 3, 4, 5, 6, 7, 1, 2, 3, 4, 5, 0, 7, 1, 2, 3, 4, 5, 6, 7};
    Ipp8u pDst[7 * 7];            /* Pointer to destination image */
    int srcStep = 7, dstStep = 7; /* Steps, in bytes, through the source/destination images */
    IppiSize roiSize = {7, 7};    /* Size of source/destination ROI in pixels */
    Ipp32s pMetrics[2] = {2, 2};  /* Array that determines metrics used */
    /* For every non-zero pixel in the source image, the functions calculate   */
    /*                     distance between that pixel and nearest zero pixel. */
    check_sts(status = ippiDistanceTransform_3x3_8u_C1R(pSrc, srcStep, pDst, dstStep, roiSize,
```

```
pMetrics));

    EXIT_MAIN
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

Result:

```
1 2 3 4 5 6 7
1 0 3 4 5 6 7
1 2 3 4 5 6 7
1 2 3 0 5 6 7    src
1 2 3 4 5 6 7
1 2 3 4 5 0 7
1 2 3 4 5 6 7


2 2 2 4 6 6 6
2 0 2 4 4 4 6
2 2 2 2 2 4 6
4 4 2 0 2 4 4    dst
6 4 2 2 2 2 2
6 4 4 4 2 0 2
6 6 6 4 2 2 2
```

GetDistanceTransformMask

Returns an optimal mask for a given type of metrics and given mask size.

Syntax

`IppStatus ippiGetDistanceTransformMask_<mod>(int kerSize, IppiNorm norm, Ipp<datatype>* pMetrics);`

Supported values for `mod`:

32s	32f

`IppStatus ippiGetDistanceTransformMask(int maskType, Ipp32f* pMetrics);`

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`kerSize`	Specifies the mask size as follows: 3 for 3x3 mask, 5 for 5x5 mask.
`norm`	Specifies the type of metrics. Possible values are:

`ippiNormInf` (0)	L_∞, $\Delta = \max(\lvert x_1 - x_2 \rvert, \lvert y_1 - y2 \rvert)$,

ippiNormL1 (1)	L_1, $\Delta = \|x_1 - x_2\| + \|y_1 - y2\|$,
ippiNormL2 (2)	L2, $\Delta = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$

maskType — Distance type.

pMetrics — Pointer to the output array to store metrics parameters. The array contains the following number of elements:

2	for 3x3 mask,
3	for 5x5 mask.

Description

This function fills up the output array with metrics parameters for the given type of metrics and size of mask. The function returns the following results:

(1, 1)	L_∞, 3x3 mask,
(1, 2)	L_1, 3x3 mask,
(2, 3)	L_2, 3x3 mask, 32s data type,
(0.955, 1.3693)	L_2, 3x3 mask, 32f data type,
(1, 1, 2)	L_∞, 5x5 mask,
(1, 2, 3)	L_1, 5x5 mask,
(4, 6, 9)	L_2, 5x5 mask, 32s data type,
(1.0, 1.4, 2.1969)	L_2, 5x5 mask, 32f data type.

For more information, see [Bor86].

NOTE For compatibility with the previous versions of the library the earlier function ippiGetDistanceTransformMask replaced by the function ippiGetDistanceTransformMask_32f in the current version is also supported.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if pMetrics pointer is NULL.
ippStsSizeErr	Indicates an error condition if kerSize has a wrong value.
ippStsBadArgErr	Indicates an error condition if kerSize or norm has a wrong value.

FastMarchingGetBufferSize

Computes the size of the working buffer for the peak search.

Syntax

```
IppStatus ippiFastMarchingGetBufferSize_8u32f_C1R(IppiSize roiSize, int* pBufferSize);
```

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`roiSize`	Maximum image size (in pixels).
`pBufferSize`	Pointer to the computed size of the buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function computes the size of the working buffer required for the ippiFastMarching function. The buffer with the length `pBufferSize[0]` can be used to filter images with width that is equal to or less than the parameter `roiSize` specified for the function ippiFastMarching.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if `pBufferSize` pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize` is lessthan 1.

FastMarching

Calculates distance transform to closest zero pixel for all non-zero pixels of source image using fast marching method.

Syntax

```
IppStatus ippiFastMarching_8u32f_C1R(const Ipp8u* pSrc, int srcStep, Ipp32f* pDst, int dstStep, IppiSize roiSize, Ipp32f radius, Ipp8u* pBuffer);
```

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image.
`roiSize`	Size of the source and destination image ROI.
`radius`	Radius of the neighborhood of the marked area.
`pBuffer`	Pointer to the working buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function computes the distance from the closest zero pixel to each image pixel according to the Fast Marching Method (FMM) [Telea04]. The FMM distance for area Ω with the border ∂Ω is a solution of the equations:

$$(|\nabla T(x,\ y)| = 1),\ \{x,\ y\} \in$$

$$(T(x,\ y) = 0),\ \{x,\ y\} \in \partial\Omega$$

The resulting distance complies with the equation

$$T(x,\ y) = min\left(\frac{T(u_1,\ v_1) + T(u_2,\ v_2) + \sqrt{2 - (T(u_1,\ v_1) - T(u_2,\ v_2))^2}}{2},\ min(T(u_1,\ v_1),\ T(u_2,\ v\right.$$

Here $\{u_1, v_1\}$ and $\{u_2, v_2\}$ are coordinates for pair of pixels adjacent to the pixel with $\{x,\ y\}$ coordinates.

The area **Ω** is defined by the non-zero pixel of the image `pSrc`. If `raduis` is positive, then the FMM distance with the negative sign is calculated in Euclidean `raduis - neighborhood` of **Ω**.

The function requires the working buffer `pBuffer` whose size should be computed by the function FastMarchingGetBufferSize beforehand.

Figure Result of the FFM Method shows the result of the fast marching method for the 7x9 image with centered 3x5 non-zero mask and `raduis = 1`.

Result of the FFM Method

```
0.0000     0.0000   -1.0000    -1.0000   -1.0000    -1.0000   -1.0000    0.0000    0.0000
0.0000     0.7071   -1.e-10    -1.e-10   -1.e-10    -1.e-10   -1.e-10    0.7071   -1.0000
-1.0000    -1.e-10    0.7071    0.9659    0.9994     0.9659    0.7071   -1.e-10   -1.0000
-1.0000    -1.e-10    0.9659    1.6730    1.9579     1.6730    0.9659   -1.e-10   -1.0000
-1.0000    -1.e-10    0.7071    0.9659    0.9994     0.9659    0.7071   -1.e-10   -1.0000
0.0000     0.7071   -1.e-10    -1.e-10   -1.e-10    -1.e-10   -1.e-10    0.7071   -1.0000
0.0000    -1.0000   -1.0000    -1.0000   -1.0000     -1.0000   -1.0000   -1.0000    0.0000
```

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize` has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if `srcStep` or `dstStep` is less than `roiSize.width` * < `pixelSize`>.
`ippStsNotEvenStepErr`	Indicates an error condition if the step value is not divisible by 4 for floating-point images.
`ippStsBadArgErr`	Indicates an error condition if `raduis` is negative.

TrueDistanceTransformGetBufSize

Calculates the size of the temporary working buffer forthe function `ippiTrueDistanceTransform`.

Syntax

```
IppStatus ippiTrueDistanceTransformGetBufferSize_8u32f_C1R(IppiSize roiSize, int*
pBufferSize);
```

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`roiSize`	Size of the imageROI in pixels.
`pBufferSize`	Pointer to the computed size of the buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function calculates the size of the work buffer required for the TrueDistanceTransform function.

Return Values

ippStsNoErr	Indicates no error. Any othervalue indicates an error or a warning.
ippStsNullPtrErr	Indicates an error conditionif one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error conditionif roiSize has a field withzero or negative value.
ippStsStepErr	Indicates an error conditionif srcStep or dstStep is less than roiSize.width*<pixelSize>.
ippStsNotEvenStepErr	Indicates an error conditionif step value is not divisible by 2 for 16u images,and by 4 for 32f images.

TrueDistanceTransform

Calculates the Eucledian distance to the closest zeropixel for all non-zero pixels of the source image.

Syntax

```
IppStatus ippiTrueDistanceTransform_8u32f_C1R(constIpp8u* pSrc, int srcStep, Ipp32f* pDst, int dstStep,IppiSize roiSize, Ipp8u* pBuffer);
```

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Pointer to the sourceimage ROI.
srcStep	Distance in bytesbetween starts of consecutive lines in the source image.
pDst	Pointer to the ROIin the destination distance image.
dstStep	Distance in bytesbetween starts of consecutive lines in the destination image.
roiSize	Size of the imageROI in pixels.
pBuffer	Pointer to the temporaryworking buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function calculates the Eucledian distance to the closest zero pixel for all non-zero pixels of the source image [Felz04].

The figure below shows the result of the integer version of the true distance transform of a 7x7 image with zero point in the center and with the scale factor -5.

136	115	101	96	101	115	136
115	91	72	64	72	91	115
101	72	45	36	45	72	101
96	64	36	0	36	64	96
101	72	45	36	45	72	101
115	91	72	64	72	91	115
136	115	101	96	101	115	136

Return Values

`ippStsNoErr`	Indicates no error. Any othervalue indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error conditionif one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error conditionif `roiSize` has a field withzero or negative value.
`ippStsStepErr`	Indicates an error conditionif `srcStep` or `dstStep` is less than `roiSize.width*<pixelSize>`.
`ippStsNotEvenStepErr`	Indicates an error conditionif step value is not divisible by 2 for `16u` images,and by 4 for `32f` images.

Image Gradients

- GradientColorToGray
- GradientVectorGetBufferSize
- GradientVectorPrewitt
- GradientVectorScharr
- GradientVectorSobel

GradientColorToGray

Converts a color gradient image to grayscale.

Syntax

```
IppStatus ippiGradientColorToGray_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, IppiNorm norm);
```

Supported values for `mod`:

`8u_C3C1R`	`16u_C3C1R`	`32f_C3C1R`

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image.
`roiSize`	Size of the source and destination image ROI.
`norm`	Type of norm to form the mask for dilation; following values are possible: • `ippiNormInf` • Infinity norm. • `ippiNormL1` • L1 norm. • `ippiNormL2` • L2 norm.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function creates the grayscale gradient image `pDst` from the source three-channel gradient image `pSrc`. The type of norm is specified by the parameter. Pixel values for destination image are computed for different type of norm in accordance with the following formula:

$$dst_{i,j} = \begin{cases} \max\{|src_{i,j,0}|, |src_{i,j,1}|, |src_{i,j,2}|\} & norm = ippiNo \\ |src_{i,j,0}| + |src_{i,j,1}| + |src_{i,j,2}| & norm = ippiNo \\ \sqrt{src_{i,j,0}^2 + src_{i,j,1}^2 + src_{i,j,2}^2} & norm = ippiNo \end{cases}$$

For integer flavors the result is scaled to the full range of the destination data type.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if srcStep or dstStep is less than roiSize.width * < pixelSize>.
ippStsNotEvenStepErr	Indicates an error condition if one of the step values is not divisible by 2 for integer images, or by 4 for floating-point images.
ippStsBadArgErr	Indicates an error condition if norm has an illegal value.

GradientVectorGetBufferSize

Computes the size of the work buffer for the ippiGradientVector{Sobel|Scharr|Prewitt} functions.

Syntax

```
IppStatus ippiGradientVectorGetBufferSize (IppiSize roiSize, IppiMaskSize mask,
IppDataType dataType, int numChannels, int* pBufferSize);
```

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

roiSize	Size of the destination ROI in pixels.
mask	Predefined mask of IppiMaskSize type.
dataType	Data type of the source image.
numChannels	Number of channels in the image.
pBufferSize	Pointer to the computed size of the external work buffer.

Description

The ippiGradientVectorGetBufferSize function computes the size (in bytes) of the external work buffer needed for the ippiGradientVector{Sobel|Scharr|Prewitt} functions and stores the result in the pBufferSize parameter.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pBufferSize is NULL.
ippStsSizeErr	Indicates an error when one of the fields of roiSize has a zero or negative value.
ippStsMaskSizeErr	Indicates an error when mask has an illegal value.
ippStsDataTypeErr	Indicates an error when dataType has an illegal value.
ippStsNumChannelsErr	Indicates an error when numChannels has an illegal value.

See Also

Structures and Enumerators

GradientVectorPrewitt Computes gradient vectors of an image using the Prewitt operator.

GradientVectorScharr Computes gradient vectors of an image using the Scharr operator.

GradientVectorSobel Computes gradient vectors of an image using the Sobel operator.

GradientVectorPrewitt

Computes gradient vectors of an image using the Prewitt operator.

Syntax

```
IppStatus ippiGradientVectorPrewitt_<mod> (const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pGx, int gxStep, Ipp<dstDatatype>* pGy, int gyStep, Ipp<dstDatatype>*
pMag, int magStep, Ipp32f* pAngle, int angleStep, IppiSize dstRoiSize, IppiMaskSize
maskSize, IppNormType normType, IppiBorderType borderType, Ipp<srcDatatype>
borderValue, Ipp8u* pBuffer);
```

Supported values for mod:

8u16s_C1R	16s32f_C1R	16u32f_C1R	32f_C1R

```
IppStatus ippiGradientVectorPrewitt_<mod> (const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pGx, int gxStep, Ipp<dstDatatype>* pGy, int gyStep, Ipp<dstDatatype>*
pMag, int magStep, Ipp32f* pAngle, int angleStep, IppiSize dstRoiSize, IppiMaskSize
maskSize, IppNormType normType, IppiBorderType borderType, const Ipp<srcDatatype>
borderValue[3], Ipp8u* pBuffer);
```

Supported values for mod:

8u16s_C3C1R	16s32f_C3C1R	16u32f_C3C1R	32f_C3C1R

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance, in bytes, between the starting points of consecutive lines in the source image.
pGx	Pointer to the destination image ROI for the X component of the gradient vector (in Cartesian form).
gxStep	Distance, in bytes, between the starting points of consecutive lines in the X-component destination image.
pGy	Pointer to the destination image ROI for the Y component of the gradient vector (in Cartesian form).
gyStep	Distance, in bytes, between the starting points of consecutive lines in the Y-component destination image.
pMag	Pointer to the magnitude of the gradient destination image ROI (in polar gradient form).
magStep	Distance, in bytes, between the starting points of consecutive lines in the magnitude of the gradient destination image.
pAngle	Pointer to the destination image ROI for the angle (in polar gradient form).
angleStep	Distance, in bytes, between the starting points of consecutive lines in the angle destination image.
dstRoiSize	Size of the source and destination image ROI in pixels.
maskSize	Predefined mask of IppiMaskSize type.
normType	Normalization mode of IppNormType type.
borderType	Type of border. Possible values are: • ippBorderConst • Values of all border pixels are set to constant. • ippBorderRepl • Border is replicated from the edge pixels.

	• ippBorderInMem • Border is obtained from the source image pixels in memory. • ippBorderMirror • Border pixels are mirrored from the source image boundary pixels.
borderValue	Constant value to assign to pixels in the constant border (not applicable for other border types).
pBuffer	Pointer to the work buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

Before using this function, compute the size of the work buffer using the ippiGradientVectorGetBufferSize function.

NOTE Any of the pGx, pGy, pMag, and pAngle output parameters can be NULL. This means that the parameter(s) is not requested.

This function operates on "gray" single-channel (C1 flavors) and color (C3 flavors) images.

Single-channel image (C1) input :

If input is a single-channel image, the ippiGradientVectorPrewitt function computes the gradient vector at each pixel of the source image ROI and stores the result either in Cartesian (pGx and pGy) and/or polar (pMag and pAngle) form, or any combination of these possible outputs.

Cartesian projections G_x and G_y are stored in the pGx and pGy buffers, respectively. The formulas below describe the algorithm for the 3x3 Prewitt operator:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} * A \quad \text{and} \quad G_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} * A$$

where

- A is the source image
- * means two-dimensional convolution
- G_x and G_y are X and Y components of the gradient

The magnitude of the gradient is computed according to the normType value by the following formulas:

L1 normalization:

$$G = |G_x| + |G_y|$$

L2 normalization:

$$G = \sqrt{G_x^2 + G_y^2}$$

The value of angle between G_x and G_y is computed by the formula:

$$angle = \arctan \frac{G_y}{G_x}$$

Color image (C3) input :

If input is a color image, the `ippiGradientVectorPrewitt` function computes the spatial image derivatives G_x and G_y for each channel of the image using the specified differential operator. For each pixel (`x`, `y`) this function chooses the derivatives for which `L2` (G_x, G_y) is the maximal value and stores them in the `pGx` and `pGy` output arrays. In other words, for each pixel of a color image the function returns the derivatives composing the largest gradient across all channels.

The examples of using this function are similar to the examples provided with the GradientVectorSobel function description.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when `pSrc` or `pBuffer` is NULL.
`ippStsSizeErr`	Indicates an error when one of the fields of `dstRoiSize` has a zero or negative value.
`ippStsMaskSizeErr`	Indicates an error when `maskSize` has an illegal value.

ippStsStepErr	Indicates an error when: • srcStep, gxStep, gySstep, magStep, or angleStep has a zero or negative value • srcStep, gxStep, gyStep, magStep, or angleStep is not a multiple of the image data size (4 for floating-point images or 2 for short integer images)
ippStsBadArgErr	Indicates an error when normType has an illegal value.
ippStsBorderErr	Indicates an error when borderType has an incorrect value.

See Also

Structures and Enumerators

Regions of Interest in Intel IPP

GradientVectorGetBufferSize Computes the size of the work buffer for the ippiGradientVector{Sobel|Scharr|Prewitt} functions.

GradientVectorScharr

Computes gradient vectors of an image using the Scharr operator.

Syntax

```
IppStatus ippiGradientVectorScharr_<mod> (const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pGx, int gxStep, Ipp<dstDatatype>* pGy, int gyStep, Ipp<dstDatatype>*
pMag, int magStep, Ipp32f* pAngle, int angleStep, IppiSize dstRoiSize, IppiMaskSize
maskSize, IppNormType normType, IppiBorderType borderType, Ipp<srcDatatype>
borderValue, Ipp8u* pBuffer);
```

Supported values for mod:

8u16s_C1R	16s32f_C1R	16u32f_C1R	32f_C1R

```
IppStatus ippiGradientVectorScharr_<mod> (const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pGx, int gxStep, Ipp<dstDatatype>* pGy, int gyStep, Ipp<dstDatatype>*
pMag, int magStep, Ipp32f* pAngle, int angleStep, IppiSize dstRoiSize, IppiMaskSize
maskSize, IppNormType normType, IppiBorderType borderType, const Ipp<srcDatatype>
borderValue[3], Ipp8u* pBuffer);
```

Supported values for mod:

8u16s_C3C1R	16s32f_C3C1R	16u32f_C3C1R	32f_C3C1R

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance, in bytes, between the starting points of consecutive lines in the source image.
`pGx`	Pointer to the destination image ROI for the X component of the gradient vector (in Cartesian form).
`gxStep`	Distance, in bytes, between the starting points of consecutive lines in the X-component destination image.
`pGy`	Pointer to the destination image ROI for the Y component of the gradient vector (in Cartesian form).
`gyStep`	Distance, in bytes, between the starting points of consecutive lines in the Y-component destination image.
`pMag`	Pointer to the magnitude of the gradient destination image ROI (in polar gradient form).
`magStep`	Distance, in bytes, between the starting points of consecutive lines in the magnitude of the gradient destination image.
`pAngle`	Pointer to the destination image ROI for the angle (in polar gradient form).
`angleStep`	Distance, in bytes, between the starting points of consecutive lines in the angle destination image.
`dstRoiSize`	Size of the source and destination image ROI in pixels.
`maskSize`	Predefined mask of IppiMaskSize type.
`normType`	Normalization mode of IppNormType type.
`borderType`	Type of border. Possible values are: • `ippBorderConst` • Values of all border pixels are set to constant. • `ippBorderRepl` • Border is replicated from the edge pixels. • `ippBorderInMem` • Border is obtained from the source image pixels in memory. • `ippBorderMirror` • Border pixels are mirrored from the source image boundary pixels.

borderValue	Constant value to assign to pixels in the constant border (not applicable for other border types).
pBuffer	Pointer to the work buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

Before using this function, compute the size of the work buffer using the ippiGradientVectorGetBufferSize function.

NOTE Any of the pGx, pGy, pMag, and pAngle output parameters can be NULL. This means that the parameter(s) is not requested.

This function operates on "gray" single-channel (C1 flavors) and color (C3 flavors) images.

Single-channel image (C1) input :

If input is a single-channel image, the ippiGradientVectorScharr function computes the gradient vector at each pixel of the source image ROI and stores the result either in Cartesian (pGx and pGy) and/or polar (pMag and pAngle) form, or any combination of these possible outputs.

Cartesian projections G_x and G_y are stored in the pGx and pGy buffers, respectively. The formulas below describe the algorithm for the 3x3 Scharr operator:

$$G_x = \begin{bmatrix} 3 & 0 & -3 \\ 10 & 0 & -10 \\ 3 & 0 & -3 \end{bmatrix} * A \qquad G_y = \begin{bmatrix} 3 & 10 & 3 \\ 0 & 0 & 0 \\ -3 & -10 & -3 \end{bmatrix} * A$$

where

- A is the source image
- * means two-dimensional convolution
- G_x and G_y are X and Y components of the gradient

The magnitude of the gradient is computed according to the normType value by the following formulas:

L1 normalization:

$$G = |G_x| + |G_y|$$

L2 normalization:

$$G = \sqrt{G_x^2 + G_y^2}$$

The value of angle between G_x and G_y is computed by the formula:

$$angle = \arctan \frac{G_y}{G_x}$$

Color image (C3) input :

If input is a color image, the `ippiGradientVectorScharr` function computes the spatial image derivatives G_x and G_y for each channel of the image using the specified differential operator. For each pixel (`x`, `y`) this function chooses the derivatives for which `L2`(G_x, G_y) is the maximal value and stores them in the `pGx` and `pGy` output arrays. In other words, for each pixel of a color image the function returns the derivatives composing the largest gradient across all channels.

The examples of using this function are similar to the examples provided with the GradientVectorSobel function description.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when `pSrc` or `pBuffer` is NULL.
`ippStsSizeErr`	Indicates an error when one of the fields of `dstRoiSize` has a zero or negative value.
`ippStsMaskSizeErr`	Indicates an error when `maskSize` has an illegal value.

ippStsStepErr	Indicates an error when: • srcStep, gxStep, gyStep, magStep, or angleStep has a zero or negative value • srcStep, gxStep, gyStep, magStep, or angleStep is not a multiple of the image data size (4 for floating-point images or 2 for short integer images)
ippStsBadArgErr	Indicates an error when normType has an illegal value.
ippStsBorderErr	Indicates an error when borderType has an incorrect value.

See Also

Structures and Enumerators

Regions of Interest in Intel IPP

GradientVectorGetBufferSize Computes the size of the work buffer for the ippiGradientVector{Sobel|Scharr|Prewitt} functions.

GradientVectorSobel

Computes gradient vectors of an image using the Sobel operator.

Syntax

```
IppStatus ippiGradientVectorSobel_<mod> (const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pGx, int gxStep, Ipp<dstDatatype>* pGy, int gyStep, Ipp<dstDatatype>*
pMag, int magStep, Ipp32f* pAngle, int angleStep, IppiSize dstRoiSize, IppiMaskSize
maskSize, IppNormType normType, IppiBorderType borderType, Ipp<srcDatatype>
borderValue, Ipp8u* pBuffer);
```

Supported values for mod:

8u16s_C1R	16s32f_C1R	16u32f_C1R	32f_C1R

```
IppStatus ippiGradientVectorSobel_<mod> (const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pGx, int gxStep, Ipp<dstDatatype>* pGy, int gyStep, Ipp<dstDatatype>*
pMag, int magStep, Ipp32f* pAngle, int angleStep, IppiSize dstRoiSize, IppiMaskSize
maskSize, IppNormType normType, IppiBorderType borderType, const Ipp<srcDatatype>
borderValue[3], Ipp8u* pBuffer);
```

Supported values for mod:

8u16s_C3C1R	16s32f_C3C1R	16u32f_C3C1R	32f_C3C1R

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance, in bytes, between the starting points of consecutive lines in the source image.
`pGx`	Pointer to the destination image ROI for the X component of the gradient vector (in Cartesian form).
`gxStep`	Distance, in bytes, between the starting points of consecutive lines in the X-component destination image.
`pGy`	Pointer to the destination image ROI for the Y component of the gradient vector (in Cartesian form).
`gyStep`	Distance, in bytes, between the starting points of consecutive lines in the Y-component destination image.
`pMag`	Pointer to the magnitude of the gradient destination image ROI (in polar gradient form).
`magStep`	Distance, in bytes, between the starting points of consecutive lines in the magnitude of the gradient destination image.
`pAngle`	Pointer to the destination image ROI for the angle (in polar gradient form).
`angleStep`	Distance, in bytes, between the starting points of consecutive lines in the angle destination image.
`dstRoiSize`	Size of the source and destination image ROI in pixels.
`maskSize`	Predefined mask of IppiMaskSize type.
`normType`	Normalization mode of IppNormType type.
`borderType`	Type of border. Possible values are: • `ippBorderConst` • Values of all border pixels are set to constant. • `ippBorderRepl` • Border is replicated from the edge pixels. • `ippBorderInMem` • Border is obtained from the source image pixels in memory. • `ippBorderMirror` • Border pixels are mirrored from the source image boundary pixels.

borderValue	Constant value to assign to pixels in the constant border (not applicable for other border types).
pBuffer	Pointer to the work buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

Before using this function, compute the size of the work buffer using the GradientVectorGetBufferSize function.

NOTE Any of the pGx, pGy, pMag, and pAngle output parameters can be NULL. This means that the parameter(s) is not requested.

This function operates on "gray" single-channel (C1 flavors) and color (C3 flavors) images.

Single-channel image (C1) input :

If input is a single-channel image, the ippiGradientVectorSobel function computes the gradient vector at each pixel of the source image ROI and stores the result either in Cartesian (pGx and pGy) and/or polar (pMag and pAngle) form, or any combination of these possible outputs.

Cartesian projections G_x and G_y are stored in the pGx and pGy buffers, respectively. The formulas below describe the algorithm for the 3x3 Sobel operator:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * A \qquad G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * A$$

for the 5x5 Sobel operator:

$$G_x = \begin{bmatrix} -1 & -2 & 0 & 2 & 1 \\ -4 & -8 & 0 & 8 & 4 \\ -6 & -12 & 0 & 12 & 6 \\ -4 & -8 & 0 & 8 & 4 \\ -1 & -2 & 0 & 2 & 1 \end{bmatrix} * A \qquad G_y = \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 0 & 8 & 12 & 8 & 2 \\ 2 & 0 & 0 & 0 & 0 \\ -2 & -8 & -12 & -8 & -2 \\ -1 & -4 & -6 & -4 & -1 \end{bmatrix} * A$$

where

- A is the source image
- * means two-dimensional convolution
- G_x and G_y are X and Y components of the gradient

The magnitude of the gradient is computed according to the normType value by the following formulas:

L1 normalization:

$$G = |G_x| + |G_y|$$

L2 normalization:

$$G = \sqrt{G_x^2 + G_y^2}$$

The value of angle between G_x and G_y is computed by the formula:

$$angle = \arctan \frac{G_y}{G_x}$$

Color image (C3) input :

If input is a color image, the `ippiGradientVectorSobel` function computes the spatial image derivatives G_x and G_y for each channel of the image using the specified differential operator. For each pixel (`x`, `y`) this function chooses the derivatives for which `L2`(G_x, G_y) is the maximal value and stores them in the `pGx` and `pGy` output arrays. In other words, for each pixel of a color image the function returns the derivatives composing the largest gradient across all channels.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when `pSrc` or `pBuffer` is NULL.

ippStsSizeErr	Indicates an error when one of the fields of dstRoiSize has a zero or negative value.
ippStsMaskSizeErr	Indicates an error when maskSize has an illegal value.
ippStsStepErr	Indicates an error when: • srcStep, gxStep, gyStep, magStep, or angleStep has a zero or negative value • srcStep, gxStep, gyStep, magStep, or angleStep is not a multiple of the image data size (4 for floating-point images or 2 for short integer images)
ippStsBadArgErr	Indicates an error when normType has an illegal value.
ippStsBorderErr	Indicates an error when borderType has an incorrect value.

Example

GradientVectorSobel1:

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

// A simple example of gradient vectors computation over an image with the Sobel
// operator using Intel(R) Integrated Performance Primitives (Intel(R) IPP) functions:
//     ippiGradientVectorGetBufferSize
//     ippiGradientVectorSobel_8u16s_C3C1R

#include <stdio.h>
#include "ipp.h"

#define WIDTH      8 /* source image width */
#define HEIGHT     4 /* source image height */
#define ROI_WIDTH  8 /* source image ROI width */
#define ROI_HEIGHT 4 /* source image ROI height */
#define NUMCHN     1 /* number of channels */

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
```

```
    exitLine: /* Label for Exit */
#define check_sts(st)        \
    if ((st) != ippStsNoErr) \
        goto exitLine; /* Go to Exit if Intel(R) IPP function returned status different from
ippStsNoErr */

static Ipp8u pSrc[(NUMCHN * WIDTH + 1) * HEIGHT] = /* Pointer to source image */
    {1, 2, 3, 4, 8, 8, 8, 8, 1, 2, 3, 4, 8, 8, 8, 8, 1, 2, 3, 4, 8, 8, 8, 8, 1, 2, 3, 4, 8, 8,
8, 8};

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status */

int main(void)
{
    IppStatus status = ippStsNoErr;
    int srcStep = WIDTH * NUMCHN;                                   /* Steps, in bytes, through
the source images */
    Ipp8u *psrc = pSrc + 1 * srcStep + 1 * NUMCHN;                  /* Pointer to source ROI image
*/
    IppiSize roiSize = {ROI_WIDTH, ROI_HEIGHT};                     /* Size of source ROI in
pixels */
    Ipp16s pGx[4 * 7], pGy[4 * 7];                                  /* Define x and y gradient
components */
    int gxStep = 4 * sizeof(Ipp16s), gyStep = 4 * sizeof(Ipp16s); /* Steps through the X and Y
component image */
    Ipp8u *pBuffer = NULL;                                          /* Pointer to the work buffer
*/
    Ipp8u borderValue = 0;
    int bufferSize = 0;
    Ipp16s pMag[ROI_WIDTH * ROI_HEIGHT];        /* Pointer to the magnitude of computed gradient
*/
    int magStep = ROI_WIDTH * sizeof(Ipp16s);   /* Step through the magnitude image */
    Ipp32f pAngle[ROI_WIDTH * ROI_HEIGHT];      /* Pointer to the angle of computed gradient */
    int angleStep = ROI_WIDTH * sizeof(Ipp32f); /* Step through the magnitude image */
    IppiMaskSize maskSize = ippMskSize3x3;
    IppiBorderType borderType = ippBorderConst;

    check_sts(status = ippiGradientVectorGetBufferSize(roiSize, maskSize, ipp8u, NUMCHN,
&bufferSize));

    pBuffer = ippsMalloc_8u(bufferSize);

    /* Compute gradient using 3x3 Sobel operator (source ROI at point x=1, y=1) */
    check_sts(status = ippiGradientVectorSobel_8u16s_C1R(psrc, srcStep, pGx, gxStep, pGy,
gyStep, pMag, magStep, pAngle, angleStep, roiSize, maskSize,
                                                         ippNormL2, borderType, borderValue,
pBuffer));

    EXIT_MAIN
    ippsFree(pBuffer);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

GradientVectorSobel2:

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

// A simple example of gradient vectors computation over an image using Sobel
// using Intel(R) Integrated Performance Primitives (Intel(R) IPP) functions:
//     ippiGradientVectorGetBufferSize
//     ippiGradientVectorSobel_8u16s_C3C1R

#include <stdio.h>
#include "ipp.h"

#define WIDTH      5 /* source image width */
#define HEIGHT     9 /* source image height */
#define ROI_WIDTH  3 /* source image ROI width */
#define ROI_HEIGHT 7 /* source image ROI height */
#define NUMCHN     3 /* number of channels */

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define check_sts(st)        \
    if ((st) != ippStsNoErr) \
        goto exitLine; /* Go to Exit if Intel(R) IPP function returned status different from
ippStsNoErr */

static Ipp8u pSrc[(NUMCHN * WIDTH + 1) * HEIGHT] = /* Pointer to source image */
    {
        0x62, 0xd6, 0x4b, 0x23, 0xc4, 0x66, 0x20, 0x1a, 0x15, 0x0e, 0xf1, 0xd5, 0x42, 0x4e,
0x5a, 0x00, 0x86, 0x1e, 0xb5, 0x2d, 0x16,
        0xfd, 0x30, 0xc7, 0x5e, 0x04, 0x1d, 0x36, 0x82, 0x9c, 0xb6, 0x00, 0x97, 0xd6, 0x16,
0x57, 0x3a, 0x1e, 0x0a, 0x55, 0xa0, 0xf3,
        0xc5, 0x97, 0x68, 0xe6, 0x64, 0x00, 0xde, 0x7d, 0x1c, 0x43, 0x4d, 0x57, 0xf8, 0xec,
0xe0, 0x9c, 0x30, 0xc3, 0xb9, 0xa2, 0x8c,
        0x00, 0x58, 0xb3, 0x10, 0x27, 0xac, 0x32, 0x59, 0x82, 0xaa, 0xe5, 0x24, 0x62, 0xd2,
0x26, 0x78, 0x00, 0x2f, 0x05, 0xd9, 0x90,
        0x01, 0x70, 0xd9, 0x5a, 0xdb, 0x36, 0x80, 0xca, 0xf7, 0x2f, 0x66, 0x00, 0xb6, 0x13,
0x6e, 0x5f, 0x72, 0x85, 0xc8, 0x97, 0x66,
        0x8b, 0x93, 0x9c, 0x74, 0x4b, 0x23, 0x00, 0x5d, 0xd3, 0x4a, 0x18, 0x43, 0x6e, 0xe9,
0x4b, 0xac, 0xdf, 0xc3, 0xa7, 0xef, 0x69,
        0xe1, 0x00, 0xdc, 0xfe, 0x20, 0x30, 0x55, 0x7a, 0x48, 0x90, 0xd7, 0x5e, 0xfd, 0x9d,
0xc5, 0x6d, 0x15, 0x00,
};

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
```

```
check and will return an appropriate status  */

int main(void)
{
    IppStatus status = ippStsNoErr;
    int srcStep = WIDTH * NUMCHN;                                   /* Steps, in bytes, through
the source images */
    Ipp8u *psrc = pSrc + 1 * srcStep + 1 * NUMCHN;                  /* Pointer to source ROI image
*/
    IppiSize roiSize = {ROI_WIDTH, ROI_HEIGHT};                     /* Size of source ROI in
pixels */
    Ipp16s pGx[4 * 7], pGy[4 * 7];                                  /* Define x and y gradient
components */
    int gxStep = 4 * sizeof(Ipp16s), gyStep = 4 * sizeof(Ipp16s); /* Steps through the X and Y
component image */
    Ipp8u *pBuffer = NULL;                                          /* Pointer to the work buffer
*/
    Ipp8u borderValue[3] = {0, 0, 0};
    int bufferSize = 0;
    Ipp16s pMag[ROI_WIDTH * ROI_HEIGHT];        /* Pointer to the magnitude of computed gradient
*/
    int magStep = ROI_WIDTH * sizeof(Ipp16s);   /* Step through the magnitude image */
    Ipp32f pAngle[ROI_WIDTH * ROI_HEIGHT];      /* Pointer to the angle of computed gradient */
    int angleStep = ROI_WIDTH * sizeof(Ipp32f); /* Step through the magnitude image */
    IppiMaskSize maskSize = ippMskSize3x3;
    IppiBorderType borderType = ippBorderInMem;

    check_sts(status = ippiGradientVectorGetBufferSize(roiSize, maskSize, ipp8u, NUMCHN,
&bufferSize));

    pBuffer = ippsMalloc_8u(bufferSize);

    /* Compute gradient using 3x3 Sobel operator (source ROI at point x=1, y=1) */
    check_sts(status = ippiGradientVectorSobel_8u16s_C3C1R(psrc, srcStep, pGx, gxStep, pGy,
gyStep, pMag, magStep, pAngle, angleStep, roiSize,
                                                           maskSize, ippNormL1, borderType,
borderValue, pBuffer));

    EXIT_MAIN
    ippsFree(pBuffer);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

See Also

Structures and Enumerators

Regions of Interest in Intel IPP

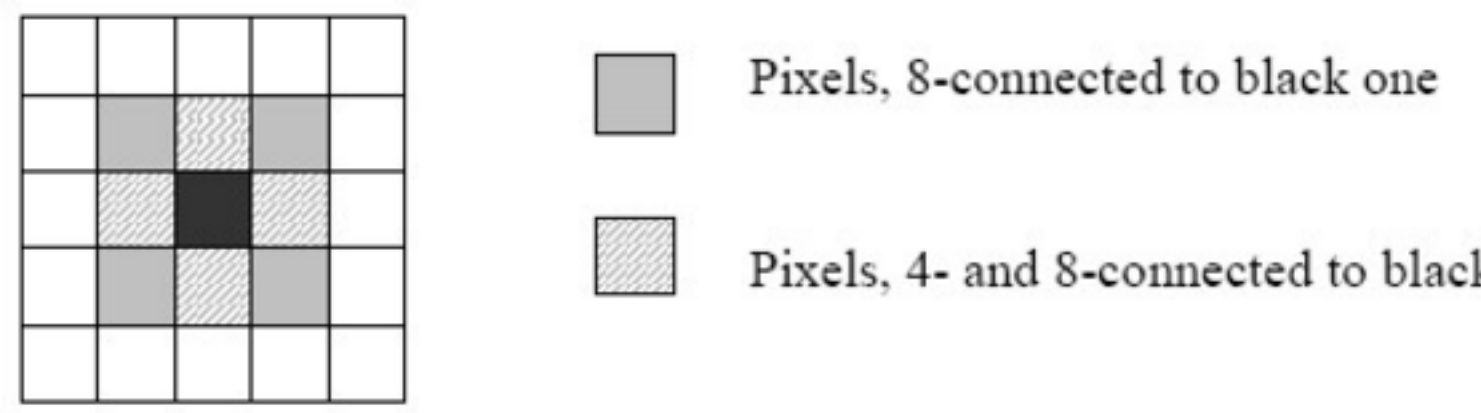
GradientVectorGetBufferSize Computes the size of the work buffer for the ippiGradientVector{Sobel|Scharr|Prewitt} functions.

Flood Fill Functions

This section describes functions performing flood filling of connected areas. *Flood filling* means that a group of connected pixels with close values is filled with, or is set to, a certain value. The flood filling process starts with a specified point ("seed") and continues until it reaches the image ROIboundary or cannot find any new pixels to fill due to a large difference in pixel values. For every pixel filled, the functions analyze neighbor pixels:

- 4 neighbors (except diagonal neighbors); this kind of connectivity is called 4-connectivity and the corresponding function name includes `4Con`, or
- 8 neighbors (diagonal neighbors included); this kind of connectivity is called 8-connectivity and the corresponding function name includes `8Con`.

Pixels Connectivity Patterns



These functions can be used for:

- segmenting a grayscale image into a set of uni-color areas,
- marking each connected component with individual color for bi-level images.

- FloodFillGetSize
- FloodFillGetSize_Grad
- FloodFill
- FloodFill_Grad
- FloodFill_Range

FloodFillGetSize

Calculates size of temporary buffer for flood filling operation.

Syntax

```
IppStatus ippiFloodFillGetSize(IppiSize roiSize, int* pBufSize);
```

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`roiSize`	Size of the source image ROI in pixels.

pBufSize	Pointer to the variable that returns the size of the temporary buffer.

Description

This function calculates the size of the temporary buffer to be used by the function ippiFloodFill.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if pBufSize pointer is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value.

FloodFillGetSize_Grad

Calculates size of temporary buffer for the gradient flood filling.

Syntax

```
IppStatus ippiFloodFillGetSize_Grad(IppiSize roiSize, int* pBufSize);
```

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

roiSize	Size of the source image ROI in pixels.
pBufSize	Pointer to the variable that returns the size of the temporary buffer.

Description

This function calculates the size of the temporary buffer to be used by the function ippiFloodFill_Grad.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if pBufSize pointer is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value.

FloodFill

Performs flood filling of connected area.

Syntax

Case 1: Operations on one-channel data

```
IppStatus ippiFloodFill_4Con_<mod>(Ipp<datatype>* pImage, int imageStep, IppiSize roiSize, IppiPoint seed, Ipp<datatype> newVal, IppiConnectedComp* pRegion, Ipp8u* pBuffer);
```

```
IppStatus ippiFloodFill_8Con_<mod>(Ipp<DataType>* pImage, int imageStep, IppiSize roiSize, IppiPoint seed, Ipp<datatype> newVal, IppiConnectedComp* pRegion, Ipp8u* pBuffer);
```

Supported values for `mod`:

`8u_C1IR`	`16u_C1IR`	`32s_C1IR`	`32f_C1IR`

Case 2: Operations on three-channel data

```
IppStatus ippiFloodFill_4Con_<mod>(Ipp<datatype>* pImage, int imageStep, IppiSize roiSize, IppiPoint seed, Ipp<datatype>* pNewVal, IppiConnectedComp* pRegion, Ipp8u* pBuffer);
```

```
IppStatus ippiFloodFill_8Con_<mod>(Ipp<DataType>* pImage, int imageStep, IppiSize roiSize, IppiPoint seed, Ipp<datatype>* pNewVal, IppiConnectedComp* pRegion, Ipp8u* pBuffer);
```

Supported values for `mod`:

`8u_C3IR`	`16u_C3IR`	`32f_C3IR`

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pImage`	Pointer to the ROI in the source and destination image (for the in-place operation).
`imageStep`	Distance in bytes between starts of consecutive lines in the image buffer.
`roiSize`	Size of the image ROI in pixels.
`seed`	Initial point.
`newVal`	Value to fill with for one-channel data.
`pNewVal`	Pointer to the vector containing values to fill with for three-channel data.

pRegion	Pointer to the connected components structure that stores information about the refilled area.
pBuffer	Pointer to the temporary buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function performs flood filling of the group of connected pixels whose pixel values are equal to the value in the seed point. Values of these pixel is set to the newVal value.

The function requires a temporary buffer whose size should be computed with the function ippiFloodFillGetSize beforehand.

The functions with the " _4con " suffixes check 4-connected neighborhood of each pixel, that is, side neighbors. The functions with the " _8con " suffixes check 8-connected neighborhood of each pixel, that is, side and corner neighbors. See Figure Pixels Connectivity Patterns".

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if imageStep is less than pRoiSize.width*<pixelSize>.
ippStsNotEvenStepErr	Indicates an error condition if steps for floating-point images are not divisible by 4, or steps for 16-bit integer images are not divisible by 2.
ippStsOutOfRangeErr	Indicates an error condition if the seed point is out of ROI.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

// A simple example of flood filling of connected area
```

```
// using Intel(R) Integrated Performance Primitives (Intel(R) IPP) functions:
//     ippiFloodFillGetSize
//     ippiFloodFill_4Con_32s_C1IR
//     ippiFloodFill_8Con_32s_C1IR

#include <stdio.h>
#include "ipp.h"

#define WIDTH  8 /* source image width */
#define HEIGHT 8 /* source image height */

static Ipp32s pImage[WIDTH * HEIGHT] = /* Pointer to ROI of initial image */
    {0, 10, 90, 0, 0, 0,  0,  0,  0, 10, 10, 90, 0, 0,  0, 0,  0, 10, 0, 10, 0, 0,  0,  0,  0,
10, 10, 90, 0,  0, 0, 0,
     0, 0,  10, 0, 0, 10, 10, 10, 0, 10, 0,  0,  0, 10, 0, 10, 0, 0,  0, 10, 0, 10, 10, 10, 0,
0,  0,  0,  10, 0, 0, 0};

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define check_sts(st)        \
    if ((st) != ippStsNoErr) \
        goto exitLine; /* Go to Exit if Intel(R) IPP function returned status different from
ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main(void)
{
    IppStatus status = ippStsNoErr;
    int imageStep = WIDTH * sizeof(Ipp32s);            /* Steps, in bytes, through the source
images */
    IppiSize roiSize = {WIDTH, HEIGHT};                /* Size of source ROI in pixels */
    IppiConnectedComp pRegion = {0};                   /* Pointer to repainted region
properties structure */
    IppiPoint seedPoint1 = {1, 0}, seedPoint2 = {5, 4}; /* Coordinates of the seed point inside
image ROI */
    int Point1NewValue = 1, Point2NewValue = 2;        /* Value to fill with for one-channel
data */
    Ipp8u *pBuffer = NULL;                             /* Pointer to the work buffer */
    int bufferSize = 0;

    /* Calculate size of temporary buffer for flood filling operation */
    check_sts(status = ippiFloodFillGetSize(roiSize, &bufferSize));

    pBuffer = ippsMalloc_8u(bufferSize);

    /* Perform flood filling of 4-connected area */
    check_sts(status = ippiFloodFill_4Con_32s_C1IR(pImage, imageStep, roiSize, seedPoint1,
Point1NewValue, &pRegion, pBuffer));

    /* Perform flood filling of 8-connected area */
    check_sts(status = ippiFloodFill_8Con_32s_C1IR(pImage, imageStep, roiSize, seedPoint2,
Point2NewValue, &pRegion, pBuffer));

    EXIT_MAIN
    ippsFree(pBuffer);
```

```
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

FloodFill_Grad

Performs gradient flood filling of connected area on an image.

Syntax

Case 1: Operations on one-channel data

```
IppStatus ippiFloodFill_Grad4Con_<mod>(Ipp<DataType>* pImage, int imageStep, IppiSize
roiSize, IppiPoint seed, Ipp<datatype> newVal, Ipp<datatype> minDelta, Ipp<datatype>
maxDelta, IppiConnectedComp* pRegion, Ipp8u* pBuffer);
```

```
IppStatus ippiFloodFill_Grad8Con_<mod>(Ipp<DataType>* pImage, int imageStep, IppiSize
roiSize, IppiPoint seed, Ipp<datatype> newVal, Ipp<datatype> minDelta, Ipp<datatype>
maxDelta, IppiConnectedComp* pRegion, Ipp8u* pBuffer);
```

Supported values for `mod`:

`8u_C1IR`	`16u_C1IR`	`32f_C1IR`

Case 2: Operations on three-channel data

```
IppStatus ippiFloodFill_Grad4Con_<mod>(Ipp<DataType>* pImage, int imageStep, IppiSize
roiSize, IppiPoint seed, Ipp<datatype>* pNewVal, Ipp<datatype>* pMinDelta,
Ipp<datatype>* pMaxDelta, IppiConnectedComp* pRegion, Ipp8u* pBuffer);
```

```
IppStatus ippiFloodFill_Grad8Con_<mod>(Ipp<DataType>* pImage, int imageStep, IppiSize
roiSize, IppiPoint seed, Ipp<datatype>* pNewVal, Ipp<datatype>* pMinDelta,
Ipp<datatype>* pMaxDelta, IppiConnectedComp* pRegion, Ipp8u* pBuffer);
```

Supported values for `mod`:

`8u_C3IR`	`16u_C3IR`	`32f_C3IR`

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pImage`	Pointer to the ROI in the source and destination image (in-place operation).
`imageStep`	Distance in bytes between starts of consecutive lines in the image buffer.
`roiSize`	Size of the image ROI in pixels.
`seed`	Initial point.

minDelta	Minimum difference between neighbor pixels for one-channel data.
maxDelta	Maximum difference between neighbor pixels for one-channel data.
newVal	Value to fill with for one-channel data.
pMinDelta	Pointer to the minimum differences between neighbor pixels for three-channel images.
pMaxDelta	Pointer to the maximum differences between neighbor pixels for three-channel images.
pNewVal	Pointer to the vector containing values to fill with for three-channel data.
pRegion	Pointer to the connected components structure that stores information about the refilled area.
pBuffer	Pointer to the temporary buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function performs flood filling
of the group of connected pixels in the seed pixel neighborhoods
whose pixel values v satisfy the following conditions:

$$v_0 - d_{lw} \le v \le v_0 + d_{up}$$

where v_0 is the value of at least one of the current pixel neighbors, which already belongs to the refilled area, and d_{lw}, d_{up} are minDelta, maxDelta, respectively. Values of these pixels are set to the newVal value.

The function requires a temporary buffer whose size should be computed with the function ippiFloodFillGetSize_Grad beforehand.

The functions with the " _4con " suffixes check 4-connected neighborhood of each pixel, i.e., side neighbors. The functions with the " _8con " suffixes check 8-connected neighborhood of each pixel, i.e., side and corner neighbors. See Figure Pixels Connectivity Patterns".

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value.

ippStsStepErr	Indicates an error condition if imageStep is less than pRoiSize.width*<pixelSize>.
ippStsNotEvenStepErr	Indicates an error condition if steps for floating-point images are not divisible by 4, or steps for 16-bit integer images are not divisible by 2.
ippStsOutOfRangeErr	Indicates an error condition if the seed point is out of ROI.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

// A simple example of gradient flood filling of connected area
// using Intel(R) Integrated Performance Primitives (Intel(R) IPP) functions:
//     ippiFloodFillGetSize_Grad
//     ippiFloodFill_Grad4Con_8u_C1IR

#include <stdio.h>
#include "ipp.h"

#define WIDTH  8 /* source image width */
#define HEIGHT 8 /* source image height */

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define check_sts(st)        \
    if ((st) != ippStsNoErr) \
        goto exitLine; /* Go to Exit if Intel(R) IPP function returned status different from
ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

static Ipp8u pImage[WIDTH * HEIGHT] = /* Pointer to ROI of initial image */
    {0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, 3, 4, 5, 6, 7,
     0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, 3, 4, 5, 6, 7,
     0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, 3, 4, 5, 6, 7,
     0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, 3, 4, 5, 6, 7,
     0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, 3, 4, 5, 6, 7,
     0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, 3, 4, 5, 6, 7,
     0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, 3, 4, 5, 6, 7,
     0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, 3, 4, 5, 6, 7};

int main(void)
{
```

```
    IppStatus status = ippStsNoErr;
    int imageStep = WIDTH;              /* Steps, in bytes, through the source images */
    IppiSize roiSize = {WIDTH, HEIGHT}; /* Size of source ROI in pixels */
    IppiConnectedComp pRegion;          /* Pointer to repainted region properties structure */
    IppiPoint seedPoint = {1, 0};       /* Coordinates of the seed point inside image ROI */
    Ipp8u PointNewValue = 1;            /* Value to fill with for one-channel data */
    Ipp8u *pBuffer = NULL;              /* Pointer to the work buffer */
    Ipp8u minDelta = 1, maxDelta = 1;   /* Minimum/maximum difference between neighbour pixels
for one-channel data */
    int bufferSize = 0;

    /* Calculate size of temporary buffer for flood filling operation */
    check_sts(status = ippiFloodFillGetSize_Grad(roiSize, &bufferSize));

    pBuffer = ippsMalloc_8u(bufferSize);

    /* Perform gradient flood filling of 4-connected area */
    check_sts(status = ippiFloodFill_Grad4Con_8u_C1IR(pImage, imageStep, roiSize, seedPoint,
PointNewValue, minDelta, maxDelta, &pRegion, pBuffer));

    EXIT_MAIN
    ippsFree(pBuffer);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

FloodFill_Range

Performs flood filling of pixels with values in the specified range in the connected area on an image.

Syntax

Case 1: Operations on one-channel data

```
IppStatus ippiFloodFill_Range4Con_<mod>(Ipp<DataType>* pImage, int imageStep, IppiSize
roiSize, IppiPoint seed, Ipp<datatype> newVal, Ipp<datatype> minDelta, Ipp<datatype>
maxDelta, IppiConnectedComp* pRegion, Ipp8u* pBuffer);
```

```
IppStatus ippiFloodFill_Range8Con_<mod>(Ipp<DataType>* pImage, int imageStep, IppiSize
roiSize, IppiPoint seed, Ipp<datatype> newVal, Ipp<datatype> minDelta, Ipp<datatype>
maxDelta, IppiConnectedComp* pRegion, Ipp8u* pBuffer);
```

Supported values for `mod`:

`8u_C1IR`	`16u_C1IR`	`32f_C1IR`

Case 2: Operations on three-channel data

```
IppStatus ippiFloodFill_Range4Con_<mod>(Ipp<DataType>* pImage, int imageStep, IppiSize
roiSize, IppiPoint seed, Ipp<datatype>* pNewVal, Ipp<datatype>* pMinDelta,
Ipp<datatype>* pMaxDelta, IppiConnectedComp* pRegion, Ipp8u* pBuffer);
```

```
IppStatus ippiFloodFill_Range8Con_<mod>(Ipp<DataType>* pImage, int imageStep, IppiSize
roiSize, IppiPoint seed, Ipp<datatype>* pNewVal, Ipp<datatype>* pMinDelta,
Ipp<datatype>* pMaxDelta, IppiConnectedComp* pRegion, Ipp8u* pBuffer);
```

Supported values for `mod`:

8u_C3IR	16u_C3IR	32f_C3IR

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pImage	Pointer to the ROI in the source and destination image (in-place operation).
imageStep	Distance in bytes between starts of consecutive lines in the image buffer.
roiSize	Size of the image ROI in pixels.
seed	Initial point.
minDelta	Minimum difference between neighbor pixels for one-channel data.
maxDelta	Maximum difference between neighbor pixels for one-channel data.
newVal	Value to fill with for one-channel data.
pMinDelta	Pointer to the minimum differences between neighbor pixels for three-channel images.
pMaxDelta	Pointer to the maximum differences between neighbor pixels for three-channel images.
pNewVal	Pointer to the vector containing values to fill with for three-channel data.
pRegion	Pointer to the connected components structure that stores information about the refilled area.
pBuffer	Pointer to the temporary buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function performs flood filling
of the group of connected pixels in the seed pixel neighborhoods
whose pixel values v satisfy the following conditions:

$$v_0 - d_{lw} \le v \le v_0 + d_{up}$$

where v_0 is the pixel value of the `seed` point, and d_{lw}, d_{up} are `minDelta`, `maxDelta`, respectively. Values of these pixels are set to the `newVal` value.

The function requires a temporary buffer whose size should be computed with the function ippiFloodFillGetSize_Grad beforehand.

The functions with the " `_4con` " suffixes check 4-connected neighborhood of each pixel, i.e., side neighbors. The functions with the " `_8con` " suffixes check 8-connected neighborhood of each pixel, i.e., side and corner neighbors. See Figure Pixels Connectivity Patterns".

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize` has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if `imageStep` is less than `pRoiSize.width*<pixelSize>`.
`ippStsNotEvenStepErr`	Indicates an error condition if steps for floating-point images are not divisible by 4, or steps for 16-bit integer images are not divisible by 2.
`ippStsOutOfRangeErr`	Indicates an error condition if the `seed` point is out of ROI.

Motion Analysis and Object Tracking

- FGMMGetBufferSize
- FGMMInit
- FGMMForeground
- FGMMBackground
- Motion Template Functions
- Optical Flow

FGMMGetBufferSize

Computes the size of the state structure for the Gaussian mixture model foreground/background subtraction.

Syntax

```
IppStatus ippiFGMMGetBufferSize_8u_C3R(IppiSize roi, int maxNGauss, int* pSpecSize);
```

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

roi	Size of the source image ROI, in pixels.
maxNGauss	Maximal size of the Gaussian mixture components.
pSpecSize	Pointer to the size of the IppFGMMSpec_8u_C3R structure.

Description

This function operates with ROI.

This function computes the size of the IppFGMMSpec_8u_C3R structure for the FGMMForeground and FGMMBackground functions.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pSpecSize is NULL.
ippStsSizeErr	Indicates an error when roi is less than, or equal to zero.
ippStsBadArgErr	Indicates an error when maxNumGauss is less than, or equal to zero.

See Also

Regions of Interest in Intel IPP

FGMMForeground Performs the Gaussian mixture model foreground subtraction.

FGMMBackground Returns the updated background image.

FGMMInit

Initializes the state structure for the Gaussian mixture model foreground/background subtraction.

Syntax

```
IppStatus ippiFGMMInit_8u_C3R(IppiSize roi, int maxNGauss, IppFGMModel* pModel,
IppFGMMState_8u_C3R* pState);
```

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

roi	Size of the source image ROI, in pixels.
maxNGauss	Maximal size of the Gaussian mixture components.

pModel	Pointer to the IppFGMModel structure containing parameters for the model. If pModel is NULL, the default parameters are applied.
pState	Pointer to the IppFGMMState_8u_C3R state structure.

Description

This function operates with ROI.

This function initializes the IppFGMMState_8u_C3R state structure for the FGMMForeground and FGMMBackground functions.

Before using this function, you need to compute the size of the state structure using the FGMMGetBufferSize function.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pModel or pState is NULL.
ippStsSizeErr	Indicates an error when roi is less than, or equal to zero.
ippStsBadArgErr	Indicates an error when maxNGauss is less than, or equal to zero.

See Also

Regions of Interest in Intel IPP

FGMMForeground Performs the Gaussian mixture model foreground subtraction.

FGMMBackground Returns the updated background image.

FGMMGetBufferSize Computes the size of the state structure for the Gaussian mixture model foreground/background subtraction.

FGMMForeground

Performs the Gaussian mixture model foreground subtraction.

Syntax

```
IppStatus ippiFGMMForeground_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep, IppiSize roi, IppFGMMState_8u_C3R* pState, IppFGMModel* pModel, double learning_rate);
```

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Pointer to the source image.
srcStep	Distance, in bytes, between the starting points of consecutive lines in the source image.
pDst	Pointer to the one-channel Ipp8u mask of foreground.
dstStep	Distance, in bytes, between the starting points of consecutive lines in the destination image.
roi	Size of the source image ROI, in pixels.
learning_rate	Speed of algorithm learning.
pState	Pointer to the IppFGMMState_8u_C3R state structure.
pModel	Pointer to the IppFGMModel structure containing parameters for the model. If pModel is NULL, the parameters are the same as in a previous call.

Description

This function operates with ROI.

This function implements the Gaussian mixture model foreground subtraction described in [ZIVKOVIC04]. The foreground mask is stored in pDst.

Before using this function, you need to compute the size of the IppFGMMState_8u_C3R state structure and initialize the structure using the FGMMGetBufferSize and FGMMInit functions, respectively.

For an example on how to use this function, refer to the example provided with the FGMMBackground function description.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when pSrc, pDst, pModel, or pState is NULL.
ippStsSizeErr	Indicates an error when roi is less than, or equal to zero.
ippStsStepErr	Indicates an error when srcStep or dstStep is less than, or equal to zero.

See Also

Regions of Interest in Intel IPP

FGMMGetBufferSize Computes the size of the state structure for the Gaussian mixture model foreground/background subtraction.

FGMMInit Initializes the state structure for the Gaussian mixture model foreground/background subtraction.

FGMMBackground Returns the updated background image.

FGMMBackground

Returns the updated background image.

Syntax

```
IppStatus ippiFGMMBackground_8u_C3R(Ipp8u* pDst, int dstStep, IppiSize roi,
IppFGMMState_8u_C3R* pState);
```

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pDst`	Pointer to the three-channel background image.
`dstStep`	Distance, in bytes, between the starting points of consecutive lines in the destination image.
`roi`	Size of the source image ROI, in pixels.
`pState`	Pointer to the `IppFGMMState_8u_C3R` state structure.

Description

This function implements the Gaussian mixture model background subtraction described in [ZIVKOVIC04]. The function returns the three-channel `Ipp8u` background image.

Before using this function, you need to compute the size of the `IppFGMMState_8u_C3R` state structure and initialize the structure using the FGMMGetBufferSize and FGMMInit functions, respectively.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when `pDst` or `pState` is `NULL`.
`ippStsSizeErr`	Indicates an error when `roi` is less than, or equal to zero.
`ippStsStepErr`	Indicates an error when `dstStep` is less than, or equal to zero.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
```

```
warranties, other than
 * those that are expressly stated in the License.
 *****************************************************************************/

// A simple example of image foreground/background segmentation
// using Intel(R) Integrated Performance Primitives (Intel(R) IPP) functions:
//     ippiFGMMGetBufferSize_8u_C3R
//     ippiFGMMInit_8u_C3R
//     ippiFGMMForeground_8u_C3R
//     ippiFGMMBackground_8u_C3R

#include <stdio.h>
#include "ipp.h"

#define WIDTH      32 /* image width  */
#define HEIGHT     32 /* image height */
#define NUM_FRAME  100
#define MAX_N_RECT 3 /* number of rectangles */

static struct {
    int x;
    int y;
    int w;
    int h;
    int dx;
    int dy;
    Ipp8u color[3];
} rect[MAX_N_RECT]; /* Parameters of rectangle */

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define check_sts(st)        \
    if ((st) != ippStsNoErr) \
        goto exitLine; /* Go to Exit if Intel(R) IPP function returned status different from
ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main(void)
{
    int r = 0, frame = 0;
    IppStatus status = ippStsNoErr;
    Ipp8u *pSrc = NULL, *pDstF = NULL, *pDstB = NULL; /* Pointers to source/foreground/
background images */
    int srcStep = 0, dstFStep = 0, dstBStep = 0;      /* Steps, in bytes, through the source/
foreground/background images */
    IppiSize roiSize = {WIDTH, HEIGHT};               /* Size of source ROI in pixels */
    int specSize = 0;                                 /* Size of the working buffer and spec
structure */
    IppFGMMState_8u_C3R *pState = NULL;               /* Pointer to the state */
    IppFGMModel pModel = {0};                         /* Pointer to the parameters structure */
    Ipp8u value[3] = {0, 0, 0};
    int maxNGauss = 5; /* Maximum number of allowed Gaussians */

    pSrc = ippiMalloc_8u_C3(roiSize.width, roiSize.height, &srcStep);
    pDstF = ippiMalloc_8u_C1(roiSize.width, roiSize.height, &dstFStep);
```

```
    pDstB = ippiMalloc_8u_C3(roiSize.width, roiSize.height, &dstBStep);

    check_sts(status = ippiFGMMGetBufferSize_8u_C3R(roiSize, maxNGauss, &specSize));

    pState = (IppFGMMState_8u_C3R *)ippsMalloc_8u(specSize);

    pModel.numFrames = 50;
    pModel.varInit = 15.0;
    pModel.varMin = 4.0;
    pModel.varMax = 75.0;
    pModel.varWBRatio = 4.0f * 4.0f;
    pModel.bckgThr = 0.9f;
    pModel.varNGRatio = 3.0f * 3.0f;
    pModel.reduction = 0.05f;
    pModel.shadowValue = 127;
    pModel.shadowFlag = 1;
    pModel.shadowRatio = 0.5;
    pModel.maxNGauss = maxNGauss;

    check_sts(status = ippiFGMMInit_8u_C3R(roiSize, maxNGauss, &pModel, pState));

    for (r = 0; r < MAX_N_RECT; r++) {
        rect[r].x = 0;
        rect[r].y = 0;
        rect[r].w = 10 + r;
        rect[r].h = 5 + r;
        rect[r].dx = 1;
        rect[r].dy = 1;
        rect[r].color[0] = (Ipp8u)(10 * r);
        rect[r].color[1] = (Ipp8u)(20 * r);
        rect[r].color[2] = (Ipp8u)(3 * r);
    }
    for (frame = 0; frame < NUM_FRAME; frame++) {
        Ipp8u valueRect[3] = {0, 0, 0};
        IppiSize roiRectSize = {0};
        /* Draw rectangle to simulate moving object */
        check_sts(status = ippiSet_8u_C3R(value, pSrc, srcStep, roiSize)); /* zero image */

        for (r = 0; r < MAX_N_RECT; r++) {
            valueRect[0] = (Ipp8u)(rect[r].color[0] + frame);
            valueRect[1] = (Ipp8u)(rect[r].color[1] + frame);
            valueRect[2] = (Ipp8u)(rect[r].color[2] + frame);
            roiRectSize.width = rect[r].w;
            roiRectSize.height = rect[r].h;
            check_sts(status = ippiSet_8u_C3R(valueRect, pSrc + srcStep * rect[r].y + 3 *
rect[r].x, srcStep, roiRectSize));

            if (rect[r].x + rect[r].w >= roiSize.width) {
                rect[r].dx = -1;
            } else if (rect[r].x <= 0) {
                rect[r].dx = 1;
            }
            if (rect[r].y + rect[r].h >= roiSize.height) {
                rect[r].dy = -1;
            } else if (rect[r].y == 0) {
                rect[r].dy = 1;
            }
            rect[r].x += rect[r].dx;
```

```
            rect[r].y += rect[r].dy;
        }
        check_sts(status = ippiFGMMForeground_8u_C3R(pSrc, srcStep, pDstF, dstFStep, roiSize,
pState, &pModel, -1));

        check_sts(status = ippiFGMMBackground_8u_C3R(pDstB, dstBStep, roiSize, pState));
    }

    EXIT_MAIN
    ippiFree(pSrc);
    ippiFree(pDstF);
    ippiFree(pDstB);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

See Also

Regions of Interest in Intel IPP

FGMMGetBufferSize Computes the size of the state structure for the Gaussian mixture model foreground/background subtraction.

FGMMInit Initializes the state structure for the Gaussian mixture model foreground/background subtraction.

Motion Template Functions

This section describes a motion templates function. This function generates motion templates images to rapidly determine where, how, and in which direction the motion occurred. The algorithms are based on [Davis97], and [Davis99]. The function operates on images that are the output of frame or background differencing, or other image segmentation operations. Thus, the input and output image types are all grayscale, that is, one color channel. The pixel types can be `8u`, `8s`, or `32f`.

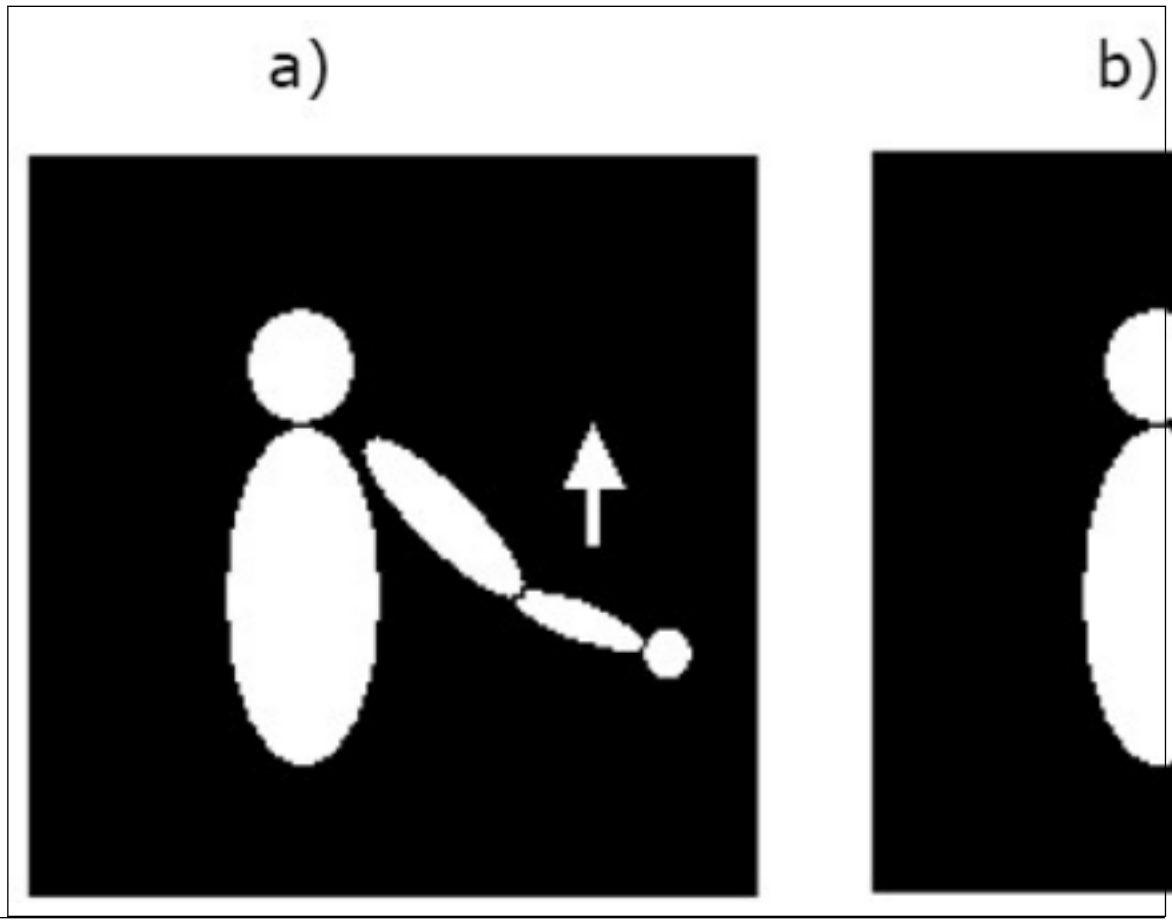
- Motion Representation
- Updating MHI Images
- UpdateMotionHistory

Motion Representation

Figure Motion Image History (a) shows capturing a foreground silhouette of the moving object or person. As the person or object moves, copying the most recent foreground silhouette as the highest values in the motion history image creates a “layered history” of the resulting motion. Typically, this “highest value” is just a floating-point timestamp of time since the code has been running in milliseconds. Figure Motion Image

History (b) shows the result that may be called the `Motion History Image (MHI)`. The MHI in Figure Motion Image History" represents how the motion took place. A pixel level or a time delta threshold, as appropriate, is set such that pixel values in the MHI that fall below that threshold are set to zero.

Motion Image History



The most recent motion has the highest value, earlier motions have decreasing values subject to a threshold below which the value is set to zero.

Updating MHI Images

Generally, floating point images are used because system time differences, that is, time elapsing since the application was launched, are read in milliseconds to be further converted into a floating point number which is the value of the most recent silhouette. Then follows writing this current silhouette over the past silhouettes with subsequent thresholding away pixels that are too old to create the MHI.

UpdateMotionHistory

Updates motion history image using motion silhouette at given timestamp.

Syntax

```
IppStatus ippiUpdateMotionHistory_<mod>(const Ipp<srcDatatype>* pSilhouette, int
silhStep, Ipp32f* pMhi, int mhiStep, IppiSize roiSize, Ipp32f timeStamp, Ipp32f
mhiDuration);
```

Supported values for `mod`:

`8u32f_C1IR`	`16u32f_C1IR`	`32f_C1IR`

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSilhouette`	Pointer to the silhouette image ROI that has non-zero values for those pixels where the motion occurs.
`roiSize`	Size of the image ROI in pixels.
`silhStep`	Distance in bytes between starts of consecutive lines in the silhouette image.
`pMhi`	Pointer to the motion history image which is both an input and output parameter.
`mhiStep`	Distance in bytes between starts of consecutive lines in the motion history image.
`timeStamp`	Timestamp in milliseconds.
`mhiDuration`	Threshold for MHI pixels. MHI motions older than this threshold are deleted.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function updates the motion history image. It sets MHI pixels to the current `timestamp` value, if their values are non-zero.

The function deletes MHI pixels, if their values are less than the `mhiDuration` timestamp, that is, the pixels are "old."

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize` has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if `mhiStep` or `silhStep` is less than `roiSize.width * <pixelSize>`.

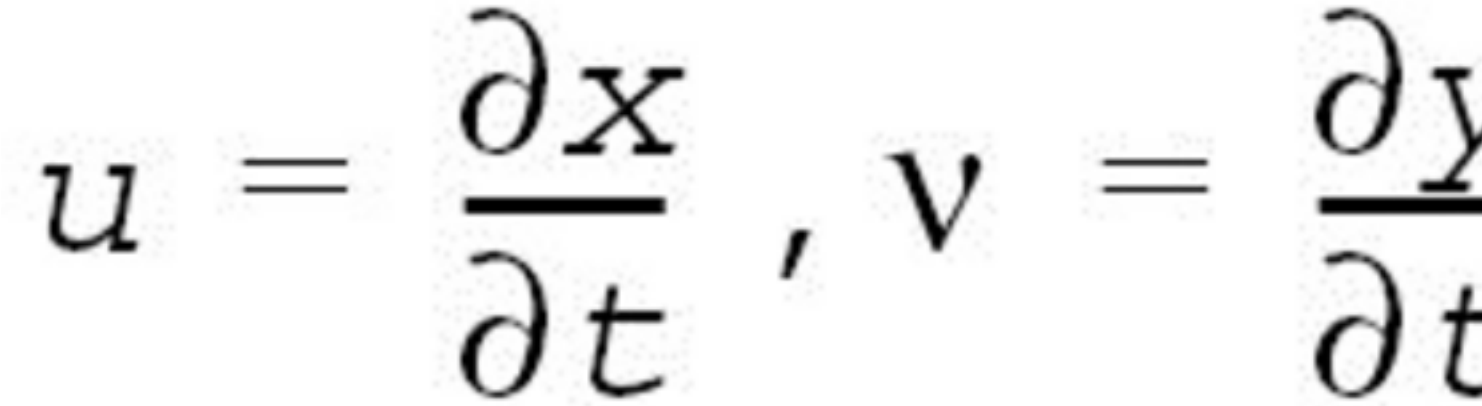
ippStsNotEvenStepErr	Indicates an error condition if the step value is not divisible by 2 for 16u images, and by 4 for 32f images.
ippStsOutOfRangeErr	Indicates an error when mhiDuration is negative.

Optical Flow

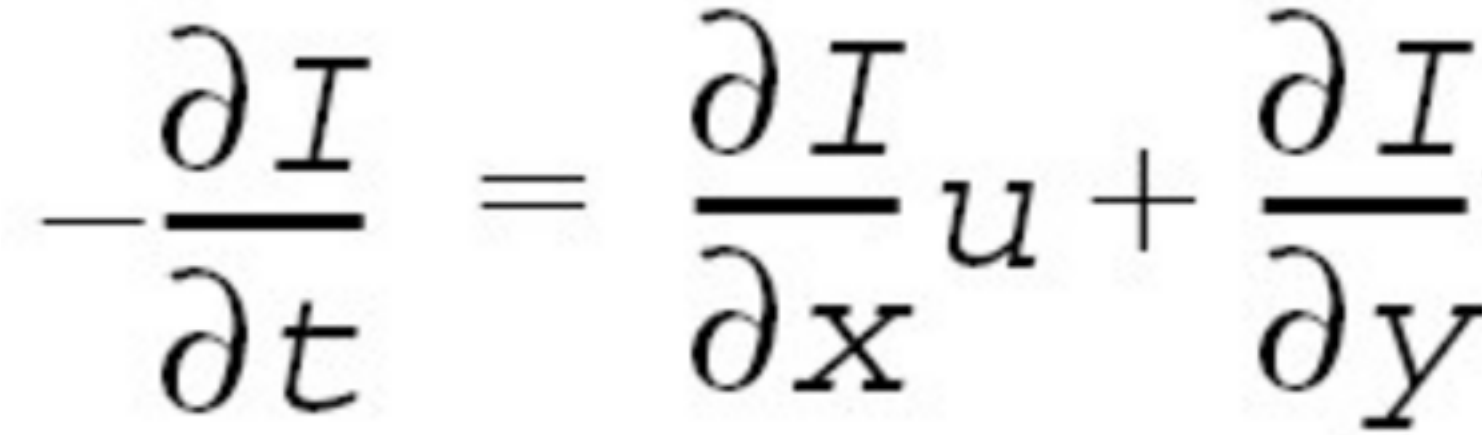
This section describes the functions that calculate the optical flow using the pyramidal Lucas-Kanade algorithm [Bou99].

The optical flow between two images is generally defined as an apparent motion of image brightness. Let I(x, y, t) be the image brightness that changes in time to provide an image sequence.

Optical flow coordinates



can be found from so called optical flow constraint equation:



The Lucas-Kanade algorithm assumes that the group of adjacent pixels has the same velocity and founds the approximate solution of the above equation using the least square method.

- OpticalFlowPyrLKGetSize
- OpticalFlowPyrLKInit

- OpticalFlowPyrLK

OpticalFlowPyrLKGetSize

Computes the size of the pyramidal optical flow state structure.

Syntax

```
IppStatus ippiOpticalFlowPyrLKGetSize(int winSize, IppiSize roiSize, IppDataType dataType, IppHintAlgorithm hint, int* pStateSize);
```

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`winSize`	Size of the search window of each pyramid level.
`roiSize`	Maximal size of the source image (zero level of the pyramid) ROI, in pixels.
`dataType`	Data type of the image. Possible values: `ipp8u`, `ipp16u`, or `ipp32f`.
`hint`	Option to select the algorithmic implementation of the transform function.
`pStateSize`	Pointer to the size of the state struture.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function computes the size of the `pState` structure that is required to calculate the optical flow between two images in the centered window of size `winSize * winSize` using the pyramidal Lucas-Kanade [Bou99] algorithm. The `hint` argument specifies the computation algorithm. The `pState` structure is used by the ippiOpticalFlowPyrLK function and can be applied to process images with size not greater than `roiSize`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error when `pStateSize` is `NULL`.
`ippStsDataTypeErr`	Indicates an error when `dataType` has an illegal value.
`ippStsSizeErr`	Indicates an error when `roiSize` has a field with a zero or negative value; or when `winSize` is less than, or equal to zero.

See Also

Regions of Interest in Intel IPP

OpticalFlowPyrLK Calculates optical flow for the set of feature points using the pyramidal Lucas-Kanade algorithm.

OpticalFlowPyrLKInit Initializes the state structure for optical flow calculation.

OpticalFlowPyrLKInit

Initializes the state structure for optical flow calculation.

Syntax

```
IppStatus ippiOpticalFlowPyrLKInit_<mod>(IppiOptFlowPyrLK_<mod>** ppState, IppiSize roiSize, int winSize, IppHintAlgorithm hint, Ipp8u* pStateBuf);
```

Supported values for `mod`:

`8u_C1R`	`16u_C1R`	`32f_C1R`

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`ppState`	Double pointer to the optical flow state structure to be initialized.
`roiSize`	Maximal size of the source image (zero level of the pyramid) ROI, in pixels.
`winSize`	Size of the search window of each pyramid level.
`hint`	Option to select the algorithmic implementation of the transform function.
`pStateBuf`	Pointer to the work buffer for the state structure.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function initializes the `ppState` structure that is required to calculate the optical flow between two images in the centered window of size `winSize * winSize` using the pyramidal Lucas-Kanade [Bou99] algorithm. The `hint` argument specifies the computation algorithm. The `ppState` structure is used by the ippiOpticalFlowPyrLK function and can be applied to process images with size not greater than `roiSize`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error when `ppState` is `NULL`.

ippStsSizeErr	Indicates an error when roiSize has a field with a zero or negative value; or when winSize is less than, or equal to zero.

See Also

Regions of Interest in Intel IPP

OpticalFlowPyrLK Calculates optical flow for the set of feature points using the pyramidal Lucas-Kanade algorithm.

OpticalFlowPyrLKGetSize Computes the size of the pyramidal optical flow state structure.

OpticalFlowPyrLK

Calculates optical flow for the set of feature points using the pyramidal Lucas-Kanade algorithm.

Syntax

```
IppStatus ippiOpticalFlowPyrLK_<mod>(IppiPyramid* pPyr1, IppiPyramid* pPyr2, const IppiPoint_32f* pPrev, IppiPoint_32f* pNext, Ipp8s* pStatus, Ipp32f* pError, int numFeat, int winSize, int maxLev, int maxIter, Ipp32f threshold, IppiOptFlowPyrLK_<mod>* pState);
```

Supported values for mod:

8u_C1R	16u_C1R	32f_C1R

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pPyr1	Pointer to the ROI in the first image pyramid structure.
pPyr2	Pointer to the ROI in the second image pyramid structure.
pPrev	Pointer to the array of initial coordinates of the feature points.
pNext	Pointer to the array of new coordinates of feature point; as input it contains hints for new coordinates.
pStatus	Pointer to the array of result indicators.
pError	Pointer to the array of differences between neighborhoods of old and new point positions.
numFeat	Number of feature points.
winSize	Size of the search window of each pyramid level.
maxLev	Pyramid level to start the operation.

maxIter	Maximum number of algorithm iterations for each pyramid level.
threshold	Threshold value.
pState	Pointer to the pyramidal optical flow structure.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function implements the iterative version of the Lucas-Kanade algorithms [Bou99]. It computes with sub-pixel accuracy new coordinates of the numFeat feature points of two images at time t and t+dt. Their initial coordinates are places in the pPrev array. Computed values of new coordinates of the feature points are stored in the array pNext, that initially contains estimations of them (or hints), for example, based on the flow values for the previous image in sequence. If there are not such hints, the pNext array contains the same initial coordinates as the pPrev array.

pStatus and pError are arrays of size numFeat with the respective data type.

The images are presented by the pyramid structures pPyr1 and pPyr2 respectively (see description of the PyramidGetSize and ippiPyramidInit functions for more details). These structures should be initialized by calling the function PyramidGetSize and ippiPyramidInit functions beforehand. The function uses the pyramidal optical flow structure pState that also should be previously initialized using OpticalFlowPyrLKGetSize and OpticalFlowPyrLKInit.

The function starts operation on the highest pyramid level (smallest image) that is specified by the maxLev parameter in the centered search window which size winSize could not exceed the corresponding value winSize that is specified in OpticalFlowPyrLKGetSize and OpticalFlowPyrLKInit. The operation for i – th feature point on the given pyramid level finishes if:

- New position of the point is found:

$$\left(\sqrt{dx^2 + dy^2} < threshold\right.$$

- Specified number of iteration maxIter is performed
- Intersection between the pyramid layer and the search window became empty

In first two cases for non-zero levels the new position coordinates are scaled to the next pyramid level and the operation continues on the next level. For zero level or for third case the operation stops, the number of the corresponding level is written to the pStatus[i] element, the new coordinates are scaled to zero level and are written to pNext[i]. The square root of the average squared difference between neighborhoods of old and new positions is written to pError[i].

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error if one of the specified pointer is NULL.
ippStsSizeErr	Indicates an error condition if numFeat or winSize has zero or negative value.

ippStsBadArgErr	Indicates an error condition if maxLev or threshold has negative value, or maxIter has zero or negative value.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

// A simple example of performing Optical Flow algorithm
// using Intel(R) Integrated Performance Primitives (Intel(R) IPP) functions:
//     ippiPyramidGetSize
//     ippiPyramidInit
//     ippiPyramidLayerDownGetSize_8u_C1R
//     ippiPyramidLayerDownInit_8u_C1R
//     ippiPyramidLayerDown_8u_C1R
//     ippiOpticalFlowPyrLKGetSize
//     ippiOpticalFlowPyrLKInit_8u_C1R

#include <stdio.h>
#include "ipp.h"

#define WIDTH  640 /* frames width  */
#define HEIGHT 480 /* frames height */

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define check_sts(st)        \
    if ((st) != ippStsNoErr) \
        goto exitLine; /* Go to Exit if Intel(R) IPP function returned status different from
ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */
void drawRect_8u(Ipp8u *pSrc, int srcStep, int x, int y, int w, int h)
{
    int i, j;
    for (j = y; j < y + h; j++) {
        for (i = x; i < x + w; i++) {
            pSrc[srcStep * j + i] = 255;
        }
    }
}
```

```
int main(void)
{
    IppStatus status = ippStsNoErr;
    Ipp8u *pPrevFrame = NULL, *pNextFrame = NULL; /* Pointers to previous/next frames */
    int prevStep = 0, nextStep = 0;               /* Steps, in bytes, through the previous/next
frames */
    IppiSize roiSize = {WIDTH, HEIGHT};           /* Size of previous/next ROI in pixels */
    int numLevel = 5;                             /* Pyramid level number */
    float rate = 2.0f;                            /* Pyramid rate  */
    Ipp16s pKernel[5] = {1, 1, 1, 1, 1};          /* Pyramid kernels */
    int kerSize = 5;                              /* Pyramid kernel size  */
    IppiPoint_32f *prevPt = NULL;                 /* Coordinates on previous frame */
    IppiPoint_32f *nextPt = NULL;                 /* Hint to coordinates on next frame */
    Ipp8s *pStatus = NULL;                        /* Array of differences between pPrev and
pNext points */
    Ipp32f *pError = NULL;                        /* Array of result indicator */
    int numFeat = 5;                              /* Point number */
    int winSize = 41;                             /* Search window size */
    int numIter = 5;                              /* Iteration number */
    float threshold = 0.0001f;

    Ipp8u *pBuf1 = NULL, *pPBuf2 = NULL, *pPBuf1 = NULL; /* Pointer to the work buffers */

    Ipp8u *pBuf2 = NULL;
    Ipp8u *pStBuf1 = NULL, *pStBuf2 = NULL; /* Pointers to the buffer to initialize state
structure for pyramids */
    Ipp8u *pOFStBuf = NULL;

    pPrevFrame = ippiMalloc_8u_C1(roiSize.width, roiSize.height, &prevStep);
    pNextFrame = ippiMalloc_8u_C1(roiSize.width, roiSize.height, &nextStep);
    prevPt = (IppiPoint_32f *)ippsMalloc_8u(numFeat * 2 * sizeof(IppiPoint_32f));
    nextPt = (IppiPoint_32f *)ippsMalloc_8u(numFeat * 2 * sizeof(IppiPoint_32f));
    pStatus = (Ipp8s *)ippsMalloc_8u(numFeat);
    pError = (Ipp32f *)ippsMalloc_8u(numFeat * sizeof(Ipp32f));

    check_sts(status = ippiSet_8u_C1R(0, pPrevFrame, prevStep, roiSize));
    check_sts(status = ippiSet_8u_C1R(0, pNextFrame, nextStep, roiSize));

    int x = 3;
    int y = 3;
    int w = 5;
    int h = 5;
    int shiftx = 3;
    int shifty = 4;

    /*imitate shift of rectangle*/
    drawRect_8u(pPrevFrame, prevStep, x, y, w, h);
    drawRect_8u(pNextFrame, nextStep, x + shiftx, y + shifty, w, h);

    check_sts(status = ippsSet_8u(0, (Ipp8u *)prevPt, numFeat * 2 * sizeof(IppiPoint_32f)));
    check_sts(status = ippsSet_8u(0, (Ipp8u *)nextPt, numFeat * 2 * sizeof(IppiPoint_32f)));

    prevPt[0].x = 0;
    prevPt[0].y = 0;
    prevPt[1].x = x;
    prevPt[1].y = y;
    prevPt[2].x = x + w;
```

```
    prevPt[2].y = y;
    prevPt[3].x = x;
    prevPt[3].y = y + h;
    prevPt[4].x = x + w;
    prevPt[4].y = y + h;

    {
        IppiPyramid *pPyr1 = NULL, *pPyr2 = NULL; /* Pointer to the pointer to the pyramid
structures */
        IppiOptFlowPyrLK *pOF = NULL;             /* Optical flow structure */
        int PyrSize, BufSize, StateSize;          /* Size of pyramid structure, pyramid external
work buffer and state structure */

        /* Compute the size of the structure for pyramids and the size of the required work
buffer (in bytes) */
        check_sts(status = ippiPyramidGetSize(&PyrSize, &BufSize, numLevel, roiSize, rate));

        pBuf1 = ippsMalloc_8u(BufSize);
        pPBuf1 = ippsMalloc_8u(PyrSize);

        /* Initialize structure for pyramids, calculates ROI for layers */
        check_sts(status = ippiPyramidInit(&pPyr1, numLevel, roiSize, rate, pPBuf1, pBuf1));
        ippsFree(pBuf1);
        pBuf1 = NULL;

        /* Compute the size of the structure for pyramids and the size of the required work
buffer (in bytes) */
        check_sts(status = ippiPyramidGetSize(&PyrSize, &BufSize, numLevel, roiSize, rate));

        pBuf1 = ippsMalloc_8u(BufSize);
        pPBuf2 = ippsMalloc_8u(PyrSize);
        /* Initialize structure for pyramids, calculates ROI for layers */
        check_sts(status = ippiPyramidInit(&pPyr2, numLevel, roiSize, rate, pPBuf2, pBuf1));
        ippsFree(pBuf1);
        pBuf1 = NULL;

        {
            IppiPyramidDownState_8u_C1R **pState1 = (IppiPyramidDownState_8u_C1R **)&(pPyr1-
>pState); /* Pyramid state structure */
            IppiPyramidDownState_8u_C1R **pState2 = (IppiPyramidDownState_8u_C1R **)&(pPyr2-
>pState); /* Pyramid state structure */
            Ipp8u **pImg1 = pPyr1-
>pImage;                                                    /* Pointer to the source
image */
            Ipp8u **pImg2 = pPyr2-
>pImage;                                                    /* Pointer to the source
image */
            int *pStep1 = pPyr1->pStep, *pStep2 = pPyr2->pStep;
            IppiSize *pRoi1 = pPyr1->pRoi, *pRoi2 = pPyr2->pRoi;
            IppHintAlgorithm hint = ippAlgHintFast;
            int i, level = pPyr1->level;
            int pStateSize1, pBufSize1, pStateSize2, pBufSize2;

            /* Calculate the size of structure for creating a lower(an upper) pyramid layer and
the size of the temporary buffer */
            check_sts(status = ippiPyramidLayerDownGetSize_8u_C1R(roiSize, rate, kerSize,
&pStateSize1, &pBufSize1));
```

```
            pBuf1 = ippsMalloc_8u(pBufSize1);
            pStBuf1 = ippsMalloc_8u(pStateSize1);

            /* Initialize the structure for creating a lower(an upper) pyramid layer */
            check_sts(status = ippiPyramidLayerDownInit_8u_C1R((IppiPyramidDownState_8u_C1R
**)pState1, roiSize, rate, pKernel, kerSize,
                                                               IPPI_INTER_LINEAR, pStBuf1,
pBuf1));
            ippsFree(pBuf1);
            pBuf1 = NULL;

            /* Calculate the size of structure for creating a lower(an upper) pyramid layer and
the size of the temporary  buffer */
            check_sts(status = ippiPyramidLayerDownGetSize_8u_C1R(roiSize, rate, kerSize,
&pStateSize2, &pBufSize2));

            pBuf2 = ippsMalloc_8u(pBufSize2);
            pStBuf2 = ippsMalloc_8u(pStateSize2);

            /* Initialize the structure for creating a lower(an upper) pyramid layer */
            check_sts(status = ippiPyramidLayerDownInit_8u_C1R((IppiPyramidDownState_8u_C1R
**)pState2, roiSize, rate, pKernel, kerSize,
                                                               IPPI_INTER_LINEAR, pStBuf2,
pBuf2));
            ippsFree(pBuf2);
            pBuf2 = NULL;

            pImg1[0] = (Ipp8u *)pPrevFrame;
            pImg2[0] = (Ipp8u *)pNextFrame;
            pStep1[0] = prevStep;
            pStep2[0] = nextStep;
            pRoi1[0] = pRoi2[0] = roiSize;
            for (i = 1; i <= level; i++) {
                pPyr1->pImage[i] = ippiMalloc_8u_C1(pRoi1[i].width, pRoi1[i].height, pStep1 + i);
                pPyr2->pImage[i] = ippiMalloc_8u_C1(pRoi2[i].width, pRoi2[i].height, pStep2 + i);
                /* Perform downsampling / upsampling of the image with 5x5 Gaussian kernel*/
                check_sts(status = ippiPyramidLayerDown_8u_C1R(pImg1[i - 1], pStep1[i - 1],
pRoi1[i - 1], pImg1[i], pStep1[i], pRoi1[i], *pState1));
                check_sts(status = ippiPyramidLayerDown_8u_C1R(pImg2[i - 1], pStep2[i - 1],
pRoi2[i - 1], pImg2[i], pStep2[i], pRoi2[i], *pState2))
            }

            check_sts(status = ippiOpticalFlowPyrLKGetSize(winSize, roiSize, ipp8u,
(IppHintAlgorithm)((int)hint), &StateSize));

            pOFStBuf = (Ipp8u *)ippsMalloc_8u(StateSize);

            /* Initialize a structure for pyramidal L-K algorithm */
            check_sts(status = ippiOpticalFlowPyrLKInit_8u_C1R((IppiOptFlowPyrLK_8u_C1R **)&pOF,
roiSize, winSize, (IppHintAlgorithm)((int)hint),
                                                               pOFStBuf));

            /* Indicate an error condition if maxLev or threshold has negative value, or maxIter
has zero or negative value */
            check_sts(status = ippiOpticalFlowPyrLK_8u_C1R(pPyr1, pPyr2, prevPt, nextPt,
pStatus, pError, numFeat, winSize, numLevel, numIter,
                                                           threshold, pOF));
```

```
            for (i = level; i > 0; i--) {
                if (pImg2[i])
                    ippiFree(pImg2[i]);
                if (pImg1[i])
                    ippiFree(pImg1[i]);
            }
        }
    }
    printf("shiftx=%d, ", shiftx);
    printf("shifty=%d\n", shifty);
    int n;
    for (n = 0; n < numFeat; n++) {
        printf("%f, %f -> %f, %f\n", prevPt[n].x, prevPt[n].y, nextPt[n].x, nextPt[n].y);
    }
    EXIT_MAIN
    ippsFree(pOFStBuf);
    ippsFree(pBuf1);
    ippsFree(pBuf2);
    ippsFree(pPBuf1);
    ippsFree(pPBuf2);
    ippsFree(pStBuf1);
    ippsFree(pStBuf2);
    ippiFree(pPrevFrame);
    ippiFree(pNextFrame);
    ippsFree(prevPt);
    ippsFree(nextPt);
    ippsFree(pStatus);
    ippsFree(pError);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

Universal Pyramids

The functions described in this section operate with universal image pyramids. These pyramids use separable symmetric kernel (not only Gaussian type) and downsampling/upsampling with arbitrary factor (not only 2). The next pyramid layer can be built for an image of an arbitrary size. These pyramids are used in some computer vision algorithms, for example, in optical flow calculations.

NOTE All universal pyramid functions use the mirrored border.

Example shows how to build pyramids and calculate the optical flow for two images.

- PyramidGetSize
- PyramidInit
- GetPyramidDownROI
- GetPyramidUpROI
- PyramidLayerDownGetSize
- PyramidLayerDownInit
- PyramidLayerDown
- PyramidLayerUpGetSize
- PyramidLayerUpInit
- PyramidLayerUp
- Example of Using General Pyramid Functions

PyramidGetSize

Computes the size of the pyramid structure and the size of the temporary buffer for the `ippiPyramidInit` function.

Syntax

```
IppStatus ippiPyramidGetSize(int* pPyrSize, int* pBufSize, int level, IppiSize roiSize, Ipp32f rate);
```

Platform-aware functions

```
IppStatus ippiPyramidGetSize_L(IppSizeL* pPyrSize, IppSizeL* pBufSize, int level, IppiSizeL roiSize, Ipp32f rate);
```

Include Files

`ippcv.h`

Flavors with the _L suffix: `ippcv_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pPyrSize`	Pointer to the size of the pyramid structure.
`pBufSize`	Pointer to the size of the external work buffer for pyramid processing.
`level`	Maximum value for the pyramid level.
`roiSize`	Size of zero level image ROI, in pixels.
`rate`	Ratio between neighbouring levels (`1< rate ≤ 10`).

Description

This function computes the size of the pyramid structure and the size of the temporary buffer, in bytes, for the ippiPyramidInit function. For an example on how to use this function, refer to the example provided with the ippiPyramidLayerDown function description.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error when `roiSize` has a field with a zero or negative value.
`ippStsBadArgErr`	Indicates an error when `level` is equal to or less than 0, or when `rate` is out of the range.

See Also

PyramidInit Initializes the pyramid structure and calculates the ROI size for pyramid layers.

PyramidLayerDown Creates a lower pyramid layer.

PyramidInit

Initializes the pyramid structure and calculates the ROI size for pyramid layers.

Syntax

```
IppStatus ippiPyramidInit(IppiPyramid** pPyr, int level, IppiSize roiSize, Ipp32f rate,
Ipp8u* pPyrBuffer, Ipp8u* pBuffer);
```

Platform-aware functions

```
IppStatus ippiPyramidInit_L(IppiPyramidL** pPyr, int level, IppiSizeL roiSize, Ipp32f
rate, Ipp8u* pPyrBuffer, Ipp8u* pBuffer);
```

Include Files

`ippcv.h`

Flavors with the -L suffix: `ippcv_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pPyr`	Pointer to the pointer to the pyramid structure.
`level`	Maximum value for the pyramid level.
`roiSize`	Size of zero level image ROI, in pixels.
`rate`	Ratio between neighbouring levels (`1< rate ≤ 10`).
`pPyrBuffer`	Pointer to the buffer for the pyramid structure initialization.
`pBuffer`	Pointer to the work buffer.

Description

This function initializes the structure for pyramid with (`level + 1`) levels. This structure is used by the ippiOpticalFlowPyrLK function for optical flow calculation. The `IppiPyramid` structure contains the following fields:

`pImage`	Pointer to the array of `(level+1)` layer images.
`pStep`	Pointer to the array of `(level+1)` image row step values.
`pRoi`	Pointer to the array of `(level+1)` layer image ROIs.
`pRate`	Pointer to the array of `(level+1)` ratios of `i – th` levels to the zero level ($rate^{-i}$).

pState	Pointer to the structure to compute the next pyramid layer.
level	Number of levels in the pyramid.

Analogue for IppiPyramid structure for L-functions is IppiPyramidL structure.

The ippiPyramidInit function fills the pRoi and pRate arrays and the level field. The value of the level field is equal to the minimum of the input value of the level parameter and the maximum possible number of layers of the pyramid with given rate and zero level size.

You need to specify other fields. To initialize the pyramid layer structure pState, use the ippiPyramidLayerDownInit or ippiPyramidLayerUpInit functions. To obtain the pyramid layer images, you can use the ippiPyramidLayerDown and ippiPyramidLayerUp functions.

For an example on how to use this function, refer to the example provided with the ippiPyramidLayerDown function description.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one of the pointers is NULL.
ippStsSizeErr	Indicates an error when roiSize has a field with a zero or negative value.
ippStsBadArgErr	Indicates an error when level is equal to or less than 0, or when rate is out of the range.

See Also

OpticalFlowPyrLK Calculates optical flow for the set of feature points using the pyramidal Lucas-Kanade algorithm.

PyramidLayerDownInit Initializes the structure for creating a lower pyramid layer.

PyramidLayerUpInit Initializes the structure for creating an upper pyramid layer.

PyramidLayerDown Creates a lower pyramid layer.

PyramidLayerUp Creates an upper pyramid layer.

GetPyramidDownROI

Computes the size of the lower pyramid layer.

Syntax

```
IppStatus ippiGetPyramidDownROI(IppiSize srcRoi, IppiSize* pDstRoi, Ipp32f rate);
```

Platform-aware functions

```
IppStatus ippiGetPyramidDownROI_L(IppiSizeL srcRoi, IppiSizeL *pDstRoi, Ipp32f rate);
```

Include Files

ippcv.h

Flavors with the _L suffix: ippcv_l.h

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`srcRoiSize`	Size of the source pyramid layer ROI in pixels.
`pDstRoiSize`	Pointer to the size of the destination (lower) pyramid layer ROI in pixels.
`rate`	Ratio between source and destination layers (`1< rate ≤ 10`).

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function computes the size of the lower pyramid layer `pDstRoiSize` for a source layer of a given size `srcRoiSize` and specified size ratio `rate` between them in accordance with the following formulas:

$$pDstRoiSize.width = max(1, min(\lceil srcRoiSize.width/\ rate \rceil, srcRoiSize.width-$$
$$pDstRoiSize.height = max(1, min(\lceil srcRoiSize.height/\ rate \rceil, srcRoiSize.heig$$

NOTE Since for the non-integer `rate` results depend on the computational precision, it is strongly recommended to use this function in computations.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if `pDstRoiSize` pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `srcRoiSize` has a field with zero or negative value.
`ippStsBadArgErr`	Indicates an error condition if `rate` is out of the range.

GetPyramidUpROI

Computes the size of the upper pyramid layer.

Syntax

```
IppStatus ippiGetPyramidUpROI(IppiSize srcRoi, IppiSize* pDstRoiMin, IppiSize*
pDstRoiMax, Ipp32f rate);
```

Platform-aware functions

```
IppStatus ippiGetPyramidUpROI_L(IppiSizeL srcRoi, IppiSizeL *pDstRoiMin, IppiSizeL
*pDstRoiMax, Ipp32f rate);
```

Include Files

ippcv.h

Flavors with the _L suffix: ippcv_l.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

srcRoiSize	Size of the source pyramid layer ROI in pixels.
pDstRoiSizeMin	Pointer to the minimal size of the destination (upper) pyramid layer ROI in pixels.
pDstRoiSizeMax	Pointer to the maximal size of the destination (upper) pyramid layer ROI in pixels.
rate	Ratio between source and destination layers (1< rate ≤ 10).

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function computes possible sizes of the upper pyramid layer pDstRoiSizeMin and pDstRoiSizeMax for a source layer of a given size srcRoiSize and specified size ratio rate between them in accordance with the following formulas:

maximum size pDstRoiSizeMax:

$$pDstRoiMax.width = max(srcRoiSize.width + 1, \lfloor srcRoiSize.width \cdot rate$$
$$pDstRoiMax.height = max(srcRoiSize.height + 1, \lfloor srcRoiSize.height \cdot$$

minimum size pDstRoiSizeMin:

if the width and height of the source layer ROI is greater than 1,

$$pDstRoiMin.width = max(srcRoiSize.width + 1, \lfloor (srcRoiSize.width - 1) \cdot rat$$
$$pDstRoiMin.height = max(srcRoiSize.height + 1, \lfloor (srcRoiSize.height - 1) \cdot$$

if the width and height of the source layer ROI is equal to 1,

$$pDstRoiMin.width = 1$$

$$pDstRoiMin.height = 1$$

NOTE Since for the non-integer `rate` results depend on the computational precision, it is strongly recommended to use this function in computations.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `srcRoiSize` has a field with zero or negative value.
`ippStsBadArgErr`	Indicates an error condition if `rate` is out of the range.

PyramidLayerDownGetSize

Computes the size of the structure for creating a lower pyramid layer and the size of the temporary buffer.

Syntax

```
IppStatus ippiPyramidLayerDownGetSize_8u_C1R(IppiSize srcRoi, Ipp32f rate, int kerSize,
int* pStateSize, int* pBufSize);
```

```
IppStatus ippiPyramidLayerDownGetSize_8u_C3R(IppiSize srcRoi, Ipp32f rate, int kerSize,
int* pStateSize, int* pBufSize);
```

```
IppStatus ippiPyramidLayerDownGetSize_16u_C1R(IppiSize srcRoi, Ipp32f rate, int
kerSize, int* pStateSize, int* pBufSize);
```

```
IppStatus ippiPyramidLayerDownGetSize_16u_C3R(IppiSize srcRoi, Ipp32f rate, int
kerSize, int* pStateSize, int* pBufSize);
```

```
IppStatus ippiPyramidLayerDownGetSize_32f_C1R(IppiSize srcRoi, Ipp32f rate, int
kerSize, int* pStateSize, int* pBufSize);
```

```
IppStatus ippiPyramidLayerDownGetSize_32f_C3R(IppiSize srcRoi, Ipp32f rate, int
kerSize, int* pStateSize, int* pBufSize);
```

Platform-aware functions

```
IppStatus ippiPyramidLayerDownGetSize_32f_C1R_L(IppiSizeL srcRoi, Ipp32f rate, int
kerSize, IppSizeL* pStateSize, IppSizeL* pBufSize);
```

Include Files

`ippcv.h`

Flavors with the _L suffix: `ippcv_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

srcRoi	Size of the source image ROI.
rate	Ratio between neighbouring levels (1< rate ≤ 10).
kerSize	Size of the kernel.
pStateSize	Pointer to the size of the pyramid layer state structure.
pBufSize	Pointer to the size of the external work buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function computes the size of the pState structure to build a lower pyramid layer and the size of the temporary buffer, in bytes. This structure is used by the ippiPyramidLayerDown function and can be applied to process images with size not greater than srcRoi. For an example on how to use this function, refer to the example provided with the ippiPyramidLayerDown function description.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one of the pointers is NULL.
ippStsSizeErr	Indicates an error when the width or height of images is less than, or equal to zero.
ippStsBadArgErr	Indicates an error when kerSize is even, or equal to or less than 0; or when rate is out of the range.

See Also

Regions of Interest in Intel IPP

PyramidLayerDown Creates a lower pyramid layer.

PyramidLayerDownInit

Initializes the structure for creating a lower pyramid layer.

Syntax

Case 1: Operating on integer data

```
IppStatus ippiPyramidLayerDownInit_8u_C1R(IppiPyramidDownState_8u_C1R** ppState,
IppiSize srcRoi, Ipp32f rate, Ipp16s* pKernel, int kerSize, int mode, Ipp8u* pStateBuf,
Ipp8u* pBuffer);
```

```
IppStatus ippiPyramidLayerDownInit_8u_C3R(IppiPyramidDownState_8u_C3R** ppState,
IppiSize srcRoi, Ipp32f rate, Ipp16s* pKernel, int kerSize, int mode, Ipp8u* pStateBuf,
Ipp8u* pBuffer);
```

```
IppStatus ippiPyramidLayerDownInit_16u_C1R(IppiPyramidDownState_16u_C1R** ppState,
IppiSize srcRoi, Ipp32f rate, Ipp16s* pKernel, int kerSize, int mode, Ipp8u* pStateBuf,
Ipp8u* pBuffer);
```

```
IppStatus ippiPyramidLayerDownInit_16u_C3R(IppiPyramidDownState_16u_C3R** ppState,
IppiSize srcRoi, Ipp32f rate, Ipp16s* pKernel, int kerSize, int mode, Ipp8u* pStateBuf,
Ipp8u* pBuffer);
```

Case 2: Operating on floating point data

```
IppStatus ippiPyramidLayerDownInit_32f_C1R(IppiPyramidDownState_32f_C1R** ppState,
IppiSize srcRoi, Ipp32f rate, Ipp32f* pKernel, int kerSize, int mode, Ipp8u* pStateBuf,
Ipp8u* pBuffer);
```

```
IppStatus ippiPyramidLayerDownInit_32f_C3R(IppiPyramidDownState_32f_C3R** ppState,
IppiSize srcRoi, Ipp32f rate, Ipp32f* pKernel, int kerSize, int mode, Ipp8u* pStateBuf,
Ipp8u* pBuffer);
```

Platform-aware functions

```
IppStatus ippiPyramidLayerDownInit_32f_C1R_L(IppiPyramidDownState_32f_C1R_L** pState,
IppiSizeL srcRoi, Ipp32f rate, const Ipp32f* pKernel, int kerSize, int mode, Ipp8u*
StateBuf, Ipp8u* Buffer);
```

Include Files

`ippcv.h`

Flavors with the _L suffix: `ippcv_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`ppState`	Pointer to the pointer to the initialized pyramid layer state structure.
`srcRoi`	Size of the source image ROI.
`rate`	Ratio between neighbouring levels (`1< rate ≤ 10`).
`pKernel`	Separable symmetric kernel of odd length.
`kerSize`	Size of the kernel.
`mode`	Interpolation method, possible value is:

`IPPI_INTER_LINEAR`	Bilinear interpolation.

`pStateBuf`	Pointer to the buffer to initialize the pyramid layer state structure.
`pBuffer`	Pointer to the external buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function initializes the `pState` structure to build a lower pyramid layer. This structure is used by the ippiPyramidLayerDown function and can be applied to process images with size not greater than `dstRoi`.

The specified kernel `pKernel` should be symmetric. If it is not symmetric, the function builds the symmetric kernel using the first part of the specified kernel and returns a warning. The symmetric separable kernel can be not Gaussian. If the sum of kernel elements is not equal to zero, the kernel is normalized.

For integer rates, the function performs downsampling by discarding rows and columns that are not multiples of the rate value. For non-integer rates, the function uses bilinear interpolation (see Linear Interpolation for more information).

For an example on how to use this function, refer to the example provided with the ippiPyramidLayerDown function description.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one of the pointers is `NULL`.
`ippStsSizeErr`	Indicates an error when the width or height of images is less than, or equal to zero.
`ippStsBadArgErr`	Indicates an error when `kerSize` is even, or equal to or less than 0; or when `rate` is out of the range.

See Also

Regions of Interest in Intel IPP

PyramidLayerDown Creates a lower pyramid layer.

Linear Interpolation

PyramidLayerDown

Creates a lower pyramid layer.

Syntax

```
IppStatus ippiPyramidLayerDown_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize srcRoiSize, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize, IppiPyramidDownState_<mod>* pState);
```

Supported values for `mod`:

`8u_C1R`	`16u_C1R`	`32f_C1R`
`8u_C3R`	`16u_C3R`	`32f_C3R`

Platform-aware functions

```
IppStatus ippiPyramidLayerDown_32f_C1R_L(const Ipp32f* pSrc, IppiSizeL srcStep, IppiSizeL srcRoiSize, Ipp32f* pDst, IppiSizeL dstStep, IppiSizeL dstRoiSize, IppiPyramidDownState_32f_C1R_L* pState);
```

Include Files

`ippcv.h`

Flavors with the _L suffix: `ippcv_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance, in bytes, between the starting points of consecutive lines in the source image.
`srcRoiSize`	Size of the source image ROI, in pixels.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance, in bytes, between the starting points of consecutive lines in the destination image.
`dstRoiSize`	Size of the destination image ROI, in pixels.
`pState`	Pointer to the pyramid layer structure.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function creates a lower pyramid layer `pDst` from the source image `pSrc`. The function applies the convolution kernel to the source image using the mirror border and then performs downsampling. Before calling `ippiPyramidLayerDown`, compute the size of the `pState` structure and work buffer using the PyramidLayerDownGetSize function and initialize the structure using the PyramidLayerDownInit function. The function can process images with `srcRoiSize` not greater than the `srcRoi` parameter specified in the PyramidLayerDownInit function.

NOTE This function uses the mirrored border.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `srcRoiSize` or `dstRoiSize` has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if `srcStep` is less than `srcRoiSize.width * <``pixelSize> , or ``dstStep` is less than `dstRoiSize.width * <pixelSize>`.`
`ippStsNotEvenStepErr`	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.

ippStsBadArgErr	Indicates an error condition if pState->rate has wrong value.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

// A simple example of downsampling of the image with
// 5x5 Gaussian kernel using Intel(R) Integrated Performance Primitives (Intel(R) IPP) functions:
//     ippiPyramidGetSize
//     ippiPyramidInit
//     ippiPyramidLayerDownGetSize_32f_C1R
//     ippiPyramidLayerDownInit_32f_C1R
//     ippiPyramidLayerDown_32f_C1R

#include <stdio.h>
#include "ipp.h"

#define WIDTH   128 /* Image width  */
#define HEIGHT  128 /* Image height */
#define KERSIZE 3   /* Kernel size  */

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define check_sts(st)        \
    if ((st) != ippStsNoErr) \
        goto exitLine; /* Go to Exit if Intel(R) IPP function returned status different from
ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main(void)
{
    IppStatus status = ippStsNoErr;
    Ipp32f *pSrc = NULL; /* Pointer to source image */
    int srcStep;         /* Step, in bytes, through the source image */

    IppiSize roiSize = {WIDTH, HEIGHT}; /* Size of source/destination ROI in pixels */
    Ipp32f rate = 2.f;                  /* Neighbor levels ratio */
    int level = 1000;                   /* Maximal number pyramid level */

    int i = 0;
```

```
    int pyrBufferSize = 0;
    int pyrStructSize = 0;

    IppiPyramid *pPyrStruct = NULL;
    Ipp8u *pPyrBuffer = NULL;
    Ipp8u *pPyrStrBuffer = NULL;

    int pyrLStateSize = 0;
    int pyrLBufferSize = 0;
    Ipp8u *pPyrLStateBuf = NULL;
    Ipp8u *pPyrLBuffer = NULL;
    Ipp32f **pPyrImage = NULL;

    Ipp32f kernel[KERSIZE] = {1.f, 1.f, 1.f}; /* Separable symmetric kernel of odd length */

    pSrc = ippiMalloc_32f_C1(roiSize.width, roiSize.height, &srcStep);

    /* Build Gaussian pyramid */
    {
        /* Computes the temporary work buffer size */
        check_sts(status = ippiPyramidGetSize(&pyrStructSize, &pyrBufferSize, level, roiSize,
rate));

        pPyrBuffer = ippsMalloc_8u(pyrBufferSize);
        pPyrStrBuffer = ippsMalloc_8u(pyrStructSize);

        /* Initializes Gaussian structure for pyramids */
        check_sts(status = ippiPyramidInit(&pPyrStruct, level, roiSize, rate, pPyrStrBuffer,
pPyrBuffer));

        /* Correct maximum scale level */
        level = pPyrStruct->level;

        /* Allocate structures to calculate pyramid layers */
        check_sts(status = ippiPyramidLayerDownGetSize_32f_C1R(roiSize, rate, KERSIZE,
&pyrLStateSize, &pyrLBufferSize));

        pPyrLStateBuf = ippsMalloc_8u(pyrLStateSize);
        pPyrLBuffer = ippsMalloc_8u(pyrLBufferSize);

        /* Initialize the structure for creating a lower pyramid layer */
        check_sts(status = ippiPyramidLayerDownInit_32f_C1R((IppiPyramidDownState_32f_C1R
**)&pPyrStruct->pState, roiSize, rate, kernel, KERSIZE,
                                                            IPPI_INTER_LINEAR, pPyrLStateBuf,
pPyrLBuffer));

        /* Allocate pyramid layers */
        pPyrImage = (Ipp32f **)pPyrStruct->pImage;
        pPyrImage[0] = pSrc;
        pPyrStruct->pStep[0] = srcStep;
        for (i = 1; i <= level; i++) {
            pPyrImage[i] = ippiMalloc_32f_C1(pPyrStruct->pRoi[i].width, pPyrStruct-
>pRoi[i].height, &pPyrStruct->pStep[i]);
        }

        /* Perform downsampling of the image with 5x5 Gaussian kernel */
        for (i = 1; i <= level; i++) {
            check_sts(status = ippiPyramidLayerDown_32f_C1R(pPyrImage[i - 1], pPyrStruct-
```

```
>pStep[i - 1], pPyrStruct->pRoi[i - 1], pPyrImage[i],
                                                        pPyrStruct->pStep[i], pPyrStruct-
>pRoi[i],
                                                        (IppiPyramidDownState_32f_C1R
*)pPyrStruct->pState));
        }
    }

    EXIT_MAIN
    if (pPyrImage)
        for (i = 1; i <= level; i++)
            ippiFree(pPyrImage[i]);
    ippiFree(pPyrLStateBuf);
    ippiFree(pPyrLBuffer);
    ippiFree(pPyrStrBuffer);
    ippiFree(pPyrBuffer);
    ippiFree(pSrc);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return status;
}
```

See Also

Regions of Interest in Intel IPP

PyramidLayerDownGetSize Computes the size of the structure for creating a lower pyramid layer and the size of the temporary buffer.

PyramidLayerDownInit Initializes the structure for creating a lower pyramid layer.

PyramidLayerUpGetSize

Computes the size of the structure for creating an upper pyramid layer and the size of the temporary buffer.

Syntax

```
IppStatus ippiPyramidLayerUpGetSize_8u_C1R(IppiSize dstRoi, Ipp32f rate, int kerSize,
int* pStateSize);
```

```
IppStatus ippiPyramidLayerUpGetSize_8u_C3R(IppiSize dstRoi, Ipp32f rate, int kerSize,
int* pStateSize);
```

```
IppStatus ippiPyramidLayerUpGetSize_16u_C1R(IppiSize dstRoi, Ipp32f rate, int kerSize,
int* pStateSize);
```

```
IppStatus ippiPyramidLayerUpGetSize_16u_C3R(IppiSize dstRoi, Ipp32f rate, int kerSize,
int* pStateSize);
```

```
IppStatus ippiPyramidLayerUpGetSize_32f_C1R(IppiSize dstRoi, Ipp32f rate, int kerSize,
int* pStateSize);
```

```
IppStatus ippiPyramidLayerUpGetSize_32f_C3R(IppiSize dstRoi, Ipp32f rate, int kerSize,
int* pStateSize);
```

Platform-aware functions

```
IppStatus ippiPyramidLayerUpGetSize_32f_C1R_L(IppiSizeL dstRoi, Ipp32f rate, int
kerSize, IppSizeL* pStateSize);
```

Include Files

`ippcv.h`

Flavors with the _L suffix: `ippcv_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`dstRoi`	Size of the destination image ROI.
`rate`	Ratio between neighbouring levels (`1< rate ≤ 10`).
`kerSize`	Size of the kernel.
`pStateSize`	Pointer to the size of the pyramid layer state structure.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function computes the size of the `pState` structure to build an upper pyramid layer and the size of the temporary buffer, in bytes. This structure is used by the ippiPyramidLayerUp function and can be applied to process images with size not greater than `dstRoi`. For an example on how to use this function, refer to the example provided with the ippiPyramidLayerDown function description.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when at least one of the pointers is `NULL`.
`ippStsSizeErr`	Indicates an error when the width or height of images is less than, or equal to zero.
`ippStsBadArgErr`	Indicates an error when `kerSize` is even, or equal to or less than 0; or when `rate` is out of the range.

See Also

Regions of Interest in Intel IPP

PyramidLayerUp Creates an upper pyramid layer.

PyramidLayerDown Creates a lower pyramid layer.

PyramidLayerUpInit

Initializes the structure for creating an upper pyramid layer.

Syntax

Case 1: Operating on integer data

```
IppStatus ippiPyramidLayerUpInit_8u_C1R(IppiPyramidUpState_8u_C1R** ppState, IppiSize
dstRoi, Ipp32f rate, Ipp16s* pKernel, int kerSize, int mode, Ipp8u* pStateBuf);

IppStatus ippiPyramidLayerUpInit_8u_C3R(IppiPyramidUpState_8u_C3R** ppState, IppiSize
dstRoi, Ipp32f rate, Ipp16s* pKernel, int kerSize, int mode, Ipp8u* pStateBuf);

IppStatus ippiPyramidLayerUpInit_16u_C1R(IppiPyramidUpState_16u_C1R** ppState, IppiSize
dstRoi, Ipp32f rate, Ipp16s* pKernel, int kerSize, int mode, Ipp8u* pStateBuf);

IppStatus ippiPyramidLayerUpInit_16u_C3R(IppiPyramidUpState_16u_C3R** ppState, IppiSize
dstRoi, Ipp32f rate, Ipp16s* pKernel, int kerSize, int mode, Ipp8u* pStateBuf);
```

Case 2: Operating on floating point data

```
IppStatus ippiPyramidLayerUpInit_32f_C1R(IppiPyramidUpState_32f_C1R** ppState, IppiSize
dstRoi, Ipp32f rate, Ipp32f* pKernel, int kerSize, int mode, Ipp8u* pStateBuf, Ipp8u*
pBuffer);

IppStatus ippiPyramidLayerUpInit_32f_C3R(IppiPyramidDownState_32f_C3R** ppState,
IppiSize dstRoi, Ipp32f rate, Ipp32f* pKernel, int kerSize, int mode, Ipp8u* pStateBuf,
Ipp8u* pBuffer);
```

Platform-aware functions

```
IppStatus ippiPyramidLayerUpInit_32f_C1R_L(IppiPyramidUpState_32f_C1R_L** ppState,
IppiSizeL dstRoi, Ipp32f rate, const Ipp32f* pKernel, int kerSize, int mode, Ipp8u*
StateBuf);
```

Include Files

`ippcv.h`

Flavors with the _L suffix: `ippcv_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`ppState`	Pointer to the pointer to the initialized pyramid state structure.
`dstRoi`	Size of the destination image ROI.
`rate`	Ratio between neighbouring levels (`1< rate ≤ 10`).
`pKernel`	Separable symmetric kernel of odd length.
`kerSize`	Size of the kernel.
`mode`	Interpolation method, possible value is:

`IPPI_INTER_LINEAR`	Bilinear interpolation.

`pStateBuf`	Pointer to the buffer to initialize the pyramid layer state structure.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function allocates memory and initializes the pState structure to build an upper pyramid layer. This structure is used by the ippiPyramidLayerUp function and can be applied to process images with size not greater than dstRoi.

The specified kernel pKernel should be symmetric. If it is not symmetric, the function builds the symmetric kernel using the first part of the specified kernel and returns a warning. The symmetric separable kernel can be not Gaussian. If the sum of kernel elements is not equal to zero, the kernel is normalized.

For integer rates, the function performs upsampling by inserting zero rows and columns that are not multiples of the rate value. For non-integer rates, the function uses bilinear interpolation (see Linear Interpolation for more information) to calculate kernel coefficients for pixels with non-integer coordinates.

For an example on how to use this function, refer to the example provided with the ippiPyramidLayerDown function description.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when at least one of the pointers is NULL.
ippStsSizeErr	Indicates an error when the width or height of images is less than, or equal to zero.
ippStsBadArgErr	Indicates an error when kerSize is even, or equal to or less than 0; or when rate is out of the range.

See Also

Regions of Interest in Intel IPP

PyramidLayerUp Creates an upper pyramid layer.

PyramidLayerDown Creates a lower pyramid layer.

Linear Interpolation

PyramidLayerUp

Creates an upper pyramid layer.

Syntax

```
IppStatus ippiPyramidLayerUp_<mod>(const Ipp<datatype>* pSrc, int srcStep, IppiSize
srcRoiSize, Ipp<datatype>* pDst, int dstStep, IppiSize dstRoiSize,
IppiPyramidUpState_<mod>* pState);
```

Supported values for mod:

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R

Platform-aware functions

```
IppStatus ippiPyramidLayerUp_32f_C1R_L(const Ipp32f* pSrc, IppiSizeL srcStep, IppiSizeL
srcRoiSize, Ipp32f* pDst, IppiSizeL dstStep, IppiSizeL dstRoiSize,
IppiPyramidUpState_32f_C1R_L* pState);
```

Include Files

ippcv.h

Flavors with the _L suffix: ippcv_l.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance, in bytes, between the starting points of consecutive lines in the source image.
srcRoiSize	Size of source image ROI, in pixels.
pDst	Pointer to the destination image ROI.
dstStep	Distance, in bytes, between the starting points of consecutive lines in the destination image.
dstRoiSize	Size of destination image ROI, in pixels.
pState	The pointer to the pyramid layer structure.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function creates an upper pyramid layer pDst from the source image pSrc. The function performs upsampling of the source image and then applies the convolution kernel using the mirror border. Before calling the ippiPyramidLayerUp function, compute the size of the pyramid layer structure pState using the PyramidLayerUpGetSize function and initialize the structure using the PyramidLayerUpInit function. The function can process images with srcRoiSize not greater than the roiSize parameter specified in the PyramidLayerUpInit function.

NOTE This function uses the mirrored border.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if srcRoiSize or dstRoiSize has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if srcStep is less than srcRoiSize.width * <``pixelSize> , or ``dstStep is less than dstRoiSize.width * <pixelSize>.

ippStsNotEvenStepErr	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
ippStsBadArgErr	Indicates an error condition if pState->rate has wrong value.

See Also

Regions of Interest in Intel IPP

PyramidLayerDownGetSize Computes the size of the structure for creating a lower pyramid layer and the size of the temporary buffer.

PyramidLayerDownInit Initializes the structure for creating a lower pyramid layer.

Example of Using General Pyramid Functions

Refer to the following example to understand how different general pyramids functions can be used to create the Gaussian and Laplacian pyramids:

Pyramid.c

Image Inpainting

The functions described in this section allows to restore the unknown image portions. They could be used to repair damaged parts of images and to remove some objects from images. Fast direct methods of inpainting that allow for run-time correcting of video frames are supported.

- InpaintGetSize
- InpaintInit
- Inpaint

InpaintGetSize

Computes the size of the state structure and work buffer for image inpainting.

Syntax

```
IppStatus ippiInpaintGetSize(const Ipp8u* pMask, int maskStep, IppiSize roiSize, Ipp32f radius, IppiInpaintFlag flags, int channels, int* pStateSize, int* pBufSize);
```

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pMask	Pointer to the mask image ROI.

maskStep	Distance, in bytes, between the starting points of consecutive lines in the mask image.
roiSize	Size of the image ROI, in pixels.
radius	Radius of the neghborhood used for inpainting.
flags	Specifies algorithm for image inpainting. Possible values:

IPP_INPAINT_TELEA	Telea algorithm
IPP_INPAINT_NS	Navier-Stokes equation

channels	Number of channels in the image.
pStateSize	Pointer to the size of the state structure.
pBufSize	Pointer to the size of the external work buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function computes the size of the state structure for direct methods of image inpainting and the size of the external work buffer. Call this function before using ippiInpaintInit. For an example on how to use this function, refer to the example provided with the ippiInpaint function description.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when width or height of the image is less than, or equal to zero.
ippStsStepErr	Indicates an error when the step in the mask image is too small.
ippStsBadArgErr	Indicates an error when radius is less than 1, or flags has an illegal value.
ippStsNumChannelsErr	Indicates an error when the specified number of image channels is invalid or not supported.

See Also

Regions of Interest in Intel IPP

Inpaint MODIFIED API. Restores unknown image pixels.

InpaintInit Initializes the state structure for image inpainting.

InpaintInit

Initializes the state structure for image inpainting.

Syntax

```
IppStatus ippiInpaintInit_8u_C1R(IppiInpaintState_8u_C1R** ppState, const Ipp32f*
pDist, int distStep, const Ipp8u* pMask, int maskStep, IppiSize roiSize, Ipp32f radius,
IppiInpaintFlag flags, Ipp8u* pStateBuf, Ipp8u* pBuf);
```

```
IppStatus ippiInpaintInit_8u_C3R(IppiInpaintState_8u_C3R** ppState, const Ipp32f*
pDist, int distStep, const Ipp8u* pMask, int maskStep, IppiSize roiSize, Ipp32f radius,
IppiInpaintFlag flags, Ipp8u* pStateBuf, Ipp8u* pBuf);
```

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`ppState`	Double pointer to the state structure for image inpaiting.
`pDist`	Pointer to the ROI of the image of distances.
`distStep`	Distance, in bytes, between the starting points of consecutive lines in the image of distances.
`pMask`	Pointer to the mask image ROI.
`maskStep`	Distance, in bytes, between the starting points of consecutive lines in the mask image.
`roiSize`	Size of the image ROI, in pixels.
`radius`	Radius of the neghborhood used for inpainting (`dist` ≤ `radius` pixels are processed).
`flags`	Specifies algorithm for image inpainting. Possible values:

IPP_INPAINT_TELEA	Telea algorithm
IPP_INPAINT_NS	Navier-Stokes equation

`pStateBuf`	Pointer to the buffer for the state structure initialization.
`pBuf`	Pointer to the external work buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function initializes the `ppState` structure for direct methods of image inpainting. This structure is used by the ippiInpaint function and can be applied to process images of the same size `roiSize`.

Zero pixels of the `pMask` image correspond to the known image pixels, non-zero pixels - to the unknown image pixels that should be restored. The distance image `pDist` specifies the order of pixel inpainting. Values of unknown pixels are restored in ascending order depending on their distances. The `radius` parameter

specifies the radius of the circular neighborhood that affects the restoration of the central pixel. The `flag` parameter specifies the method of direct inpainting. Two methods are supported: Telea algorithm [Telea04] and Navier-Stokes equation [Bert01].

For an example on how to use this function, refer to the example provided with the ippiInpaint function description.

NOTE The image ROI must not exceed the maximum width and height of `roiSize` specified in the initialization function.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error when one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error when width or height of the image is less than, or equal to zero.
`ippStsStepErr`	Indicates an error when the step of the mask or distance image ROI is too small.
`ippStsNotEvenStepErr`	Indicates an error when the step value is not divisible by the `pDist` element.
`ippStsBadArgErr`	Indicates an error when `radius` is less than 1, or `flags` has an illegal value.

See Also

Regions of Interest in Intel IPP

Inpaint MODIFIED API. Restores unknown image pixels.

InpaintGetSize Computes the size of the state structure and work buffer for image inpainting.

Inpaint

MODIFIED API. Restores unknown image pixels.

Syntax

```
IppStatus ippiInpaint_8u_C1R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep,
IppiSize roiSize, IppiInpaintState_8u_C1R* pState, Ipp8u* pBuffer);
```

```
IppStatus ippiInpaint_8u_C3R(const Ipp8u* pSrc, int srcStep, Ipp8u* pDst, int dstStep,
IppiSize roiSize, IppiInpaintState_8u_C1R* pState, Ipp8u* pBuffer);
```

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance in bytes between starts of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance in bytes between starts of consecutive lines in the destination image.
`roiSize`	Size of the image ROI in pixels.
`pState`	The pointer to the inpainting structure.
`pBuffer`	Pointer to the work buffer.

Description

NOTE
Important
The API of this function has been modified in Intel IPP 9.0 release.

Before using this function, compute the size of the state structure and work buffer using InpaintGetSize and initialize the structure using InpaintInit.

This function operates with ROI (see Regions of Interest in Intel IPP).

This function reconstructs damaged part of the image, or removes a selected object (see Figure Image Inpainting). The image part to restore is defined by the mask that is created when the inpainting structure `pState` is initialized by the InpaintInit function. Different distant transforms can be used, but the Fast Marching method (ippiFastMarching) provides the best results. The order of pixel restoration is defined by the distance through the initialization the inpainting structure `pState` by the InpaintInit function. Pixels are restored in according to the growing of their distance value. When a pixel is inpainted, it is treated as the known one.

Two algorithms of direct inpainting are supported (controlled by the parameter `flags` of the InpaintInit function):

- image restoration of the unknown pixel by the weighted sum of approximations by known pixels in the neighborhood (`flags = IPP_INPAINT_TELEA`) [Telea04],
- image restoration based on the Navier-Stokes equations (`flags = IPP_INPAINT_NS`) [Bert01].

The inpainting structure `pState` can be used to perform restoration of several different images of the same size `roiSize`.

Image Inpainting



Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error when one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error when `roiSize` has a field with zero or negative value, or if differs from the corresponding parameter that is specified when the inpainting structure is initialized by InpaintInit.
`ippStsStepErr`	Indicates an error condition if `srcStep` or `dstStep` is less than `roiSize.width` * < `pixelSize`>.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
```

```
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *****************************************************************************/

// A simple example of restoring damaged image area by direct inpainting
// using Intel(R) Integrated Performance Primitives (Intel(R) IPP) functions:
//     ippiFastMarchingGetBufferSize_8u32f_C1R
//     ippiFastMarching_8u32f_C1R
//     ippiInpaintGetSize
//     ippiInpaintInit_8u_C1R
//     ippiInpaint_8u_C1R

#include <stdio.h>
#include "ipp.h"

#define WIDTH         128            /* Image width*/
#define HEIGHT        128            /* Image height*/
#define RADIUS        3.f            /* Neighborhood radius*/
#define INPAINT_FLAGS IPP_INPAINT_NS /* Specifies algorithm for image inpainting*/

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define check_sts(st)        \
    if ((st) != ippStsNoErr) \
        goto exitLine; /* Go to Exit if Intel(R) IPP function returned status different from
ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main(void)
{
    IppStatus status = ippStsNoErr;

    Ipp8u *pSrc = NULL, *pDst = NULL;   /* Pointers to source/destination images */
    int srcStep = 0, dstStep = 0;       /* Steps, in bytes, through the source/destination
images */
    IppiSize roiSize = {WIDTH, HEIGHT}; /* Size of source/destination ROI in pixels */

    Ipp8u *pMask = NULL; /* Pointer to mask image */
    int maskStep;        /* Step, in bytes, through the mask image */

    Ipp32f *pFMDist = NULL;  /* Pointer to distances image */
    int fmDistStep = 0;      /* Step, in bytes, through the distances image */
    Ipp8u *pFMBuffer = NULL; /* Pointer to the Fast marching work buffer */
    int fmBufferSize = 0;

    IppiInpaintState_8u_C1R *pInpState = NULL; /* Pointer to inpainting structure */
    Ipp8u *pInpBuffer = NULL;                  /* Pointer to the Inpaint work buffer */
    int inpBufferSize = 0;
    Ipp8u *pInpStateBuf = NULL; /* Pointer to the Inpaint structure buffer */
    int inpStateSize = 0;

    pSrc = ippiMalloc_8u_C1(roiSize.width, roiSize.height, &srcStep);
    pDst = ippiMalloc_8u_C1(roiSize.width, roiSize.height, &dstStep);
    pMask = ippiMalloc_8u_C1(roiSize.width, roiSize.height, &maskStep);
```

```
    pFMDist = ippiMalloc_32f_C1(roiSize.width, roiSize.height, &fmDistStep);

    /* Compute the temporary work buffer size for Fast Marching */
    check_sts(status = ippiFastMarchingGetBufferSize_8u32f_C1R(roiSize, &fmBufferSize));

    pFMBuffer = ippsMalloc_8u(fmBufferSize);

    /* Calculate distance by the Fast Marching Method */
    check_sts(status = ippiFastMarching_8u32f_C1R(pMask, maskStep, pFMDist, fmDistStep, roiSize,
RADIUS, pFMBuffer));

    /* Calculate State and work buffer sizes for Inpaint */
    check_sts(status = ippiInpaintGetSize(pMask, maskStep, roiSize, RADIUS, INPAINT_FLAGS,
ippC1, &inpStateSize, &inpBufferSize));

    /* Memory allocation for State and work buffer */
    pInpStateBuf = ippsMalloc_8u(inpStateSize);
    pInpBuffer = ippsMalloc_8u(inpBufferSize);

    /* Initialize the Inpaint State for direct methods of image inpainting */
    check_sts(status =
                   ippiInpaintInit_8u_C1R(&pInpState, pFMDist, fmDistStep, pMask, maskStep,
roiSize, RADIUS, INPAINT_FLAGS, pInpStateBuf, pInpBuffer));

    /* Restores damaged image area by direct inpainting */
    check_sts(status = ippiInpaint_8u_C1R(pSrc, srcStep, pDst, dstStep, roiSize, pInpState,
pInpBuffer));

    EXIT_MAIN
    ippsFree(pFMBuffer);
    ippsFree(pFMDist);
    ippsFree(pInpBuffer);
    ippiFree(pInpStateBuf);
    ippiFree(pSrc);
    ippiFree(pDst);
    ippiFree(pMask);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

See Also

InpaintGetSize Computes the size of the state structure and work buffer for image inpainting.

InpaintInit Initializes the state structure for image inpainting.

Image Segmentation

THis section describes the functions that perform image segmentation using different techniques. These functions allow to extract parts of the image that can be associated with objects of the real world. Watershed and gradient segmentation are region-based methods to split image into the distinctive areas.

Background/foreground segmentation allows for distinguishing between moving objects and stable areas of the background.

- LabelMarkersGetBufferSize
- LabelMarkers
- MarkSpecklesGetBufferSize

- MarkSpeckles
- SegmentWatershedGetBufferSize
- SegmentWatershed
- SegmentGradientGetBufferSize
- SegmentGradient
- BoundSegments

LabelMarkersGetBufferSize

Computes the size of the working buffer for the marker labeling.

Syntax

```
IppStatus ippiLabelMarkersGetBufferSize_8u_C1R(IppiSize roiSize, int* pBufSize);
IppStatus ippiLabelMarkersGetBufferSize_8u32s_C1R(IppiSize roiSize, int* pBufSize);
IppStatus ippiLabelMarkersGetBufferSize_16u_C1R(IppiSize roiSize, int* pBufSize);
```

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`roiSize`	Size of the source image ROI in pixels.
`pBufSize`	Pointer to the computed size of the working buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function computes the size of the working buffer required for the ippiLabelMarkers function. The buffer with the length `pBufSize[0]` can be used to segment images with width and/or height that is equal to or less than the corresponding field of the parameter `roiSize`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if the pointer `pBufSize` is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize` has a field with zero or negative value.

LabelMarkers

Labels markers in image with different values.

Syntax

```
IppStatus ippiLabelMarkers_8u_C1IR(Ipp8u* pMarker, int markerStep, IppiSize roiSize,
int minLabel, int maxLabel, IppiNorm norm, int* pNumber, Ipp8u* pBuffer);
```

```
IppStatus ippiLabelMarkers_8u32s_C1R(Ipp8u* pSrcMarker, int srcMarkerStep, Ipp32s*
pDstMarker, int dstMarkerStep, IppiSize roiSize, int minLabel, int maxLabel, IppiNorm
norm, int* pNumber, Ipp8u* pBuffer);
```

```
IppStatus ippiLabelMarkers_16u_C1IR(Ipp16u* pMarker, int markerStep, IppiSize roiSize,
int minLabel, int maxLabel, IppiNorm norm, int* pNumber, Ipp8u* pBuffer);
```

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pMarker`	Pointer to the source and destination image ROI (for in-place flavors).
`markerStep`	Distance in bytes between starts of consecutive lines in the source and destination image.
`pSrcMarker`	Pointer to the source image ROI (for not-in-place flavors).
`srcMarkerStep`	Distance, in bytes, between the starting points of consecutive lines in the source image (for not-in-place flavors).
`pDstMarker`	Pointer to the source and destination image ROI (for not-in-place flavors).
`dstMarkerStep`	Distance, in bytes, between the starting points of consecutive lines in the destination image (for not-in-place flavors).
`minLabel`	Minimal value of the marker label (`0< minLabel ≤ maxLabel`).
`maxLabel`	Maximal value of the marker label (`minLabel ≤ maxLabel` < 255 for 8-bit markers, and `minLabel ≤ maxLabel < 65535` for 16-bit markers, and `minLabel ≤ maxLabel` < (2^{31}-1) for 32-bit markers).
`roiSize`	Size of the source and destination image ROI in pixels.
`norm`	Specifies type of the norm to form the mask for marker propagation:

`ippiNormInf`	Infinity norm (8-connectivity);
`ippiNormL1`	L1 norm (4-connectivity).

`pNumber`	Pointer to the number of markers.
`pBuffer`	Pointer to the working buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function labels markers in the destination image with different integer values. Each connected set of non-zero image pixels is treated as the separate marker. 4- or 8-connectivity can be used depending on the norm type. All pixels belonging to the same marker are set to the same value from the interval [`minLabel`, `maxLabel`]. Two markers can be labeled with the same value if the number of connected components exceeds `minLabel-maxLabel + 1`. The image with labeled markers can be used as the seed image for segmentation by functions ippiSegmentWatershed or ippiSegmentGradient functions.

The function requires the working buffer `pBuffer` whose size should be computed by the function ippiLabelMarkersGetBufferSize beforehand.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize` has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if `markerStep` is less than `roiSize.width` * < `pixelSize`>.
`ippStsNotEvenStepErr`	Indicates an error condition if `markerStep`, `srcMarkerStep`, or `dstMarkerStep` is not divisible by `respective<pixelSize>`.
`ippStsBadArgErr`	Indicates an error condition if one of the `minLabel`, `maxLabel`, and `norm` has an illegal value.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

// A simple example of labelling connected non-zero components with
// different label values using Intel(R) Integrated Performance Primitives (Intel(R) IPP)
functions:
//     ippiLabelMarkersGetBufferSize_16u_C1R
//     ippiLabelMarkers_16u_C1IR

#include <stdio.h>
#include "ipp.h"
```

```
#define WIDTH  8 /* source image width */
#define HEIGHT 8 /* source image height */

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define check_sts(st)        \
    if ((st) != ippStsNoErr) \
        goto exitLine; /* Go to Exit if Intel(R) IPP function returned status different from
ippStsNoErr */

static Ipp16u pMarker[WIDTH * HEIGHT] = /* Pointer to source and destination marker image */
    {0, 12, 13, 0,  0, 0,  0,  0,  0, 22, 23, 24, 0, 0,  0, 0,  0, 32, 33, 34, 0, 0,  0,  0,  0,
42, 43, 44, 0,  0, 0, 0,
     0, 0,  53, 54, 0, 56, 57, 58, 0, 0,  0,  0,  0, 66, 0, 68, 0, 0,  0,  57, 0, 76, 77, 78, 0,
0,  0,  0,  68, 0, 0, 0};

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main(void)
{
    IppStatus status = ippStsNoErr;
    int markerStep = WIDTH * sizeof(Ipp16u); /* Steps, in bytes, through the source images */
    IppiSize roiSize = {WIDTH, HEIGHT};      /* Size of source ROI in pixels */
    int minLabel = 1;                        /* Minimal label value */
    int maxLabel = 30000;                    /* Maximal label value */
    int markersNum = 0;                      /* Pointer to number of markers */
    Ipp8u *pBuffer = NULL;                   /* Pointer to the work buffer */
    int bufferSize = 0;

    /* Calculate size of temporary buffer */
    check_sts(status = ippiLabelMarkersGetBufferSize_16u_C1R(roiSize, &bufferSize));

    pBuffer = ippsMalloc_8u(bufferSize);

    /* Label connected non-zero components with different label values */
    check_sts(status = ippiLabelMarkers_16u_C1IR(pMarker, markerStep, roiSize, minLabel,
maxLabel, ippiNormL1, &markersNum, pBuffer));

    EXIT_MAIN
    ippsFree(pBuffer);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

MarkSpecklesGetBufferSize

Computes the size of the external work buffer for speckle marking.

Syntax

```
IppStatus ippiMarkSpecklesGetBufferSize(IppiSize roiSize, IppDataType dataType, int
numChannels, int* pBufferSize);
```

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

roiSize	Size of the source and destination image ROI in pixels.
dataType	Data type of the source and destination image.
numChannels	Number of channels in the image. Possible value is 1.
pBufferSize	Pointer to the computed size (in bytes) of the external work buffer.

Description

This function computes the size of the external work buffer for the MarkSpeckles function.

For an example on how to use this function, refer to the example provided with the MarkSpeckles function description.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error when pBufferSize is NULL.
ippStsSizeErr	Indicates an error when roiSize has a field with a zero or negative value.
ippStsDataTypeErr	Indicates an error when dataType has an illegal value.
ippStsNumChannelErr	Indicates an error when numChannels has an illegal value.

See Also

MarkSpeckles Marks small noise blobs (speckles) in an image.

MarkSpeckles

Marks small noise blobs (speckles) in an image.

Syntax

```
IppStatus ippiMarkSpeckles_<mod>(Ipp<datatype>* pSrcDst, int srcDstStep, IppiSize
roiSize, Ipp<datatype> speckleVal, int maxSpeckleSize, Ipp<datatype> maxPixDiff,
IppiNorm norm, Ipp8u* pBuffer);
```

Supported values for `mod`:

`8u_C1IR`	`16u_C1IR`	`16s_C1IR`	`32f_C1IR`

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrcDst`	Pointer to the source and destination image.
`srcDstStep`	Distance, in bytes, between the starting points of consecutive lines in the source and destination image.
`roiSize`	Size of the source and destination image ROI in pixels.
`speckleVal`	Value to set to the speckles.
`maxSpeckleSize`	Maximum size of the image component to consider it as a speckle.
`maxPixDiff`	Maximum difference between neighboring disparity pixels to put them into the same component.
`norm`	Type of the norm to form the mask for marker propagation. Possible value is: • `ippiNormL1` • L1 norm (4-connectivity)
`pBuffer`	Pointer to the work buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function marks small noise blobs (speckles) in the source image.

The `pSrcDst` parameter points to the processed source and destination image ROI.

The function finds small connected components and set them to the `speckleVal` value. This function marks only components with size that is less than, or equal to `maxSpeckleSize`. Pixels of the image belong to the same connected component if the difference between adjacent pixels (considering 4-connected adjacency) is less than, or equal to the `maxSpeckleSize` value.

NOTE This release does not support 8-connectivity.

The function does not process the pixels of the image that already have the `speckleVal` value.

Before using the `ippiMarkSpeckles` function, compute the size of the external buffer using the MarkSpecklesGetBufferSize function.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when `pSrcDst` or `pBufferSize` is `NULL`.
`ippStsSizeErr`	Indicates an error when `roiSize` has a field with a zero or negative value.
`ippStsNotEvenStepErr`	Indicates an error when one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
`ippStsNormErr`	Indicates an error when `norm` has an incorrect or not supported value.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

// A simple example of performing filtering of small noise blobs (speckles)
// in the image using Intel(R) Integrated Performance Primitives (Intel(R) IPP) functions:
//     ippiMarkSpecklesGetBufferSize
//     ippiMarkSpeckles_8u_C1IR

#include <stdio.h>
#include "ipp.h"

#define WIDTH  128 /* image width  */
#define HEIGHT 64  /* image height */

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define check_sts(st)        \
    if ((st) != ippStsNoErr) \
        goto exitLine; /* Go to Exit if Intel(R) IPP function returned status different from
ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
```

```
check and will return an appropriate status  */

int main(void)
{
    IppStatus status = ippStsNoErr;
    int numChannels = 1;
    Ipp8u *pSrcDst = NULL;              /* Pointers to source/destination images */
    int srcDstStep = 0;                 /* Steps, in bytes, through the source/destination
images */
    IppiSize roiSize = {WIDTH, HEIGHT}; /* Size of source/destination ROI in pixels */
    Ipp8u speckleVal = 255;             /* Disparity value */
    Ipp8u maxPixDif = 1;                /* Maximum difference between neighbor pixels */
    Ipp32s maxSpeckleSize = 8;          /* Maximum difference between neighbor pixels */
    Ipp8u *pBuffer = NULL;              /* Pointer to the work buffer */
    int bufferSize = 0;

    /* Computes the temporary work buffer size */
    check_sts(status = ippiMarkSpecklesGetBufferSize(roiSize, ipp8u, numChannels, &bufferSize));

    pBuffer = ippsMalloc_8u(bufferSize);
    pSrcDst = ippiMalloc_8u_C1(roiSize.width, roiSize.height, &srcDstStep);

    check_sts(status = ippiMarkSpeckles_8u_C1IR(pSrcDst, srcDstStep, roiSize, speckleVal,
maxSpeckleSize, maxPixDif, ippiNormL1, pBuffer));

    EXIT_MAIN
    ippiFree(pSrcDst);
    ippsFree(pBuffer);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

See Also

Regions of Interest in Intel IPP

User-defined Border Types

FilterGaussianGetBufferSize Computes the size of the Gaussian specification structure and the size of the external work buffer for Gaussian filtering with user-defined borders.

FilterGaussianInit Initializes the Gaussian context structure.

SegmentWatershedGetBufferSize

Computes the size of the working buffer for the watershed segmentation.

Syntax

```
IppStatus ippiSegmentWatershedGetBufferSize_8u_C1R(IppiSize roiSize, int* pBufSize);

IppStatus ippiSegmentWatershedGetBufferSize_8u16u_C1R(IppiSize roiSize, int* pBufSize);

IppStatus ippiSegmentWatershedGetBufferSize_32f16u_C1R(IppiSize roiSize, int*
pBufSize);
```

Include Files

```
ippcv.h
```

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`roiSize`	Size of the source image ROI in pixels.
`pBufSize`	Pointer to the computed size of the working buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function computes the size of the working buffer required for the ippiSegmentWatershed function. The buffer with the length `pBufSize[0]` can be used to segment images with width and/or height that is equal to or less than the corresponding field of the parameter `roiSize`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if the pointer `pBufSize` is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize` has a field with zero or negative value.

SegmentWatershed

Performs marker-controlled watershed segmentation of an image.

Syntax

```
IppStatus ippiSegmentWatershed_8u_C1IR(const Ipp8u* pSrc, int srcStep, Ipp8u* pMarker,
int markerStep, IppiSize roiSize, IppiNorm norm, int flag, Ipp8u* pBuffer);
```

```
IppStatus ippiSegmentWatershed_8u16u_C1IR(const Ipp8u* pSrc, int srcStep, Ipp16u*
pMarker, int markerStep, IppiSize roiSize, IppiNorm norm, int flag, Ipp8u* pBuffer);
```

```
IppStatus ippiSegmentWatershed_32f16u_C1IR(const Ipp32f* pSrc, int srcStep, Ipp16u*
pMarker, int markerStep, IppiSize roiSize, IppiNorm norm, int flag, Ipp8u* pBuffer);
```

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance, in bytes, between the starting points of consecutive lines in the source image.
pMarker	Pointer to the ROI of the source and destination image of markers.
markerStep	Distance, in bytes, between the starting points of consecutive lines in the image with markers.
roiSize	Size of the source and destination image ROI, in pixels.
norm	Specifies the type of the norm to form the mask for marker propagation:

ippiNormInf	Infinity norm (8-connectivity, 3x3 rectangular mask)
ippiNormL1	L1 norm (4-connectivity, 3x3 cross mask)
ippiNormL2	Approximation of L2 norm (8-connectivity, 3x3 mask) [Bor86]
ippiNormFM	Fast marching distance [Telea04]

flag — Specifies the algorithm of segmentation. The value is a logical sum of the following mandatory values:

IPP_SEGMENT_QUEUE	Priority queue is used to define the order of pixel processing.
IPP_SEGMENT_DISTANCE	Distance transform algorithm is used for segmentation.

IPP_SEGMENT_BORDER_4	Pixels of the 4-connectivity border between image segments are marked with the IPP_MAX_8U (255) value.
IPP_SEGMENT_BORDER_8	Pixels of the 8-connectivity border between image segments are marked with the IPP_MAX_8U (255) value.

pBuffer	Pointer to the working buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function performs marker-controlled watershed segmentation of the source image. Non-zero pixels of the pMarker image belong to water source markers. Marker values propagate through the whole image according to the watershed algorithm. Image segments are formed by groups of connected pMarker pixels with the same value. The parameter norm controls marker propagation connectivity. Watershed segmentation is preferable for images with local minimums, for example, gradient images. Image markers generally correspond to these local minimums and can be created, for example, manually or using morphological reconstruction.

The parameter `flag` specifies how watershed segmentation is performed. This parameter is a logical sum of two values among the following supported values:

- Mandatory values specifying the algorithm of segmentation:

`IPP_SEGMENT_QUEUE`	Classic watershed segmentation scheme with the priority queue [Vincent91]
`IPP_SEGMENT_DISTANCE`	Watershed segmentation by calculating the topographic distance for each pixel [Lotufo00], [Meyer94]

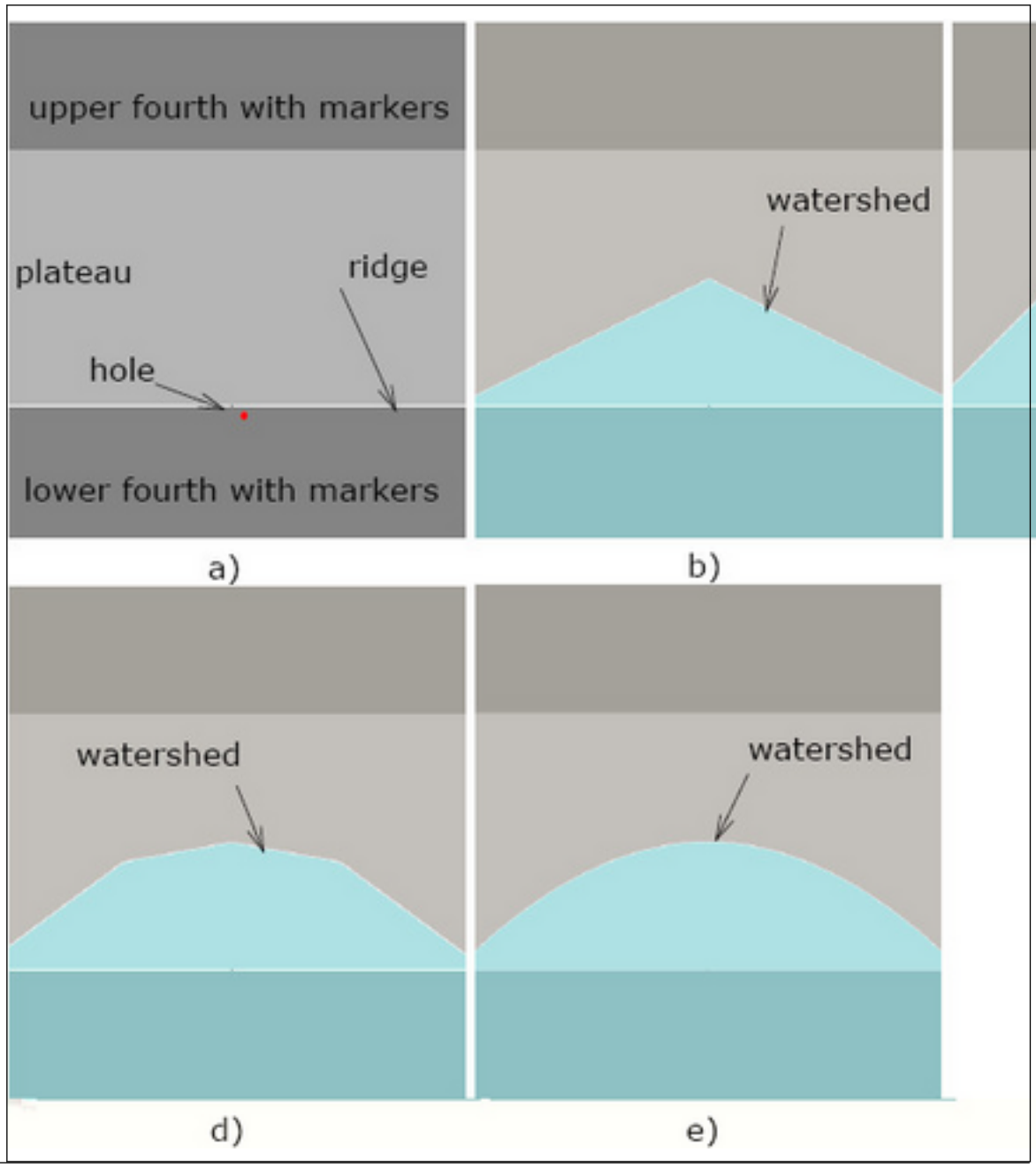
- Optional additional values of the `flag`: `IPP_SEGMENT_BORDER_4` and `IPP_SEGMENT_BORDER_8` specify the border of the segments. All pixels adjacent to the differently marked pixels are considered as border pixels, and their values are set to `IPP_MAX_8U` (255) for 8-bit markers, or `IPP_MAX_16U` (65535) for 16-bit markers. In this case, the value `IPP_MAX_8U(IPP_MAX_16U)` should not be used to mark segments. If these optional values are not specified, segments are formed without borders.

The function requires the working buffer `pBuffer`, which size should be computed by the function ippiSegmentWatershedGetBufferSize beforehand.

Figure Watershed Segmentation with Different Norms shows the plateau filling through the watershed segmentation with different values of the `norm` parameters. Initial image (a) has the labeled with markers upper and lower fourths with low pixel value, the central plateau between them, the ridge between the

plateau and the lower forth with one pixel hole in the center of it. The following pictures are segmentation results: b) - for L1 norm (block distance), c)- Linf norm (chessboard distance), d) - approximate L2 (Euclidian) norm [Bor86], e) Fast Marching distance.

Watershed Segmentation with Different Norms



Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if one of the srcStep or markerStep is less than roiSize.width * <pixelSize>.
ippNotEvenStsStepErr	Indicates an error condition if one of the srcStep or markerStep for 16-bit integer images is not divisible by 2.
ippStsBadArgErr	Indicates an error condition if norm has an illegal value.

SegmentGradientGetBufferSize

Computes the size of the working buffer for the gradient segmentation.

Syntax

```
IppStatus ippiSegmentGradientGetBufferSize_8u_C1R(IppiSize roiSize, int* pBufferSize);
IppStatus ippiSegmentGradientGetBufferSize_8u_C3R(IppiSize roiSize, int* pBufSize);
```

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

roiSize	Size of the source image ROI in pixels.
pBufSize, pBufferSize	Pointer to the computed size of the working buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function computes the size of the working buffer required for the ippiSegmentGradient function. The buffer with the length pBufSize[0] can be used to segment images with width and/or height that is equal to or less than the corresponding field of the parameter roiSize.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr	Indicates an error condition if the pointer pBufSize is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value.

SegmentGradient

Performs image segmentation by region growing to the least gradient direction.

Syntax

```
IppStatus ippiSegmentGradient_8u_C1IR(const Ipp8u* pSrc, int srcStep, Ipp8u* pMarker,
int markerStep, IppiSize roiSize, IppiNorm norm, int flags, Ipp8u* pBuffer);
```

```
IppStatus ippiSegmentGradient_8u_C3IR(const Ipp8u* pSrc, int srcStep, Ipp8u* pMarker,
int markerStep, IppiSize roiSize, IppiNorm norm, int flags, Ipp8u* pBuffer);
```

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Pointer to the source image ROI.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pMarker	Pointer to the ROI of the source and destination image of markers.
markerStep	Distance in bytes between starts of consecutive lines in the image of markers.
roiSize	Size of the source and destination image ROI in pixels.
norm	Specifies type of the norm to form the mask for marker propagation:

ippiNormInf	Infinity norm (8-connectivity, 3x3 rectangular mask);
ippiNormL1	L1 norm (4-connectivity, 3x3 cross mask);

flags — optional flag:

IPP_SEGMENT_BORDER_4	pixels of the 4-connectivity border between image segments are marked with value (IPP_MAX_8U)-1 (254).
IPP_SEGMENT_BORDER_8	pixels of the 8-connectivity border between image segments are marked with value (IPP_MAX_8U)-1 (254).

pBuffer	Pointer to the working buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function performs image segmentation by region growing with markers. Non-zero pixels of pMarker image belong to initial image regions. Marker values propagate through the whole image in the direction of the least value of the absolute value of the image gradient. For 3-channel image the gradient is calculated as the maximum of channel gradients. Image segments are formed by groups of connected pMarker pixels with the same value. The parameter norm controls marker propagation connectivity. Gradient segmentation is generally done for an image without explicit calculation of the image gradient. [Meyer92]

If IPP_SEGMENT_BORDER flag is defined, then the pixels adjacent to differently marked pixels are assumed to be border pixels and are set to a special value (254). This value must not be used to mark segments in this case.

Another special value (255) is used inside the function and can not be used to mark segment inany case.

The function requires the working buffer pBuffer whose size should be computed by the function ippiSegmentGradientGetBufferSize beforehand.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error condition if one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if one of the srcStep or markerStep is less than roiSize.width * <pixelSize>.
ippStsBadArgErr	Indicates an error condition if norm has an illegal value.

BoundSegments

Marks pixels belonging to segment boundaries.

Syntax

```
IppStatus ippiBoundSegments_8u_C1IR(Ipp8u* pMarker, int markerStep, IppiSize roiSize, Ipp8u val, IppiNorm norm);
```

```
IppStatus ippiBoundSegments_16u_C1IR(Ipp16u* pMarker, int markerStep, IppiSize roiSize, Ipp16u val, IppiNorm norm);
```

Include Files

ippcv.h

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pMarker`	Pointer to the ROI of the source and destination image of markers.
`markerStep`	Distance in bytes between starts of consecutive lines in the image of markers.
`roiSize`	Size of the source and destination image ROI in pixels.
`val`	Value of the boundary pixel.
`norm`	Specifies type of the norm gor pixel neighborhood:

ippiNormInf	Infinity norm (8-connectivity);
ippiNormL1	L1 norm (4-connectivity.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function detects segment boundaries in the `pMarker` image and sets border pixels to the value `val`. A segment is the set of connected pixels of the `pMarker` image with the same value not equal to `val`. After boundaries are marked, the `pMarker` image does not contain any pair of adjacent in `norm` pixels with the same value that not equal to `val`.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error condition if one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `roiSize` has a field with zero or negative value.
`ippStsStepErr`	Indicates an error condition if `markerStep` is less than `roiSize.width` * < `pixelSize`>.
`ippNotEvenStsStepErr`	Indicates an error condition if `markerStep` for 16-bit integer images is not divisible by 2.
`ippStsBadArgErr`	Indicates an error condition if `norm` has an illegal value.

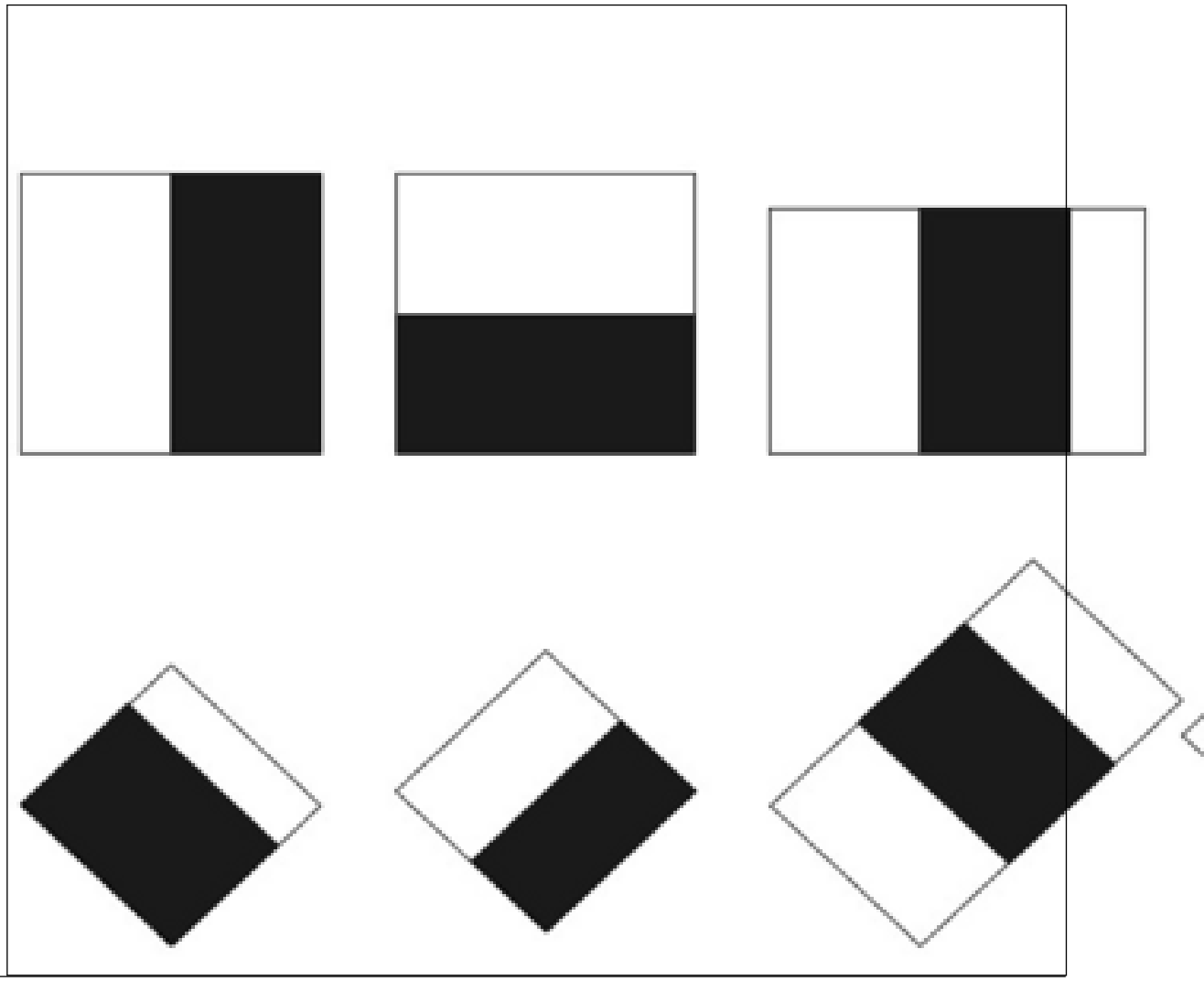
Pattern Recognition

- Object Detection Using Haar-like Features
- Local Binary Pattern (LBP) Operator

Object Detection Using Haar-like Features

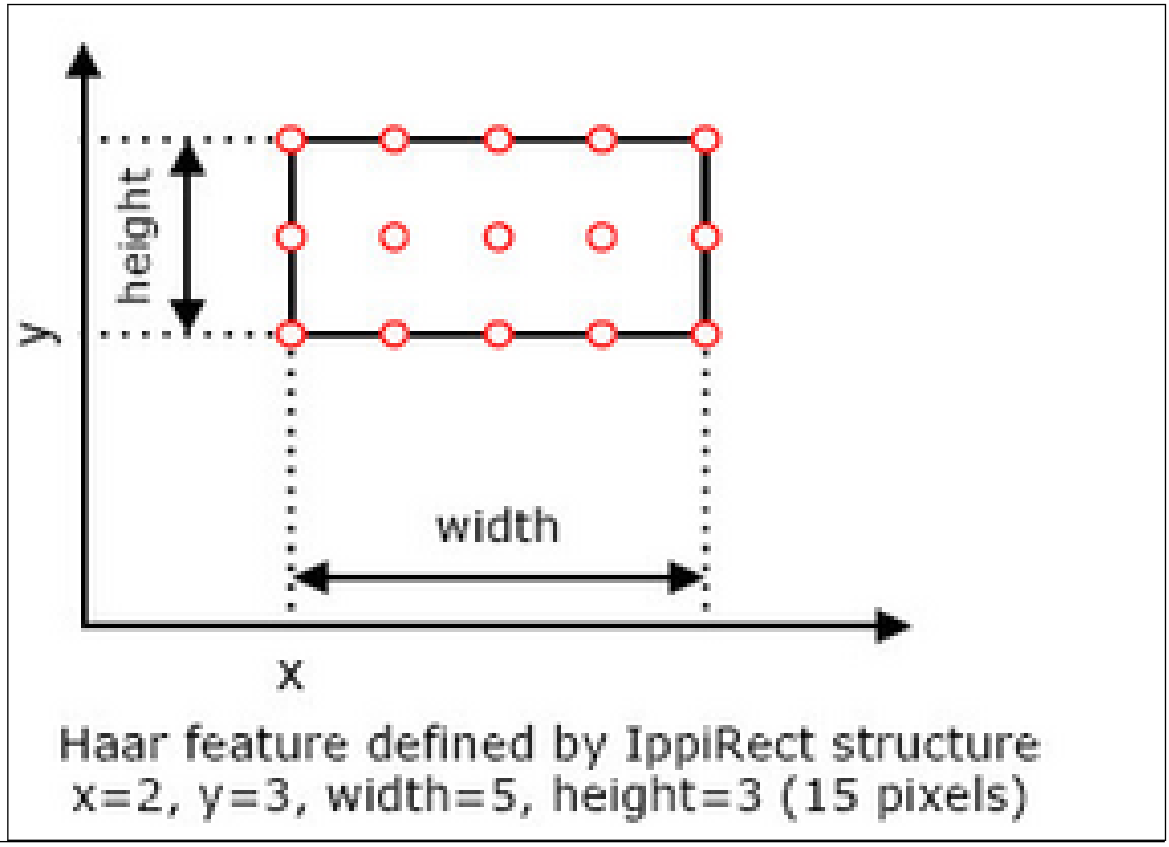
The object detector described in [Viola01] and [Lein02] is based on Haar classifiers. Each classifier uses `k` rectangular areas (Haar features) to make decision if the region of the image looks like the predefined image or not. Figure Types of Haar Features shows different types of Haar features.

Types of Haar Features



In the Intel IPP Haar features are represented using `IppRect` structure. Figure Representing Haar Features shows how it can be done for common and tilted features.

Representing Haar Features



When the classifier K_t is applied to the pixel (`i`, `j`) of the image `A`, it yields the value `val1(t)` if

image:: _images/GUID-A0744D92-6164-4F83-8096-F84C532AF2BE-low.jpg

and `val2(t)` otherwise.

Here w_l is a feature weight, `norm(i, j)` is the norm factor (generally the standard deviation on the rectangle containing all features), `threshold(t)`, `val1(t)` and `val2(t)` are parameters of the classifier. For fast computation the integral representation of an image is used. Haar classifiers are organized in sequences called `stages(classification stages)`. The stage value is the sum of its classifier values. During feature detecting stages are consequently applied to the region of the image until the stage value becomes less than the threshold value or all stages are passed.

- HaarClassifierGetSize
- HaarClassifierInit
- GetHaarClassifierSize
- TiltedHaarClassifierInit
- TiltHaarFeatures
- ApplyHaarClassifier
- ApplyMixedHaarClassifier

HaarClassifierGetSize

Computes the size of the structure for standard Haar classifiers.

Syntax

```
IppStatus ippiHaarClassifierGetSize(IppDataType dataType, IppiSize roiSize, const int* pNum, int length, int* pSize);
```

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`dataType`	Data type of the source image. Possible values: `ipp32f`, `ipp32s`.
`roiSize`	Maximal size of the source image ROI, in pixels.
`pNum`	Pointer to the array of Haar classifier lengths.
`length`	Number of classifiers in the stage.
`pSize`	Pointer to the size of Haar classifier structure.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function computes the size of the `pState` structure that is required to calculate the sequence of Haar classifiers - classification stage. The `i – th` classifier in the stage has `pNum[i]` rectangular features.

The length of the `pThreshold`, `pVal1`, and `pVal2` vectors used in the ippiHaarClassifierInit function is equal to `length`.

The length of the `pFeature` and `pWeight` vectors is equal to:

$$\sum_{i=0}^{length-1} pNum[i]$$

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error when any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error when: • `length` or one of the `pNum[i]` values is less than, or equal to zero

	• roiSize has a field with a zero or negative value
ippStsBadArgErr	Indicates an error when one of the features is defined incorrectly.
ippStsDataTypeErr	Indicates an error when dataType has an illegal value.

See Also

Regions of Interest in Intel IPP

HaarClassifierInit Initializes the structure for standard Haar classifiers.

ApplyHaarClassifier Applies a Haar classifier to an image.

HaarClassifierInit

Initializes the structure for standard Haar classifiers.

Syntax

```
IppStatus ippiHaarClassifierInit_32f(IppiHaarClassifier_32f** ppState, const IppiRect* pFeature, const Ipp32f* pWeight, const Ipp32f* pThreshold, const Ipp32f* pVal1, const Ipp32f* pVal2, const int* pNum, int length);
```

```
IppStatus ippiHaarClassifierInit_32s(IppiHaarClassifier_32s** ppState, const IppiRect* pFeature, const Ipp32s* pWeight, const Ipp32s* pThreshold, const Ipp32s* pVal1, const Ipp32s* pVal2, const int* pNum, int length);
```

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

ppState	Double pointer to the Haar classifier structure.
pFeature	Pointer to the array of features.
pWeight	Pointer to the array of feature weights.
pThreshold	Pointer to the array of classifier threshold values.
pVal1, pVal2	Pointers to the arrays of classifier result values.
pNum	Pointer to the array of Haar classifier lengths.
length	Number of classifiers in the stage.

Description

This function initializes the state structure that is required to calculate the sequence of Haar classifiers - classification stage. The `i – th` classifier in the stage has `pNum[i]` rectangular features. Each feature is defined by a certain rectangle with horizontal and vertical sides. The length of the `pThreshold`, `pVal1`, and `pVal2` vectors is equal to `length`. The length of `pFeature` and `pWeight` is equal to:

$$\sum_{i=0}^{length-1} pNum[i]$$

Result of applying classifiers to the image is computed using the formula in "Object Detection Using Haar-like Features".

All features of the classifier initialized by the `ippiHaarClassifierInit` function have vertical and horizontal sides (left part of Figure "Representing Haar Features"). Some of these features then can be tilted using the ippiTiltHaarFeatures function.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error when any of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error when `length` or one of the `pNum[i]` values is less than, or equal to zero.
`ippStsBadArgErr`	Indicates an error when one of the features is defined incorrectly.

See Also

HaarClassifierGetSize Computes the size of the structure for standard Haar classifiers.

ApplyHaarClassifier Applies a Haar classifier to an image.

Object Detection Using Haar-like Features

TiltHaarFeatures Modifies a Haar classifier by tilting specified features.

GetHaarClassifierSize

Returns the size of the Haar classifier.

Syntax

```
IppStatus ippiGetHaarClassifierSize_32f(IppiHaarClassifier_32f* pState, IppiSize* pSize);
```

```
IppStatus ippiGetHaarClassifierSize_32s(IppiHaarClassifier_32s* pState, IppiSize* pSize);
```

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pState`	Pointer to the Haar classifier structure.
`pSize`	Pointer to the size of Haar classifier structure.

Description

This function computes the minimum size of the window containing all features of the Haar classifier described by the `pState`.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error condition if the `pState` pointer is `NULL`.

TiltedHaarClassifierInit

Initializes the structure for tilted Haar classifiers.

Syntax

```
IppStatus ippiTiltedHaarClassifierInit_32f(IppiHaarClassifier_32f* pState, const IppiRect* pFeature, const Ipp32f* pWeight, const Ipp32f* pThreshold, const Ipp32f* pVal1, const Ipp32f* pVal2, const int* pNum, int length);
```

```
IppStatus ippiTiltedHaarClassifierInit_32s(IppiHaarClassifier_32s* pState, const IppiRect* pFeature, const Ipp32s* pWeight, const Ipp32s* pThreshold, const Ipp32s* pVal1, const Ipp32s* pVal2, const int* pNum, int length);
```

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pState`	Double pointer to the Haar classifier structure.
`pFeature`	Pointer to the array of features.
`pWeight`	Pointer to the array of feature weights.
`pThreshold`	Pointer to the array of classifier threshold values.

pVal1, pVal2	Pointers to the arrays of classifier result values.
pNum	Pointer to the array of Haar classifier lengths.
length	Number of classifiers in the stage.

Description

This function initializes the state structure that is required to calculate the sequence of Haar classifiers - classification stage. The i – th classifier in the stage has pNum[i] rectangular features. Each feature is defined by a certain rectangle with sides tilted by 45 degrees. You should specify the points with minimum and maximum row numbers. The length of the pFeature, pFeature, pWeight, pVal1, and pVal2 vectors is equal to:

$$\sum_{i=0}^{length-1} pNum[i]$$

Result of applying classifiers to the image is computed using the formula in "Object Detection Using Haar-like Features".

All features of the classifier initialized by the ippiTiltedHaarClassifierInit function have tilted sides (right part of Figure "Representing Haar Features").

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when any of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when: • length or one of the pNum[i] values is less than, or equal to zero • Sum of all elements of pNum is not equal to length
ippStsBadArgErr	Indicates an error when one of the features is defined incorrectly.

See Also

ApplyHaarClassifier Applies a Haar classifier to an image.

Object Detection Using Haar-like Features

TiltHaarFeatures Modifies a Haar classifier by tilting specified features.

TiltHaarFeatures

Modifies a Haar classifier by tilting specified features.

Syntax

```
IppStatus ippiTiltHaarFeatures_32f(const Ipp8u* pMask, int flag,
IppiHaarClassifier_32f* pState);
```

```
IppStatus ippiTiltHaarFeatures_32s(const Ipp8u* pMask, int flag,
IppiHaarClassifier_32s* pState);
```

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pMask`	Pointer to the mask vector.
`flag`	Flag to choose the direction of feature tilting.
`pState`	Pointer to the Haar classifier structure.

Description

This function tilts specified features of the Haar classifier. Before using this function, compute the size of the Haar classifier state structure using HaarClassifierGetSize and initialize the structure using TiltedHaarClassifierInit. Non-zero elements of previously prepared vector `pMask` indicates the features that are tilted. The `flag` parameter specifies how the features are tilted:

- if `flag` is equal to 0, the feature is tilted around the left top corner clockwise
- if `flag` is equal to 1, the feature is tilted around the bottom left corner counter-clockwise

This mixed classifier containing both common and tilted features can be used by the function ippiApplyMixedHaarClassifier.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when one of the specified pointers is `NULL`.
`ippStsBadArgErr`	Indicates an error when the classifier is tilted already.

See Also

HaarClassifierGetSize Computes the size of the structure for standard Haar classifiers.

TiltedHaarClassifierInit Initializes the structure for tilted Haar classifiers.

ApplyHaarClassifier

Applies a Haar classifier to an image.

Syntax

```
IppStatus ippiApplyHaarClassifier_32f_C1R(const Ipp32f* pSrc, int srcStep, const Ipp32f* pNorm, int normStep, Ipp8u* pMask, int maskStep, IppiSize roiSize, int* pPositive, Ipp32f threshold, IppiHaarClassifier_32f* pState);

IppStatus ippiApplyHaarClassifier_32s32f_C1R(const Ipp32s* pSrc, int srcStep, const Ipp32f* pNorm, int normStep, Ipp8u* pMask, int maskStep, IppiSize roiSize, int* pPositive, Ipp32f threshold, IppiHaarClassifier_32f* pState);

IppStatus ippiApplyHaarClassifier_32s_C1RSfs(const Ipp32s* pSrc, int srcStep, const Ipp32s* pNorm, int normStep, Ipp8u* pMask, int maskStep, IppiSize roiSize, int* pPositive, Ipp32s threshold, IppiHaarClassifier_32s* pState, int scaleFactor);
```

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the ROI in the source image of integrals.
`srcStep`	Distance, in bytes, between the starting points of consecutive lines in the source image.
`pNorm`	Pointer to the ROI in the source image of norm factors.
`normStep`	Distance, in bytes, between the starting points of consecutive lines in the image of the norm factors.
`pMask`	Pointer to the source and destination image of classification decisions.
`maskStep`	Distance, in bytes, between the starting points of consecutive lines in the image of classification decisions.
`pPositive`	Pointer to the number of positive decisions.
`roiSize`	Size of the source and destination images ROI in pixels.
`threshold`	Stage threshold value.
`pState`	Pointer to the Haar classifier structure.
`scaleFactor`	Scale factor (see Integer Result Scaling).

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function applies the Haar classifier to pixels of the source image ROI `pSrc`. The source image should be in the integral representation, it can be obtained by calling one of the integral functions beforehand. The sum of pixels on feature rectangles is computed as:

$$\sum_{l=1}^{k}(pSrc[i+y_l, j+x_l]-pSrc[i+Y_l, j+x_l]-pSrc[i+y_l, j+X_l]+pSrc[i+Y_l, j+$$

Here (y_l, x_l) and (Y_l, X_l) are coordinates of top left and right bottom pixels of `l - th` rectangle of the feature, and w_l is the feature weight. For `i = 0. roiSize.height - 1, j = 0. roiSize.width - 1` all pixels referred in the above formula should be allocated in memory.

The input value of `pPositive[0]` is used as a hint to choose the calculation algorithm. If it is greater than or equal to `roiSize.width*roiSize.height`, the value of the classifier is calculated in accordance with the above formula for all pixels of the input image. Otherwise the value of the classifier is calculated for all non-zero pixels of `pMask` image. If the sum is less than `threshold` than the negative decision is made and the value of the corresponding pixel of the `pMask` image is set to zero. The number of positive decisions is assigned to the `pPositive[0]`.

Before using this function, you need to compute the size of the state structure using HaarClassifierGetSize and initialize the structure using HaarClassifierInit or TiltedHaarClassifierInit.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error when `roiSize` has a field with a zero or negative value.
`ippStsStepErr`	Indicates an error when one of the image step values is less than `roiSize.width*<pixelSize>`.
`ippStsNorEvenStepErr`	Indicates an error when one of the image step values is not divisible by 4 for 32-bit images.

See Also

HaarClassifierGetSize Computes the size of the structure for standard Haar classifiers.

HaarClassifierInit Initializes the structure for standard Haar classifiers.

TiltedHaarClassifierInit Initializes the structure for tilted Haar classifiers.

ApplyMixedHaarClassifier

Applies a mixed Haar classifier to an image.

Syntax

```
IppStatus ippiApplyMixedHaarClassifier_32f_C1R(const Ipp32f* pSrc, int srcStep, const Ipp32f* pTilt, int tiltStep, const Ipp32f* pNorm, int normStep, Ipp8u* pMask, int maskStep, IppiSize roiSize, int* pPositive, Ipp32f threshold, IppiHaarClassifier_32f* pState);
```

```
IppStatus ippiApplyMixedHaarClassifier_32s32f_C1R(const Ipp32s* pSrc, int srcStep,
const Ipp32s* pTilt, int tiltStep, const Ipp32f* pNorm, int normStep, Ipp8u* pMask, int
maskStep, IppiSize roiSize, int* pPositive, Ipp32f threshold, IppiHaarClassifier_32f*
pState);

IppStatus ippiApplyMixedHaarClassifier_32s_C1RSfs(const Ipp32s* pSrc, int srcStep,
const Ipp32s* pTilt, int tiltStep, const Ipp32s* pNorm, int normStep, Ipp8u* pMask, int
maskStep, IppiSize roiSize, int* pPositive, Ipp32s threshold, IppiHaarClassifier_32s*
pState, int scaleFactor);
```

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pSrc`	Pointer to the ROI in the source image of integrals.
`srcStep`	Distance, in bytes, between the starting points of consecutive lines in the source image of integrals.
`pTilt`	Pointer to the ROI in the source image of tilted integrals.
`tiltStep`	Distance, in bytes, between the starting points of consecutive lines in the source image of tilted integrals.
`pNorm`	Pointer to the ROI in the source image of norm factors.
`normStep`	Distance, in bytes, between the starting points of consecutive lines in the image of the norm factors.
`pMask`	Pointer to the source and destination image of classification decisions.
`maskStep`	Distance, in bytes, between the starting points of consecutive lines in the image of classification decisions.
`pPositive`	Pointer to the number of positive decisions.
`roiSize`	Size of the source and destination images ROI in pixels.
`threshold`	Stage threshold value.
`pState`	Pointer to the mixed Haar classifier structure.
`scaleFactor`	Scale factor (see Integer Integer Result Scaling).

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function applies the mixed Haar classifier `pState` to the ROI of the source images `pSrc` and `pTilt`. The mixed Haar classifier is a classifier initialized by HaarClassifierInit

and then modified by the TiltHaarFeatures function. The source images must be in the integral representation, they can be obtained by calling one of the integral functions beforehand. Common features are applied to the `pSrc` image, and tilted features are applied to the `pTilt` image. The sum of pixels on feature rectangles is computed as:

$$\sum_{l=1}^{k} (pSrc[i+y_l, j+x_l] - pSrc[i+Y_l, j+x_l] - pSrc[i+y_l, j+X_l] + pSrc[i+Y_l, j+X_l]$$

or

$$\sum_{l=1}^{k} (pTilt[i+y_l, j+x_l] - pTilt[i+Y_l, j+x_l] - pTilt[i+y_l, j+X_l] + pTilt[i+Y_l, j+$$

Here (y_l, x_l) and (Y_l, X_l) are coordinates of top left and right bottom pixels of `l - th` rectangle of the feature , and w_l is the feature weight. For `i = 0. roiSize.height - 1`, `j = 0. roiSize.width - 1` all pixels referred in the above formula should be allocated in memory.

The input value of `pPositive[0]` is used as a hint to choose the calculation algorithm. If it is greater than or equal to `roiSize.width*roiSize.height` the value of the classifier is calculated in accordance with the above formula for all pixels of the input image. Otherwise the value of the classifier is calculated for all non-zero pixels of `pMask` image. If the sum is less than `threshold` than the negative decision is made and the value of the corresponding pixel of the `pMask` image is set to zero. The number of positive decisions is assigned to the `pPositive[0]`.

Return Values

`ippStsNoErr`	Indicates no error.
`ippStsNullPtrErr`	Indicates an error when one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error when `roiSize` has a field with a zero or negative value.
`ippStsStepErr`	Indicates an error when one of the image step values is less than `roiSize.width*<pixelSize>`.
`ippStsNorEvenStepErr`	Indicates an error when one of the image step values is not divisible by 4 for 32-bit images.

See Also

Regions of Interest in Intel IPP

HaarClassifierInit Initializes the structure for standard Haar classifiers.

TiltHaarFeatures Modifies a Haar classifier by tilting specified features.

Local Binary Pattern (LBP) Operator

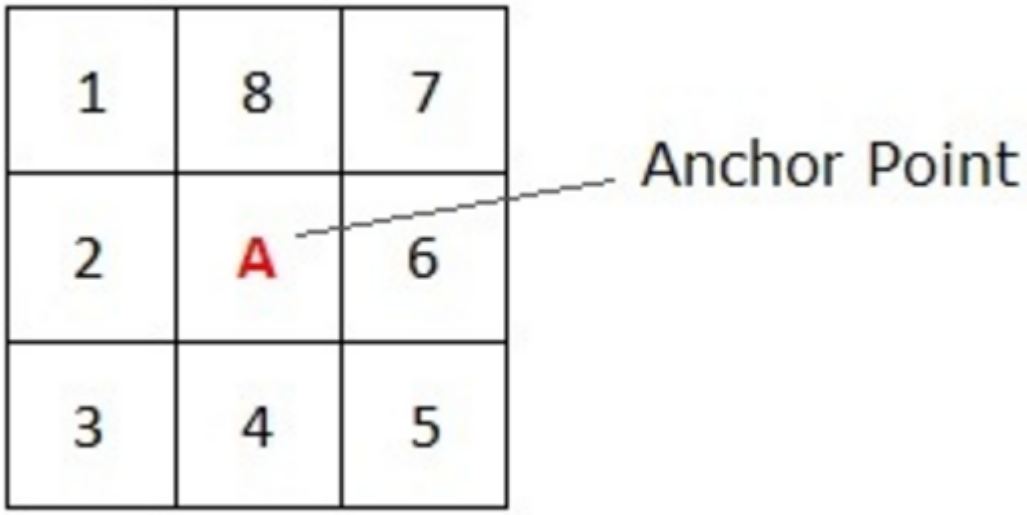
The local binary pattern (LBP) operator transforms an image into an array, or to an image with integer labels. Integer labels describe small-scale view of the image. For grayscale images, these labels represent a texture descriptor of the image. Integer labels statistics are used for image analysis. Changes of the monotonic gray level do not affect the LBP operator.

Intel® IPP functions described in this section use LBP operators with mask size 3x3 and 5x5.

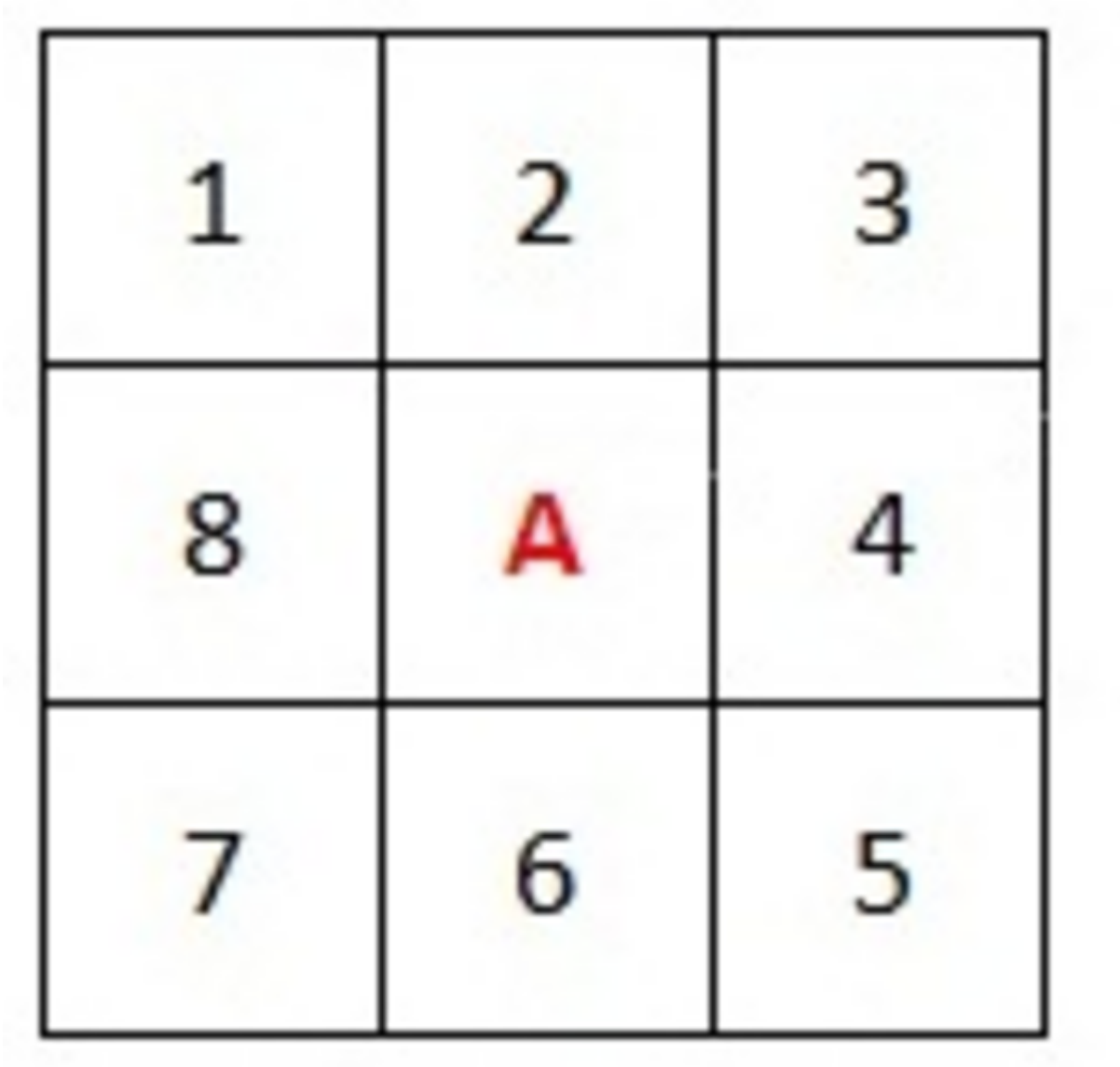
The `LBPImageMode` functions support four modes of LBP calculation set by the `mode` parameter. The `LBPImage` functions compute LBP similar to the `LBPImageMode` functions with the `mode` value equal to 1.

The LBP operator with 3x3 mask uses neighborhood consisting of eight pixels, as shown in the figures below.

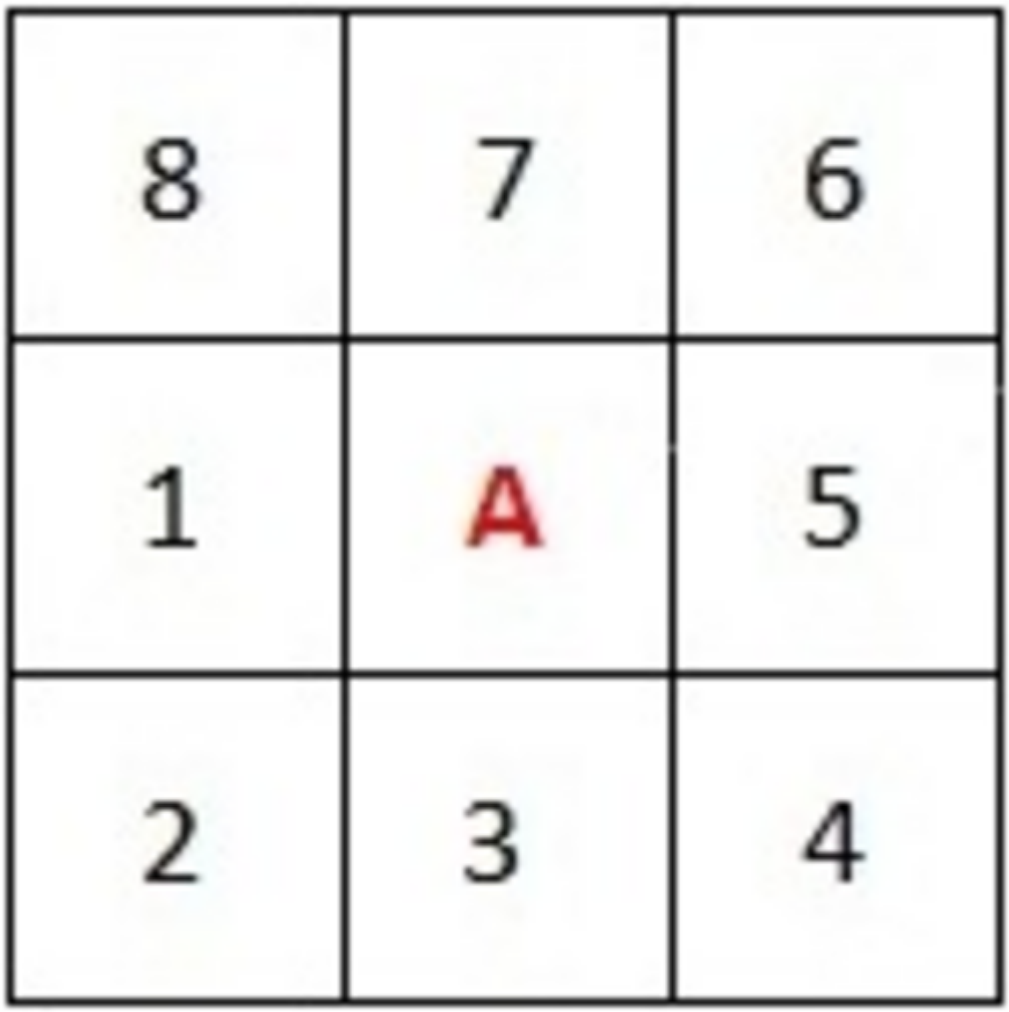
`mode = 0`:



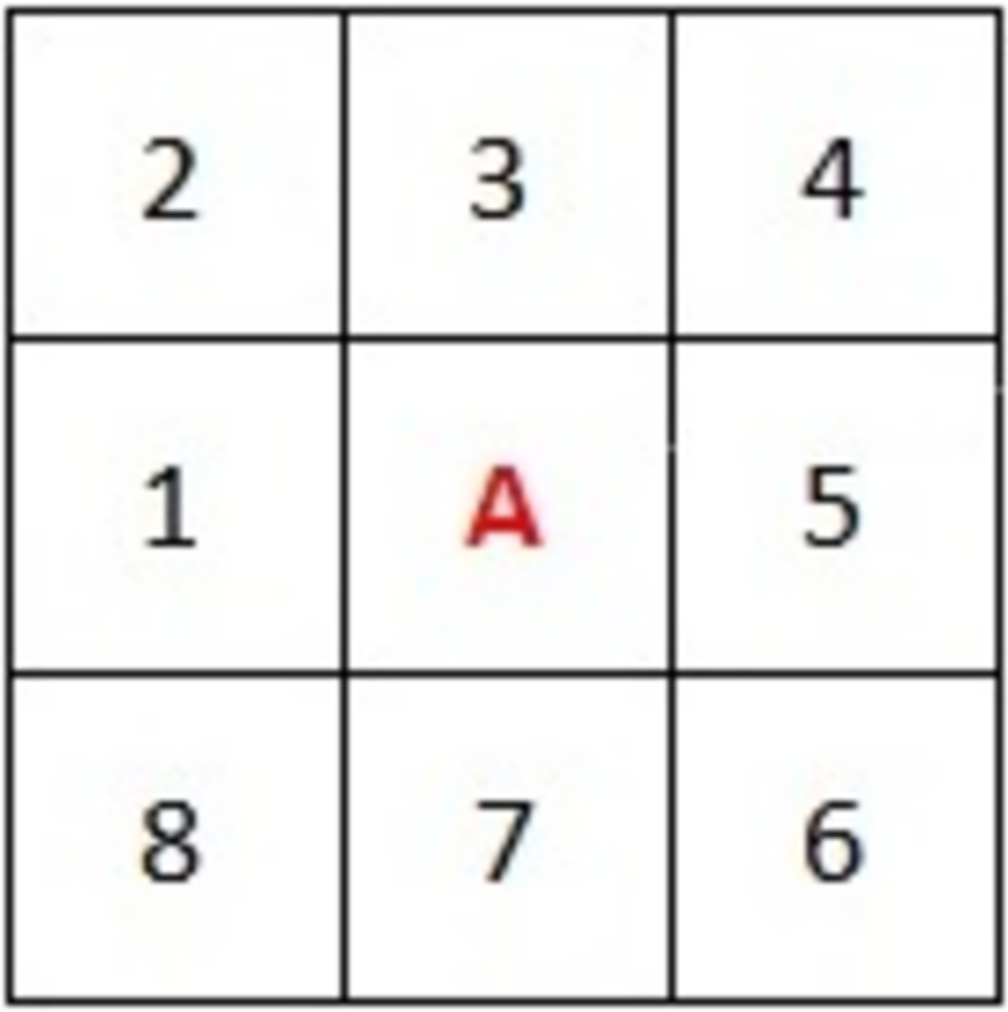
`mode = 1`:



mode = 2:

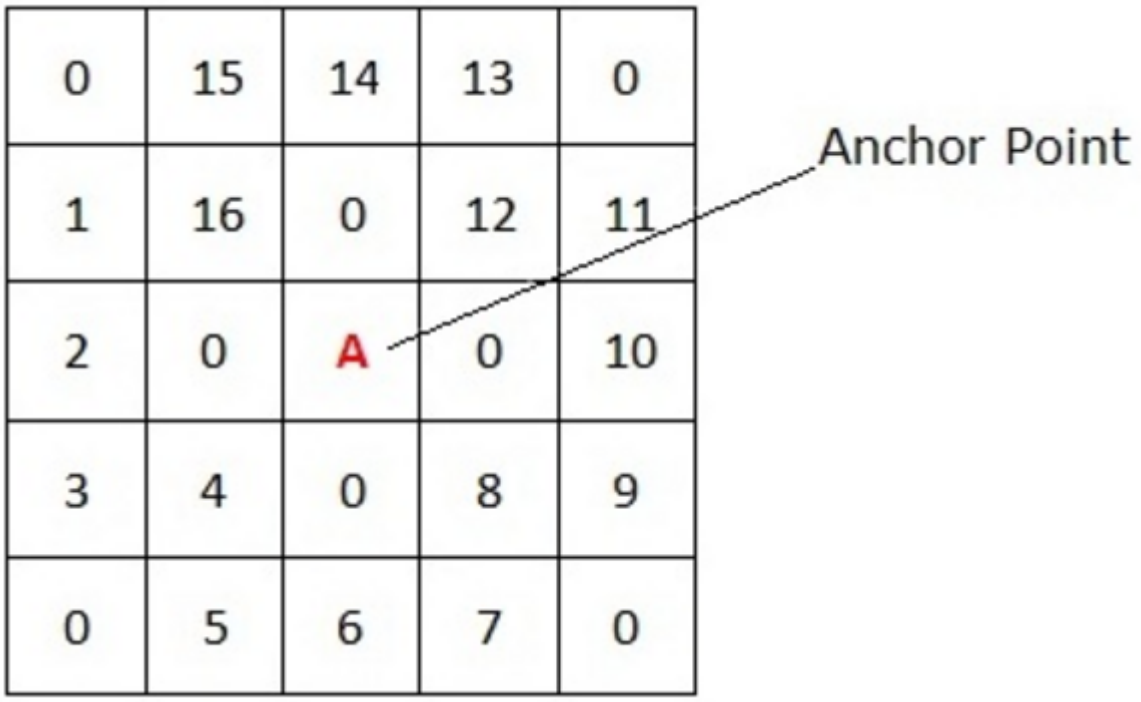


mode = 3:



The LBP operator with 5x5 mask uses neighborhood consisting of 16 pixels, as shown in the figures below.

`mode = 0`:



mode = 1:

0	3	4	5	0
1	2	0	6	7
16	0	A	0	8
15	14	0	10	9
0	13	12	11	0

mode = 2:

0	14	13	12	0
16	15	0	11	10
1	0	A	0	9
2	3	0	7	8
0	4	5	6	0

mode = 3:

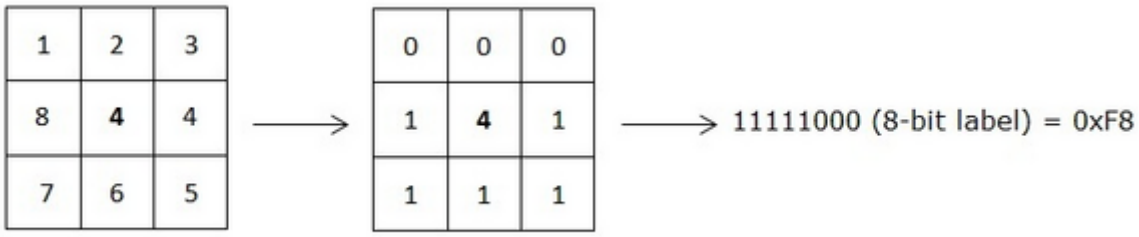
0	4	5	6	0
2	3	0	7	8
1	0	A	0	9
16	15	0	11	10
0	14	13	12	0

In the above figures:

- Numbers indicate the position of the corresponding bit in a resulting label
- The A letter indicates the anchor point position.

The LBP operator does the following when processing an image:

- Compares each pixel neighboring to the anchor with the anchor pixel in accordance with the neighboring pixel order. If the neighboring pixel value is more than, or equal to the anchor point value, the result is 1. If the neighboring pixel value is less than the anchor point value, the result is 0.
- Puts the result of comparison to the corresponding bit of the resulting label, as shown in the figure below.



- LBPImageMode
- LBPImageHorizCorr

LBPImageMode

Calculates LBP of the image according to the specified mode.

Syntax

```
IppStatus ippiLBPImageMode3x3_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize, int mode, IppiBorderType
borderType, const Ipp<srcDatatype>* borderValue);
```

Supported values for `mod`:

`8u_C1R`	`32f8u_C1R`

```
IppStatus ippiLBPImageMode5x5_<mod>(const Ipp<srcDatatype>* pSrc, int srcStep,
Ipp<dstDatatype>* pDst, int dstStep, IppiSize dstRoiSize, int mode, IppiBorderType
borderType, const Ipp<srcDatatype>* borderValue);
```

`8u_C1R`	`8u16u_C1R`	`32f8u_C1R`	`32f16u_C1R`

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source image ROI.
`srcStep`	Distance, in bytes, between the starting points of consecutive lines in the source image.
`pDst`	Pointer to the destination image ROI.
`dstStep`	Distance, in bytes, between the starting points of consecutive lines in the destination image.
`dstRoiSize`	Size of the destination ROI, in pixels.

mode	Mode for LBP calculation. Supported values are 0, 1, 2, 3.
borderType	Type of border. Possible values are: • ippBorderRepl • Border is replicated from the edge pixels. • ippBorderInMem • Border is obtained from the source image pixels in memory.
borderValue	Constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.

Description

These functions operate with ROI (see Regions of Interest in Intel IPP).

The ippiLBPImageMode3x3 and ippiLBPImageMode5x5 functions calculate LBP of the pSrc image ROI according to the mode value. The result is stored in the pDst destination image.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error if dstRoiSize has a field with a zero or negative value.
ippStsBadArgErr	Indicates an error when border has an illegal value.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

// A simple example of calculating the LBP of the image
// using Intel(R) Integrated Performance Primitives (Intel(R) IPP) function:
//     ippiLBPImageMode3x3_8u_C1R

#include <stdio.h>
```

```
#include "ipp.h"

#define WIDTH  128 /* image width  */
#define HEIGHT 64  /* image height */

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define check_sts(st)        \
    if ((st) != ippStsNoErr) \
        goto exitLine; /* Go to Exit if Intel(R) IPP function returned status different from
ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main(void)
{
    IppStatus status = ippStsNoErr;
    Ipp8u *pSrc = NULL, *pDst = NULL;   /* Pointers to source/destination images */
    int srcStep = 0, dstStep = 0;       /* Steps, in bytes, through the source/destination
images */
    IppiSize roiSize = {WIDTH, HEIGHT}; /* Size of source/destination ROI in pixels */
    int mode = 0;                       /* Specifies how LBP is created */
    const Ipp8u borderValue[1] = {3};

    pSrc = ippiMalloc_8u_C1(roiSize.width, roiSize.height, &srcStep);
    pDst = ippiMalloc_8u_C1(roiSize.width, roiSize.height, &dstStep);

    check_sts(status = ippiLBPImageMode3x3_8u_C1R(pSrc, srcStep, pDst, dstStep, roiSize, mode,
ippBorderRepl, borderValue));

    EXIT_MAIN
    ippiFree(pSrc);
    ippiFree(pDst);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

See Also

Regions of Interest in Intel IPP

LBPImageHorizCorr

Calculates a correlation between two LBPs.

Syntax

```
IppStatus ippiLBPImageHorizCorr_<mod>(const Ipp<datatype>* pSrc1, int src1Step, const
Ipp<datatype>* pSrc2, int src2Step, Ipp<datatype>* pDst, int dstStep, IppiSize
dstRoiSize, int horShift, IppiBorderType borderType, const Ipp<datatype>* borderValue);
```

Supported values for `mod`:

`8u_C1R`	`16u_C1R`

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc1, pSrc2	Pointers to the source image ROI.
src1Step, src2Step	Distance, in bytes, between the starting points of consecutive lines in the source image.
pDst	Pointer to the destination image ROI.
dstStep	Distance, in bytes, between the starting points of consecutive lines in the destination image.
dstRoiSize	Size of the destination ROI in pixels.
horShift	Horizontal shift of the pSrc2 image.
borderType	Type of border. Possible values are: • ippBorderRepl • Border is replicated from the edge pixels. • ippBorderInMem • Border is obtained from the source image pixels in memory.
borderValue	Constant value to assign to pixels of the constant border. This parameter is applicable only to the ippBorderConst border type.

Description

This function operates with ROI.

This function calculates the difference between two LBP images. The result is stored in the pDst destination image.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error if dstRoiSize has a field with zero or negative value.
ippStsBadArgErr	Indicates an error when border has an illegal value.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

// A simple example of calculating a correlation between
// two LBPs using Intel(R) Integrated Performance Primitives (Intel(R) IPP) function:
//     ippiLBPImageHorizCorr_8u_C1R

#include <stdio.h>
#include "ipp.h"

#define WIDTH  128 /* image width */
#define HEIGHT 64  /* image height */

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define check_sts(st)        \
    if ((st) != ippStsNoErr) \
        goto exitLine; /* Go to Exit if Intel(R) IPP function returned status different from
ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main(void)
{
    IppStatus status = ippStsNoErr;
    Ipp8u *pSrc1 = NULL, *pSrc2 = NULL, *pDst = NULL; /* Pointers to source/destination images */
    int srcStep1 = 0, srcStep2 = 0, dstStep = 0;      /* Steps, in bytes, through the source/
destination images */
    IppiSize roiSize = {WIDTH, HEIGHT};               /* Size of source/destination ROI in
pixels */
    const Ipp8u borderValue[1] = {3};
    int shift = 4; /* Horizontal shift of the pSrc2 image */

    pSrc1 = ippiMalloc_8u_C1(roiSize.width, roiSize.height, &srcStep1);
    pSrc2 = ippiMalloc_8u_C1(roiSize.width, roiSize.height, &srcStep2);
    pDst = ippiMalloc_8u_C1(roiSize.width, roiSize.height, &dstStep);

    check_sts(status = ippiLBPImageHorizCorr_8u_C1R(pSrc1, srcStep1, pSrc2, srcStep2, pDst,
dstStep, roiSize, shift, ippBorderRepl, borderValue));

    EXIT_MAIN
    ippiFree(pSrc1);
```

```
    ippiFree(pSrc2);
    ippiFree(pDst);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

See Also

Regions of Interest in Intel IPP

Borders in Neighborhood Operations

User-defined Border Types

Camera Calibration and 3D Reconstruction

- Correction of Camera Lens Distortion

Correction of Camera Lens Distortion

Digital camera usually introduces significant distortion caused by the camera and lens. These distortions cause errors in any analysis of the image. The functions described in this section correct these distortion using intrinsic camera parameters and distortion coefficients. These intrinsic camera parameters are focal lengths `fx`, `fy`, and principal point coordinates `cx`, `cy`. The distortion is characterized by two coefficients of radial distortions k_1, k_2 and two coefficients of tangential distortions p_1, p_2.

The undistorted coordinates x_u and y_u of point with coordinates (x_d, y_d) are computed in accordance with the following formulas:

$$x_u = x_d \cdot \left(1 + k_1 r^2 + k_2 r^4\right) + 2p_1 x_d y_d + p_2 \cdot \left(r^2 + 2x\right.$$

$$y_u = y_d \cdot \left(1 + k_1 r^2 + k_2 r^4\right) + 2p_2 x_d y_d + p_1 \cdot \left(r^2 + 2y\right.$$

Here $r^2 = x_d^2 + y_d^2$, $x_d = (j-cx)/fx$, $y_d = (i-cy)/fy$; `i` and `j` are row and columns numbers of the pixel. The pixel value is computed using bilinear interpolation of four nearest pixel of the source image. If undistorted coordinates are outside the image, then the destination pixel is not changed.

- UndistortGetSize
- UndistortRadial
- CreateMapCameraUndistort

UndistortGetSize

Computes the size of the external buffer.

Syntax

```
IppStatus ippiUndistortGetSize(IppiSize roiSize, int* pBufferSize);
```

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

roiSize	Size of source and destination images ROI in pixels.
pBufferSize	Pointer to the computed value of the buffer size.

Description

This function computes the size of the temporary external buffer that is used by the functions ippiUndistortRadial. The buffer of the computed size can be used to process smaller images as well.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if pBufferSize is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value.

UndistortRadial

Corrects radial distortions of the single image.

Syntax

```
IppStatus ippiUndistortRadial_<mod>(const Ipp<datatype>* pSrc, int srcStep,
Ipp<datatype>* pDst, int dstStep, IppiSize roiSize, Ipp32f fx, Ipp32f fy, Ipp32f cx,
Ipp32f cy, Ipp32f k1, Ipp32f k2, Ipp8u* pBuffer);
```

Supported values for mod:

8u_C1R	16u_C1R	32f_C1R
8u_C3R	16u_C3R	32f_C3R

Include Files

ippcv.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h, ippi.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib, ippi.lib

Parameters

pSrc	Pointer to the ROI in the source distorted image.
srcStep	Distance in bytes between starts of consecutive lines in the source image.
pDst	Pointer to the ROI in the destination corrected image.
dstStep	Distance in bytes between starts of consecutive lines in the destination image.
roiSize	Size of source and destination images ROI in pixels.
fx	Focal lengths along the x axis.
fy	Focal lengths along the y axis.
cx	x-coordinate of the principal point.
cy	y-coordinate of the principal point.
k1	First coefficient of radial distortion.
k2	Second coefficient of radial distortion.
pBuffer	Pointer to the external buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function corrects radial distortions of the single source image pSrc and stores corrected image in the pDst. Correction is performed accounting camera parameters fx, fy, cx, cy and radial distortion parameters k1, k2. The function can also pass the pointer to the external buffer pBuffer whose size should be computed previously using the function ippiUndistortGetSize. If a null pointer is passed, slower computations without an external buffer will be performed.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if pSrc or pDst is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if srcStep is less than roiSize.width * <``pixelSize> , or ``dstStep is less than roiSize.width * <pixelSize>.
ippStsNotEvenStepErr	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images, or by 2 for short-integer images.
ippStsBadArgErr	Indicates an error if fx or fy is equal to 0.

CreateMapCameraUndistort

Creates look-up tables of coordinates of corrected image.

Syntax

```
IppStatus ippiCreateMapCameraUndistort_32f_C1R(Ipp32f* pxMap, int xStep, Ipp32f* pyMap,
int yStep, IppiSize roiSize, Ipp32f fx, Ipp32f fy, Ipp32f cx, Ipp32f cy, Ipp32f k1,
Ipp32f k2, Ipp32f p1, Ipp32f p2, Ipp8u* pBuffer);
```

Include Files

`ippcv.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`, `ippi.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`, `ippi.lib`

Parameters

`pxMap`	Pointer to the destination x coordinate look-up buffer.
`xStep`	Distance in bytes between starts of consecutive lines in the `pxMap` image.
`pyMap`	Pointer to the destination y coordinate look-up buffer.
`yStep`	Distance in bytes between starts of consecutive lines in the `pyMap` image.
`roiSize`	Size of source and destination images ROI in pixels.
`fx`	Focal lengths along the `x` axis.
`fy`	Focal lengths along the `y` axis.
`cx`	`x - coordinate` of the principal point.
`cy`	`y - coordinate` of the principal point.
`k1`	First coefficient of radial distortion.
`k2`	Second coefficient of radial distortion.
`p1`	First coefficient of tangential distortion.
`p2`	Second coefficient of tangential distortion.
`pBuffer`	Pointer to the external buffer.

Description

This function operates with ROI (see Regions of Interest in Intel IPP).

This function creates the look-up tables of `x - and y - coordinatespxMap` and `pyMap` respectively. These coordinates are computed in accordance with camera parameters `fx`, `fy`, `cx`, `cy`, and distortion parameters `k1`, `k2`, `p1`, `p2`. The created tables can be used by the Intel IPP function ippiRemap to remap the distorted source image and get the corrected image.

To accelerate the computations the function can pass the pointer to the external buffer `pBuffer` whose size should be computed previously using the function ippiUndistortGetSize. If a null pointer is passed, slower computations without an external buffer will be performed.

Return Values

ippStsNoErr	Indicates no error.
ippStsNullPtrErr	Indicates an error if pxMap or pyMap is NULL.
ippStsSizeErr	Indicates an error condition if roiSize has a field with zero or negative value.
ippStsStepErr	Indicates an error condition if : xStep is less than roiSize.width * <``pixelSize> , or ``yStep is less than roiSize.width * <pixelSize>.
ippStsNotEvenStepErr	Indicates an error condition if one of the step values is not divisible by 4 for floating-point images.
ippStsBadArgErr	Indicates an error when fx or fy is equal to 0.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

// A simple example of initialization of x and y maps for undistortion
// using Intel(R) Integrated Performance Primitives (Intel(R) IPP) functions:
//     ippiUndistortGetSize
//     ippiCreateMapCameraUndistort_32f_C1R
//     ippiRemap_32f_C1R

#include <stdio.h>
#include "ipp.h"

#define WIDTH  640 /* Source image width */
#define HEIGHT 480 /* Destination image width */

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define check_sts(st)        \
    if ((st) != ippStsNoErr) \
        goto exitLine; /* Go to Exit if Intel(R) IPP function returned status different from
ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
```

```
check and will return an appropriate status  */

int main(void)
{
    IppStatus status = ippStsNoErr;
    Ipp32f *pSrc = NULL, *pDst = NULL;   /* Pointers to source/destination images */
    int srcStep = 0, dstStep = 0;        /* Steps, in bytes, through the source/destination
images */
    IppiSize roiSize = {WIDTH, HEIGHT};  /* Size of source/destination ROI in pixels */
    Ipp32f *pxMap = NULL, *pyMap = NULL; /* Pointers to x and y maps */
    Ipp8u *pBuffer = NULL;               /* Pointer to work buffer */
    int buflen = 0;
    int xStep = 0, yStep = 0;                   /* Steps, in bytes, through the x and y maps */
    IppiRect rect = {0, 0, WIDTH, HEIGHT};      /* Region of interest in the source image */
    Ipp32f fx = WIDTH / 3.f, fy = HEIGHT / 2.f; /* Focal lengths */
    Ipp32f cx = WIDTH / 2.f, cy = HEIGHT / 2.f; /* Coordinates of principal point */
    Ipp32f k1 = 0.04f, k2 = 0.004f;             /* Coeffs of radial distortion */
    Ipp32f p1 = 0.02f, p2 = 0.02f;              /* Coeffs of tangential distortion */

    pSrc = ippiMalloc_32f_C1(roiSize.width, roiSize.height, &srcStep);
    pDst = ippiMalloc_32f_C1(roiSize.width, roiSize.height, &dstStep);
    pxMap = ippiMalloc_32f_C1(roiSize.width, roiSize.height, &xStep);
    pyMap = ippiMalloc_32f_C1(roiSize.width, roiSize.height, &yStep);

    check_sts(status = ippiSet_32f_C1R(0.1234567f, pSrc, srcStep, roiSize));

    check_sts(status = ippiUndistortGetSize(roiSize, &buflen));

    pBuffer = ippsMalloc_8u(buflen);

    /* Initialize x and y maps for undistortion by the ippiRemap function */
    check_sts(status = ippiCreateMapCameraUndistort_32f_C1R(pxMap, xStep, pyMap, yStep, roiSize,
fx, fy, cx, cy, k1, k2, p1, p2, pBuffer));

    /* Transform the source image by remapping its pixels */
    check_sts(status = ippiRemap_32f_C1R(pSrc, roiSize, srcStep, rect, pxMap, xStep, pyMap,
yStep, pDst, dstStep, roiSize, IPPI_INTER_LINEAR));

    EXIT_MAIN
    ippiFree(pyMap);
    ippiFree(pxMap);
    ippsFree(pBuffer);
    ippiFree(pSrc);
    ippiFree(pDst);

    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

3D Data Processing Functions

This section describes the Intel® Integrated Performance Primitives (Intel® IPP) functions that perform 3D data transforms - resizing, affine transform, and remapping, as well as functions for 3D data linear filtering.

- CopyConstBorder
- CopyReplicateBorder

- Filter
- FilterGetBufSize
- FilterBorder
- FilterBorderInit
- FilterBorderGetSize
- FilterMedian
- FilterMedianInit
- FilterMedianGetSize
- ResizeGetBufSize
- GetResizeCuboid
- Resize
- Remap
- WarpAffineGetBufSize
- WarpAffine

CopyConstBorder

Copies pixel values between two 3D images and adds border pixels with a constant value.

Syntax

```
IppStatus ipprCopyConstBorder_8u_C1V(const Ipp8u* pSrc, int srcPlaneStep, int srcStep,
IpprVolume srcRoiVolume, Ipp8u* pDst, int dstPlaneStep, int dstStep, IpprVolume
dstRoiVolume, int topBorderHeight, int leftBorderWidth, int forwardBorderDepth, const
Ipp8u* value);
```

```
IppStatus ipprCopyConstBorder_16u_C1V(const Ipp16u* pSrc, int srcPlaneStep, int
srcStep, IpprVolume srcRoiVolume, Ipp16u* pDst, int dstPlaneStep, int dstStep,
IpprVolume dstRoiVolume, int topBorderHeight, int leftBorderWidth, int
forwardBorderDepth, const Ipp16u* value);
```

```
IppStatus ipprCopyConstBorder_16s_C1V(const Ipp16s* pSrc, int srcPlaneStep, int
srcStep, IpprVolume srcRoiVolume, Ipp16s* pDst, int dstPlaneStep, int dstStep,
IpprVolume dstRoiVolume, int topBorderHeight, int leftBorderWidth, int
forwardBorderDepth, const Ipp16s* value);
```

```
IppStatus ipprCopyConstBorder_32f_C1V(const Ipp32f* pSrc, int srcPlaneStep, int
srcStep, IpprVolume srcRoiVolume, Ipp32f* pDst, int dstPlaneStep, int dstStep,
IpprVolume dstRoiVolume, int topBorderHeight, int leftBorderWidth, int
forwardBorderDepth, const Ipp32f* value);
```

```
IppStatus ipprCopyConstBorder_64f_C1V(const Ipp64f* pSrc, int srcPlaneStep, int
srcStep, IpprVolume srcRoiVolume, Ipp64f* pDst, int dstPlaneStep, int dstStep,
IpprVolume dstRoiVolume, int topBorderHeight, int leftBorderWidth, int
forwardBorderDepth, const Ipp64f* value);
```

Platform-aware functions

```
IppStatus ipprCopyConstBorder_8u_C1V_L(const Ipp8u* pSrc, IppSizeL srcPlaneStep,
IppSizeL srcStep, IpprVolumeL srcRoiVolume, Ipp8u* pDst, IppSizeL dstPlaneStep,
IppSizeL dstStep, IpprVolumeL dstRoiVolume, IppSizeL topBorderHeight, IppSizeL
leftBorderWidth, IppSizeL forwardBorderDepth, const Ipp8u* value);
```

```
IppStatus ipprCopyConstBorder_16u_C1V_L(const Ipp16u* pSrc, IppSizeL srcPlaneStep,
IppSizeL srcStep, IpprVolumeL srcRoiVolume, Ipp16u* pDst, IppSizeL dstPlaneStep,
IppSizeL dstStep, IpprVolumeL dstRoiVolume, IppSizeL topBorderHeight, IppSizeL
leftBorderWidth, IppSizeL forwardBorderDepth, const Ipp16u* value);
```

```
IppStatus ipprCopyConstBorder_16s_C1V_L(const Ipp16s* pSrc, IppSizeL srcPlaneStep,
IppSizeL srcStep, IpprVolumeL srcRoiVolume, Ipp16s* pDst, IppSizeL dstPlaneStep,
IppSizeL dstStep, IpprVolumeL dstRoiVolume, IppSizeL topBorderHeight, IppSizeL
leftBorderWidth, IppSizeL forwardBorderDepth, const Ipp16s* value);

IppStatus ipprCopyConstBorder_32f_C1V_L(const Ipp32f* pSrc, IppSizeL srcPlaneStep,
IppSizeL srcStep, IpprVolumeL srcRoiVolume, Ipp32f* pDst, IppSizeL dstPlaneStep,
IppSizeL dstStep, IpprVolumeL dstRoiVolume, IppSizeL topBorderHeight, IppSizeL
leftBorderWidth, IppSizeL forwardBorderDepth, const Ipp32f* value);

IppStatus ipprCopyConstBorder_64f_C1V_L(const Ipp64f* pSrc, IppSizeL srcPlaneStep,
IppSizeL srcStep, IpprVolumeL srcRoiVolume, Ipp64f* pDst, IppSizeL dstPlaneStep,
IppSizeL dstStep, IpprVolumeL dstRoiVolume, IppSizeL topBorderHeight, IppSizeL
leftBorderWidth, IppSizeL forwardBorderDepth, const Ipp64f* value);
```

Include Files

`ippi.h`

`ippi_l.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

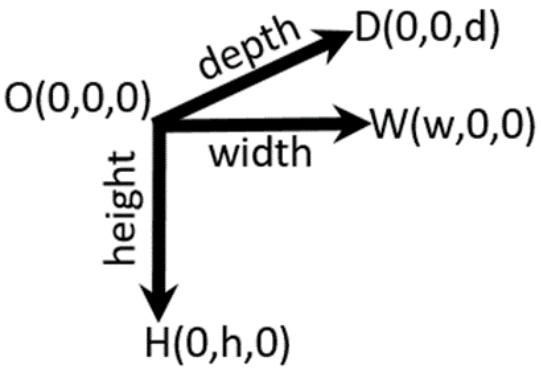
`pSrc`	Array of pointers to the planes in the source volume.
`srcStep`	Distance, in bytes, between the starting points of consecutive lines in each plane of the source volume.
`srcPlaneStep`	Distance, in bytes, between the starting points of consecutive images in every plane of the source volume.
`srcRoiVolume`	Volume of the source ROI in pixels.
`pDst`	Array of pointers to the planes in the destination volume.
`dstStep`	Distance, in bytes, between the starting points of consecutive lines in each plane of the destination volume.
`dstPlaneStep`	Distance, in bytes, between the starting points of consecutive images in every plane of the source volume.
`dstRoiVolume`	Volume of the destination ROI in pixels.
`topBorderHeight`	Height of the top border in pixels.
`leftBorderWidth`	Width of the left border in pixels.

forwardBorderDepth	Depth of the forward border in pixels.
value	Constant value to assign to the border pixels.

Description

This function operates with VOI. This function copies the source image pSrc with the volume srcRoiVolume to the destination image pDst with the volume dstRoiVolume and creates a border outside the copied area. The function sets pixel values of the border to the specified constant value that is passed by the value argument.

The image below shows the mapping of the parameters topBorderHeight, leftBorderWidth, and forwardBorderDeprh onto the dimensions of the three-dimensional space.



Return Values

ippStsNullPtrErr	Indicates an error condition if pSrc, pDst, or value pointer is NULL.
ippStsStepErr	Indicates an error condition if srcPlaneStep, srcStep, dstPlaneStep, or dstStep has a field with negative value.
ippStsSizeErr	Indicates an error condition if leftBorderWidth, topBorderHeight, or forwardBorderDepth has a field with negative value.

See Also

CopyReplicateBorder Copies pixel values between two 3D images and adds replicated border pixels.

Structures and Enumerators for Platform-Aware Functions

CopyReplicateBorder

Copies pixel values between two 3D images and adds replicated border pixels.

Syntax

```
IppStatus ipprCopyReplicateBorder_8u_C1V(const Ipp8u* pSrc, int srcPlaneStep, int
srcStep, IpprVolume srcRoiVolume, Ipp8u* pDst, int dstPlaneStep, int dstStep,
IpprVolume dstRoiVolume, int topBorderHeight, int leftBorderWidth, int
forwardBorderDepth);

IppStatus ipprCopyReplicateBorder_16u_C1V(const Ipp16u* pSrc, int srcPlaneStep, int
srcStep, IpprVolume srcRoiVolume, Ipp16u* pDst, int dstPlaneStep, int dstStep,
IpprVolume dstRoiVolume, int topBorderHeight, int leftBorderWidth, int
forwardBorderDepth);

IppStatus ipprCopyReplicateBorder_16s_C1V(const Ipp16s* pSrc, int srcPlaneStep, int
srcStep, IpprVolume srcRoiVolume, Ipp16s* pDst, int dstPlaneStep, int dstStep,
IpprVolume dstRoiVolume, int topBorderHeight, int leftBorderWidth, int
forwardBorderDepth);

IppStatus ipprCopyReplicateBorder_32f_C1V(const Ipp32f* pSrc, int srcPlaneStep, int
srcStep, IpprVolume srcRoiVolume, Ipp32f* pDst, int dstPlaneStep, int dstStep,
IpprVolume dstRoiVolume, int topBorderHeight, int leftBorderWidth, int
forwardBorderDepth);

IppStatus ipprCopyReplicateBorder_64f_C1V(const Ipp64f* pSrc, int srcPlaneStep, int
srcStep, IpprVolume srcRoiVolume, Ipp64f* pDst, int dstPlaneStep, int dstStep,
IpprVolume dstRoiVolume, int topBorderHeight, int leftBorderWidth, int
forwardBorderDepth);
```

Platform-aware functions

```
IppStatus ipprCopyReplicateBorder_8u_C1V_L(const Ipp8u* pSrc, IppSizeL srcPlaneStep,
IppSizeL srcStep, IpprVolumeL srcRoiVolume, Ipp8u* pDst, IppSizeL dstPlaneStep,
IppSizeL dstStep, IpprVolumeL dstRoiVolume, IppSizeL topBorderHeight, IppSizeL
leftBorderWidth, IppSizeL forwardBorderDepth);

IppStatus ipprCopyReplicateBorder_16u_C1V_L(const Ipp16u* pSrc, IppSizeL srcPlaneStep,
IppSizeL srcStep, IpprVolumeL srcRoiVolume, Ipp16u* pDst, IppSizeL dstPlaneStep,
IppSizeL dstStep, IpprVolumeL dstRoiVolume, IppSizeL topBorderHeight, IppSizeL
leftBorderWidth, IppSizeL forwardBorderDepth);

IppStatus ipprCopyReplicateBorder_16s_C1V_L(const Ipp16s* pSrc, IppSizeL srcPlaneStep,
IppSizeL srcStep, IpprVolumeL srcRoiVolume, Ipp16s* pDst, IppSizeL dstPlaneStep,
IppSizeL dstStep, IpprVolumeL dstRoiVolume, IppSizeL topBorderHeight, IppSizeL
leftBorderWidth, IppSizeL forwardBorderDepth);

IppStatus ipprCopyReplicateBorder_32f_C1V_L(const Ipp32f* pSrc, IppSizeL srcPlaneStep,
IppSizeL srcStep, IpprVolumeL srcRoiVolume, Ipp32f* pDst, IppSizeL dstPlaneStep,
IppSizeL dstStep, IpprVolumeL DstRoiVolume, IppSizeL topBorderHeight, IppSizeL
leftBorderWidth, IppSizeL forwardBorderDepth);

IppStatus ipprCopyReplicateBorder_64f_C1V_L(const Ipp64f* pSrc, IppSizeL srcPlaneStep,
IppSizeL srcStep, IpprVolumeL srcRoiVolume, Ipp64f* pDst, IppSizeL dstPlaneStep,
IppSizeL dstStep, IpprVolumeL dstRoiVolume, IppSizeL topBorderHeight, IppSizeL
leftBorderWidth, IppSizeL forwardBorderDepth);
```

Include Files

ippi.h

ippi_l.h

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

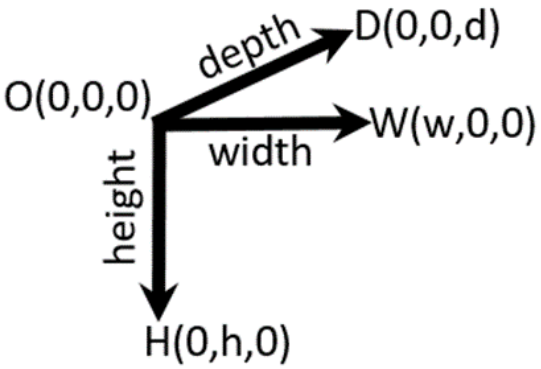
Parameters

`pSrc`	Array of pointers to the planes in the source volume.
`srcStep`	Distance, in bytes, between the starting points of consecutive lines in each plane of the source volume.
`srcPlaneStep`	Distance, in bytes, between the starting points of consecutive images in every plane of the source volume.
`srcRoiVolume`	Volume of the source ROI in pixels.
`pDst`	Array of pointers to the planes in the destination volume.
`dstStep`	Distance, in bytes, between the starting points of consecutive lines in each plane of the destination volume.
`dstPlaneStep`	Distance, in bytes, between the starting points of consecutive images in every plane of the source volume.
`dstRoiVolume`	Volume of the destination ROI in pixels.
`topBorderHeight`	Height of the top border in pixels.
`leftBorderWidth`	Width of the left border in pixels.
`forwardBorderDepth`	Depth of the forward border in pixels.

Description

This function operates with VOI. This function copies the source image `pSrc` with the volume `srcRoiVolume` to the destination image `pDst` with the volume `dstRoiVolume`. The function fills pixels (‘border’) outside the copied area in the destination image with the values of the source image pixels.

The image below shows the mapping of the parameters `topBorderHeight`, `leftBorderWidth`, and `forwardBorderWidth` onto the dimensions of the three-dimensional space.



Return Values

`ippStsNullPtrErr`	Indicates an error condition if `pSrc` or `pDst` pointer is `NULL`.
`ippStsStepErr`	Indicates an error condition if `srcPlaneStep` value is less than `srcStep` value or if `dstPlaneStep` value is less than `dstStep` value.
`ippStsSizeErr`	Indicates an error condition if `leftBorderWidth`, `topBorderHeight` or `forwardBorderDepth` has a field with negative value.

See Also

CopyConstBorder Copies pixel values between two 3D images and adds border pixels with a constant value.

Structures and Enumerators for Platform-Aware Functions

Filter

Filters a volume using a general cuboidal kernel.

Syntax

```
IppStatus ipprFilter_16s_C1PV(const Ipp16s* const pSrc[], int srcStep, const Ipp16s*
pDst[], int dstStep, IpprVolume dstVolume, const Ipp32s* pKernel, IpprVolume
kernelVolume, IpprPoint anchor, int divisor, Ipp8u* pBuffer);
```

```
IppStatus ipprFilter_16s_C1V(const Ipp16s* pSrc, int srcStep, int srcPlaneStep, Ipp16s*
pDst, int dstStep, int dstPlaneStep, IpprVolume dstVolume, const Ipp32s* pKernel,
IpprVolume kernelVolume, IpprPoint anchor, int divisor, Ipp8u* pBuffer);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

pSrc	Array of pointers to the planes in the source volume.
srcStep	Distance, in bytes, between the starting points of consecutive lines in each plane of the source volume.
srcPlaneStep	Distance, in bytes, between the starting points of consecutive lines in every plane of the source volume (for the 16s_C1V flavor).
pDst	Array of pointers to the planes in the destination volume.
dstStep	Distance, in bytes, between the starting points of consecutive lines in each plane of the destination volume.
dstPlaneStep	Distance, in bytes, between the starting points of consecutive lines in every plane of the destination volume (for the 16s_C1V flavor).
dstVolume	Size of the processed volume.
pKernel	Pointers to the kernel values.
kernelVolume	Size of the kernel volume.
anchor	Anchor 3d-cell specifying the cuboidal kernel alignment with respect to the position of the input voxel.
divisor	The integer value by which the computed result is divided.
pBuffer	Pointer to the external buffer.

Description

This function operates with VOI. This function uses the general cuboidal kernel of size kernelVolume to filter a volume VOI. This function sums the products between the kernel coefficients pKernel and voxel values taken over the source voxel neighborhood defined by kernelVolume and anchor. The anchor 3d-cell is specified by its coordinates anchor.x, anchor.y and anchor.z in the coordinate system associated with the right bottom back corner of the kernel. Note the kernel coefficients are used in inverse order. The sum is written to the destination voxel. To ensure valid operation when volume boundary voxels are processed, the application must correctly define additional border voxels.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if pSrc, pDst, pKernel or pBuffer pointer is NULL.

ippStsSizeErr	Indicates an error condition if dstVolume or kernelVolume has a field with zero or negative value.
ippStsDivisorErr	Indicates an error condition if the divisor value is zero.

FilterGetBufSize

Calculates the size of the working buffer.

Syntax

```
IppStatus ipprFilterGetBufSize(IpprVolume dstVolume, IpprVolume kernelVolume, int nChannel, int* pSize);
```

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

dstVolume	Size of the processed volume.
kernelVolume	Size of the kernel volume.
nChannel	Number of channels or planes, possible value is one.
pSize	Pointer to the size of the external buffer.

Description

This function operates with VOI. This function computes the size of the working buffer pSize that is required for the function ipprFilter.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error condition if pSize pointer is NULL.
ippStsNumChannelErr	Indicates an error condition if nChannel has an illegal value.

ippStsSizeErr	Indicates an error condition if dstVolume or kernelVolume has a field with zero or negative value.

FilterBorder

Filters a 3D image using a rectangular filter.

Syntax

```
IppStatus ipprFilterBorder_8u_C1V(const Ipp8u* pSrc, int srcPlaneStep, int srcStep,
Ipp8u* pDst, int dstPlaneStep, int dstStep, IpprVolume dstRoiVolume, IpprBorderType
borderType, const Ipp8u borderValue[1], const IpprFilterBorderSpec* pSpec, Ipp8u*
pBuffer);

IppStatus ipprFilterBorder_16s_C1V(const Ipp16s* pSrc, int srcPlaneStep, int srcStep,
Ipp16s* pDst, int dstPlaneStep, int dstStep, IpprVolume dstRoiVolume, IpprBorderType
borderType, const Ipp16s borderValue[1], const IpprFilterBorderSpec* pSpec, Ipp8u*
pBuffer);

IppStatus ipprFilterBorder_16u_C1V(const Ipp16u* pSrc, int srcPlaneStep, int srcStep,
Ipp16u* pDst, int dstPlaneStep, int dstStep, IpprVolume dstRoiVolume, IpprBorderType
borderType, const Ipp16u borderValue[1], const IpprFilterBorderSpec* pSpec, Ipp8u*
pBuffer);

IppStatus ipprFilterBorder_32f_C1V(const Ipp32f* pSrc, int srcPlaneStep, int srcStep,
Ipp32f* pDst, int dstPlaneStep, int dstStep, IpprVolume dstRoiVolume, IpprBorderType
borderType, const Ipp32f borderValue[1], const IpprFilterBorderSpec* pSpec, Ipp8u*
pBuffer);

IppStatus ipprFilterBorder_64f_C1V(const Ipp64f* pSrc, int srcPlaneStep, int srcStep,
Ipp64f* pDst, int dstPlaneStep, int dstStep, IpprVolume dstRoiVolume, IpprBorderType
borderType, const Ipp64f borderValue[1], const IpprFilterBorderSpec* pSpec, Ipp8u*
pBuffer);
```

Platform-aware functions

```
IppStatus ipprFilterBorder_8u_C1V_L(const Ipp8u* pSrc, IppSizeL srcPlaneStep, IppSizeL
srcStep, Ipp8u* pDst, IppSizeL dstPlaneStep, IppSizeL dstStep, IpprVolumeL
dstRoiVolume, IpprBorderType borderType, const Ipp8u borderValue[1], const
IpprFilterBorderSpec* pSpec, Ipp8u* pBuffer);

IppStatus ipprFilterBorder_16s_C1V_L(const Ipp16s* pSrc, IppSizeL srcPlaneStep,
IppSizeL srcStep, Ipp16s* pDst, IppSizeL dstPlaneStep, IppSizeL dstStep, IpprVolumeL
dstRoiVolume, IpprBorderType borderType, const Ipp16s borderValue[1], const
IpprFilterBorderSpec* pSpec, Ipp8u* pBuffer);

IppStatus ipprFilterBorder_16u_C1V_L(const Ipp16u* pSrc, IppSizeL srcPlaneStep,
IppSizeL srcStep, Ipp16u* pDst, IppSizeL dstPlaneStep, IppSizeL dstStep, IpprVolumeL
dstRoiVolume, IpprBorderType borderType, const Ipp16u borderValue[1], const
IpprFilterBorderSpec* pSpec, Ipp8u* pBuffer);

IppStatus ipprFilterBorder_32f_C1V_L(const Ipp32f* pSrc, IppSizeL srcPlaneStep,
IppSizeL srcStep, Ipp32f* pDst, IppSizeL dstPlaneStep, IppSizeL dstStep, IpprVolumeL
dstRoiVolume, IpprBorderType borderType, const Ipp32f borderValue[1], const
IpprFilterBorderSpec* pSpec, Ipp8u* pBuffer);
```

```
IppStatus ipprFilterBorder_64f_C1V_L(const Ipp64f* pSrc, IppSizeL srcPlaneStep,
IppSizeL srcStep, Ipp64f* pDst, IppSizeL dstPlaneStep, IppSizeL dstStep, IpprVolumeL
dstRoiVolume, IpprBorderType borderType, const Ipp64f borderValue[1], const
IpprFilterBorderSpec* pSpec, Ipp8u* pBuffer);
```

Threading Layer (TL) functions based on the Platform Aware API

```
IppStatus ipprFilterBorder_8u_C1V_LT(const Ipp8u* pSrc, IppSizeL srcPlaneStep, IppSizeL
srcStep, Ipp8u* pDst, IppSizeL dstPlaneStep, IppSizeL dstStep, IpprVolumeL
dstRoiVolume, IpprBorderType borderType, const Ipp8u borderValue[1], const
IpprFilterBorderSpec_LT* pSpec, Ipp8u* pBuffer);

IppStatus ipprFilterBorder_16s_C1V_LT(const Ipp16s* pSrc, IppSizeL srcPlaneStep,
IppSizeL srcStep, Ipp16s* pDst, IppSizeL dstPlaneStep, IppSizeL dstStep, IpprVolumeL
dstRoiVolume, IpprBorderType borderType, const Ipp16s borderValue[1], const
IpprFilterBorderSpec_LT* pSpec, Ipp8u* pBuffer);

IppStatus ipprFilterBorder_16u_C1V_LT(const Ipp16u* pSrc, IppSizeL srcPlaneStep,
IppSizeL srcStep, Ipp16u* pDst, IppSizeL dstPlaneStep, IppSizeL dstStep, IpprVolumeL
dstRoiVolume, IpprBorderType borderType, const Ipp16u borderValue[1], const
IpprFilterBorderSpec_LT* pSpec, Ipp8u* pBuffer);

IppStatus ipprFilterBorder_32f_C1V_LT(const Ipp32f* pSrc, IppSizeL srcPlaneStep,
IppSizeL srcStep, Ipp32f* pDst, IppSizeL dstPlaneStep, IppSizeL dstStep, IpprVolumeL
dstRoiVolume, IpprBorderType borderType, const Ipp32f borderValue[1], const
IpprFilterBorderSpec_LT* pSpec, Ipp8u* pBuffer);

IppStatus ipprFilterBorder_64f_C1V_LT(const Ipp64f* pSrc, IppSizeL srcPlaneStep,
IppSizeL srcStep, Ipp64f* pDst, IppSizeL dstPlaneStep, IppSizeL dstStep, IpprVolumeL
dstRoiVolume, IpprBorderType borderType, const Ipp64f borderValue[1], const
IpprFilterBorderSpec_LT* pSpec, Ipp8u* pBuffer);
```

Threading Layer (TL) functions based on the Classic API

```
IppStatus ipprFilterBorder_8u_C1V_T(const Ipp8u* pSrc, int srcPlaneStep, int srcStep,
Ipp8u* pDst, int dstPlaneStep, int dstStep, IpprVolume dstRoiVolume, IpprBorderType
borderType, const Ipp8u borderValue[1], const IpprFilterBorderSpec_T* pSpec, Ipp8u*
pBuffer);

IppStatus ipprFilterBorder_16s_C1V_T(const Ipp16s* pSrc, int srcPlaneStep, int srcStep,
Ipp16s* pDst, int dstPlaneStep, int dstStep, IpprVolume dstRoiVolume, IpprBorderType
borderType, const Ipp16s borderValue[1], const IpprFilterBorderSpec_T* pSpec, Ipp8u*
pBuffer);

IppStatus ipprFilterBorder_16u_C1V_T(const Ipp16u* pSrc, int srcPlaneStep, int srcStep,
Ipp16u* pDst, int dstPlaneStep, int dstStep, IpprVolume dstRoiVolume, IpprBorderType
borderType, const Ipp16u borderValue[1], const IpprFilterBorderSpec_T* pSpec, Ipp8u*
pBuffer);

IppStatus ipprFilterBorder_32f_C1V_T(const Ipp32f* pSrc, int srcPlaneStep, int srcStep,
Ipp32f* pDst, int dstPlaneStep, int dstStep, IpprVolume dstRoiVolume, IpprBorderType
borderType, const Ipp32f borderValue[1], const IpprFilterBorderSpec_T* pSpec, Ipp8u*
pBuffer);

IppStatus ipprFilterBorder_64f_C1V_T(const Ipp64f* pSrc, int srcPlaneStep, int srcStep,
Ipp64f* pDst, int dstPlaneStep, int dstStep, IpprVolume dstRoiVolume, IpprBorderType
borderType, const Ipp64f borderValue[1], const IpprFilterBorderSpec_T* pSpec, Ipp8u*
pBuffer);
```

Include Files

ippi.h

ippi_l.h

ippi_tl.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

pSrc	Array of pointers to the planes in the source volume.
srcStep	Distance, in bytes, between the starting points of consecutive lines in each plane of the source volume.
srcPlaneStep	Distance, in bytes, between the starting points of consecutive images in every plane of the source volume.
pDst	Array of pointers to the planes in the destination volume.
dstStep	Distance, in bytes, between the starting points of consecutive lines in each plane of the destination volume.
dstPlaneStep	Distance, in bytes, between the starting points of consecutive images in every plane of the source volume.
dstRoiVolume	Volume of the destination ROI in pixels.
borderType	Type of the border. Possible values are: • ipprBorderInMem • Border is obtained from the source image pixels in memory. • ipprBorderRepl • Border is replicated from the edge pixels. • ipprBorderConst • Border is replicated from the edge pixels.
borderValue	Constant value to assign to pixels of the constant border.
pSpec	Pointer to the filter specification structure.
pBuffer	Pointer to the work buffer for filtering operations.

Description

Before using this function, you need to initialize the filter specification structure for 3D image processing using the ipprFilterBorderInit function.

This function operates with VOI. This function performs linear filtering on a source image with the volume. Type of the image border is defined by the value of the border parameter.

Return Values

ippStsNoErr	Indicates no error condition. Any other value indicates an error condition.
ippStsStepErr	Indicates an error condition if srcPlaneStep, srcStep, dstPlaneStep, or dstStep has a field with negative value.
ippStsNullPtrErr	Indicates an error condition if the pSrc, pDst, pSpec, or pBuffer pointer is NULL.
ippStsSizeErr	Indicates an error condition if dstRoiVolume has a field with zero or negative value.

Example

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

//   A simple example of performing filtering an image using a general integer rectangular kernel
// implemented with Intel(R) Integrated Performance Primitives (Intel(R) IPP) functions:
//     ipprFilterBorderGetSize
//     ipprFilterBorderInit_16s
//     ipprFilterBorder_8u_C1R

#include <stdio.h>
#include "ipp.h"

#define WIDTH  128 /* image width */
#define HEIGHT 64  /* image height */
#define DEPTH  64  /* image depth */
const Ipp16s kernel[3 * 3 * 3] = {0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0,
1, 0, 1, 0, 1, 0};

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
```

```
#define check_sts(st)        \
    if ((st) != ippStsNoErr) \
        goto exitLine; /* Go to Exit if Intel(R) IPP function returned status different from
ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main(void)
{
    IppStatus status = ippStsNoErr;
    Ipp8u *pSrc = NULL, *pDst = NULL;                                    /* Pointers to source/
destination images */
    IppSizeL srcPlaneStep = 0, srcStep = 0, dstPlaneStep, dstStep = 0; /* Steps, in bytes,
through the source/destination images */
    IpprVolumeL roiVolume = {WIDTH, HEIGHT, DEPTH};                    /* Size of source/
destination ROI in pixels */
    IpprVolumeL kernelVolume = {3, 3, 3};
    Ipp8u *pBuffer = NULL;                    /* Pointer to the work buffer */
    IpprFilterBorderSpec *pSpec = NULL;       /* context structure */
    IppSizeL iTmpBufSize = 0, iSpecSize = 0; /* Common work buffer size */
    IpprBorderType borderType = ipprBorderRepl;
    Ipp8u borderValue = 0;
    int numChannels = 1;

    pSrc = ippsMalloc_8u_L(roiVolume.width * roiVolume.height * roiVolume.depth);
    pDst = ippsMalloc_8u_L(roiVolume.width * roiVolume.height * roiVolume.depth);
    srcStep = roiVolume.width;
    srcPlaneStep = roiVolume.height * srcStep;
    dstStep = roiVolume.width;
    dstPlaneStep = roiVolume.height * srcStep;

    check_sts(status = ipprFilterBorderGetSize_L(kernelVolume, roiVolume, ipp8u, ipp16s,
numChannels, &iSpecSize, &iTmpBufSize));

    pSpec = (IpprFilterBorderSpec *)ippsMalloc_8u(iSpecSize);
    pBuffer = ippsMalloc_8u_L(iTmpBufSize);

    check_sts(status = ipprFilterBorderInit_16s_L(kernel, kernelVolume, 4, ipp8u, numChannels,
pSpec));

    check_sts(status = ipprFilterBorder_8u_C1V_L(pSrc, srcPlaneStep, srcStep, pDst,
dstPlaneStep, dstStep, roiVolume, borderType, &borderValue, pSpec,
                                                 pBuffer));

    EXIT_MAIN
    ippsFree(pBuffer);
    ippsFree(pSpec);
    ippiFree(pSrc);
    ippiFree(pDst);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

See Also

FilterBorderInit Initializes the filter specification structure for 3D image processing.

FilterBorderGetSize Computes the size of the filter specification structure and the size of the work buffer for 3D image processing.

Structures and Enumerators

Structures and Enumerators for Platform-Aware Functions

FilterBorderInit

Initializes the filter specification structure for 3D image processing.

Syntax

```
IppStatus ipprFilterBorderInit_16s(const Ipp16s* pKernel, IpprVolume kernelVolume, int divisor, IppDataType dataType, int numChannels, IpprFilterBorderSpec* pSpec);
```

```
IppStatus ipprFilterBorderInit_32f(const Ipp32f* pKernel, IpprVolume kernelVolume, IppDataType dataType, int numChannels, IpprFilterBorderSpec* pSpec);
```

```
IppStatus ipprFilterBorderInit_64f(const Ipp64f* pKernel, IpprVolume kernelVolume, IppDataType dataType, int numChannels, IpprFilterBorderSpec* pSpec);
```

Platform-aware functions

```
IppStatus ipprFilterBorderInit_16s_L(const Ipp16s* pKernel, IpprVolumeL kernelVolume, int divisor, IppDataType dataType, int numChannels, IpprFilterBorderSpec* pSpec);
```

```
IppStatus ipprFilterBorderInit_32f_L(const Ipp32f* pKernel, IpprVolumeL kernelVolume, IppDataType dataType, int numChannels, IpprFilterBorderSpec* pSpec);
```

```
IppStatus ipprFilterBorderInit_64f_L(const Ipp64f* pKernel, IpprVolumeL kernelVolume, IppDataType dataType, int numChannels, IpprFilterBorderSpec* pSpec);
```

Threading Layer (TL) functions based on the Platform Aware API

```
IppStatus ipprFilterBorderInit_16s_LT(const Ipp16s* pKernel, IpprVolumeL kernelVolume, int divisor, IppDataType dataType, int numChannels, IpprFilterBorderSpec_LT* pSpec);
```

```
IppStatus ipprFilterBorderInit_32f_LT(const Ipp32f* pKernel, IpprVolumeL kernelVolume, IppDataType dataType, int numChannels, IpprFilterBorderSpec_LT* pSpec);
```

```
IppStatus ipprFilterBorderInit_64f_LT(const Ipp64f* pKernel, IpprVolumeL kernelVolume, IppDataType dataType, int numChannels, IpprFilterBorderSpec_LT* pSpec);
```

Threading Layer (TL) functions based on the Classic API

```
IppStatus ipprFilterBorderInit_16s_T(const Ipp16s* pKernel, IpprVolume kernelVolume, int divisor, IppDataType dataType, int numChannels, IpprFilterBorderSpec_T* pSpec);
```

```
IppStatus ipprFilterBorderInit_32f_T(const Ipp32f* pKernel, IpprVolume kernelVolume, IppDataType dataType, int numChannels, IpprFilterBorderSpec_T* pSpec);
```

```
IppStatus ipprFilterBorderInit_64f_T(const Ipp64f* pKernel, IpprVolume kernelVolume, IppDataType dataType, int numChannels, IpprFilterBorderSpec_T* pSpec);
```

Include Files

```
ippi.h
```

```
ippi_l.h
```

```
ippi_tl.h
```

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`kernelVolume`	Size of the kernel volume.
`pKernel`	Pointers to the kernel values.
`dataType`	Data type of the source image. Possible values are `ipp8u`, `ipp16u`, `ipp16s`, `ipp32f`, and `ipp64f`.
`divisor`	The integer value by which the computed result is divided.
`numChannels`	Number of channels in the image. Possible value is 1.
`pSpec`	Pointer to the filter specification structure.

Description

This function operates with VOI. This function initializes the filter specification structure `pSpec`. Before using this function, you need to compute the size of the specification structure for 3D image processing using the ipprFilterBorderGetSize function.

Return Values

`ippStsNoErr`	Indicates no error condition. Any other value indicates an error condition.
`ippStsNullPtrErr`	Indicates an error condition if the `pSpec` or `pKernel` pointer is `NULL`.
`ippStsSizeErr`	Indicates an error condition if `kernelVolume` has a field with zero or negative value.
`ippStsChannelErr`	Indicates an error condition if `numChannels` has an illegal value.
`ippStsDataTypeErr`	Indicates an error condition if `dataType` has an illegal value.
`ippStsDivisorErr`	Indicates an error condition if the `divisor` value is zero.

See Also

FilterBorder Filters a 3D image using a rectangular filter.

FilterBorderGetSize Computes the size of the filter specification structure and the size of the work buffer for 3D image processing.

Structures and Enumerators for Platform-Aware Functions

FilterBorderGetSize

Computes the size of the filter specification structure and the size of the work buffer for 3D image processing.

Syntax

```
IppStatus ipprFilterBorderGetSize(IpprVolume kernelVolume, IpprVolume dstRoiVolume,
IppDataType dataType, IppDataType kernelType, int numChannels, int* pSpecSize, int*
pBufferSize);
```

Platform-aware function

```
IppStatus ipprFilterBorderGetSize_L(IpprVolumeL kernelVolume, IpprVolumeL dstRoiVolume,
IppDataType dataType, IppDataType kernelType, int numChannels, IppSizeL* pSpecSize,
IppSizeL* pBufferSize);
```

Threading Layer (TL) function based on the Platform Aware API

```
IppStatus ipprFilterBorderGetSize_LT(IpprVolumeL kernelVolume, IpprVolumeL
dstRoiVolume, IppDataType dataType, IppDataType kernelType, int numChannels, IppSizeL*
pSpecSize, IppSizeL* pBufferSize);
```

Threading Layer (TL) function based on the Classic API

```
IppStatus ipprFilterBorderGetSize_T(IpprVolume kernelVolume, IpprVolume dstRoiVolume,
IppDataType dataType, IppDataType kernelType, int numChannels, int* pSpecSize, int*
pBufferSize);
```

Include Files

`ippi.h`

`ippi_l.h`

`ippi_tl.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`kernelVolume`	Size of the kernel volume.
`dstRoiVolume`	Maximal size of the destination image ROI (in pixels).
`dataType`	Data type of the source image. Possible values are `ipp8u`, `ipp16u`, `ipp16s`, `ipp32f`, and `ipp64f`.
`kernelType`	Data type of the filter kernel. Possible values are `ipp16s`, `ipp32f`, and `ipp64f`.
`numChannels`	Number of channels in the image. Possible value is 1.
`pSpecSize`	Pointer to the size of the filter specification structure.

pBufferSize	Pointer to the size of the work buffer required for filtering.

Description

This function operates with VOI. This function computes the size of the filter specification structure pSpec and the size of the buffer required for 3D image filtering operations. Call this function before using the ipprFilterBorderInit function.

Return Values

ippStsNoErr	Indicates no error condition. Any other value indicates an error condition.
ippStsNullPtrErr	Indicates an error condition if the pSpecSize or pBufferSize pointer is NULL.
ippStsSizeErr	Indicates an error condition if dstRoiVolume or kernelVolume has a field with zero or negative value.
ippStsChannelErr	Indicates an error condition if numChannels has an illegal value.
ippStsDataTypeErr	Indicates an error condition if the combination of kernelType and dataType has an illegal value.

See Also

FilterBorder Filters a 3D image using a rectangular filter.

FilterBorderInit Initializes the filter specification structure for 3D image processing.

Structures and Enumerators for Platform-Aware Functions

FilterMedian

Filters a 3D image using a median filter.

Syntax

```
IppStatus ipprFilterMedian_8u_C1V(const Ipp8u* pSrc, int srcPlaneStep, int srcStep,
Ipp8u* pDst, int dstPlaneStep, int dstStep, IpprVolume dstRoiVolume, IpprBorderType
borderType, const Ipp8u* pBorderValue, const IpprFilterMedianSpec* pSpec, Ipp8u*
pBuffer);
```

```
IppStatus ipprFilterMedian_16u_C1V(const Ipp16u* pSrc, int srcPlaneStep, int srcStep,
Ipp16u* pDst, int dstPlaneStep, int dstStep, IpprVolume dstRoiVolume, IpprBorderType
borderType, const Ipp16u* pBorderValue, const IpprFilterMedianSpec* pSpec, Ipp8u*
pBuffer);
```

```
IppStatus ipprFilterMedian_16s_C1V(const Ipp16s* pSrc, int srcPlaneStep, int srcStep,
Ipp16s* pDst, int dstPlaneStep, int dstStep, IpprVolume dstRoiVolume, IpprBorderType
borderType, const Ipp16s* pBorderValue, const IpprFilterMedianSpec* pSpec, Ipp8u*
pBuffer);
```

```
IppStatus ipprFilterMedian_32f_C1V(const Ipp32f* pSrc, int srcPlaneStep, int srcStep,
Ipp32f* pDst, int dstPlaneStep, int dstStep, IpprVolume dstRoiVolume, IpprBorderType
borderType, const Ipp32f* pBorderValue, const IpprFilterMedianSpec* pSpec, Ipp8u*
pBuffer);

IppStatus ipprFilterMedian_64f_C1V(const Ipp64f* pSrc, int srcPlaneStep, int srcStep,
Ipp64f* pDst, int dstPlaneStep, int dstStep, IpprVolume dstRoiVolume, IpprBorderType
borderType, const Ipp64f* pBorderValue, const IpprFilterMedianSpec* pSpec, Ipp8u*
pBuffer);
```

Threading Layer (TL) functions

```
IppStatus ipprFilterMedian_8u_C1V_T(const Ipp8u* pSrc, int srcPlaneStep, int srcStep,
Ipp8u* pDst, int dstPlaneStep, int dstStep, IpprVolume dstRoiVolume, IpprBorderType
borderType, const Ipp8u* pBorderValue, const IpprFilterMedianSpec_T* pSpec, Ipp8u*
pBuffer);

IppStatus ipprFilterMedian_16u_C1V_T(const Ipp16u* pSrc, int srcPlaneStep, int srcStep,
Ipp16u* pDst, int dstPlaneStep, int dstStep, IpprVolume dstRoiVolume, IpprBorderType
borderType, const Ipp16u* pBorderValue, const IpprFilterMedianSpec_T* pSpec, Ipp8u*
pBuffer);

IppStatus ipprFilterMedian_16s_C1V_T(const Ipp16s* pSrc, int srcPlaneStep, int srcStep,
Ipp16s* pDst, int dstPlaneStep, int dstStep, IpprVolume dstRoiVolume, IpprBorderType
borderType, const Ipp16s* pBorderValue, const IpprFilterMedianSpec_T* pSpec, Ipp8u*
pBuffer);

IppStatus ipprFilterMedian_32f_C1V_T(const Ipp32f* pSrc, int srcPlaneStep, int srcStep,
Ipp32f* pDst, int dstPlaneStep, int dstStep, IpprVolume dstRoiVolume, IpprBorderType
borderType, const Ipp32f* pBorderValue, const IpprFilterMedianSpec_T* pSpec, Ipp8u*
pBuffer);

IppStatus ipprFilterMedian_64f_C1V_T(const Ipp64f* pSrc, int srcPlaneStep, int srcStep,
Ipp64f* pDst, int dstPlaneStep, int dstStep, IpprVolume dstRoiVolume, IpprBorderType
borderType, const Ipp64f* pBorderValue, const IpprFilterMedianSpec_T* pSpec, Ipp8u*
pBuffer);
```

Include Files

`ippi.h`

`ippi_tl.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Array of pointers to the planes in the source volume.
`srcStep`	Distance, in bytes, between the starting points of consecutive lines in each plane of the source volume.

srcPlaneStep	Distance, in bytes, between the starting points of consecutive images in every plane of the source volume.
pDst	Array of pointers to the planes in the destination volume.
dstStep	Distance, in bytes, between the starting points of consecutive lines in each plane of the destination volume.
dstPlaneStep	Distance, in bytes, between the starting points of consecutive images in every plane of the source volume.
dstRoiVolume	Volume of the destination ROI in pixels.
borderType	Type of the border. Possible values are: • ipprBorderInMem • Border is obtained from the source image pixels in memory. • ipprBorderRepl • Border is replicated from the edge pixels. • ipprBorderConst • Border is replicated from the edge pixels.
pBorderValue	Pointer to the constant value to assign to pixels of the constant border.
pSpec	Pointer to the filter specification structure.
pBuffer	Pointer to the work buffer for filtering operations.

Description

Before using this function, you need to initialize the filter specification structure for 3D image processing using the ipprFilterMedianInit function.

This function operates with VOI. This function performs median filtering on a source image with the volume. Type of the image border is defined by the value of the border parameter.

Return Values

ippStsNoErr	Indicates no error condition. Any other value indicates an error condition.
ippStsStepErr	Indicates an error condition if srcPlaneStep, srcStep, dstPlaneStep, or dstStep has a field with negative value.
ippStsNullPtrErr	Indicates an error condition if the pSrc, pDst, pSpec, pBorderValue, or pBuffer pointer is NULL.
ippStsSizeErr	Indicates an error condition if dstRoiVolume has a field with zero or negative value.

See Also

FilterMedianInit Initializes the filter specification structure for 3D image processing with a median filter.

FilterMedianGetSize Computes the size of the filter specification structure and the size of the work buffer for 3D image processing with a median filter.

Structures and Enumerators

FilterMedianInit

Initializes the filter specification structure for 3D image processing with a median filter.

Syntax

```
IppStatus ipprFilterMedianInit(IpprVolume maskVolume, IppDataType dataType, int numChannels, IpprFilterMedianSpec* pSpec);
```

Threading Layer (TL) function

```
IppStatus ipprFilterMedianInit_T(IpprVolume maskVolume, IppDataType dataType, int numChannels, IpprFilterMedianSpec_T* pSpec);
```

Include Files

`ippi.h`

`ippi_tl.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`maskVolume`	Size of the mask volume.
`dataType`	Data type of the source image. Possible values are `ipp8u`, `ipp16u`, `ipp16s`, `ipp32f`, and `ipp64f`.
`numChannels`	Number of channels in the image. Possible value is 1.
`pSpec`	Pointer to the filter specification structure.

Description

This function operates with VOI. This function initializes the filter specification structure `pSpec` for 3D image processing with a median filter. Before using this function, you need to compute the size of the corresponding specification structure using the ipprFilterMedianGetSize function.

Return Values

`ippStsNoErr`	Indicates no error condition. Any other value indicates an error condition.

ippStsNullPtrErr	Indicates an error condition if the pSpec pointer is NULL.
ippStsSizeErr	Indicates an error condition if maskVolume has a field with zero or negative value.
ippStsChannelErr	Indicates an error condition if numChannels has an illegal value.
ippStsDataTypeErr	Indicates an error condition if dataType has an illegal value.

See Also

FilterMedianGetSize Computes the size of the filter specification structure and the size of the work buffer for 3D image processing with a median filter.

FilterMedian Filters a 3D image using a median filter.

FilterMedianGetSize

Computes the size of the filter specification structure and the size of the work buffer for 3D image processing with a median filter.

Syntax

```
IppStatus ipprFilterMmedianGetSize(IpprVolume maskVolume, IpprVolume dstRoiVolume,
IppDataType dataType, int numChannels, int* pSpecSize, int* pBufferSize);
```

Threading Layer (TL) function

```
IppStatus ipprFilterMedianGetSize_T(IpprVolume maskVolume, IpprVolume dstRoiVolume,
IppDataType dataType, int numChannels, int* pSpecSize, int* pBufferSize);
```

Include Files

ippi.h

ippi_tl.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

maskVolume	Size of the mask volume.
dstRoiVolume	Maximal size of the destination image ROI (in pixels).
dataType	Data type of the source image. Possible values are ipp8u, ipp16u, ipp16s, ipp32f, and ipp64f.
numChannels	Number of channels in the image. Possible value is 1.

pSpecSize	Pointer to the size of the filter specification structure.
pBufferSize	Pointer to the size of the work buffer required for filtering.

Description

This function operates with VOI. This function computes the size of the filter specification structure pSpec and the size of the buffer required for 3D image filtering operations with a median filter. Call this function before using the ipprFilterMedianInit function.

Return Values

ippStsNoErr	Indicates no error condition. Any other value indicates an error condition.
ippStsNullPtrErr	Indicates an error condition if the pSpecSize or pBufferSize pointer is NULL.
ippStsSizeErr	Indicates an error condition if dstRoiVolume or maskVolume has a field with zero or negative value.
ippStsChannelErr	Indicates an error condition if numChannels has an illegal value.
ippStsDataTypeErr	Indicates an error condition if dataType has an illegal value.

See Also

FilterMedianInit Initializes the filter specification structure for 3D image processing with a median filter.

FIlterMedian Filters a 3D image using a median filter.

ResizeGetBufSize

Calculates the size of the external work buffer for the function ipprResize.

Syntax

```
IppStatus ipprResizeGetBufSize(IpprCuboid srcVoi, IpprCuboid dstVoi, int nChannel, int interpolation, int* pSize);
```

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

srcVoi	Volume of interest in the source volume.
dstVoi	Volume of interest in the destination volume.
nChannel	Number of channels, possible value: 1.
interpolation	Type of interpolation, the following values are possible: • IPPI_INTER_NN – nearest neighbor interpolation, • IPPI_INTER_LINEAR- trilinear interpolation, • IPPI_INTER_CUBIC – tricubic interpolation, • IPPI_INTER_CUBIC2P_BSPLINE- B-spline, • IPPI_INTER_CUBIC2P_CATMULLROM - Catmull - Rom spline, • IPPI_INTER_CUBIC2P_B05C03 - special two-parameters filter(1/2, 3/10).
pSize	Pointer to the size of the external buffer.

Description

This function calculates the size of the external buffer required for the ipprResize function.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error.
ippStsNullPtrErr	Indicates an error when the pSize pointer is NULL.
ippStsSizeErr	Indicates an error when width, or height, or depth of the srcVoi or dstVoi is less than, or equal to 0.
ippStsNumChannelErr	Indicates an error when nChannel is not equal to 1.
ippStsInterpolationErr	Indicates an error when interpolation has an illegal value.

See Also

Resize Resizes the source volume.

GetResizeCuboid

Computes coordinates of the destination cuboid.

Syntax

```
IppStatus ipprGetResizeCuboid(IpprCuboid srcVoi, IpprCuboid* pDstCuboid, double xFactor, double yFactor, double zFactor, double xShift, double yShift, double zShift, int interpolation);
```

Include Files

ippi.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

srcVoi	Volume of interest of the source volume.
pDstCuboid	Pointer to the destination cuboid.
x-, y-, zFactor	Factors by which the x, y, z dimensions of the source VOI are changed.
x-, y-, zShift	Shift values in the x, y, and z directions respectively.
interpolation	Type of interpolation, the following values are possible:

Description

This function operates with volume of interest (VOI).

This function computes the coordinates of the resultant cuboid which is obtained if the source volume srcVoi is resized with the specified parameters. The resize operation is not performed.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when pDstCuboid is NULL.
ippStsSizeErr	Indicates an error when width, or height, or depth of the source and destination volumes is less than, or equal to 0.
ippStsResizeFactorErr	Indicates an error when one of the xFactor, yFactor, zFactor values is less than, or equal to 0.
ippStsInterpolationErr	Indicates an error when interpolation has an illegal value.

Resize

Resizes the source volume.

Syntax

```
IppStatus ipprResize_8u_C1V(const Ipp8u* pSrc, IpprVolume srcVolume, int srcStep, int srcPlaneStep, IpprCuboid srcVoi, Ipp8u* pDst, int dstStep, int dstPlaneStep, IpprCuboid dstVoi, double xFactor, double yFactor, double zFactor, double xShift, double yShift, double zShift, int interpolation, Ipp8u* pBuffer);

IppStatus ipprResize_16u_C1V(const Ipp16u* pSrc, IpprVolume srcVolume, int srcStep, int srcPlaneStep, IpprCuboid srcVoi, Ipp16u* pDst, int dstStep, int dstPlaneStep, IpprCuboid dstVoi, double xFactor, double yFactor, double zFactor, double xShift, double yShift, double zShift, int interpolation, Ipp8u* pBuffer);

IppStatus ipprResize_16s_C1V(const Ipp16s* pSrc, IpprVolume srcVolume, int srcStep, int srcPlaneStep, IpprCuboid srcVoi, Ipp16s* pDst, int dstStep, int dstPlaneStep, IpprCuboid dstVoi, double xFactor, double yFactor, double zFactor, double xShift, double yShift, double zShift, int interpolation, Ipp8u* pBuffer);

IppStatus ipprResize_8u_C1PV(const Ipp8u* const pSrc[], IpprVolume srcVolume, int srcStep, IpprCuboid srcVoi, Ipp8u* const pDst[], int dstStep, IpprCuboid dstVoi, double xFactor, double yFactor, double zFactor, double xShift, double yShift, double zShift, int interpolation, Ipp8u* pBuffer);

IppStatus ipprResize_16u_C1PV(const Ipp16u* const pSrc[], IpprVolume srcVolume, int srcStep, IpprCuboid srcVoi, Ipp16u* const pDst[], int dstStep, IpprCuboid dstVoi, double xFactor, double yFactor, double zFactor, double xShift, double yShift, double zShift, int interpolation, Ipp8u* pBuffer);

IppStatus ipprResize_16s_C1PV(const Ipp16s* const pSrc[], IpprVolume srcVolume, int srcStep, IpprCuboid srcVoi, Ipp16s* const pDst[], int dstStep, IpprCuboid dstVoi, double xFactor, double yFactor, double zFactor, double xShift, double yShift, double zShift, int interpolation, Ipp8u* pBuffer);

IppStatus ipprResize_32f_C1PV(const Ipp32f* const pSrc[], IpprVolume srcVolume, int srcStep, IpprCuboid srcVoi, Ipp32f* const pDst[], int dstStep, IpprCuboid dstVoi, double xFactor, double yFactor, double zFactor, double xShift, double yShift, double zShift, int interpolation, Ipp8u* pBuffer);

IppStatus ipprResize_32f_C1V(const Ipp32f* pSrc, IpprVolume srcVolume, int srcStep, int srcPlaneStep, IpprCuboid srcVoi, Ipp32f* pDst, int dstStep, int dstPlaneStep, IpprCuboid dstVoi, double xFactor, double yFactor, double zFactor, double xShift, double yShift, double zShift, int interpolation, Ipp8u* pBuffer);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Pointer to the source volume origin. An array of pointers to the source planes for non-contiguous volume.

srcVolume	Size of the source volume.
srcStep	Distance, in bytes, between the starting points of consecutive lines in each plane of the source volume.
srcPlaneStep	Distance, in bytes, between the planes of the source contiguous volume.
srcVoi	Volume of interest of the source volume.
pDst	Pointer to the destination volume origin. An array of pointers to the destination planes for non-contiguous volume.
dstStep	Distance, in bytes, between the starting points of consecutive lines in each plane of the the destination volume.
dstPlaneStep	Distance, in bytes, between the planes of the destination contiguous volume.
dstVoi	Volume of interest of the destination volume.
x-, y-, zFactor	Factors by which the x, y, z dimensions of the source VOI are changed.
x-, y-, zShift	Shift values in the x, y, and z directions respectively.
interpolation	Type of interpolation, the following values are possible: • IPPI_INTER_NN - nearest neighbor interpolation, • IPPI_INTER_LINEAR - trilinear interpolation, • IPPI_INTER_CUBIC - tricubic interpolation, • IPPI_INTER_CUBIC2P_BSPLINE - B - spline • IPPI_INTER_CUBIC2P_CATMULLROM - Catmull - Rom spline, • IPPI_INTER_CUBIC2P_B05C03- special two-parameters filter (1/2, 3/10).
pBuffer	Pointer to the external buffer.

Description

This function operates with volume of interest (VOI).

This function resizes the source volume srcVoi by xFactor in the x direction, yFactor in the y direction and zFactor in the z direction. The volume size can be reduced or increased in each direction, depending on the values of xFactor, yFactor, zFactor. If the value of the certain factor is greater than 1, the volume size is increased, and if it is less than 1, the volume size is reduced in the corresponding direction. The result is resampled using the interpolation method specified by the interpolation parameter, and written to the destination volumeVOI.

Coordinates `x'`, `y'`, and `z'` in the resized volume are obtained from the equations:

`x' = xFactor * x + xShift`

`y' = yFactor * y + yShift`

`z' = zFactor * z + zShift`

where `x`, `y`, and `z` denote the coordinates of the element in the source volume. The right coordinate system (RCS) is used here.

Descriptor `PV` means working with non-contiguous volume data, where the location of the planes is passed to the function using pointers on the plane. In functions with descriptor `V`, the location of the planes is determined by the distance in bytes between the starting points of successive lines in each plane of the original volume (contiguous volume data).

In case of resize:

- In `IppStatus ipprResize_8u_C1V` function `V` means:

  `const Ipp8u* pSrc` - Pointer to the source volume origin. `srcPlaneStep` - Distance in bytes, between the planes of the source contiguous volume.
- In `IppStatus ipprResize_8u_C1PV` function `PV` means:

  `const Ipp8u* const pSrc[]` - An array of pointers to the source planes for non-contiguous volume.

Before using this function, compute the size of the external buffer `pBuffer` using ipprResizeGetBufSize.

Return Values

`ippStsNoErr`	Indicates no error. Any other value indicates an error or a warning.
`ippStsNullPtrErr`	Indicates an error when one of the specified pointers is `NULL`.
`ippStsSizeErr`	Indicates an error when width, or height, or depth of the source and destination volumes is less than or equal to 0.
`ippStsResizeFactorErr`	Indicates an error when one of the `xFactor`, `yFactor`, `zFactor` pointers is less than, or equal to 0.
`ippStsInterpolationErr`	Indicates an error when `interpolation` has an illegal value.
`ippStsWrongIntersectVOI`	Indicates a warning when `srcVoi` has no intersection with the source volume, operation is not performed.

See Also

ResizeGetBufSize Calculates the size of the external work buffer for the function ipprResize.

Remap

Performs the look-up coordinate mapping of the elements of the source volume.

Syntax

Case 1: Operation on non-contiguous volume data

```
IppStatus ipprRemap_8u_C1PV(const Ipp8u* pSrc[], IpprVolume srcVolume, int srcStep,
IpprCuboid srcVoi, const Ipp32f* pxMap[], const Ipp32f* pyMap[], const Ipp32f* pzMap[],
int mapStep, Ipp8u* pDst[], int dstStep, IpprVolume dstVolume, int interpolation);

IppStatus ipprRemap_16u_C1PV(const Ipp16u* pSrc[], IpprVolume srcVolume, int srcStep,
IpprCuboid srcVoi, const Ipp32f* pxMap[], const Ipp32f* pyMap[], const Ipp32f* pzMap[],
int mapStep, Ipp16u* pDst[], int dstStep, IpprVolume dstVolume, int interpolation);

IppStatus ipprRemap_32f_C1PV(const Ipp32f* pSrc[], IpprVolume srcVolume, int srcStep,
IpprCuboid srcVoi, const Ipp32f* pxMap[], const Ipp32f* pyMap[], const Ipp32f* pzMap[],
int mapStep, Ipp32f* pDst[], int dstStep, IpprVolume dstVolume, int interpolation);
```

Case 2: Operation on contiguous volume data

```
IppStatus ipprRemap_8u_C1V(const Ipp8u* pSrc, IpprVolume srcVolume, int srcStep, int
srcPlaneStep, IpprCuboid srcVoi, const Ipp32f* pxMap, const Ipp32f* pyMap, const
Ipp32f* pzMap, int mapStep, int mapPlaneStep, Ipp8u* pDst, int dstStep, int
dstPlaneStep, IpprVolume dstVolume, int interpolation);

IppStatus ipprRemap_16u_C1V(const Ipp16u* pSrc, IpprVolume srcVolume, int srcStep, int
srcPlaneStep, IpprCuboid srcVoi, const Ipp32f* pxMap, const Ipp32f* pyMap, const
Ipp32f* pzMap, int mapStep, int mapPlaneStep, Ipp16u* pDst, int dstStep, int
dstPlaneStep, IpprVolume dstVolume, int interpolation);

IppStatus ipprRemap_32f_C1V(const Ipp32f* pSrc, IpprVolume srcVolume, int srcStep, int
srcPlaneStep, IpprCuboid srcVoi, const Ipp32f* pxMap, const Ipp32f* pyMap, const
Ipp32f* pzMap, int mapStep, int mapPlaneStep, Ipp32f* pDst, int dstStep, int
dstPlaneStep, IpprVolume dstVolume, int interpolation);
```

Include Files

`ippi.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Array of pointers to the planes in the source volume.
`srcVolume`	Size of the source volume.
`srcStep`	Distance, in bytes, between the starting points of consecutive lines in every plane of the source volume.
`srcPlaneStep`	Distance, in bytes, between the starting points of consecutive lines in every plane of the source volume (for `8u_C1V`, `16u_C1V`, and `32f_C1V` flavors).
`srcVoi`	Region of interest in the source volume.

pxMap, pyMap, pzMap	Arrays of pointers to the starts of the 2D buffers, containing tables of the x-, y- and z-coordinates. If the referenced coordinates correspond to a voxel outside of the source VOI, no mapping of the source pixel is done.
mapStep	Step, in bytes, through the buffers containing tables of the x-, y- and z-coordinates.
mapPlaneStep	Distance, in bytes, between the starting points of consecutive lines in every plane of the buffers containing tables (for 8u_C1V, 16u_C1V, and 32f_C1V flavors).
pDst	Array of the pointers to the planes in the destination volume.
dstStep	Distance, in bytes, between the starting points of consecutive lines in every plane of the destination volume.
dstPlaneStep	Distance, in bytes, between the starting points of consecutive lines in every plane of the destination volume (for 8u_C1V, 16u_C1V, and 32f_C1V flavors).
dstVolume	Size of the destination volume.
interpolation	The type of interpolation, the following values are possible:

Description

This function operates with volume of interest (VOI).

This function transforms the source volume by remapping its voxels. Voxel remapping is performed using pxMap, pyMap and pzMap buffers to look-up the coordinates of the source volume voxel that is written to the target destination volume voxel. The application has to supply these look-up tables.

The remapping of the source voxels to the destination voxels is made according to the following formulas:

dst_voxel[i, j, k] = src_voxel [pxMap[i, j, k], pyMap[i,j,k], pzMap[i , j,k]]

where i, j, k are the *x*-, *y*- and *z*-coordinates of the target destination volume voxel dst_voxel;

pxMap[i, j, k] contains the *x*-coordinates of the source volume voxels src_voxel that are written to dst_voxel.

pyMap[i, j, k] contains the *y*-coordinates of the source volume voxels src_voxel that are written to dst_voxel.

pzMap[i, j, k] contains the *z*-coordinates of the source volume voxels src_voxel that are written to dst_voxel.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.

ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when width, or height, or depth of the source and destination volumes has zero or negative value.
ippStsInterpolationErr	Indicates an error when interpolation has an illegal value.
ippStsWrongIntersectVOI	Indicates a warning when srcVoi has no intersection with the source volume, operation is not performed.

WarpAffineGetBufSize

Calculates the size of the external buffer for the affine transform.

Syntax

```
IppStatus ipprWarpAffineGetBufSize(IpprVolume srcVolume, IpprCuboid srcVoi, IpprCuboid dstVoi, const double coeffs[3][4], int nChannel, int interpolation, int* pSize);
```

Threading Layer (TL) function

```
IppStatus ipprWarpAffineGetBufSize_T (IpprVolume srcVolume, IpprCuboid srcVoi, IpprCuboid dstVoi, const double coeffs[3][4], int nChannel, int interpolation, int* pSize);
```

Include Files

ippi.h

ippi_tl.h

Domain Dependencies

Headers: ippcore.h, ippvm.h, ipps.h

Libraries: ippcore.lib, ippvm.lib, ipps.lib

Parameters

srcVolume	Size of the source volume.
srcVoi	Region of interest of the source volume.
dstVoi	Region of interest of the destination volume.
coeffs	Affine transform matrix.
nChannel	Number of channel or planes, possible value is 1.
interpolation	Type of interpolation, the • IPPI_INTER_NN - nearest neighbor interpolation, • IPPI_INTER_LINEAR - trilinear interpolation,

	following values are possible:	• IPPI_INTER_CUBIC - tricubic interpolation, • IPPI_INTER_CUBIC2P_BSPLINE - B - spline • IPPI_INTER_CUBIC2P_CATMULLROM - Catmull - Rom spline, • IPPI_INTER_CUBIC2P_B05C03- special two-parameters filter (1/2, 3/10).
pSize	Pointer to the size of the external buffer.	

Description

This function calculates the size (in bytes) of the external buffer that is required for the ipprWarpAffine function. (In some cases the function returns zero size of the buffer).

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when pSize or coeffs is NULL.
ippStsSizeErr	Indicates an error if width, or height, or depth of the srcVoi or dstVoi is less than, or equal to zero.
ippStsNumChannelErr	Indicates an error when nChannel has an illegal value.
ippStsInterpolationErr	Indicates an error when interpolation has an illegal value.

See Also

WarpAffine Performs the general affine transform of the source volume.

WarpAffine

Performs the general affine transform of the source volume.

Syntax

```
IppStatus ipprWarpAffine_8u_C1PV(const Ipp8u* const pSrc[], IpprVolume srcVolume, int srcStep, IpprCuboid srcVoi, Ipp8u* const pDst[], int dstStep, IpprCuboid dstVoi, const double coeffs[3][4], int interpolation, Ipp8u* pBuffer);

IppStatus ipprWarpAffine_16u_C1PV(const Ipp16u* const pSrc[], IpprVolume srcVolume, int srcStep, IpprCuboid srcVoi, Ipp16u* const pDst[], int dstStep, IpprCuboid dstVoi, const double coeffs[3][4], int interpolation, Ipp8u* pBuffer);

IppStatus ipprWarpAffine_32f_C1PV(const Ipp32f* const pSrc[], IpprVolume srcVolume, int srcStep, IpprCuboid srcVoi, Ipp32f* const pDst[], int dstStep, IpprCuboid dstVoi, const double coeffs[3][4], int interpolation, Ipp8u* pBuffer);
```

```
IppStatus ipprWarpAffine_8u_C1V(const Ipp8u* pSrc, IpprVolume srcVolume, int srcStep,
int srcPlaneStep, IpprCuboid srcVoi, Ipp8u* pDst, int dstStep, int dstPlaneStep,
IpprCuboid dstVoi, const double coeffs[3][4], int interpolation, Ipp8u* pBuffer);

IppStatus ipprWarpAffine_16u_C1V(const Ipp16u* pSrc, IpprVolume srcVolume, int srcStep,
int srcPlaneStep, IpprCuboid srcVoi, Ipp16u* pDst, int dstStep, int dstPlaneStep,
IpprCuboid dstVoi, const double coeffs[3][4], int interpolation, Ipp8u* pBuffer);

IppStatus ipprWarpAffine_32f_C1V(const Ipp32f* pSrc, IpprVolume srcVolume, int srcStep,
int srcPlaneStep, IpprCuboid srcVoi, Ipp32f* pDst, int dstStep, int dstPlaneStep,
IpprCuboid dstVoi, const double coeffs[3][4], int interpolation, Ipp8u* pBuffer);
```

Threading Layer (TL) functions

```
IppStatus ipprWarpAffine_8u_C1PV_T(const Ipp8u* const pSrc[], IpprVolume srcVolume, int
srcStep, IpprCuboid srcVoi, Ipp8u* const pDst[], int dstStep, IpprCuboid dstVoi, const
double coeffs[3][4], int interpolation, Ipp8u* pBuffer);

IppStatus ipprWarpAffine_16u_C1PV_T(const Ipp16u* const pSrc[], IpprVolume srcVolume,
int srcStep, IpprCuboid srcVoi, Ipp16u* const pDst[], int dstStep, IpprCuboid dstVoi,
const double coeffs[3][4], int interpolation, Ipp8u* pBuffer);

IppStatus ipprWarpAffine_32f_C1PV_T(const Ipp32f* const pSrc[], IpprVolume srcVolume,
int srcStep, IpprCuboid srcVoi, Ipp32f* const pDst[], int dstStep, IpprCuboid dstVoi,
const double coeffs[3][4], int interpolation, Ipp8u* pBuffer);

IppStatus ipprWarpAffine_8u_C1V_T(const Ipp8u* pSrc, IpprVolume srcVolume, int srcStep,
int srcPlaneStep, IpprCuboid srcVoi, Ipp8u* pDst, int dstStep, int dstPlaneStep,
IpprCuboid dstVoi, const double coeffs[3][4], int interpolation, Ipp8u* pBuffer);

IppStatus ipprWarpAffine_16u_C1V_T(const Ipp16u* pSrc, IpprVolume srcVolume, int
srcStep, int srcPlaneStep, IpprCuboid srcVoi, Ipp16u* pDst, int dstStep, int
dstPlaneStep, IpprCuboid dstVoi, const double coeffs[3][4], int interpolation, Ipp8u*
pBuffer);

IppStatus ipprWarpAffine_16u_C1V_T(const Ipp16u* pSrc, IpprVolume srcVolume, int
srcStep, int srcPlaneStep, IpprCuboid srcVoi, Ipp16u* pDst, int dstStep, int
dstPlaneStep, IpprCuboid dstVoi, const double coeffs[3][4], int interpolation, Ipp8u*
pBuffer);
```

Include Files

`ippi.h`

`ippi_tl.h`

Domain Dependencies

Headers: `ippcore.h`, `ippvm.h`, `ipps.h`

Libraries: `ippcore.lib`, `ippvm.lib`, `ipps.lib`

Parameters

`pSrc`	Array of pointers to the planes in the source volume.
`srcVolume`	Size, in pixels, of the source volume.

srcStep	Distance, in bytes, between the starting points of consecutive lines in each plane of the source volume.
srcPlaneStep	Distance, in bytes, between the starting points of consecutive lines in every plane of the source volume (for 8u_C1V, 16u_C1V, and 32f_C1V flavors).
dstStep	Distance, in bytes, between the starting points of consecutive lines in each plane of the destination volume.
srcVoi	Volume of interest of the source volume.
pDst	Array of pointers to the planes in the destination volume.
dstVoi	Volume of interest of the destination volume.
dstStep	Distance, in bytes, between the starting points of consecutive lines in every plane of the destination volume.
dstPlaneStep	Distance, in bytes, between the starting points of consecutive lines in every plane of the destination volume (for 8u_C1V, 16u_C1V, and 32f_C1V flavors).
coeffs	Coefficients of the affine transform.
interpolation	Type of interpolation, the following values are possible: • IPPI_INTER_NN - nearest neighbor interpolation, • IPPI_INTER_LINEAR - trilinear interpolation, • IPPI_INTER_CUBIC - tricubic interpolation, • IPPI_INTER_CUBIC2P_BSPLINE - B - spline • IPPI_INTER_CUBIC2P_CATMULLROM - Catmull - Rom spline, • IPPI_INTER_CUBIC2P_B05C03- special two-parameters filter (1/2, 3/10).
pBuffer	Pointer to the external buffer.

Description

This function operates with volume of interest (VOI).

This affine warp function transforms the coordinates (x,y,z) of the source volume voxels according to the following formulas:

$x' = c_{00}*x + c_{01}*y + c_{02}*z+c_{03}$

$y' = c_{10}*x + c_{11}*y + c_{12}*z + c_{13}$

$z' = c_{20}*x + c_{21}*y + c_{22}*z + c_{23}$

where x', y' and z' denote the voxel coordinates in the transformed volume, and c_{ij} are the affine transform coefficients stored in the array coeffs.

Before calling this function, compute the size of the external buffer pBuffer using the ipprWarpAffineGetBufSize function.

Return Values

ippStsNoErr	Indicates no error. Any other value indicates an error or a warning.
ippStsNullPtrErr	Indicates an error when one of the specified pointers is NULL.
ippStsSizeErr	Indicates an error when width, or height, or depth of the source and destination volumes is less than, or equal to zero.
ippStsCoeffErr	Indicates an error when determinant of the transform matrix c_{ij} is equal to zero.
ippStsInterpolationErr	Indicates an error when interpolation has an illegal value.
ippStsWrongIntersectVOI	Indicates a warning when srcVoi has no intersection with the source volume, operation is not performed.

See Also

WarpAffineGetBufSize Calculates the size of the external buffer for the affine transform.

Appendix A: Handling of Special Cases

Some mathematical functions implemented in Intel IPP are not defined for all possible argument values. This appendix describes how the corresponding Intel IPP image processing functions handle situations when their input arguments fall outside the range of function definition or may lead to ambiguously determined output results.

Table Special Cases for Intel IPP Image Processing Functions below summarizes these special cases for different functions and lists result values together with status codes returned by the functions. The status codes ending with Err (except for the ippStsNoErr status) indicate an error. When an error occurs, the function execution is interrupted. All other status codes indicate that the input argument is outside the range, but the function execution is continued with the corresponding result value.

Function Base Name	Data Type	Case Description	Result Value	Status Code
ippiSqrt	16s32f	Sqrt (x<0)Sqrt (x<0)	0NAN_32F	ippStsSqrtNegArgippStsSqrtNegArg
ippiDiv	8u16s32f16sc32sc32fc	Div (0/0)Div (x/0)Div (0/0)Div (x/0),	0IPP_MAX_8U0IPP_MAX_16SIPP_MIN_16SNAN_32FINF_	ippStsDivByZeroippStsDivByZeroippStsDivByZero

		x>0Div (x/0), x<0Div (0/0)Div (x/0), x>0Div (x/0), x<0Div (0/0)Div (x/0),Div (0/0)Div (x/0), x>0Div (x/0), x<0Div (0/0)Div (x/0), x>0Div (x/0), x<0	32FINF_NEG_32F000IPP_MAX_32SIPP_MIN_32SNAN_32FINF_32FINF_NEG_32F	ippStsDivByZero ippStsDivByZero ippStsDivByZero ippStsDivByZero ippStsDivByZero ippStsDivByZero ippStsDivByZero ippStsDivByZero ippStsDivByZero ippStsDivByZero
ippiDivC	all	Div(x/const), const=0	-	ippStsDivByZeroErr
ippiLn	8u16s32f	Ln (0)Ln (0)Ln (x<0)Ln (x<0)Ln(x<IPP_MINABS_32F)	0IPP_MIN_16SIPP_MIN_16SNAN_32FINF_NEG_32F	ippStsLnZeroArg ippStsLnZeroArg ippStsLnNegArgippStsLnNegArgippStsLnZeroArg
ippiExp	8u16s32f	overflowoverflowoverflow	IPP_MAX_8UIPP_MAX_16SINF_32F	ippStsNoErrippStsNoErrippStsNoErr

Here `x` denotes an input value. For the definition of the constants used, see Image Data Types and Ranges in Intel® Integrated Performance Primitives Concepts.

Note that flavors of the same math function operating on different data types may produce diferent results for the equal argument values. However, for a given function and a fixed data type, handling of special cases is the same for all function flavors that have different descriptors in their names. For example, logarithm function `ippiLn` operating on `16s` data treats zero argument values in the same way for all its flavors `ippiLn_16s_C1RSfs`, `ippiLn_16s_C3RSfs`, `ippiLn_16s_C1IRSfs`, and `ippiLn_16s_C3IRSfs`.

Appendix B: Interpolation in Image Geometric Transform Functions

This appendix describes the interpolation algorithms used in the geometric transformation functions of Intel IPP. For more information about each of the geometric transform functions, see Image Geometry Transforms.

- Overview of Interpolation Modes
- Mathematical Notation
- Nearest Neighbor Interpolation
- Linear Interpolation
- Cubic Interpolation
- Super Sampling
- Lanczos Interpolation
- Tricubic Interpolation
- Trilinear Interpolation
- Interpolation with Two-Parameter Cubic Filters

Overview of Interpolation Modes

In geometric transformations, the grid of input image pixels is not necessarily mapped onto the grid of pixels in the output image. Therefore, to compute the pixel intensities in the output image, the geometric transform functions need to interpolate the intensity values of several input pixels that are mapped to a certain neighborhood of the output pixel.

Geometric transformations can use various interpolation algorithms. The library supports the following interpolation modes depending on how the type of interpolation algorithm is specified:

- **Type 1:** Application code specifies the interpolation mode by passing the `interpolation` parameter of `int` type to a processing function. The following interpolation modes are supported:
  - Nearest neighbor interpolation (`interpolation` = `IPPI_INTER_NN`)
  - Linear interpolation (`interpolation = IPPI_INTER_LINEAR`)
  - Cubic interpolation (`interpolation = IPPI_INTER_CUBIC`)
  - Interpolation with Lanczos window function (`interpolation` = `IPPI_INTER_LANCZOS`)
  - Interpolation with two-parameter cubic filters with the fixed coefficients (`interpolation` can be set to the following:
    - `IPPI_INTER_CUBIC2P_BSPLINE` (B=1;C=0)
    - `IPPI_INTER_CUBIC2P_CATMULLROM` (B=0;C=0.5)
    - `IPPI_INTER_CUBIC2P_B05C03` (B=0.5;C=0.3)
- **Type 2:** Interpolation mode is specified explicitly in a function name suffix:
  - Nearest neighbor interpolation (pass `interpolation` = `ippNearest` to `GetSize` functions, use the functions with the `Nearest` suffix, for example, `ResizeNearestInit` or `ResizeNearest`)
  - Linear interpolation (pass `interpolation = ippLinear` to `GetSize` functions, use the functions with the `Linear` suffix, for example, `ResizeLinearInit` or `ResizeLinear`)
  - Interpolation with two-parameter cubic filters (pass `interpolation = ippCubic` to `GetSize` functions, use the functions with the `Cubic` suffix, for example, `ResizeCubicInit` or `ResizeCubic`)
  - Supersampling (pass `interpolation = ippSuper` to `GetSize` functions, use the functions with the `Super` suffix, for example, `ResizeSuperInit` or `ResizeSuper`)
  - Interpolation with Lanczos window function (pass `interpolation` = `ippLanczos` to `GetSize` functions, use the functions with the `Lanczos` suffix, for example, `ResizeLanczosInit` or `ResizeLanczos`)

For certain functions of type 1, the specified interpolation modes can be combined with additional *smoothing of edges* to which the borders of the original image are transformed. To use this option, for the `interpolation` parameter, specify the edge smoothing flag and the desired interpolation mode using the bitwise `OR` operation. For example, in order to rotate an image with cubic interpolation and smooth the rotated image edges, pass the following value to `ippiRotate()`:

`interpolation= IPPI_INTER_CUBIC | IPPI_SMOOTH_EDGE.`

To enable edge smoothing for functions of type 2, pass the special flag to the `Init` function (if it exists), for example, you can pass this flag to `Init` functions for `WarpAffine` and `WarpPerspective` function groups.

Interpolation with edge smoothing option can be used only in those geometric transform functions where this option is explicitly listed in the parameters definition section.

Table Interpolation Modes Supported by Image Geometric Transform Functions lists the supported interpolation modes for the main geometric transform functions that use interpolation.

Function Base Name	NN	Lin	Cub2P	Cub	CR	La2	La3	SS	AA	ES
Resize Cubic			x		x *)					
Resize Lanczos						x	x			

ResizeLinear		x							
ResizeNearest	x								
ResizeSuper							x		
WarpAffineCubic			x		x *)				
WarpAffineLinear		x							
WarpAffineNearest	x								
WarpPerspectiveCubic			x		x *)				
WarpPerspectiveLinear		x							
WarpPerspectiveNearest	x								
Remap	x	x		x	x	x	n/a	n/a	x
WarpBilinear	x	x		x			n/a	n/a	x
WarpBilinearBack	x	x		x			n/a	n/a	
WarpBilinearQuad	x	x		x			n/a	n/a	x

*) The function ippiResizeCubic supports the interpolation with two-parameter cubic filters, where parameters B and C can be specified explicitly.

Here **NN** - nearest neighbor interpolation, **Lin** - linear interpolation, **Cub2P** - interpolation with two-parameter cubic filters, **Cub** - cubic interpolation, **CR** - Catmull-Rom spline, **La2, La3** - interpolation with the Lanczos window, **SS** - super sampling interpolation, **AA** - antialiasing, **ES** - edge smoothing.

The sections that follow provide more details on each interpolation mode.

Mathematical Notation

In this appendix the following notation is used:

(x_D, y_D)	pixel coordinates in the destination image (integer values);
(x_S, y_S)	the computed coordinates of a point in the source image that is mapped exactly to (x_D, y_D);
$S(x,y)$	pixel value (intensity) in the source image;
$D(x,y)$	pixel value (intensity) in the destination image.

Nearest Neighbor Interpolation

This is the fastest and least accurate interpolation mode. The pixel value in the destination image is set to the value of the source image pixel closest to the point

$(x_S, y_S): D(x_D, y_D) = S(\text{round}(x_S), \text{round}(y_S))$.

To use the nearest neighbor interpolation, set the `interpolation` parameter to `IPPI_INTER_NN` or use the functions with the `Nearest` suffix (pass `interpolation = ippNearest` to `GetSize` functions).

Linear Interpolation

The linear interpolation is slower but more accurate than the nearest neighbor interpolation. On the other hand, it is faster but less accurate than cubic interpolation. The linear interpolation algorithm uses source image intensities at the four pixels (x_{S0}, y_{S0}), (x_{S1}, y_{S0}), (x_{S0}, y_{S1}), (x_{S1}, y_{S1}) that are closest to (x_S, y_S) in the source image:

$x_{S0} = \text{int}(x_S)$, $x_{S1} = x_{S0} + 1$, $y_{S0} = \text{int}(y_S)$, $y_{S1} = y_{S0} + 1$.

First, the intensity values are interpolated along the x-axis to produce two intermediate results I_0 and I_1 (see Figure Linear Interpolation):

$I_0 = S(x_S, y_{S0}) = S(x_{S0}, y_{S0})*(x_{S1} - x_S) + S(x_{S1}, y_{S0})*(x_S - x_{S0})$

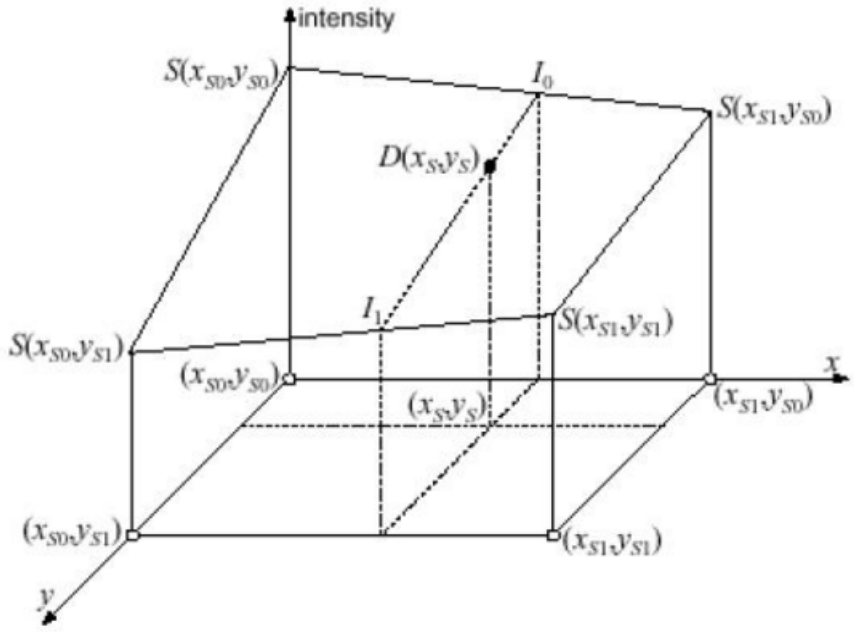
$I_1 = S(x_S, y_{S1}) = S(x_{S0}, y_{S1})*(x_{S1} - x_S) + S(x_{S1}, y_{S1})*(x_S - x_{S0})$.

Then, the sought-for intensity $D(x_D, y_D)$ is computed by interpolating the intermediate values I_0 and I_1 along the `y – axis`:

$D(x_D, y_D) = I_0 *(y_{S1} - y_S) + I_1 *(y_S - y_{S0})$.

To use the linear interpolation, set the `interpolation` parameter to `IPPI_INTER_LINEAR` or use the functions with the `Linear` suffix (pass `interpolation = ippLinear` to `GetSize` functions). For images with 8-bit unsigned color channels, the `ippiWarpAffine` and `ippiResizeLinear` functions compute the coordinates (x_S, y_S) with the accuracy $2^{-16} = 1/65536$. For images with 16-bit unsigned color channels, these functions compute the coordinates with floating-point precision.

Linear Interpolation |



Cubic Interpolation

The cubic interpolation algorithm (see Figure Cubic Interpolation) uses source image intensities at sixteen pixels in the neighborhood of the point (x_S, y_S) in the source image:

$x_{S0} = int(x_S) - 1$; $x_{S1} = x_{S0} + 1$; $x_{S2} = x_{S0} + 2$; $x_{S3} = x_{S0} + 3$;

$y_{S0} = int(y_S) - 1$; $y_{S1} = y_{S0} + 1$; $y_{S2} = y_{S0} + 2$; $y_{S3} = y_{S0} + 3$.

First, for each y_{Sk} the algorithm determines four cubic polynomials $F_0(x)$, $F_1(x)$, $F_2(x)$, and $F_3(x)$:

$F_k(x) = a_k x^3 + b_k x^2 + c_k x + d_k$, $0 \le k \le 3$

such that

$F_k(x_{S0}) = S(x_{S0}, y_{Sk})$; $F_k(x_{S1}) = S(x_{S1}, y_{Sk})$,

$F_k(x_{S2}) = S(x_{S2}, y_{Sk})$; $F_k(x_{S3}) = S(x_{S3}, y_{Sk})$.

In Figure Cubic Interpolation , these polynomials are shown by solid curves.

Then, the algorithm determines a cubic polynomial $F_y(y)$ such that

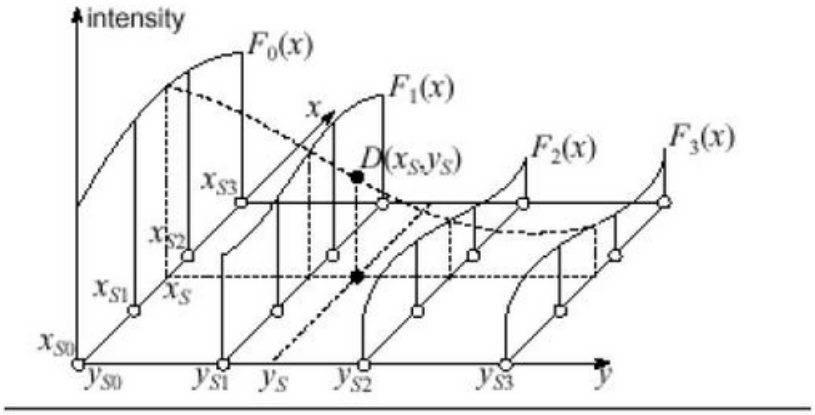
$F_y(y_{S0}) = F_0(x_S)$, $F_y(y_{S1}) = F_1(x_S)$, $F_y(y_{S2}) = F_2(x_S)$, $F_y(y_{S3}) = F_3(x_S)$.

The polynomial $F_y(y)$ is represented by the dashed curve in Figure Cubic Interpolation.

Finally, the sought intensity $D(x_D, y_D)$ is set to the value $F_y(y_S)$.

To use the cubic interpolation, set the `interpolation` parameter to `IPPI_INTER_CUBIC`.

Cubic Interpolation |



Super Sampling

If the destination image is much smaller than the source image, the other interpolation algorithms may skip some pixels in the source image (that is, these algorithms not necessarily use all source pixels when computing intensity of the destination pixels).

The super-sampling algorithm is as follows:

1. Divide the source image rectangular ROI (or the whole image, if there is no ROI) into equal rectangles, each rectangle corresponding to some pixel in the destination image. Note that each source pixel is represented by a 1x1 square.
2. Compute a weighted sum of source pixel values for all pixels that are in the rectangle or have a non-zero intersection with the rectangle. If a source pixel is fully contained in the rectangle, the value of that pixel is taken with weight 1. If the rectangle and the square of the source pixel have an intersection of area `a < 1`, that pixel's value is taken with weight `a`.

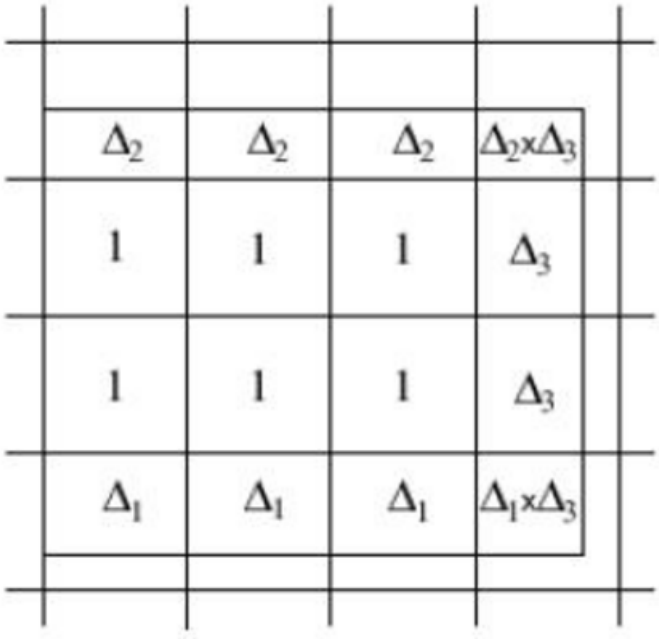
   Figure Supersampling Weights shows the corresponding weight value for each source pixel intersecting with the rectangle.
3. To compute the pixel value in the destination image, divide this weighted sum by the ratio of the source and destination rectangle areas S_{Src}/ S_{Dst} = `1/ xFactor * yFactor`.

   Here `xFactor`, and `yFactor` are the parameters of the functions that specify the factors by which the `x` and `y` dimensions of the source image ROI are changed.

Note that supersampling interpolation can be used only for `xFactor < 1`, and `yFactor < 1`.

To use the supersampling interpolation, use the functions with the `Super` suffix (pass `interpolation = ippSuper` to `GetSize` functions).

Supersampling Weights |



Lanczos Interpolation

This method is based on the 2-lobed or 3-lobed Lanczos window function as the interpolation function.

Interpolation with the 2-lobed Lanczos Window Function

The interpolation algorithm uses source image intensities at 16 pixels in the neighborhood of the point (x_S, y_S) in the source image:

x_{S0} = int(x_S) - 1; x_{S1} = x_{S0} + 1; x_{S2} = x_{S0} + 2; x_{S3} = x_{S0} + 3;

y_{S0} = int(y_S) - 1; y_{S1} = y_{S0} + 1; y_{S2} = y_{S0} + 2; y_{S3} = y_{S0} + 3;

First, the intensity values are interpolated along the `x – axis` to produce four intermediate results I_0, I_1, I_2, I_3:

$$I_k = \sum_{i=0}^{3} a_i \cdot s(x_{Si}, y_{Sk}), 0 \le k \le 3$$

Then the intensity $D(x_D, y_D)$ is computed by interpolating the intermediate values I_k along the `y – axis`:

$$D(x_D, y_D) = \sum_{k=0}^{3} b_k \cdot I_k$$

Here a_i and b_k are the coefficients defined as

$a_i = L(x_S - x_{Si})$, $b_k = L(y_S - y_{Si})$,

where $L(x)$ is the Lanczos windowed sinc function:

$$L(x) = \sin c(x) \cdot Lanczos2(x) = \begin{cases} \frac{\sin(\pi x)}{\pi x} \cdot \frac{\sin((\pi x)/2)}{(\pi x)/2}, & 0 \le |x| < 2 \\ 0, & 2 \le |x| \end{cases}$$

To use this interpolation, use the `ippiResizeLanczos` function.

Interpolation with the 3-lobed Lanczos Window Function

The interpolation algorithm uses source image intensities at 36 pixels in the neighborhood of the point (x_S, y_S) in the source image:

$x_{S0} = \text{int}(x_S) - 2$; $x_{S1} = x_{S0} + 1$; $x_{S2} = x_{S0} + 2$; $x_{S3} = x_{S0} + 3$; $x_{S3} = x_{S0} + 4$; $x_{S3} = x_{S0} + 5$;

$y_{S0} = \text{int}(y_S) - 2$; $y_{S1} = y_{S0} + 1$; $y_{S2} = y_{S0} + 2$; $y_{S3} = y_{S0} + 3$; $y_{S2} = y_{S0} + 4$; $y_{S2} = y_{S0} + 5$;

First, the intensity values are interpolated along the `x – axis` to produce six intermediate results I_0, I_1, … I_5:

$$I_k = \sum_{i=0}^{5} a_i \cdot s(x_{Si}, y_{Sk}), 0 \le k \le 5$$

Then the intensity $D(x_D, y_D)$ is computed by interpolating the intermediate values I_k along the `y – axis`:

$$D(x_D, y_D) = \sum_{k=0}^{5} b_k \cdot I_k$$

Here a_i and b_k are the coefficients defined as

$a_i = L(x_S - x_{Si})$, $b_k = L(y_S - y_{Si})$,

where $L(x)$ is the Lanczos windowed sinc function:

$$L(x) = \sin c(x) \cdot Lanczos3(x) = \begin{cases} \frac{\sin(\pi x)}{\pi x} \cdot \frac{\sin((\pi x)/3)}{(\pi x)/3}, & 0 \le |x| < 3 \\ 0, & 3 \le |x| \end{cases}$$

To use this interpolation, set the `interpolation` parameter to `IPPI_INTER_LANCZOS`, or use the functions with the `Lanczos` suffix (pass `interpolation = ippLanczos` to `GetSize` functions).

Tricubic Interpolation

Short Description

The tricubic interpolation algorithm uses source image intensities at 64 pixels in the neighborhood of the point (x_s, y_s, z_s) in the source image:

$$x_{s_0} = int(x_s) - 1 \qquad y_{s_0} = int(y_s) - 1$$
$$x_{s_1} = x_{s_0} + 1 \qquad y_{s_1} = y_{s_0} + 1$$
$$x_{s_2} = x_{s_0} + 2 \qquad y_{s_2} = y_{s_0} + 2$$
$$x_{s_3} = x_{s_0} + 3 \qquad y_{s_3} = y_{s_0} + 3$$

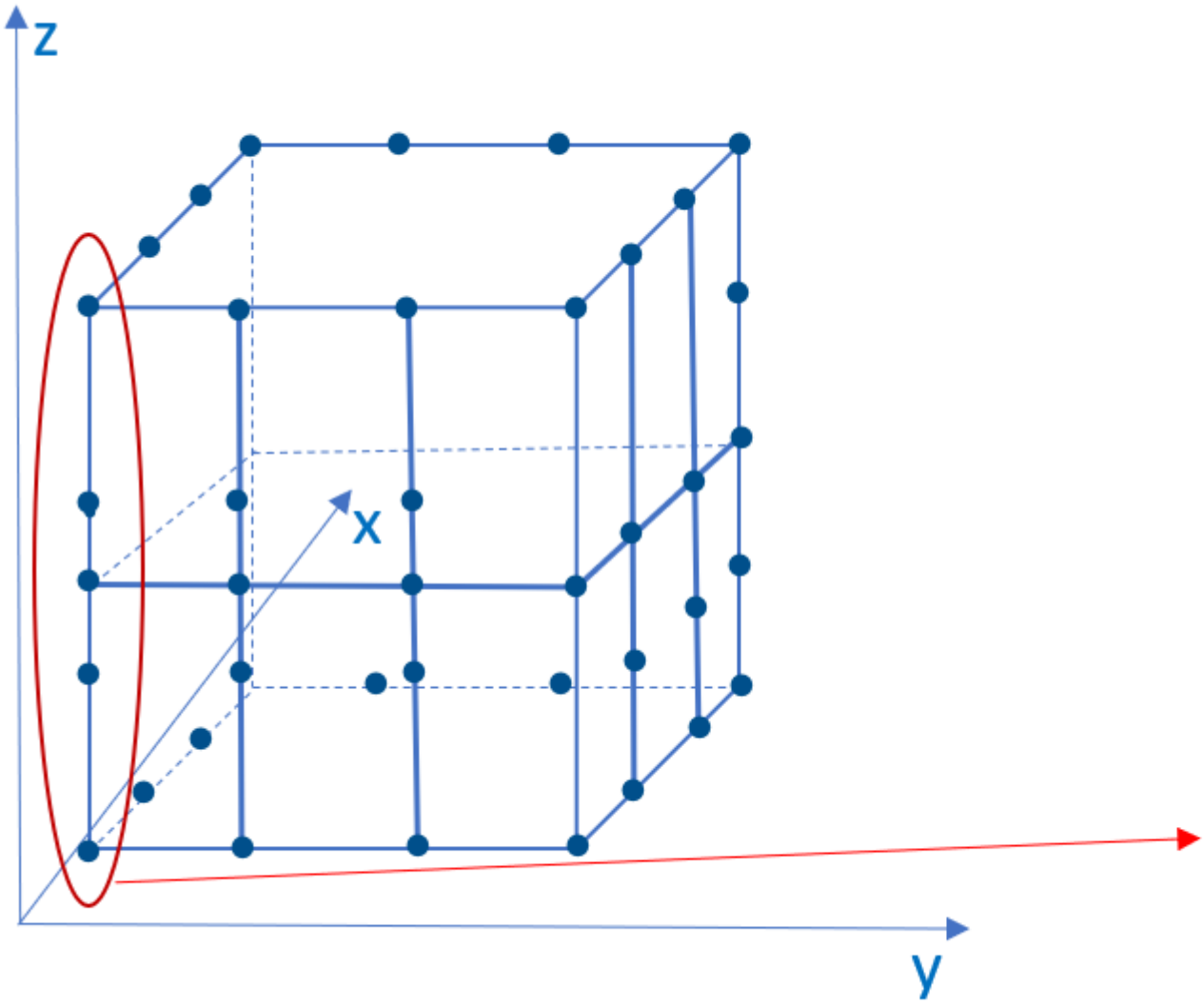
First, for each z_{sk} the algorithm determines 16 cubic polynomials $F_{nm}(z)$, $0 \le m, n \le 15$:

$F_{nm}(z) = a_{mn} z^3 + b_{nm} z^2 + c_{nm} z + d_{nm}$

For fixed values z_s , the function $F_{nm}(z_{snm})$ is equal to:

$$F00_k(z_{s_k}) = S(x_{s_0}, y_{s_0}, z_{s_k})$$
$$F01_k(z_{s_k}) = S(x_{s_0}, y_{s_1}, z_{s_k})$$
$$F10_k(z_{s_k}) = S(x_{s_1}, y_{s_0}, z_{s_k})$$
$$F11_k(z_{s_k}) = S(x_{s_1}, y_{s_1}, z_{s_k})$$
$$F20_k(z_{s_k}) = S(x_{s_2}, y_{s_0}, z_{s_k})$$
$$F21_k(z_{s_k}) = S(x_{s_2}, y_{s_1}, z_{s_k})$$
$$F30_k(z_{s_k}) = S(x_{s_3}, y_{s_0}, z_{s_k})$$
$$F31_k(z_{s_k}) = S(x_{s_3}, y_{s_1}, z_{s_k})$$

There are sixteen points with fixed z_s $F_{00}(z)$, $F_{01}(z)$, $F_{02}(z)$, $F_{03}(z)$, $F_{10}(z)$, $F_{11}(z)$, $F_{12}(z)$, $F_{13}(z)$, $F_{20}(z)$, $F_{21}(z)$, $F_{22}(z)$, $F_{23}(z)$, $F_{30}(z)$, $F_{31}(z)$, $F_{32}(z)$, $F_{33}(z)$. The need is to fix y_s and x_s. This situation is described in the Cubic Interpolation, refer to this page to visualize the operation of the algorithm as well.



To use the cubic interpolation, set the interpolation parameter to `IPPI_INTER_CUBIC`.

Trilinear Interpolation

The trilinear interpolation is slower but more accurate than the 3D neighbor interpolation. It is also faster but less accurate than the tricubic interpolation. The trilinear interpolation algorithm uses source image intensities at the eight pixels: (x_{s0}, y_{s0}, z_{s0}), (x_{s0}, y_{s0}, z_{s1}), (x_{s0}, y_{s1}, z_{s0}), (x_{s0}, y_{s1}, z_{s1}), (x_{s1}, y_{s0}, z_{s0}), (x_{s1}, y_{s0}, z_{s1}), (x_{s1}, y_{s1}, z_{s0}), (x_{s1}, y_{s1}, z_{s1}) that are the closest to (x_s, y_s, z_s) in the source image:

$$x_{s_0} = int(x_s)$$
$$y_{s_0} = int(y_s)$$
$$z_{s_0} = int(z_s)$$

also with the conditions:

$$x_{s_0} < x_s < x_{s_1}$$
$$y_{s_0} < y_s < y_{s_1}$$
$$z_{s_0} < z_s < z_{s_1}$$

First, the intensity values are interpolated along the x-axis to produce four intermediate results I_0, I_1, I_2, I_3 :

$$I_0 = S(x_s, y_{s_0}, z_{s_0}) = S(x_{s_0}, y_{s_0}, z_{s_0})(x_{s_1} - x_s) + S($$

$$I_1 = S(x_s, y_{s_0}, z_{s_1}) = S(x_{s_0}, y_{s_0}, z_{s_1})(x_{s_1} - x_s) + S($$

$$I_2 = S(x_s, y_{s_1}, z_{s_0}) = S(x_{s_0}, y_{s_1}, z_{s_0})(x_{s_1} - x_s) + S($$

$$I_3 = S(x_s, y_{s_1}, z_{s_1}) = S(x_{s_0}, y_{s_1}, z_{s_1})(x_{s_1} - x_s) + S($$

Second, the intensity values are interpolated along the y-axis using the intermediate value I_0, I_1, I_2, I_3:

$$T_0 = S(x_s, y_s, z_{s_0}) = I_0(y_{s_1} - y_s) + I_2(y_s - y_{s_0})$$

$$T_1 = S(x_s, y_s, z_{s_1}) = I_1(y_{s_1} - y_s) + I_3(y_s - y_{s_0})$$

Then, the sought-for intensity $D(x_D, y_D, z_D) = T_0 (z_{s1}-z_s) + T_1 (z_s-z_{s0})$

To use the linear interpolation, set the `interpolation` parameter to `IPPI_INTER_LINEAR` or use the functions with the `Linear` suffix (pass `interpolation = ippLinear` to `GetSize` functions).

To visualize the operation of the algorithm, look at the Linear interpolation page .

Interpolation with Two-Parameter Cubic Filters

The two-parameter family of cubic filters have kernels of the form:

$$k(x) = \frac{1}{6}\begin{cases}(12-9B-6C)|x|^3+(-18+12B+6C)|x|^2+(6-2B) & |x|<1\\(-B-6C)|x|^3+(6B+30C)|x|^2+(-12B-48C)|x|+(8B+24C) & 1\le|x|<2\\0 & \text{otherwise}\end{cases}$$

where `B` and `C` are two parameters; their variations give different approximation.

These interpolation methods additionally filter the output to improve quality of an image.

To get more information about how `B` and `C` values affect the result, refer to [Mitchell88].

Interpolation with two parameter cubic filters is presented as following functions:

- B-spline
- Catmull-Rom spline
- Special two parameters filter

B-spline

B-spline is a more general type of curve than Bezier curves. This spline function has the smallest support for a given degree, order of smoothness, and domain partitioning.

Parameters `B` and `C` are equal to 1 and 0 respectively.

$$k(x) = \frac{1}{6}\begin{cases}3|x|^3-6|x|^2+4 & |x|<1\\-|x|^3+6|x|^2-12|x|+8 & 1\le|x|<2\\0 & otherwise\end{cases}$$

To use the interpolation with two-parameter cubic filters, set the interpolation parameter to `IPPI_INTER_CUBIC2P_BSPLINE`.

To learn more about B-splines, refer to ScrewBender

Catmull-Rom Spline

The Catmull-Rom spline is a kind of cubic Hermite spline. Its peculiarity is that the interpolated function is given not only by its values at points but also by its first derivatives .

Parameters B and C are equal to 0 and 0.5 respectively.

$$k(x) = \frac{1}{6}\begin{cases} 9|x|^3 - 15|x|^2 + 6 & |x| < 1 \\ -3|x|^3 + 15|x|^2 - 24|x| + 12 & 1 \le |x| < 2 \\ 0 & otherwise \end{cases}$$

These interpolation methods additionally filter the output to improve the quality of an image.

To use the interpolation with two-parameter cubic filters, set the interpolation parameter `IPPI_INTER_CUBIC2P_CATMULLROM`.

Special Two-Parameters Filter

Parameters `B` and `C` are equal to 0.5 and 0.3 respectively.

$$k(x) = \frac{1}{6}\begin{cases} 5.7|x|^3 - 10.2|x|^2 + 5 & |x| < 1 \\ -2.3|x|^3 + 12|x|^2 - 20.4|x| + 11.2 & 1 \le |x| < 2 \\ 0 & otherwise \end{cases}$$

To use the interpolation with two-parameter cubic filters, set the interpolation parameter `IPPI_INTER_CUBIC2P_ B05C03` .

Appendix C: Removed Functions for Image and Video Processing

This appendix contains tables that list the functions removed from Intel IPP 9.0. If an application created with the previous versions calls a function listed here, then the source code must be modified. The tables specify the corresponding Intel IPP 9.0 functions or workaround to replace the removed functions:

- ippcc.h
- ippcv.h
- ippi.h
- ippj.h - the whole domain is removed
- ippvc.h - the whole domain is removed

> **NOTE** To get information on possible alternatives to the removed functions that do not have substitution or workaround in Intel IPP , file a support request at Online Service Center.

ippcc.h:

Removed from 9.0	Substitution or Workaround
`ippiBGR555ToYCbCr411_16u8u_C3P3R`	N/A
`ippiBGR555ToYCbCr420_16u8u_C3P3R`	N/A
`ippiBGR555ToYCbCr422_16u8u_C3C2R`	N/A

ippiBGR555ToYCbCr422_16u8u_C3P3R	N/A
ippiBGR555ToYCrCb420_16u8u_C3P3R	N/A
ippiBGR555ToYUV420_16u8u_C3P3R	N/A
ippiBGR565ToBGR_16u8u_C3R	N/A
ippiBGR565ToYCbCr411_16u8u_C3P3R	N/A
ippiBGR565ToYCbCr420_16u8u_C3P3R	N/A
ippiBGR565ToYCbCr422_16u8u_C3C2R	N/A
ippiBGR565ToYCbCr422_16u8u_C3P3R	N/A
ippiBGR565ToYCrCb420_16u8u_C3P3R	N/A
ippiBGR565ToYUV420_16u8u_C3P3R	N/A
ippiBGRToBGR565_8u16u_C3R	N/A
ippiColorTwist32f_8s_AC4IR	8s is not supported anymore, use another data type
ippiColorTwist32f_8s_AC4R	8s is not supported anymore, use another data type
ippiColorTwist32f_8s_C3IR	8s is not supported anymore, use another data type
ippiColorTwist32f_8s_C3R	8s is not supported anymore, use another data type
ippiColorTwist32f_8s_IP3R	8s is not supported anymore, use another data type
ippiColorTwist32f_8s_P3R	8s is not supported anymore, use another data type
ippiRGB565ToRGB_16u8u_C3R	N/A
ippiRGB565ToYUV420_16u8u_C3P3R	N/A
ippiRGB565ToYUV422_16u8u_C3P3R	N/A
ippiRGBToRGB565_8u16u_C3R	N/A
ippiYCbCr411ToBGR555_8u16u_P3C3R	N/A
ippiYCbCr411ToBGR565_8u16u_P3C3R	N/A
ippiYCbCr420ToBGR444Dither_8u16u_P3C3R	N/A
ippiYCbCr420ToBGR444_8u16u_P3C3R	N/A
ippiYCbCr420ToBGR555Dither_8u16u_P3C3R	N/A
ippiYCbCr420ToBGR555_8u16u_P3C3R	N/A
ippiYCbCr420ToBGR565Dither_8u16u_P3C3R	N/A
ippiYCbCr420ToBGR565_8u16u_P3C3R	N/A
ippiYCbCr420ToRGB444Dither_8u16u_P3C3R	N/A
ippiYCbCr420ToRGB444_8u16u_P3C3R	N/A

ippiYCbCr420ToRGB555Dither_8u16u_P3C3R	N/A
ippiYCbCr420ToRGB555_8u16u_P3C3R	N/A
ippiYCbCr420ToRGB565Dither_8u16u_P3C3R	N/A
ippiYCbCr420ToRGB565_8u16u_P3C3R	N/A
ippiYCbCr422ToBGR444Dither_8u16u_C2C3R	N/A
ippiYCbCr422ToBGR444Dither_8u16u_P3C3R	N/A
ippiYCbCr422ToBGR444_8u16u_C2C3R	N/A
ippiYCbCr422ToBGR444_8u16u_P3C3R	N/A
ippiYCbCr422ToBGR555Dither_8u16u_C2C3R	N/A
ippiYCbCr422ToBGR555Dither_8u16u_P3C3R	N/A
ippiYCbCr422ToBGR555_8u16u_C2C3R	N/A
ippiYCbCr422ToBGR555_8u16u_P3C3R	N/A
ippiYCbCr422ToBGR565Dither_8u16u_C2C3R	N/A
ippiYCbCr422ToBGR565Dither_8u16u_P3C3R	N/A
ippiYCbCr422ToBGR565_8u16u_C2C3R	N/A
ippiYCbCr422ToBGR565_8u16u_P3C3R	N/A
ippiYCbCr422ToRGB444Dither_8u16u_C2C3R	N/A
ippiYCbCr422ToRGB444Dither_8u16u_P3C3R	N/A
ippiYCbCr422ToRGB444_8u16u_C2C3R	N/A
ippiYCbCr422ToRGB444_8u16u_P3C3R	N/A
ippiYCbCr422ToRGB555Dither_8u16u_C2C3R	N/A
ippiYCbCr422ToRGB555Dither_8u16u_P3C3R	N/A
ippiYCbCr422ToRGB555_8u16u_C2C3R	N/A
ippiYCbCr422ToRGB555_8u16u_P3C3R	N/A
ippiYCbCr422ToRGB565Dither_8u16u_C2C3R	N/A
ippiYCbCr422ToRGB565Dither_8u16u_P3C3R	N/A
ippiYCbCr422ToRGB565_8u16u_C2C3R	N/A
ippiYCbCr422ToRGB565_8u16u_P3C3R	N/A
ippiYCbCrToBGR444Dither_8u16u_C3R	N/A
ippiYCbCrToBGR444Dither_8u16u_P3C3R	N/A
ippiYCbCrToBGR444_8u16u_C3R	N/A

ippiYCbCrToBGR444_8u16u_P3C3R	N/A
ippiYCbCrToBGR555Dither_8u16u_C3R	N/A
ippiYCbCrToBGR555Dither_8u16u_P3C3R	N/A
ippiYCbCrToBGR555_8u16u_C3R	N/A
ippiYCbCrToBGR555_8u16u_P3C3R	N/A
ippiYCbCrToBGR565Dither_8u16u_C3R	N/A
ippiYCbCrToBGR565Dither_8u16u_P3C3R	N/A
ippiYCbCrToBGR565_8u16u_C3R	N/A
ippiYCbCrToBGR565_8u16u_P3C3R	N/A
ippiYCbCrToRGB444Dither_8u16u_C3R	N/A
ippiYCbCrToRGB444Dither_8u16u_P3C3R	N/A
ippiYCbCrToRGB444_8u16u_C3R	N/A
ippiYCbCrToRGB444_8u16u_P3C3R	N/A
ippiYCbCrToRGB555Dither_8u16u_C3R	N/A
ippiYCbCrToRGB555Dither_8u16u_P3C3R	N/A
ippiYCbCrToRGB555_8u16u_C3R	N/A
ippiYCbCrToRGB555_8u16u_P3C3R	N/A
ippiYCbCrToRGB565Dither_8u16u_C3R	N/A
ippiYCbCrToRGB565Dither_8u16u_P3C3R	N/A
ippiYCbCrToRGB565_8u16u_C3R	N/A
ippiYCbCrToRGB565_8u16u_P3C3R	N/A
ippiYUV420ToBGR444Dither_8u16u_P3C3R	N/A
ippiYUV420ToBGR444_8u16u_P3C3R	N/A
ippiYUV420ToBGR555Dither_8u16u_P3C3R	N/A
ippiYUV420ToBGR555_8u16u_P3C3R	N/A
ippiYUV420ToBGR565Dither_8u16u_P3C3R	N/A
ippiYUV420ToBGR565_8u16u_P3C3R	N/A
ippiYUV420ToRGB444Dither_8u16u_P3C3R	N/A
ippiYUV420ToRGB444_8u16u_P3C3R	N/A
ippiYUV420ToRGB555Dither_8u16u_P3C3R	N/A
ippiYUV420ToRGB555_8u16u_P3C3R	N/A

ippiYUV420ToRGB565Dither_8u16u_P3C3R	N/A
ippiYUV420ToRGB565_8u16u_P3C3R	N/A

ippcv.h:

Removed from 9.0	Substitution or Workaround
ippiAddProduct_8s32f_C1IMR	8s is not supported anymore, use another data type
ippiAddProduct_8s32f_C1IR	8s is not supported anymore, use another data type
ippiAddSquare_8s32f_C1IMR	8s is not supported anymore, use another data type
ippiAddSquare_8s32f_C1IR	8s is not supported anymore, use another data type
ippiAddWeighted_8s32f_C1IMR	8s is not supported anymore, use another data type
ippiAddWeighted_8s32f_C1IR	8s is not supported anymore, use another data type
ippiAdd_8s32f_C1IMR	8s is not supported anymore, use another data type
ippiAdd_8s32f_C1IR	8s is not supported anymore, use another data type
ippiDilateBorderReplicate_32f_C1R	ippiDilateBorder_32f_C1R
ippiDilateBorderReplicate_32f_C3R	ippiDilateBorder_32f_C3R
ippiDilateBorderReplicate_32f_C4R	ippiDilateBorder_32f_C4R
ippiDilateBorderReplicate_8u_C1R	ippiDilateBorder_8u_C1R
ippiDilateBorderReplicate_8u_C3R	ippiDilateBorder_8u_C3R
ippiDilateBorderReplicate_8u_C4R	ippiDilateBorder_8u_C4R
ippiErodeBorderReplicate_32f_C1R	ippiErodeBorder_32f_C1R
ippiErodeBorderReplicate_32f_C3R	ippiErodeBorder_32f_C3R
ippiErodeBorderReplicate_32f_C4R	ippiErodeBorder_32f_C4R
ippiErodeBorderReplicate_8u_C1R	ippiErodeBorder_8u_C1R
ippiErodeBorderReplicate_8u_C3R	ippiErodeBorder_8u_C3R
ippiErodeBorderReplicate_8u_C4R	ippiErodeBorder_8u_C4R
ippiFilterGaussBorder_32f_C1R	ippiFilterGaussianBorder_32f_C1R
ippiFilterGaussGetBufferSize_32f_C1R	ippiFilterGaussianGetBufferSize
ippiFilterLaplacianBorder_8u8s_C1R	8s data type is not supported anymore - use another data type
ippiFilterLaplacianGetBufferSize_8u8s_C1R	8s data type is not supported anymore - use another data type
ippiFilterMaxBorderReplicate_32f_C1R	ippiFilterMaxBorder_32f_C1R

ippiFilterMaxBorderReplicate_32f_C3R	ippiFilterMaxBorder_32f_C3R
ippiFilterMaxBorderReplicate_32f_C4R	ippiFilterMaxBorder_32f_C4R
ippiFilterMaxBorderReplicate_8u_C1R	ippiFilterMaxBorder_8u_C1R
ippiFilterMaxBorderReplicate_8u_C3R	ippiFilterMaxBorder_8u_C3R
ippiFilterMaxBorderReplicate_8u_C4R	ippiFilterMaxBorder_8u_C4R
ippiFilterMaxGetBufferSize_32f_C1R	ippiFilterMaxBorderGetBufferSize with dataType=32f and numChannels=1
ippiFilterMaxGetBufferSize_32f_C3R	ippiFilterMaxBorderGetBufferSize with dataType=32f and numChannels=3
ippiFilterMaxGetBufferSize_32f_C4R	ippiFilterMaxBorderGetBufferSize with dataType=32f and numChannels=4
ippiFilterMaxGetBufferSize_8u_C1R	ippiFilterMaxBorderGetBufferSize with dataType=8u and numChannels=1
ippiFilterMaxGetBufferSize_8u_C3R	ippiFilterMaxBorderGetBufferSize with dataType=8u and numChannels=3
ippiFilterMaxGetBufferSize_8u_C4R	ippiFilterMaxBorderGetBufferSize with dataType=8u and numChannels=4
ippiFilterMinBorderReplicate_32f_C1R	ippiFilterMinBorder_32f_C1R
ippiFilterMinBorderReplicate_32f_C3R	ippiFilterMinBorder_32f_C3R
ippiFilterMinBorderReplicate_32f_C4R	ippiFilterMinBorder_32f_C4R
ippiFilterMinBorderReplicate_8u_C1R	ippiFilterMinBorder_8u_C1R
ippiFilterMinBorderReplicate_8u_C3R	ippiFilterMinBorder_8u_C3R
ippiFilterMinBorderReplicate_8u_C4R	ippiFilterMinBorder_8u_C4R
ippiFilterMinGetBufferSize_32f_C1R	ippiFilterMinBorderGetBufferSize with dataType=32f and numChannels=1
ippiFilterMinGetBufferSize_32f_C3R	ippiFilterMinBorderGetBufferSize with dataType=32f and numChannels=3
ippiFilterMinGetBufferSize_32f_C4R	ippiFilterMinBorderGetBufferSize with dataType=32f and numChannels=4
ippiFilterMinGetBufferSize_8u_C1R	ippiFilterMinBorderGetBufferSize with dataType=8u and numChannels=1
ippiFilterMinGetBufferSize_8u_C3R	ippiFilterMinBorderGetBufferSize with dataType=8u and numChannels=3
ippiFilterMinGetBufferSize_8u_C4R	ippiFilterMinBorderGetBufferSize with dataType=8u and numChannels=4

ippiFilterScharrHorizBorder_32f_C1R	ippiFilterScharrHorizMaskBorder_32f_C1R from the ippIP domain
ippiFilterScharrHorizBorder_8u16s_C1R	ippiFilterScharrHorizMaskBorderGetBufferSize from the ippIP domain
ippiFilterScharrHorizBorder_8u8s_C1R	8s data type is not supported anymore - use another data type
ippiFilterScharrHorizGetBufferSize_32f_C1R	ippiFilterScharrHorizMaskBorderGetBufferSize from the ippIP domain
ippiFilterScharrHorizGetBufferSize_8u16s_C1R	ippiFilterScharrHorizMaskBorderGetBufferSize from the ippIP domain
ippiFilterScharrHorizGetBufferSize_8u8s_C1R	8s data type is not supported anymore - use another data type
ippiFilterScharrVertBorder_32f_C1R	ippiFilterScharrVertMaskBorder_32f_C1R from the ippIP domain
ippiFilterScharrVertBorder_8u16s_C1R	ippiFilterScharrVertMaskBorderGetBufferSize from the ippIP domain
ippiFilterScharrVertBorder_8u8s_C1R	8s data type is not supported anymore - use another data type
ippiFilterScharrVertGetBufferSize_32f_C1R	ippiFilterScharrVertMaskBorderGetBufferSize from the ippIP domain
ippiFilterScharrVertGetBufferSize_8u16s_C1R	ippiFilterScharrVertMaskBorderGetBufferSize from the ippIP domain
ippiFilterScharrVertGetBufferSize_8u8s_C1R	8s data type is not supported anymore - use another data type
ippiFilterSobelCrossBorder_8u8s_C1R	8s data type is not supported anymore - use another data type
ippiFilterSobelCrossGetBufferSize_8u8s_C1R	8s data type is not supported anymore - use another data type
ippiFilterSobelHorizBorder_8u8s_C1R	8s data type is not supported anymore - use another data type
ippiFilterSobelHorizGetBufferSize_8u8s_C1R	8s data type is not supported anymore - use another data type
ippiFilterSobelHorizSecondBorder_32f_C1R	ippiFilterSobelHorizSecondBorder_32f_C1R from the ippIP domain
ippiFilterSobelHorizSecondBorder_8u16s_C1R	ippiFilterSobelHorizSecondBorderGetBufferSize from the ippIP domain
ippiFilterSobelHorizSecondBorder_8u8s_C1R	8s data type is not supported anymore - use another data type

ippiFilterSobelHorizSecondGetBufferSize_32f_C1R	ippiFilterSobelHorizSecondBorderGetBufferSize from the ippIP domain
ippiFilterSobelHorizSecondGetBufferSize_8u16s_C1R	ippiFilterSobelHorizSecondBorderGetBufferSize
ippiFilterSobelHorizSecondGetBufferSize_8u8s_C1R	8s data type is not supported anymore - use another data type
ippiFilterSobelNegVertBorder_32f_C1R	ippiFilterSobelNegVertBorder_32f_C1R from the ippIP domain
ippiFilterSobelNegVertBorder_8u16s_C1R	ippiFilterSobelNegVertBorderGetBufferSize from the ippIP domain
ippiFilterSobelNegVertBorder_8u8s_C1R	8s data type is not supported anymore - use another data type
ippiFilterSobelNegVertGetBufferSize_32f_C1R	ippiFilterSobelNegVertBorderGetBufferSize from the ippIP domain
ippiFilterSobelNegVertGetBufferSize_8u16s_C1R	ippiFilterSobelNegVertBorderGetBufferSize from the ippIP domain
ippiFilterSobelNegVertGetBufferSize_8u8s_C1R	8s data type is not supported anymore - use another data type
ippiFilterSobelVertBorder_8u8s_C1R	8s data type is not supported anymore - use another data type
ippiFilterSobelVertGetBufferSize_8u8s_C1R	8s data type is not supported anymore - use another data type
ippiFilterSobelVertSecondBorder_32f_C1R	ippiFilterSobelVertSecondBorder_32f_C1R from the ippIP domain
ippiFilterSobelVertSecondBorder_8u16s_C1R	ippiFilterSobelVertSecondBorderGetBufferSize from the ippIP domain
ippiFilterSobelVertSecondBorder_8u8s_C1R	8s data type is not supported anymore - use another data type
ippiFilterSobelVertSecondGetBufferSize_32f_C1R	ippiFilterSobelVertSecondBorderGetBufferSize from the ippIP domain
ippiFilterSobelVertSecondGetBufferSize_8u16s_C1R	ippiFilterSobelVertSecondBorderGetBufferSize
ippiFilterSobelVertSecondGetBufferSize_8u8s_C1R	8s data type is not supported anymore - use another data type
ippiForegroundGaussianFree_8u_C1R	N/A
ippiForegroundGaussianFree_8u_C3R	N/A
ippiForegroundGaussianInitAlloc_8u_C1R	N/A
ippiForegroundGaussianInitAlloc_8u_C3R	N/A

ippiForegroundGaussian_8u_C1R	N/A
ippiForegroundGaussian_8u_C3R	N/A
ippiForegroundHistogramFree_8u_C1R	N/A
ippiForegroundHistogramFree_8u_C3R	N/A
ippiForegroundHistogramInitAlloc_8u_C1R	N/A
ippiForegroundHistogramInitAlloc_8u_C3R	N/A
ippiForegroundHistogramUpdate_8u_C1R	N/A
ippiForegroundHistogramUpdate_8u_C3R	N/A
ippiForegroundHistogram_8u_C1R	N/A
ippiForegroundHistogram_8u_C3R	N/A
ippiHaarClassifierFree_32f	ippFree
ippiHaarClassifierFree_32s	ippFree
ippiHaarClassifierInitAlloc_32f	ippiHaarClassifierGetSize + ippMalloc + ippiHaarClassifierInit_32f
ippiHaarClassifierInitAlloc_32s	ippiHaarClassifierGetSize + ippMalloc + ippiHaarClassifierInit_32s
ippiInpaintFree_8u_C1R	ippFree
ippiInpaintFree_8u_C3R	ippFree
ippiInpaintInitAlloc_8u_C1R	ippiInpaintGetSize + ippMalloc + ippiInpaintInit_8u_C1R
ippiInpaintInitAlloc_8u_C3R	ippiInpaintGetSize + ippMalloc + ippiInpaintInit_8u_C3R
ippiMean_8s_C1MR	8s data type is not supported anymore - use another data type
ippiMean_8s_C3CMR	8s data type is not supported anymore - use another data type
ippiMean_StdDev_8s_C1MR	8s data type is not supported anymore - use another data type
ippiMean_StdDev_8s_C1R	8s data type is not supported anymore - use another data type
ippiMean_StdDev_8s_C3CMR	8s data type is not supported anymore - use another data type
ippiMean_StdDev_8s_C3CR	8s data type is not supported anymore - use another data type
ippiMinMaxIndx_8s_C1MR	8s data type is not supported anymore - use another data type

ippiMinMaxIndx_8s_C1R	8s data type is not supported anymore - use another data type
ippiMinMaxIndx_8s_C3CMR	8s data type is not supported anymore - use another data type
ippiMinMaxIndx_8s_C3CR	8s data type is not supported anymore - use another data type
ippiMorphAdvFree	ippFree
ippiMorphAdvInitAlloc_32f_C1R	ippiMorphAdvGetSize_32f_C1R + ippMalloc + ippiMorphAdvInit_32f_C1R
ippiMorphAdvInitAlloc_32f_C3R	ippiMorphAdvGetSize_32f_C3R + ippMalloc + ippiMorphAdvInit_32f_C3R
ippiMorphAdvInitAlloc_32f_C4R	ippiMorphAdvGetSize_32f_C4R + ippMalloc + ippiMorphAdvInit_32f_C4R
ippiMorphAdvInitAlloc_8u_C1R	ippiMorphAdvGetSize_8u_C1R + ippMalloc + ippiMorphAdvInit_8u_C1R
ippiMorphAdvInitAlloc_8u_C3R	ippiMorphAdvGetSize_8u_C3R + ippMalloc + ippiMorphAdvInit_8u_C3R
ippiMorphAdvInitAlloc_8u_C4R	ippiMorphAdvGetSize_8u_C4R + ippMalloc + ippiMorphAdvInit_8u_C4R
ippiMorphGrayFree_32f_C1R	ippFree
ippiMorphGrayFree_8u_C1R	ippFree
ippiMorphGrayInitAlloc_32f_C1R	ippiMorphGrayGetSize_32f_C1R + ippMalloc + ippiMorphGrayInit_32f_C1R
ippiMorphGrayInitAlloc_8u_C1R	ippiMorphGrayGetSize_8u_C1R + ippMalloc + ippiMorphGrayInit_8u_C1R
ippiMorphReconstructGetBufferSize_16u_C1	Use new ippiMorphReconstructGetBufferSize
ippiMorphReconstructGetBufferSize_32f_C1	Use new ippiMorphReconstructGetBufferSize
ippiMorphReconstructGetBufferSize_64f_C1	Use new ippiMorphReconstructGetBufferSize
ippiMorphReconstructGetBufferSize_8u_C1	Use new ippiMorphReconstructGetBufferSize
ippiMorphologyFree	ippFree
ippiMorphologyGetSize_32f_C1R	ippiMorphologyBorderGetSize_32f_C1R
ippiMorphologyGetSize_32f_C3R	ippiMorphologyBorderGetSize_32f_C3R
ippiMorphologyGetSize_32f_C4R	ippiMorphologyBorderGetSize_32f_C4R
ippiMorphologyGetSize_8u_C1R	ippiMorphologyBorderGetSize_8u_C1R
ippiMorphologyGetSize_8u_C3R	ippiMorphologyBorderGetSize_8u_C3R

ippiMorphologyGetSize_8u_C4R	ippiMorphologyBorderGetSize_8u_C4R
ippiMorphologyInitAlloc_32f_C1R	ippiMorphologyBorderGetSize_32f_C1R + ippMalloc_8u + ippiMorphologyBorderInit_32f_C1R
ippiMorphologyInitAlloc_32f_C3R	ippiMorphologyBorderGetSize_32f_C3R + ippMalloc_8u + ippiMorphologyBorderInit_32f_C3R
ippiMorphologyInitAlloc_32f_C4R	ippiMorphologyBorderGetSize_32f_C4R + ippMalloc_8u + ippiMorphologyBorderInit_32f_C4R
ippiMorphologyInitAlloc_8u_C1R	ippiMorphologyBorderGetSize_8u_C1R + ippMalloc_8u + ippiMorphologyBorderInit_8u_C1R
ippiMorphologyInitAlloc_8u_C3R	ippiMorphologyBorderGetSize_8u_C3R + ippMalloc_8u + ippiMorphologyBorderInit_8u_C3R
ippiMorphologyInitAlloc_8u_C4R	ippiMorphologyBorderGetSize_8u_C4R + ippMalloc_8u + ippiMorphologyBorderInit_8u_C4R
ippiMorphologyInit_32f_C1R	ippiMorphologyBorderInit_32f_C1R
ippiMorphologyInit_32f_C3R	ippiMorphologyBorderInit_32f_C3R
ippiMorphologyInit_32f_C4R	ippiMorphologyBorderInit_32f_C4R
ippiMorphologyInit_8u_C1R	ippiMorphologyBorderInit_8u_C1R
ippiMorphologyInit_8u_C3R	ippiMorphologyBorderInit_8u_C3R
ippiMorphologyInit_8u_C4R	ippiMorphologyBorderInit_8u_C4R
ippiNormDiff_Inf_8s_C1MR	8s data type is not supported anymore - use another data type
ippiNormDiff_Inf_8s_C3CMR	8s data type is not supported anymore - use another data type
ippiNormDiff_L1_8s_C1MR	8s data type is not supported anymore - use another data type
ippiNormDiff_L1_8s_C3CMR	8s data type is not supported anymore - use another data type
ippiNormDiff_L2_8s_C1MR	8s data type is not supported anymore - use another data type
ippiNormDiff_L2_8s_C3CMR	8s data type is not supported anymore - use another data type
ippiNormRel_Inf_8s_C1MR	8s data type is not supported anymore - use another data type

ippiNormRel_Inf_8s_C3CMR	8s data type is not supported anymore - use another data type
ippiNormRel_L1_8s_C1MR	8s data type is not supported anymore - use another data type
ippiNormRel_L1_8s_C3CMR	8s data type is not supported anymore - use another data type
ippiNormRel_L2_8s_C1MR	8s data type is not supported anymore - use another data type
ippiNormRel_L2_8s_C3CMR	8s data type is not supported anymore - use another data type
ippiNorm_Inf_8s_C1MR	8s data type is not supported anymore - use another data type
ippiNorm_Inf_8s_C3CMR	8s data type is not supported anymore - use another data type
ippiNorm_L1_8s_C1MR	8s data type is not supported anymore - use another data type
ippiNorm_L1_8s_C3CMR	8s data type is not supported anymore - use another data type
ippiNorm_L2_8s_C1MR	8s data type is not supported anymore - use another data type
ippiNorm_L2_8s_C3CMR	8s data type is not supported anymore - use another data type
ippiOpticalFlowPyrLKFree_16u_C1R	ippFree
ippiOpticalFlowPyrLKFree_32f_C1R	ippFree
ippiOpticalFlowPyrLKFree_8u_C1R	ippFree
ippiOpticalFlowPyrLKInitAlloc_16u_C1R	ippiOpticalFlowPyrLKGetSize (with 16u dataType)+ ippMalloc + ippiOpticalFlowPyrLKInit_16u_C1R
ippiOpticalFlowPyrLKInitAlloc_32f_C1R	ippiOpticalFlowPyrLKGetSize (with 32f dataType)+ ippMalloc + ippiOpticalFlowPyrLKInit_32f_C1R
ippiOpticalFlowPyrLKInitAlloc_8u_C1R	ippiOpticalFlowPyrLKGetSize (with 8u dataType)+ ippMalloc + ippiOpticalFlowPyrLKInit_8u_C1R
ippiPyrDownGetBufSize_Gauss5x5	ippiPyramidLayerDownGetSize_8u_C1R
ippiPyrDown_Gauss5x5_32f_C1R	ippiPyramidLayerDown_32f_C1R
ippiPyrDown_Gauss5x5_32f_C3R	ippiPyramidLayerDown_32f_C3R
ippiPyrDown_Gauss5x5_8s_C1R	8s data type is not supported anymore - use another data type

ippiPyrDown_Gauss5x5_8s_C3R	8s data type is not supported anymore - use another data type
ippiPyrDown_Gauss5x5_8u_C1R	ippiPyramidLayerDown_8u_C1R
ippiPyrDown_Gauss5x5_8u_C3R	ippiPyramidLayerDown_8u_C3R
ippiPyrUpGetBufSize_Gauss5x5	ippiPyramidLayerUpGetSize_8u_C1R
ippiPyrUp_Gauss5x5_32f_C1R	ippiPyramidLayerUp_32f_C1R
ippiPyrUp_Gauss5x5_32f_C3R	ippiPyramidLayerUp_32f_C3R
ippiPyrUp_Gauss5x5_8s_C1R	8s data type is not supported anymore - use another data type
ippiPyrUp_Gauss5x5_8s_C3R	8s data type is not supported anymore - use another data type
ippiPyrUp_Gauss5x5_8u_C1R	ippiPyramidLayerUp_8u_C1R
ippiPyrUp_Gauss5x5_8u_C3R	ippiPyramidLayerUp_8u_C3R
ippiPyramidFree	ippFree
ippiPyramidInitAlloc	ippiPyramidGetSize + ippiPyramidInit
ippiPyramidLayerDownFree_16u_C1R	ippFree
ippiPyramidLayerDownFree_16u_C3R	ippFree
ippiPyramidLayerDownFree_32f_C1R	ippFree
ippiPyramidLayerDownFree_32f_C3R	ippFree
ippiPyramidLayerDownFree_8u_C1R	ippFree
ippiPyramidLayerDownFree_8u_C3R	ippFree
ippiPyramidLayerDownInitAlloc_16u_C1R	ippiPyramidLayerDownGetSize_16u_C1R + ippMalloc + ippiPyramidLayerDownInit_16u_C1R
ippiPyramidLayerDownInitAlloc_16u_C3R	ippiPyramidLayerDownGetSize_16u_C3R + ippMalloc + ippiPyramidLayerDownInit_16u_C3R
ippiPyramidLayerDownInitAlloc_32f_C1R	ippiPyramidLayerDownGetSize_32f_C1R + ippMalloc + ippiPyramidLayerDownInit_32u_C1R
ippiPyramidLayerDownInitAlloc_32f_C3R	ippiPyramidLayerDownGetSize_32f_C3R + ippMalloc + ippiPyramidLayerDownInit_32f_C3R
ippiPyramidLayerDownInitAlloc_8u_C1R	ippiPyramidLayerDownGetSize_8u_C1R + ippMalloc + ippiPyramidLayerDownInit_8u_C1R

ippiPyramidLayerDownInitAlloc_8u_C3R	ippiPyramidLayerDownGetSize_8u_C3R + ippMalloc + ippiPyramidLayerDownInit_8u_C3R
ippiPyramidLayerUpFree_16u_C1R	ippFree
ippiPyramidLayerUpFree_16u_C3R	ippFree
ippiPyramidLayerUpFree_32f_C1R	ippFree
ippiPyramidLayerUpFree_32f_C3R	ippFree
ippiPyramidLayerUpFree_8u_C1R	ippFree
ippiPyramidLayerUpFree_8u_C3R	ippFree
ippiPyramidLayerUpInitAlloc_16u_C1R	ippiPyramidLayerUpGetSize_16u_C1R + ippMalloc + ippiPyramidLayerUpInit_16u_C1R
ippiPyramidLayerUpInitAlloc_16u_C3R	ippiPyramidLayerUpGetSize_16u_C3R + ippMalloc + ippiPyramidLayerUpInit_16u_C3R
ippiPyramidLayerUpInitAlloc_32f_C1R	ippiPyramidLayerUpGetSize_32f_C1R + ippMalloc + ippiPyramidLayerUpInit_32u_C1R
ippiPyramidLayerUpInitAlloc_32f_C3R	ippiPyramidLayerUpGetSize_32f_C3R + ippMalloc + ippiPyramidLayerUpInit_32f_C3R
ippiPyramidLayerUpInitAlloc_8u_C1R	ippiPyramidLayerUpGetSize_8u_C1R + ippMalloc + ippiPyramidLayerUpInit_8u_C1R
ippiPyramidLayerUpInitAlloc_8u_C3R	ippiPyramidLayerUpGetSize_8u_C3R + ippMalloc + ippiPyramidLayerUpInit_8u_C3R
ippiSRHNCalcResidual_PSF2x2_16u32f_C1R	N/A
ippiSRHNCalcResidual_PSF2x2_8u32f_C1R	N/A
ippiSRHNCalcResidual_PSF3x3_16u32f_C1R	N/A
ippiSRHNCalcResidual_PSF3x3_8u32f_C1R	N/A
ippiSRHNFree_PSF2x2	N/A
ippiSRHNFree_PSF3x3	N/A
ippiSRHNInitAlloc_PSF2x2	N/A
ippiSRHNInitAlloc_PSF3x3	N/A
ippiSRHNUpdateGradient_PSF2x2_32f_C1R	N/A
ippiSRHNUpdateGradient_PSF3x3_32f_C1R	N/A

ippiTiltedHaarClassifierInitAlloc_32f	ippiHaarClassifierGetSize + ippMalloc + ippiTiltedHaarClassifierInit_32f
ippiTiltedHaarClassifierInitAlloc_32s	ippiHaarClassifierGetSize + ippMalloc + ippiTiltedHaarClassifierInit_32s
ippibFastArctan_32f	ippsAtan_32f_A11

ippi.h:

Removed from 9.0	Substitution or Workaround
ippiAddC_16sc_AC4IRSfs	N/A
ippiAddC_16sc_AC4RSfs	N/A
ippiAddC_16sc_C1IRSfs	N/A
ippiAddC_16sc_C1RSfs	N/A
ippiAddC_16sc_C3IRSfs	N/A
ippiAddC_16sc_C3RSfs	N/A
ippiAddC_32fc_AC4IR	N/A
ippiAddC_32fc_AC4R	N/A
ippiAddC_32fc_C1IR	N/A
ippiAddC_32fc_C1R	N/A
ippiAddC_32fc_C3IR	N/A
ippiAddC_32fc_C3R	N/A
ippiAddC_32sc_AC4IRSfs	N/A
ippiAddC_32sc_AC4RSfs	N/A
ippiAddC_32sc_C1IRSfs	N/A
ippiAddC_32sc_C1RSfs	N/A
ippiAddC_32sc_C3IRSfs	N/A
ippiAddC_32sc_C3RSfs	N/A
ippiAddRandGauss_Direct_16s_AC4IR	ippiAddRandGauss_16s_AC4IR
ippiAddRandGauss_Direct_16s_C1IR	ippiAddRandGauss_16s_C1IR
ippiAddRandGauss_Direct_16s_C3IR	ippiAddRandGauss_16s_C3IR
ippiAddRandGauss_Direct_16s_C4IR	ippiAddRandGauss_16s_C4IR
ippiAddRandGauss_Direct_16u_AC4IR	ippiAddRandGauss_16u_AC4IR
ippiAddRandGauss_Direct_16u_C1IR	ippiAddRandGauss_16u_C1IR

ippiAddRandGauss_Direct_16u_C3IR	ippiAddRandGauss_16u_C3IR
ippiAddRandGauss_Direct_16u_C4IR	ippiAddRandGauss_16u_C4IR
ippiAddRandGauss_Direct_32f_AC4IR	ippiAddRandGauss_32f_AC4IR
ippiAddRandGauss_Direct_32f_C1IR	ippiAddRandGauss_32f_C1IR
ippiAddRandGauss_Direct_32f_C3IR	ippiAddRandGauss_32f_C3IR
ippiAddRandGauss_Direct_32f_C4IR	ippiAddRandGauss_32f_C4IR
ippiAddRandGauss_Direct_8u_AC4IR	ippiAddRandGauss_8u_AC4IR
ippiAddRandGauss_Direct_8u_C1IR	ippiAddRandGauss_8u_C1IR
ippiAddRandGauss_Direct_8u_C3IR	ippiAddRandGauss_8u_C3IR
ippiAddRandGauss_Direct_8u_C4IR	ippiAddRandGauss_8u_C4IR
ippiAddRandUniform_Direct_16s_AC4IR	ippiAddRandUniform_16s_AC4IR
ippiAddRandUniform_Direct_16s_C1IR	ippiAddRandUniform_16s_C1IR
ippiAddRandUniform_Direct_16s_C3IR	ippiAddRandUniform_16s_C3IR
ippiAddRandUniform_Direct_16s_C4IR	ippiAddRandUniform_16s_C4IR
ippiAddRandUniform_Direct_16u_AC4IR	ippiAddRandUniform_16u_AC4IR
ippiAddRandUniform_Direct_16u_C1IR	ippiAddRandUniform_16u_C1IR
ippiAddRandUniform_Direct_16u_C3IR	ippiAddRandUniform_16u_C3IR
ippiAddRandUniform_Direct_16u_C4IR	ippiAddRandUniform_16u_C4IR
ippiAddRandUniform_Direct_32f_AC4IR	ippiAddRandUniform_32f_AC4IR
ippiAddRandUniform_Direct_32f_C1IR	ippiAddRandUniform_32f_C1IR
ippiAddRandUniform_Direct_32f_C3IR	ippiAddRandUniform_32f_C3IR
ippiAddRandUniform_Direct_32f_C4IR	ippiAddRandUniform_32f_C4IR
ippiAddRandUniform_Direct_8u_AC4IR	ippiAddRandUniform_8u_AC4IR
ippiAddRandUniform_Direct_8u_C1IR	ippiAddRandUniform_8u_C1IR
ippiAddRandUniform_Direct_8u_C3IR	ippiAddRandUniform_8u_C3IR
ippiAddRandUniform_Direct_8u_C4IR	ippiAddRandUniform_8u_C4IR
ippiAddRotateShift	N/A
ippiAdd_16sc_AC4IRSfs	N/A
ippiAdd_16sc_AC4RSfs	N/A
ippiAdd_16sc_C1IRSfs	N/A
ippiAdd_16sc_C1RSfs	N/A

ippiAdd_16sc_C3IRSfs	N/A
ippiAdd_16sc_C3RSfs	N/A
ippiAdd_32fc_AC4IR	N/A
ippiAdd_32fc_AC4R	N/A
ippiAdd_32fc_C1IR	N/A
ippiAdd_32fc_C1R	N/A
ippiAdd_32fc_C3IR	N/A
ippiAdd_32fc_C3R	N/A
ippiAdd_32sc_AC4IRSfs	N/A
ippiAdd_32sc_AC4RSfs	N/A
ippiAdd_32sc_C1IRSfs	N/A
ippiAdd_32sc_C1RSfs	N/A
ippiAdd_32sc_C3IRSfs	N/A
ippiAdd_32sc_C3RSfs	N/A
ippiAlphaCompC_8s_C1R	ippiConvert_8s16s + ippiAlphaCompC_16s
ippiAlphaComp_8s_AC1R	ippiConvert_8s16s + ippiAlphaComp_16s
ippiAlphaComp_8s_AC4R	ippiConvert_8s16s + ippiAlphaComp_16s
ippiComplement_32s_C1IR	N/A
ippiConvFull_16s_AC4R	ippiConv_16s_C4R (with algType=ippiROIFull)
ippiConvFull_16s_C1R	ippiConv_16s_C1R (with algType=ippiROIFull)
ippiConvFull_16s_C3R	ippiConv_16s_C3R (with algType=ippiROIFull)
ippiConvFull_32f_AC4R	ippiConv_32f_C4R (with algType=ippiROIFull)
ippiConvFull_32f_C1R	ippiConv_32f_C1R (with algType=ippiROIFull)
ippiConvFull_32f_C3R	ippiConv_32f_C3R (with algType=ippiROIFull)
ippiConvFull_8u_AC4R	ippiConv_8u_C4R (with algType=ippiROIFull)
ippiConvFull_8u_C1R	ippiConv_8u_C1R (with algType=ippiROIFull)
ippiConvFull_8u_C3R	ippiConv_8u_C3R (with algType=ippiROIFull)
ippiConvValid_16s_AC4R	ippiConv16s_C4R (with algType=ippiROIValid)
ippiConvValid_16s_C1R	ippiConv16s_C1R (with algType=ippiROIValid)
ippiConvValid_16s_C3R	ippiConv16s_C3R (with algType=ippiROIValid)
ippiConvValid_32f_AC4R	ippiConv32f_C4R (with algType=ippiROIValid)

ippiConvValid_32f_C1R	ippiConv32f_C1R (with algType=ippiROIValid)
ippiConvValid_32f_C3R	ippiConv32f_C3R (with algType=ippiROIValid)
ippiConvValid_8u_AC4R	ippiConv8u_C4R (with algType=ippiROIValid)
ippiConvValid_8u_C1R	ippiConv8u_C1R (with algType=ippiROIValid)
ippiConvValid_8u_C3R	ippiConv8u_C3R (with algType=ippiROIValid)
ippiConvert_1u8u_C1R	ippiBinToGray_1u8u_C1R
ippiCplxExtendToPack_16sc16s_C1R	ippiConvert_16s32f + ippiCplxExtendToPack_32f
ippiCplxExtendToPack_16sc16s_C3R	ippiConvert_16s32f + ippiCplxExtendToPack_32f
ippiCplxExtendToPack_32sc32s_C1R	ippiConvert_32s32f + ippCplxExtendToPack_32f
ippiCplxExtendToPack_32sc32s_C3R	ippiConvert_32s32f + ippCplxExtendToPack_32f
ippiCrossCorrFull_NormLevel_16u32f_AC4R	ippiCopy_16u_C4C1R + ippiCrossCorrNorm_16u32f_C1R + algType = ippiROIFull | ippiNormCoefficient
ippiCrossCorrFull_NormLevel_16u32f_C1R	ippiCrossCorrNorm_16u32f_C1R + algType = ippiROIFull | ippiNormCoefficient
ippiCrossCorrFull_NormLevel_16u32f_C3R	ippiCopy_16u_C3C1R + ippiCrossCorrNorm_16u32f_C1R + algType = ippiROIFull | ippiNormCoefficient
ippiCrossCorrFull_NormLevel_16u32f_C4R	ippiCopy_16u_C4C1R + ippiCrossCorrNorm_16u32f_C1R + algType = ippiROIFull | ippiNormCoefficient
ippiCrossCorrFull_NormLevel_32f_AC4R	ippiCopy_32f_C4C1R + ippiCrossCorrNorm_32f_C1R + algType = ippiROIFull | ippiNormCoefficient
ippiCrossCorrFull_NormLevel_32f_C1R	ippiCrossCorrNorm_32f_C1R + algType = ippiROIFull | ippiNormCoefficient
ippiCrossCorrFull_NormLevel_32f_C3R	ippiCopy_32f_C3C1R + ippiCrossCorrNorm_32f_C1R + algType = ippiROIFull | ippiNormCoefficient
ippiCrossCorrFull_NormLevel_32f_C4R	ippiCopy_32f_C4C1R + ippiCrossCorrNorm_32f_C1R + algType = ippiROIFull | ippiNormCoefficient
ippiCrossCorrFull_NormLevel_64f_C1R	N/A

ippiCrossCorrFull_NormLevel_8s32f_AC4R	ippiConvert_8s32f + ippiCopy_32f_C4C1R + ippiCrossCorrNorm_32f_C1R + algType = ippiROIFull | ippiNormCoefficient
ippiCrossCorrFull_NormLevel_8s32f_C1R	ippiConvert_8s32f + ippiCrossCorrNorm_32f_C1R + algType = ippiROIFull | ippiNormCoefficient
ippiCrossCorrFull_NormLevel_8s32f_C3R	ippiConvert_8s32f + ippiCopy_32f_C3C1R + ippiCrossCorrNorm_32f_C1R + algType = ippiROIFull | ippiNormCoefficient
ippiCrossCorrFull_NormLevel_8s32f_C4R	ippiConvert_8s32f + ippiCopy_32f_C4C1R + ippiCrossCorrNorm_32f_C1R + algType = ippiROIFull | ippiNormCoefficient
ippiCrossCorrFull_NormLevel_8u32f_AC4R	ippiCopy_8u_C4C1R + ippiCrossCorrNorm_8u32f_C1R + algType = ippiROIFull | ippiNormCoefficient
ippiCrossCorrFull_NormLevel_8u32f_C1R	ippiCrossCorrNorm_8u32f_C1R + algType = ippiROIFull | ippiNormCoefficient
ippiCrossCorrFull_NormLevel_8u32f_C3R	ippiCopy_8u_C3C1R + ippiCrossCorrNorm_8u32f_C1R + algType = ippiROIFull | ippiNormCoefficient
ippiCrossCorrFull_NormLevel_8u32f_C4R	ippiCopy_8u_C4C1R + ippiCrossCorrNorm_8u32f_C1R + algType = ippiROIFull | ippiNormCoefficient
ippiCrossCorrFull_NormLevel_8u_AC4RSfs	ippiCopy_8u_C4C1R + ippiCrossCorrNorm_8u_C1RSfs + algType = ippiROIFull | ippiNormCoefficient
ippiCrossCorrFull_NormLevel_8u_C1RSfs	ippiCrossCorrNorm_8u_C1RSfs + algType = ippiROIFull | ippiNormCoefficient
ippiCrossCorrFull_NormLevel_8u_C3RSfs	ippiCopy_8u_C3C1R + ippiCrossCorrNorm_8u_C1RSfs + algType = ippiROIFull | ippiNormCoefficient
ippiCrossCorrFull_NormLevel_8u_C4RSfs	ippiCopy_8u_C4C1R + ippiCrossCorrNorm_8u_C1RSfs + algType = ippiROIFull | ippiNormCoefficient
ippiCrossCorrFull_Norm_16u32f_AC4R	ippiCopy_16u_C4C1R + ippiCrossCorrNorm_16u32f_C1R + algType = ippiROIFull | ippiNorm
ippiCrossCorrFull_Norm_16u32f_C1R	ippiCrossCorrNorm_16u32f_C1R + algType = ippiROIFull | ippiNorm

ippiCrossCorrFull_Norm_16u32f_C3R	ippiCopy_16u_C3C1R + ippiCrossCorrNorm_16u32f_C1R + algType = ippiROIFull | ippiNorm
ippiCrossCorrFull_Norm_16u32f_C4R	ippiCopy_16u_C4C1R + ippiCrossCorrNorm_16u32f_C1R + algType = ippiROIFull | ippiNorm
ippiCrossCorrFull_Norm_32f_AC4R	ippiCopy_32f_C4C1R + ippiCrossCorrNorm_32f_C1R + algType = ippiROIFull | ippiNorm
ippiCrossCorrFull_Norm_32f_C1R	ippiCrossCorrNorm_32f_C1R + algType = ippiROIFull | ippiNorm
ippiCrossCorrFull_Norm_32f_C3R	ippiCopy_32f_C3C1R + ippiCrossCorrNorm_32f_C1R + algType = ippiROIFull | ippiNorm
ippiCrossCorrFull_Norm_32f_C4R	ippiCopy_32f_C4C1R + ippiCrossCorrNorm_32f_C1R + algType = ippiROIFull | ippiNorm
ippiCrossCorrFull_Norm_8s32f_AC4R	ippiConvert_8s32f + ippiCopy_32f_C4C1R + ippiCrossCorrNorm_32f_C1R + algType = ippiROIFull | ippiNorm
ippiCrossCorrFull_Norm_8s32f_C1R	ippiConvert_8s32f + ippiCrossCorrNorm_32f_C1R + algType = ippiROIFull | ippiNorm
ippiCrossCorrFull_Norm_8s32f_C3R	ippiConvert_8s32f + ippiCopy_32f_C3C1R + ippiCrossCorrNorm_32f_C1R + algType = ippiROIFull | ippiNorm
ippiCrossCorrFull_Norm_8s32f_C4R	ippiConvert_8s32f + ippiCopy_32f_C4C1R + ippiCrossCorrNorm_32f_C1R + algType = ippiROIFull | ippiNorm
ippiCrossCorrFull_Norm_8u32f_AC4R	ippiCopy_8u_C4C1R + ippiCrossCorrNorm_8u32f_C1R + algType = ippiROIFull | ippiNorm
ippiCrossCorrFull_Norm_8u32f_C1R	ippiCrossCorrNorm_8u32f_C1R + algType = ippiROIFull | ippiNorm
ippiCrossCorrFull_Norm_8u32f_C3R	ippiCopy_8u_C3C1R + ippiCrossCorrNorm_8u32f_C1R + algType = ippiROIFull | ippiNorm
ippiCrossCorrFull_Norm_8u32f_C4R	ippiCopy_8u_C4C1R + ippiCrossCorrNorm_8u32f_C1R + algType = ippiROIFull | ippiNorm

ippiCrossCorrFull_Norm_8u_AC4RSfs	ippiCopy_8u_C4C1R + ippiCrossCorrNorm_8u_C1RSfs + algType = ippiROIFull | ippiNorm
ippiCrossCorrFull_Norm_8u_C1RSfs	ippiCrossCorrNorm_8u_C1RSfs + algType = ippiROIFull | ippiNorm
ippiCrossCorrFull_Norm_8u_C3RSfs	ippiCopy_8u_C3C1R + ippiCrossCorrNorm_8u_C1RSfs + algType = ippiROIFull | ippiNorm
ippiCrossCorrFull_Norm_8u_C4RSfs	ippiCopy_8u_C4C1R + ippiCrossCorrNorm_8u_C1RSfs + algType = ippiROIFull | ippiNorm
ippiCrossCorrSame_NormLevel_16u32f_AC4R	ippiCopy_16u_C4C1R + ippiCrossCorrNorm_16u32f_C1R + algType = ippiROISame | ippiNormCoefficient
ippiCrossCorrSame_NormLevel_16u32f_C1R	ippiCrossCorrNorm_16u32f_C1R + algType = ippiROISame | ippiNormCoefficient
ippiCrossCorrSame_NormLevel_16u32f_C3R	ippiCopy_16u_C3C1R + ippiCrossCorrNorm_16u32f_C1R + algType = ippiROISame | ippiNormCoefficient
ippiCrossCorrSame_NormLevel_16u32f_C4R	ippiCopy_16u_C4C1R + ippiCrossCorrNorm_16u32f_C1R + algType = ippiROISame | ippiNormCoefficient
ippiCrossCorrSame_NormLevel_32f_AC4R	ippiCopy_32f_C4C1R + ippiCrossCorrNorm_32f_C1R + algType = ippiROISame | ippiNormCoefficient
ippiCrossCorrSame_NormLevel_32f_C1R	ippiCrossCorrNorm_32f_C1R + algType = ippiROISame | ippiNormCoefficient
ippiCrossCorrSame_NormLevel_32f_C3R	ippiCopy_32f_C3C1R + ippiCrossCorrNorm_32f_C1R + algType = ippiROISame | ippiNormCoefficient
ippiCrossCorrSame_NormLevel_32f_C4R	ippiCopy_32f_C4C1R + ippiCrossCorrNorm_32f_C1R + algType = ippiROISame | ippiNormCoefficient
ippiCrossCorrSame_NormLevel_8s32f_AC4R	ippiConvert_8s32f + ippiCopy_32f_C4C1R + ippiCrossCorrNorm_32f_C1R + algType = ippiROISame | ippiNormCoefficient
ippiCrossCorrSame_NormLevel_8s32f_C1R	ippiConvert_8s32f + ippiCrossCorrNorm_32f_C1R + algType = ippiROISame | ippiNormCoefficient

ippiCrossCorrSame_NormLevel_8s32f_C3R	ippiConvert_8s32f + ippiCopy_32f_C3C1R + ippiCrossCorrNorm_32f_C1R + algType = ippiROISame | ippiNormCoefficient
ippiCrossCorrSame_NormLevel_8s32f_C4R	ippiConvert_8s32f + ippiCopy_32f_C4C1R + ippiCrossCorrNorm_32f_C1R + algType = ippiROISame | ippiNormCoefficient
ippiCrossCorrSame_NormLevel_8u32f_AC4R	ippiCopy_8u_C4C1R + ippiCrossCorrNorm_8u32f_C1R + algType = ippiROISame | ippiNormCoefficient
ippiCrossCorrSame_NormLevel_8u32f_C1R	ippiCrossCorrNorm_8u32f_C1R + algType = ippiROISame | ippiNormCoefficient
ippiCrossCorrSame_NormLevel_8u32f_C3R	ippiCopy_8u_C3C1R + ippiCrossCorrNorm_8u32f_C1R + algType = ippiROISame | ippiNormCoefficient
ippiCrossCorrSame_NormLevel_8u32f_C4R	ippiCopy_8u_C4C1R + ippiCrossCorrNorm_8u32f_C1R + algType = ippiROISame | ippiNormCoefficient
ippiCrossCorrSame_NormLevel_8u_AC4RSfs	ippiCopy_8u_C4C1R + ippiCrossCorrNorm_8u_C1RSfs + algType = ippiROISame | ippiNormCoefficient
ippiCrossCorrSame_NormLevel_8u_C1RSfs	ippiCrossCorrNorm_8u_C1RSfs + algType = ippiROISame | ippiNormCoefficient
ippiCrossCorrSame_NormLevel_8u_C3RSfs	ippiCopy_8u_C3C1R + ippiCrossCorrNorm_8u_C1RSfs + algType = ippiROISame | ippiNormCoefficient
ippiCrossCorrSame_NormLevel_8u_C4RSfs	ippiCopy_8u_C4C1R + ippiCrossCorrNorm_8u_C1RSfs + algType = ippiROISame | ippiNormCoefficient
ippiCrossCorrSame_Norm_16u32f_AC4R	ippiCopy_16u_C4C1R + ippiCrossCorrNorm_16u32f_C1R + algType = ippiROISame | ippiNorm
ippiCrossCorrSame_Norm_16u32f_C1R	ippiCrossCorrNorm_16u32f_C1R + algType = ippiROISame | ippiNorm
ippiCrossCorrSame_Norm_16u32f_C3R	ippiCopy_16u_C3C1R + ippiCrossCorrNorm_16u32f_C1R + algType = ippiROISame | ippiNorm
ippiCrossCorrSame_Norm_16u32f_C4R	ippiCopy_16u_C4C1R + ippiCrossCorrNorm_16u32f_C1R + algType = ippiROISame | ippiNorm

ippiCrossCorrSame_Norm_32f_AC4R	ippiCopy_32f_C4C1R + ippiCrossCorrNorm_32f_C1R + algType = ippiROISame | ippiNorm
ippiCrossCorrSame_Norm_32f_C1R	ippiCrossCorrNorm_32f_C1R + algType = ippiROISame | ippiNorm
ippiCrossCorrSame_Norm_32f_C3R	ippiCopy_32f_C3C1R + ippiCrossCorrNorm_32f_C1R + algType = ippiROISame | ippiNorm
ippiCrossCorrSame_Norm_32f_C4R	ippiCopy_32f_C4C1R + ippiCrossCorrNorm_32f_C1R + algType = ippiROISame | ippiNorm
ippiCrossCorrSame_Norm_8s32f_AC4R	ippiConvert_8s32f + ippiCopy_32f_C4C1R + ippiCrossCorrNorm_32f_C1R + algType = ippiROISame | ippiNorm
ippiCrossCorrSame_Norm_8s32f_C1R	ippiConvert_8s32f + ippiCrossCorrNorm_32f_C1R + algType = ippiROISame | ippiNorm
ippiCrossCorrSame_Norm_8s32f_C3R	ippiConvert_8s32f + ippiCopy_32f_C3C1R + ippiCrossCorrNorm_32f_C1R + algType = ippiROISame | ippiNorm
ippiCrossCorrSame_Norm_8s32f_C4R	ippiConvert_8s32f + ippiCopy_32f_C4C1R + ippiCrossCorrNorm_32f_C1R + algType = ippiROISame | ippiNorm
ippiCrossCorrSame_Norm_8u32f_AC4R	ippiCopy_8u_C4C1R + ippiCrossCorrNorm_8u32f_C1R + algType = ippiROISame | ippiNorm
ippiCrossCorrSame_Norm_8u32f_C1R	ippiCrossCorrNorm_8u32f_C1R + algType = ippiROISame | ippiNorm
ippiCrossCorrSame_Norm_8u32f_C3R	ippiCopy_8u_C3C1R + ippiCrossCorrNorm_8u32f_C1R + algType = ippiROISame | ippiNorm
ippiCrossCorrSame_Norm_8u32f_C4R	ippiCopy_8u_C4C1R + ippiCrossCorrNorm_8u32f_C1R + algType = ippiROISame | ippiNorm
ippiCrossCorrSame_Norm_8u_AC4RSfs	ippiCopy_8u_C4C1R + ippiCrossCorrNorm_8u_C1RSfs + algType = ippiROISame | ippiNorm
ippiCrossCorrSame_Norm_8u_C1RSfs	ippiCrossCorrNorm_8u_C1RSfs + algType = ippiROISame | ippiNorm

ippiCrossCorrSame_Norm_8u_C3RSfs	ippiCopy_8u_C3C1R + ippiCrossCorrNorm_8u_C1RSfs + algType = ippiROISame | ippiNorm
ippiCrossCorrSame_Norm_8u_C4RSfs	ippiCopy_8u_C4C1R + ippiCrossCorrNorm_8u_C1RSfs + algType = ippiROISame | ippiNorm
ippiCrossCorrValid_16u32f_C1R	ippiCrossCorrNorm_16u32f_C1R + algType = ippiROIValid | ippiNormNone
ippiCrossCorrValid_32f_C1R	ippiCrossCorrNorm_32f_C1R + algType = ippiROIValid | ippiNormNone
ippiCrossCorrValid_8s32f_C1R	ippiConvert_8s32f + ippiCrossCorrNorm_32f_C1R + algType = ippiROIValid | ippiNormNone
ippiCrossCorrValid_8u32f_C1R	ippiCrossCorrNorm_8u32f_C1R + algType = ippiROIValid | ippiNormNone
ippiCrossCorrValid_NormLevel_16u32f_AC4R	ippiCopy_16u_C4C1R + ippiCrossCorrNorm_16u32f_C1R + algType = ippiROIValid | ippiNormCoefficient
ippiCrossCorrValid_NormLevel_16u32f_C1R	ippiCrossCorrNorm_16u32f_C1R + algType = ippiROIValid | ippiNormCoefficient
ippiCrossCorrValid_NormLevel_16u32f_C3R	ippiCopy_16u_C3C1R + ippiCrossCorrNorm_16u32f_C1R + algType = ippiROIValid | ippiNormCoefficient
ippiCrossCorrValid_NormLevel_16u32f_C4R	ippiCopy_16u_C4C1R + ippiCrossCorrNorm_16u32f_C1R + algType = ippiROIValid | ippiNormCoefficient
ippiCrossCorrValid_NormLevel_32f_AC4R	ippiCopy_32f_C4C1R + ippiCrossCorrNorm_32f_C1R + algType = ippiROIValid | ippiNormCoefficient
ippiCrossCorrValid_NormLevel_32f_C1R	ippiCrossCorrNorm_32f_C1R + algType = ippiROIValid | ippiNormCoefficient
ippiCrossCorrValid_NormLevel_32f_C3R	ippiCopy_32f_C3C1R + ippiCrossCorrNorm_32f_C1R + algType = ippiROIValid | ippiNormCoefficient
ippiCrossCorrValid_NormLevel_32f_C4R	ippiCopy_32f_C4C1R + ippiCrossCorrNorm_32f_C1R + algType = ippiROIValid | ippiNormCoefficient
ippiCrossCorrValid_NormLevel_64f_C1R	N/A
ippiCrossCorrValid_NormLevel_8s32f_AC4R	ippiConvert_8s32f + ippiCopy_32f_C4C1R + ippiCrossCorrNorm_32f_C1R + algType = ippiROIValid | ippiNormCoefficient

ippiCrossCorrValid_NormLevel_8s32f_C1R	ippiConvert_8s32f + ippiCrossCorrNorm_32f_C1R + algType = ippiROIValid | ippiNormCoefficient
ippiCrossCorrValid_NormLevel_8s32f_C3R	ippiConvert_8s32f + ippiCopy_32f_C3C1R + ippiCrossCorrNorm_32f_C1R + algType = ippiROIValid | ippiNormCoefficient
ippiCrossCorrValid_NormLevel_8s32f_C4R	ippiConvert_8s32f + ippiCopy_32f_C4C1R + ippiCrossCorrNorm_32f_C1R + algType = ippiROIValid | ippiNormCoefficient
ippiCrossCorrValid_NormLevel_8u32f_AC4R	ippiCopy_8u_C4C1R + ippiCrossCorrNorm_8u32f_C1R + algType = ippiROIValid | ippiNormCoefficient
ippiCrossCorrValid_NormLevel_8u32f_C1R	ippiCrossCorrNorm_8u32f_C1R + algType = ippiROIValid | ippiNormCoefficient
ippiCrossCorrValid_NormLevel_8u32f_C3R	ippiCopy_8u_C3C1R + ippiCrossCorrNorm_8u32f_C1R + algType = ippiROIValid | ippiNormCoefficient
ippiCrossCorrValid_NormLevel_8u32f_C4R	ippiCopy_8u_C4C1R + ippiCrossCorrNorm_8u32f_C1R + algType = ippiROIValid | ippiNormCoefficient
ippiCrossCorrValid_NormLevel_8u_AC4RSfs	ippiCopy_8u_C4C1R + ippiCrossCorrNorm_8u_C1RSfs + algType = ippiROIValid | ippiNormCoefficient
ippiCrossCorrValid_NormLevel_8u_C1RSfs	ippiCrossCorrNorm_8u_C1RSfs + algType = ippiROIValid | ippiNormCoefficient
ippiCrossCorrValid_NormLevel_8u_C3RSfs	ippiCopy_8u_C3C1R + ippiCrossCorrNorm_8u_C1RSfs + algType = ippiROIValid | ippiNormCoefficient
ippiCrossCorrValid_NormLevel_8u_C4RSfs	ippiCopy_8u_C4C1R + ippiCrossCorrNorm_8u_C1RSfs + algType = ippiROIValid | ippiNormCoefficient
ippiCrossCorrValid_Norm_16u32f_AC4R	ippiCopy_16u_C4C1R + ippiCrossCorrNorm_16u32f_C1R + algType = ippiROIValid | ippiNorm
ippiCrossCorrValid_Norm_16u32f_C1R	ippiCrossCorrNorm_16u32f_C1R + algType = ippiROIValid | ippiNorm
ippiCrossCorrValid_Norm_16u32f_C3R	ippiCopy_16u_C3C1R + ippiCrossCorrNorm_16u32f_C1R + algType = ippiROIValid | ippiNorm

ippiCrossCorrValid_Norm_16u32f_C4R	ippiCopy_16u_C4C1R + ippiCrossCorrNorm_16u32f_C1R + algType = ippiROIValid | ippiNorm
ippiCrossCorrValid_Norm_32f_AC4R	ippiCopy_32f_C4C1R + ippiCrossCorrNorm_32f_C1R + algType = ippiROIValid | ippiNorm
ippiCrossCorrValid_Norm_32f_C1R	ippiCrossCorrNorm_32f_C1R + algType = ippiROIValid | ippiNorm
ippiCrossCorrValid_Norm_32f_C3R	ippiCopy_32f_C3C1R + ippiCrossCorrNorm_32f_C1R + algType = ippiROIValid | ippiNorm
ippiCrossCorrValid_Norm_32f_C4R	ippiCopy_32f_C4C1R + ippiCrossCorrNorm_32f_C1R + algType = ippiROIValid | ippiNorm
ippiCrossCorrValid_Norm_8s32f_AC4R	ippiConvert_8s32f + ippiCopy_32f_C4C1R + ippiCrossCorrNorm_32f_C1R + algType = ippiROIValid | ippiNorm
ippiCrossCorrValid_Norm_8s32f_C1R	ippiConvert_8s32f + ippiCrossCorrNorm_32f_C1R + algType = ippiROIValid | ippiNorm
ippiCrossCorrValid_Norm_8s32f_C3R	ippiConvert_8s32f + ippiCopy_32f_C3C1R + ippiCrossCorrNorm_32f_C1R + algType = ippiROIValid | ippiNorm
ippiCrossCorrValid_Norm_8s32f_C4R	ippiConvert_8s32f + ippiCopy_32f_C4C1R + ippiCrossCorrNorm_32f_C1R + algType = ippiROIValid | ippiNorm
ippiCrossCorrValid_Norm_8u32f_AC4R	ippiCopy_8u_C4C1R + ippiCrossCorrNorm_8u32f_C1R + algType = ippiROIValid | ippiNorm
ippiCrossCorrValid_Norm_8u32f_C1R	ippiCrossCorrNorm_8u32f_C1R + algType = ippiROIValid | ippiNorm
ippiCrossCorrValid_Norm_8u32f_C3R	ippiCopy_8u_C3C1R + ippiCrossCorrNorm_8u32f_C1R + algType = ippiROIValid | ippiNorm
ippiCrossCorrValid_Norm_8u32f_C4R	ippiCopy_8u_C4C1R + ippiCrossCorrNorm_8u32f_C1R + algType = ippiROIValid | ippiNorm
ippiCrossCorrValid_Norm_8u_AC4RSfs	ippiCopy_8u_C4C1R + ippiCrossCorrNorm_8u_C1RSfs + algType = ippiROIValid | ippiNorm

ippiCrossCorrValid_Norm_8u_C1RSfs	ippiCrossCorrNorm_8u_C1RSfs + algType = ippiROIValid | ippiNorm
ippiCrossCorrValid_Norm_8u_C3RSfs	ippiCopy_8u_C3C1R + ippiCrossCorrNorm_8u_C1RSfs + algType = ippiROIValid | ippiNorm
ippiCrossCorrValid_Norm_8u_C4RSfs	ippiCopy_8u_C4C1R + ippiCrossCorrNorm_8u_C1RSfs + algType = ippiROIValid | ippiNorm
ippiDCTFwdFree_32f	ippiFree
ippiDCTFwdGetBufSize_32f	N/A
ippiDCTFwdInitAlloc_32f	ippiDCTGetSize + ippiMalloc + ippiDCTInit
ippiDCTInvFree_32f	ippiFree
ippiDCTInvGetBufSize_32f	N/A
ippiDCTInvInitAlloc_32f	ippiDCTGetSize + ippiMalloc + ippiDCTInit
ippiDFTFree_C_32fc	ippiFree
ippiDFTFree_R_32f	ippiFree
ippiDFTFree_R_32s	ippiFree
ippiDFTFwd_RToPack_8u32s_AC4RSfs	ippiConvert_8u32f + ippiDFTFwd_RToPack_32f
ippiDFTFwd_RToPack_8u32s_C1RSfs	ippiConvert_8u32f + ippiDFTFwd_RToPack_32f
ippiDFTFwd_RToPack_8u32s_C3RSfs	ippiConvert_8u32f + ippiDFTFwd_RToPack_32f
ippiDFTFwd_RToPack_8u32s_C4RSfs	ippiConvert_8u32f + ippiDFTFwd_RToPack_32f
ippiDFTGetBufSize_C_32fc	N/A
ippiDFTGetBufSize_R_32f	N/A
ippiDFTGetBufSize_R_32s	N/A
ippiDFTInitAlloc_C_32fc	ippiDFTGetSize + ippiMalloc + ippiDFTInit
ippiDFTInitAlloc_R_32f	ippiDFTGetSize + ippiMalloc + ippiDFTInit
ippiDFTInitAlloc_R_32s	ippiConvert_32s32f (or to 64f)+ ippiDFTGetSize + ippiMalloc + ippiDFTInit
ippiDFTInv_PackToR_32s8u_AC4RSfs	ippiDFTFwd_PackToR_32f + ippiConvert_32f8u

ippiDFTInv_PackToR_32s8u_C1RSfs	ippiDFTFwd_PackToR_32f + ippiConvert_32f8u
ippiDFTInv_PackToR_32s8u_C3RSfs	ippiDFTFwd_PackToR_32f + ippiConvert_32f8u
ippiDFTInv_PackToR_32s8u_C4RSfs	ippiDFTFwd_PackToR_32f + ippiConvert_32f8u
ippiDeconvFFTFree_32f_C1R	ippFree
ippiDeconvFFTFree_32f_C3R	ippFree
ippiDeconvFFTInitAlloc_32f_C1R	ippiDeconvFFTGetSize_32f + ippMalloc_8u + ippiDeconvFFTInit_32f_C1R
ippiDeconvFFTInitAlloc_32f_C3R	ippiDeconvFFTGetSize_32f + ippMalloc_8u + ippiDeconvFFTInit_32f_C3R
ippiDeconvLRFree_32f_C1R	ippFree
ippiDeconvLRFree_32f_C3R	ippFree
ippiDeconvLRInitAlloc_32f_C1R	ippiDeconvLRGetSize_32f + ippMalloc_8u + ippiDeconvLRInit_32f_C1R
ippiDeconvLRInitAlloc_32f_C3R	ippiDeconvLRGetSize_32f + ippMalloc_8u + ippiDeconvLRInit_32f_C3R
ippiDilate3x3_16u_AC4IR	Use not-in-place flavor
ippiDilate3x3_16u_AC4R	ippiDilateBorder_16u_C4R with 3x3 mask
ippiDilate3x3_16u_C1IR	Use not-in-place flavor
ippiDilate3x3_16u_C1R	ippiDilateBorder_16u_C1R with 3x3 mask
ippiDilate3x3_16u_C3IR	Use not-in-place flavor
ippiDilate3x3_16u_C3R	ippiDilateBorder_16u_C3R with 3x3 mask
ippiDilate3x3_16u_C4IR	Use not-in-place flavor
ippiDilate3x3_16u_C4R	ippiDilateBorder_16u_C4R with 3x3 mask
ippiDilate3x3_32f_AC4IR	Use not-in-place flavor
ippiDilate3x3_32f_AC4R	ippiDilateBorder_32f_C4R with 3x3 mask
ippiDilate3x3_32f_C1IR	Use not-in-place flavor
ippiDilate3x3_32f_C1R	ippiDilateBorder_32f_C1R with 3x3 mask
ippiDilate3x3_32f_C3IR	Use not-in-place flavor
ippiDilate3x3_32f_C3R	ippiDilateBorder_32f_C3R with 3x3 mask
ippiDilate3x3_32f_C4IR	Use not-in-place flavor

ippiDilate3x3_32f_C4R	ippiDilateBorder_32f_C4R with 3x3 mask
ippiDilate3x3_8u_AC4IR	Use not-in-place flavor
ippiDilate3x3_8u_AC4R	ippiDilateBorder_8u_C4R with 3x3 mask
ippiDilate3x3_8u_C1IR	Use not-in-place flavor
ippiDilate3x3_8u_C1R	ippiDilateBorder_8u_C1R with 3x3 mask
ippiDilate3x3_8u_C3IR	Use not-in-place flavor
ippiDilate3x3_8u_C3R	ippiDilateBorder_8u_C3R with 3x3 mask
ippiDilate3x3_8u_C4IR	Use not-in-place flavor
ippiDilate3x3_8u_C4R	ippiDilateBorder_8u_C4R with 3x3 mask
ippiDilate_16u_AC4IR	Use not-in-place flavor
ippiDilate_16u_AC4R	ippiDilateBorder_16u_C4R
ippiDilate_16u_C1IR	Use not-in-place flavor
ippiDilate_16u_C1R	ippiDilateBorder_16u_C1R
ippiDilate_16u_C3IR	Use not-in-place flavor
ippiDilate_16u_C3R	ippiDilateBorder_16u_C3R
ippiDilate_16u_C4R	ippiDilateBorder_16u_C4R
ippiDilate_32f_AC4IR	Use not-in-place flavor
ippiDilate_32f_AC4R	ippiDilateBorder_32f_C4R
ippiDilate_32f_C1IR	Use not-in-place flavor
ippiDilate_32f_C1R	ippiDilateBorder_32f_C1R
ippiDilate_32f_C3IR	Use not-in-place flavor
ippiDilate_32f_C3R	ippiDilateBorder_32f_C3R
ippiDilate_32f_C4R	ippiDilateBorder_32f_C4R
ippiDilate_8u_AC4IR	Use not-in-place flavor
ippiDilate_8u_AC4R	ippiDilateBorder_8u_C4R
ippiDilate_8u_C1IR	Use not-in-place flavor
ippiDilate_8u_C1R	ippiDilateBorder_8u_C1R
ippiDilate_8u_C3IR	Use not-in-place flavor
ippiDilate_8u_C3R	ippiDilateBorder_8u_C3R
ippiDilate_8u_C4R	ippiDilateBorder_8u_C4R
ippiDivC_16sc_AC4IRSfs	N/A

ippiDivC_16sc_AC4RSfs	N/A
ippiDivC_16sc_C1IRSfs	N/A
ippiDivC_16sc_C1RSfs	N/A
ippiDivC_16sc_C3IRSfs	N/A
ippiDivC_16sc_C3RSfs	N/A
ippiDivC_32fc_AC4IR	N/A
ippiDivC_32fc_AC4R	N/A
ippiDivC_32fc_C1IR	N/A
ippiDivC_32fc_C1R	N/A
ippiDivC_32fc_C3IR	N/A
ippiDivC_32fc_C3R	N/A
ippiDivC_32sc_AC4IRSfs	N/A
ippiDivC_32sc_AC4RSfs	N/A
ippiDivC_32sc_C1IRSfs	N/A
ippiDivC_32sc_C1RSfs	N/A
ippiDivC_32sc_C3IRSfs	N/A
ippiDivC_32sc_C3RSfs	N/A
ippiDiv_16sc_AC4IRSfs	N/A
ippiDiv_16sc_AC4RSfs	N/A
ippiDiv_16sc_C1IRSfs	N/A
ippiDiv_16sc_C1RSfs	N/A
ippiDiv_16sc_C3IRSfs	N/A
ippiDiv_16sc_C3RSfs	N/A
ippiDiv_32fc_AC4IR	N/A
ippiDiv_32fc_AC4R	N/A
ippiDiv_32fc_C1IR	N/A
ippiDiv_32fc_C1R	N/A
ippiDiv_32fc_C3IR	N/A
ippiDiv_32fc_C3R	N/A
ippiDiv_32sc_AC4IRSfs	N/A
ippiDiv_32sc_AC4RSfs	N/A

ippiDiv_32sc_C1IRSfs	N/A
ippiDiv_32sc_C1RSfs	N/A
ippiDiv_32sc_C3IRSfs	N/A
ippiDiv_32sc_C3RSfs	N/A
ippiDotProd_8s64f_AC4R	ippiConvert_8s16s + ippiDotProd_16s64f
ippiDotProd_8s64f_C1R	ippiConvert_8s16s + ippiDotProd_16s64f
ippiDotProd_8s64f_C3R	ippiConvert_8s16s + ippiDotProd_16s64f
ippiDotProd_8s64f_C4R	ippiConvert_8s16s + ippiDotProd_16s64f
ippiErode3x3_16u_AC4IR	Use not-in-place flavor
ippiErode3x3_16u_AC4R	ippiErodeBorder_16u_C4R with 3x3 mask
ippiErode3x3_16u_C1IR	Use not-in-place flavor
ippiErode3x3_16u_C1R	ippiErodeBorder_16u_C1R with 3x3 mask
ippiErode3x3_16u_C3IR	Use not-in-place flavor
ippiErode3x3_16u_C3R	ippiErodeBorder_16u_C3R with 3x3 mask
ippiErode3x3_16u_C4IR	Use not-in-place flavor
ippiErode3x3_16u_C4R	ippiErodeBorder_16u_C4R with 3x3 mask
ippiErode3x3_32f_AC4IR	Use not-in-place flavor
ippiErode3x3_32f_AC4R	ippiErodeBorder_32f_C4R with 3x3 mask
ippiErode3x3_32f_C1IR	Use not-in-place flavor
ippiErode3x3_32f_C1R	ippiErodeBorder_32f_C1R with 3x3 mask
ippiErode3x3_32f_C3IR	Use not-in-place flavor
ippiErode3x3_32f_C3R	ippiErodeBorder_32f_C3R with 3x3 mask
ippiErode3x3_32f_C4IR	Use not-in-place flavor
ippiErode3x3_32f_C4R	ippiErodeBorder_32f_C4R with 3x3 mask
ippiErode3x3_8u_AC4IR	Use not-in-place flavor
ippiErode3x3_8u_AC4R	ippiErodeBorder_8u_C4R with 3x3 mask
ippiErode3x3_8u_C1IR	Use not-in-place flavor
ippiErode3x3_8u_C1R	ippiErodeBorder_8u_C1R with 3x3 mask
ippiErode3x3_8u_C3IR	Use not-in-place flavor
ippiErode3x3_8u_C3R	ippiErodeBorder_8u_C3R with 3x3 mask
ippiErode3x3_8u_C4IR	Use not-in-place flavor

ippiErode3x3_8u_C4R	ippiErodeBorder_8u_C4R with 3x3 mask
ippiErode_16u_AC4IR	Use not-in-place flavor
ippiErode_16u_AC4R	ippiErodeBorder_16u_C4R
ippiErode_16u_C1IR	Use not-in-place flavor
ippiErode_16u_C1R	ippiErodeBorder_16u_C1R
ippiErode_16u_C3IR	Use not-in-place flavor
ippiErode_16u_C3R	ippiErodeBorder_16u_C3R
ippiErode_16u_C4R	ippiErodeBorder_16u_C4R
ippiErode_32f_AC4IR	Use not-in-place flavor
ippiErode_32f_AC4R	ippiErodeBorder_32f_C4R
ippiErode_32f_C1IR	Use not-in-place flavor
ippiErode_32f_C1R	ippiErodeBorder_32f_C1R
ippiErode_32f_C3IR	Use not-in-place flavor
ippiErode_32f_C3R	ippiErodeBorder_32f_C3R
ippiErode_32f_C4R	ippiErodeBorder_32f_C4R
ippiErode_8u_AC4IR	Use not-in-place flavor
ippiErode_8u_AC4R	ippiErodeBorder_8u_C4R
ippiErode_8u_C1IR	Use not-in-place flavor
ippiErode_8u_C1R	ippiErodeBorder_8u_C1R
ippiErode_8u_C3IR	Use not-in-place flavor
ippiErode_8u_C3R	ippiErodeBorder_8u_C3R
ippiErode_8u_C4R	ippiErodeBorder_8u_C4R
ippiFFTFree_C_32fc	ippiFree
ippiFFTFree_R_32f	ippiFree
ippiFFTFree_R_32s	ippiFree
ippiFFTFwd_RToPack_8u32s_AC4RSfs	ippiConvert_8u32f + ippiFFTFwd_RToPack_32f
ippiFFTFwd_RToPack_8u32s_C1RSfs	ippiConvert_8u32f + ippiFFTFwd_RToPack_32f
ippiFFTFwd_RToPack_8u32s_C3RSfs	ippiConvert_8u32f + ippiFFTFwd_RToPack_32f

ippiFFTFwd_RToPack_8u32s_C4RSfs	ippiConvert_8u32f + ippiFFTFwd_RToPack_32f
ippiFFTGetBufSize_C_32fc	N/A
ippiFFTGetBufSize_R_32f	N/A
ippiFFTGetBufSize_R_32s	N/A
ippiFFTInitAlloc_C_32fc	ippiFFTGetSize + ippiMalloc + ippiFFTInit
ippiFFTInitAlloc_R_32f	ippiFFTGetSize + ippiMalloc + ippiFFTInit
ippiFFTInitAlloc_R_32s	ippiConvert_32s32f (or to 64f)+ ippiFFTGetSize + ippiMalloc + ippiFFTInit
ippiFFTInv_PackToR_32s8u_AC4RSfs	ippiFFTFwd_PackToR_32f + ippiConvert_32f8u
ippiFFTInv_PackToR_32s8u_C1RSfs	ippiFFTFwd_PackToR_32f + ippiConvert_32f8u
ippiFFTInv_PackToR_32s8u_C3RSfs	ippiFFTFwd_PackToR_32f + ippiConvert_32f8u
ippiFFTInv_PackToR_32s8u_C4RSfs	ippiFFTFwd_PackToR_32f + ippiConvert_32f8u
ippiFilter32f_16s_AC4R	ippiFilterBorderInit_32f + ippiFilterBorder_16s_C4R
ippiFilter32f_16s_C1R	ippiFilterBorderInit_32f + ippiFilterBorder_16s_C1R
ippiFilter32f_16s_C3R	ippiFilterBorderInit_32f + ippiFilterBorder_16s_C3R
ippiFilter32f_16s_C4R	ippiFilterBorderInit_32f + ippiFilterBorder_16s_C4R
ippiFilter32f_16u_AC4R	ippiFilterBorderInit_32f + ippiFilterBorder_16u_C4R
ippiFilter32f_16u_C1R	ippiFilterBorderInit_32f + ippiFilterBorder_16u_C1R
ippiFilter32f_16u_C3R	ippiFilterBorderInit_32f + ippiFilterBorder_16u_C3R
ippiFilter32f_16u_C4R	ippiFilterBorderInit_32f + ippiFilterBorder_16u_C4R
ippiFilter32f_32s_C1R	ippiFilterBorderInit_32f + ippiConvert_32s32f + ippiFilterBorder_32f_C1R

ippiFilter32f_32s_C3R	ippiFilterBorderInit_32f + ippiConvert_32s32f + ippiFilterBorder_32f_C3R
ippiFilter32f_32s_C4R	ippiFilterBorderInit_32f + ippiConvert_32s32f + ippiFilterBorder_32f_C4R
ippiFilter32f_8s16s_C1R	ippiFilterBorderInit_32f + ippiConvert_8s16s + ippiFilterBorder_16s_C1R
ippiFilter32f_8s16s_C3R	ippiFilterBorderInit_32f + ippiConvert_8s16s + ippiFilterBorder_16s_C3R
ippiFilter32f_8s16s_C4R	ippiFilterBorderInit_32f + ippiConvert_8s16s + ippiFilterBorder_16s_C4R
ippiFilter32f_8s_C1R	ippiFilterBorderInit_32f + ippiConvert_8s16s + ippiFilterBorder_16s_C1R
ippiFilter32f_8s_C3R	ippiFilterBorderInit_32f + ippiConvert_8s16s + ippiFilterBorder_16s_C3R
ippiFilter32f_8s_C4R	ippiFilterBorderInit_32f + ippiConvert_8s16s + ippiFilterBorder_16s_C4R
ippiFilter32f_8u16s_C1R	ippiFilterBorderInit_32f + ippiConvert_8u16s + ippiFilterBorder_16s_C1R
ippiFilter32f_8u16s_C3R	ippiFilterBorderInit_32f + ippiConvert_8u16s + ippiFilterBorder_16s_C3R
ippiFilter32f_8u16s_C4R	ippiFilterBorderInit_32f + ippiConvert_8u16s + ippiFilterBorder_16s_C4R
ippiFilter32f_8u_AC4R	ippiFilterBorderInit_32f + ippiFilterBorder_8u_C4R
ippiFilter32f_8u_C1R	ippiFilterBorderInit_32f + ippiFilterBorder_8u_C1R
ippiFilter32f_8u_C3R	ippiFilterBorderInit_32f + ippiFilterBorder_8u_C3R
ippiFilter32f_8u_C4R	ippiFilterBorderInit_32f + ippiFilterBorder_8u_C4R

ippiFilterBilateralGetBufSize_8u_C1R	ippiFilterBilateralBorderGetBufferSize
ippiFilterBilateralInit_8u_C1R	ippiFilterBilateralBorderInit
ippiFilterBilateral_8u_C1R	ippiFilterBilateralBorder_8u_C1R
ippiFilterBox_16s_AC4IR	Use ippiFilterBoxBorder
ippiFilterBox_16s_AC4R	Use ippiFilterBoxBorder
ippiFilterBox_16s_C1IR	Use ippiFilterBoxBorder
ippiFilterBox_16s_C1R	Use ippiFilterBoxBorder
ippiFilterBox_16s_C3IR	Use ippiFilterBoxBorder
ippiFilterBox_16s_C3R	Use ippiFilterBoxBorder
ippiFilterBox_16s_C4IR	Use ippiFilterBoxBorder
ippiFilterBox_16s_C4R	Use ippiFilterBoxBorder
ippiFilterBox_16u_AC4IR	Use ippiFilterBoxBorder
ippiFilterBox_16u_AC4R	Use ippiFilterBoxBorder
ippiFilterBox_16u_C1IR	Use ippiFilterBoxBorder
ippiFilterBox_16u_C1R	Use ippiFilterBoxBorder
ippiFilterBox_16u_C3IR	Use ippiFilterBoxBorder
ippiFilterBox_16u_C3R	Use ippiFilterBoxBorder
ippiFilterBox_16u_C4IR	Use ippiFilterBoxBorder
ippiFilterBox_16u_C4R	Use ippiFilterBoxBorder
ippiFilterBox_32f_AC4IR	Use ippiFilterBoxBorder
ippiFilterBox_32f_AC4R	Use ippiFilterBoxBorder
ippiFilterBox_32f_C1IR	Use ippiFilterBoxBorder
ippiFilterBox_32f_C1R	Use ippiFilterBoxBorder
ippiFilterBox_32f_C3IR	Use ippiFilterBoxBorder
ippiFilterBox_32f_C3R	Use ippiFilterBoxBorder
ippiFilterBox_32f_C4IR	Use ippiFilterBoxBorder
ippiFilterBox_32f_C4R	Use ippiFilterBoxBorder
ippiFilterBox_8u_AC4IR	Use ippiFilterBoxBorder
ippiFilterBox_8u_AC4R	Use ippiFilterBoxBorder
ippiFilterBox_8u_C1IR	Use ippiFilterBoxBorder
ippiFilterBox_8u_C1R	Use ippiFilterBoxBorder

ippiFilterBox_8u_C3IR	Use ippiFilterBoxBorder
ippiFilterBox_8u_C3R	Use ippiFilterBoxBorder
ippiFilterBox_8u_C4IR	Use ippiFilterBoxBorder
ippiFilterBox_8u_C4R	Use ippiFilterBoxBorder
ippiFilterColumn_64f_C1R	Use ippiFilter_64f_C1R
ippiFilterColumn32f_16s_AC4R	ippiFilterBorderInit_32f + ippiFilterBorder_16s_C4R (with filter.width=1)
ippiFilterColumn32f_16s_C1R	ippiFilterBorderInit_32f + ippiFilterBorder_16s_C1R (with filter.width=1)
ippiFilterColumn32f_16s_C3R	ippiFilterBorderInit_32f + ippiFilterBorder_16s_C3R (with filter.width=1)
ippiFilterColumn32f_16s_C4R	ippiFilterBorderInit_32f + ippiFilterBorder_16s_C4R (with filter.width=1)
ippiFilterColumn32f_16u_AC4R	ippiFilterBorderInit_32f + ippiFilterBorder_16u_C4R (with filter.width=1)
ippiFilterColumn32f_16u_C1R	ippiFilterBorderInit_32f + ippiFilterBorder_16u_C1R (with filter.width=1)
ippiFilterColumn32f_16u_C3R	ippiFilterBorderInit_32f + ippiFilterBorder_16u_C3R (with filter.width=1)
ippiFilterColumn32f_16u_C4R	ippiFilterBorderInit_32f + ippiFilterBorder_16u_C4R (with filter.width=1)
ippiFilterColumn32f_8u_AC4R	ippiFilterBorderInit_32f + ippiFilterBorder_8u_C4R (with filter.width=1)
ippiFilterColumn32f_8u_C1R	ippiFilterBorderInit_32f + ippiFilterBorder_8u_C1R (with filter.width=1)
ippiFilterColumn32f_8u_C3R	ippiFilterBorderInit_32f + ippiFilterBorder_8u_C3R (with filter.width=1)
ippiFilterColumn32f_8u_C4R	ippiFilterBorderInit_32f + ippiFilterBorder_8u_C4R (with filter.width=1)
ippiFilterColumn_16s_AC4R	ippiFilterBorderInit_16s + ippiFilterBorder_16s_C4R (with filter.width=1)
ippiFilterColumn_16s_C1R	ippiFilterBorderInit_16s + ippiFilterBorder_16s_C1R (with filter.width=1)
ippiFilterColumn_16s_C3R	ippiFilterBorderInit_16s + ippiFilterBorder_16s_C3R (with filter.width=1)
ippiFilterColumn_16s_C4R	ippiFilterBorderInit_16s + ippiFilterBorder_16s_C4R (with filter.width=1)

ippiFilterColumn_16u_AC4R	ippiFilterBorderInit_16s + ippiFilterBorder_16u_C4R (with filter.width=1)
ippiFilterColumn_16u_C1R	ippiFilterBorderInit_16s + ippiFilterBorder_16u_C1R (with filter.width=1)
ippiFilterColumn_16u_C3R	ippiFilterBorderInit_16s + ippiFilterBorder_16u_C3R (with filter.width=1)
ippiFilterColumn_16u_C4R	ippiFilterBorderInit_16s + ippiFilterBorder_16u_C4R (with filter.width=1)
ippiFilterColumn_32f_AC4R	ippiFilterBorderInit_32f + ippiFilterBorder_32f_C4R (with filter.width=1)
ippiFilterColumn_32f_C1R	ippiFilterBorderInit_32f + ippiFilterBorder_32f_C1R (with filter.width=1)
ippiFilterColumn_32f_C3R	ippiFilterBorderInit_32f + ippiFilterBorder_32f_C3R (with filter.width=1)
ippiFilterColumn_32f_C4R	ippiFilterBorderInit_32f + ippiFilterBorder_32f_C4R (with filter.width=1)
ippiFilterColumn_8u_AC4R	ippiFilterBorderInit_16s + ippiFilterBorder_8u_C4R (with filter.width=1)
ippiFilterColumn_8u_C1R	ippiFilterBorderInit_16s + ippiFilterBorder_8u_C1R (with filter.width=1)
ippiFilterColumn_8u_C3R	ippiFilterBorderInit_16s + ippiFilterBorder_8u_C3R (with filter.width=1)
ippiFilterColumn_8u_C4R	ippiFilterBorderInit_16s + ippiFilterBorder_8u_C4R (with filter.width=1)
ippiFilterGauss_16s_AC4R	ippiCopy_C4C1R + ippiFilterGaussianBorder_16s_C1R
ippiFilterGauss_16s_C1R	ippiFilterGaussianBorder_16s_C1R
ippiFilterGauss_16s_C3R	ippiFilterGaussianBorder_16s_C3R
ippiFilterGauss_16s_C4R	ippiCopy_C4C1R + ippiFilterGaussianBorder_16s_C1R
ippiFilterGauss_16u_AC4R	ippiCopy_C4C1R + ippiFilterGaussianBorder_16u_C1R
ippiFilterGauss_16u_C1R	ippiFilterGaussianBorder_16u_C1R
ippiFilterGauss_16u_C3R	ippiFilterGaussianBorder_16u_C3R
ippiFilterGauss_16u_C4R	ippiCopy_C4C1R + ippiFilterGaussianBorder_16u_C1R

ippiFilterGauss_32f_AC4R	ippiCopy_C4C1R + ippiFilterGaussianBorder_32f_C1R
ippiFilterGauss_32f_C1R	ippiFilterGaussianBorder_32f_C1R
ippiFilterGauss_32f_C3R	ippiFilterGaussianBorder_32f_C3R
ippiFilterGauss_32f_C4R	ippiCopy_C4C1R + ippiFilterGaussianBorder_32f_C1R
ippiFilterGauss_8u_AC4R	ippiCopy_C4C1R + ippiFilterGaussianBorder_8u_C1R
ippiFilterGauss_8u_C1R	ippiFilterGaussianBorder_8u_C1R
ippiFilterGauss_8u_C3R	ippiFilterGaussianBorder_8u_C3R
ippiFilterGauss_8u_C4R	ippiCopy_C4C1R + ippiFilterGaussianBorder_8u_C1R
ippiFilterHipass_16s_AC4R	ippiFilterHipassBorder_16s_AC4R
ippiFilterHipass_16s_C1R	ippiFilterHipassBorder_16s_C1R
ippiFilterHipass_16s_C3R	ippiFilterHipassBorder_16s_C3R
ippiFilterHipass_16s_C4R	ippiFilterHipassBorder_16s_C4R
ippiFilterHipass_32f_AC4R	ippiFilterHipassBorder_32f_AC4R
ippiFilterHipass_32f_C1R	ippiFilterHipassBorder_32f_C1R
ippiFilterHipass_32f_C3R	ippiFilterHipassBorder_32f_C3R
ippiFilterHipass_32f_C4R	ippiFilterHipassBorder_32f_C4R
ippiFilterHipass_8u_AC4R	ippiFilterHipassBorder_8u_AC4R
ippiFilterHipass_8u_C1R	ippiFilterHipassBorder_8u_C1R
ippiFilterHipass_8u_C3R	ippiFilterHipassBorder_8u_C3R
ippiFilterHipass_8u_C4R	ippiFilterHipassBorder_8u_C4R
ippiFilterLaplace_16s_AC4R	ippiFilterLaplaceBorder_16s_AC4R
ippiFilterLaplace_16s_C1R	ippiFilterLaplaceBorder_16s_C1R
ippiFilterLaplace_16s_C3R	ippiFilterLaplaceBorder_16s_C3R
ippiFilterLaplace_16s_C4R	ippiFilterLaplaceBorder_16s_C4R
ippiFilterLaplace_32f_AC4R	ippiFilterLaplaceBorder_32f_AC4R
ippiFilterLaplace_32f_C1R	ippiFilterLaplaceBorder_32f_C1R
ippiFilterLaplace_32f_C3R	ippiFilterLaplaceBorder_32f_C3R
ippiFilterLaplace_32f_C4R	ippiFilterLaplaceBorder_32f_C4R

ippiFilterLaplace_8s16s_C1R	ippiConvert_8s16s + ippiFilterLaplaceBorder_16s
ippiFilterLaplace_8u16s_C1R	ippiConvert_8u16s + ippiFilterLaplaceBorder_16s
ippiFilterLaplace_8u_AC4R	ippiFilterLaplaceBorder_8u_AC4R
ippiFilterLaplace_8u_C1R	ippiFilterLaplaceBorder_8u_C1R
ippiFilterLaplace_8u_C3R	ippiFilterLaplaceBorder_8u_C3R
ippiFilterLaplace_8u_C4R	ippiFilterLaplaceBorder_8u_C4R
ippiFilterLowpass_16s_AC4R	ippiFilterBoxBorder_16s_AC4R
ippiFilterLowpass_16s_C1R	ippiFilterBoxBorder_16s_C1R
ippiFilterLowpass_16s_C3R	ippiFilterBoxBorder_16s_C3R
ippiFilterLowpass_16u_AC4R	ippiFilterBoxBorder_16u_AC4R
ippiFilterLowpass_16u_C1R	ippiFilterBoxBorder_16u_C1R
ippiFilterLowpass_16u_C3R	ippiFilterBoxBorder_16u_C3R
ippiFilterLowpass_32f_AC4R	ippiFilterBoxBorder_32f_AC4R
ippiFilterLowpass_32f_C1R	ippiFilterBoxBorder_32f_C1R
ippiFilterLowpass_32f_C3R	ippiFilterBoxBorder_32f_C3R
ippiFilterLowpass_8u_AC4R	ippiFilterBoxBorder_8u_AC4R
ippiFilterLowpass_8u_C1R	ippiFilterBoxBorder_8u_C1R
ippiFilterLowpass_8u_C3R	ippiFilterBoxBorder_8u_C3R
ippiFilterMax_16s_AC4R	Use ippiFilterMaxBorder
ippiFilterMax_16s_C1R	Use ippiFilterMaxBorder
ippiFilterMax_16s_C3R	Use ippiFilterMaxBorder
ippiFilterMax_16s_C4R	Use ippiFilterMaxBorder
ippiFilterMax_16u_AC4R	Use ippiFilterMaxBorder
ippiFilterMax_16u_C1R	Use ippiFilterMaxBorder
ippiFilterMax_16u_C3R	Use ippiFilterMaxBorder
ippiFilterMax_16u_C4R	Use ippiFilterMaxBorder
ippiFilterMax_32f_AC4R	Use ippiFilterMaxBorder
ippiFilterMax_32f_C1R	Use ippiFilterMaxBorder
ippiFilterMax_32f_C3R	Use ippiFilterMaxBorder

ippiFilterMax_32f_C4R	Use ippiFilterMaxBorder
ippiFilterMax_8u_AC4R	Use ippiFilterMaxBorder
ippiFilterMax_8u_C1R	Use ippiFilterMaxBorder
ippiFilterMax_8u_C3R	Use ippiFilterMaxBorder
ippiFilterMax_8u_C4R	Use ippiFilterMaxBorder
ippiFilterMedianHoriz_16s_AC4R	Use ippiFilterMedianBorder with horiz mask (mask.height=1)
ippiFilterMedianHoriz_16s_C1R	Use ippiFilterMedianBorder with horiz mask (mask.height=1)
ippiFilterMedianHoriz_16s_C3R	Use ippiFilterMedianBorder with horiz mask (mask.height=1)
ippiFilterMedianHoriz_16s_C4R	Use ippiFilterMedianBorder with horiz mask (mask.height=1)
ippiFilterMedianHoriz_16u_AC4R	Use ippiFilterMedianBorder with horiz mask (mask.height=1)
ippiFilterMedianHoriz_16u_C1R	Use ippiFilterMedianBorder with horiz mask (mask.height=1)
ippiFilterMedianHoriz_16u_C3R	Use ippiFilterMedianBorder with horiz mask (mask.height=1)
ippiFilterMedianHoriz_16u_C4R	Use ippiFilterMedianBorder with horiz mask (mask.height=1)
ippiFilterMedianHoriz_8u_AC4R	Use ippiFilterMedianBorder with horiz mask (mask.height=1)
ippiFilterMedianHoriz_8u_C1R	Use ippiFilterMedianBorder with horiz mask (mask.height=1)
ippiFilterMedianHoriz_8u_C3R	Use ippiFilterMedianBorder with horiz mask (mask.height=1)
ippiFilterMedianHoriz_8u_C4R	Use ippiFilterMedianBorder with horiz mask (mask.height=1)
ippiFilterMedianVert_16s_AC4R	Use ippiFilterMedianBorder with vertical mask (mask.width=1)
ippiFilterMedianVert_16s_C1R	Use ippiFilterMedianBorder with vertical mask (mask.width=1)
ippiFilterMedianVert_16s_C3R	Use ippiFilterMedianBorder with vertical mask (mask.width=1)
ippiFilterMedianVert_16s_C4R	Use ippiFilterMedianBorder with vertical mask (mask.width=1)

ippiFilterMedianVert_16u_AC4R	Use ippiFilterMedianBorder with vertical mask (mask.width=1)
ippiFilterMedianVert_16u_C1R	Use ippiFilterMedianBorder with vertical mask (mask.width=1)
ippiFilterMedianVert_16u_C3R	Use ippiFilterMedianBorder with vertical mask (mask.width=1)
ippiFilterMedianVert_16u_C4R	Use ippiFilterMedianBorder with vertical mask (mask.width=1)
ippiFilterMedianVert_8u_AC4R	Use ippiFilterMedianBorder with vertical mask (mask.width=1)
ippiFilterMedianVert_8u_C1R	Use ippiFilterMedianBorder with vertical mask (mask.width=1)
ippiFilterMedianVert_8u_C3R	Use ippiFilterMedianBorder with vertical mask (mask.width=1)
ippiFilterMedianVert_8u_C4R	Use ippiFilterMedianBorder with vertical mask (mask.width=1)
ippiFilterMedian_16s_AC4R	Use ippiFilterMedianBorder
ippiFilterMedian_16s_C1R	Use ippiFilterMedianBorder
ippiFilterMedian_16s_C3R	Use ippiFilterMedianBorder
ippiFilterMedian_16s_C4R	Use ippiFilterMedianBorder
ippiFilterMedian_16u_AC4R	Use ippiFilterMedianBorder
ippiFilterMedian_16u_C1R	Use ippiFilterMedianBorder
ippiFilterMedian_16u_C3R	Use ippiFilterMedianBorder
ippiFilterMedian_16u_C4R	Use ippiFilterMedianBorder
ippiFilterMedian_32f_C1R	Use ippiFilterMedianBorder
ippiFilterMedian_8u_AC4R	Use ippiFilterMedianBorder
ippiFilterMedian_8u_C1R	Use ippiFilterMedianBorder
ippiFilterMedian_8u_C3R	Use ippiFilterMedianBorder
ippiFilterMedian_8u_C4R	Use ippiFilterMedianBorder
ippiFilterMin_16s_AC4R	Use ippiFilterMinBorder
ippiFilterMin_16s_C1R	Use ippiFilterMinBorder
ippiFilterMin_16s_C3R	Use ippiFilterMinBorder
ippiFilterMin_16s_C4R	Use ippiFilterMinBorder
ippiFilterMin_16u_AC4R	Use ippiFilterMinBorder

ippiFilterMin_16u_C1R	Use ippiFilterMinBorder
ippiFilterMin_16u_C3R	Use ippiFilterMinBorder
ippiFilterMin_16u_C4R	Use ippiFilterMinBorder
ippiFilterMin_32f_AC4R	Use ippiFilterMinBorder
ippiFilterMin_32f_C1R	Use ippiFilterMinBorder
ippiFilterMin_32f_C3R	Use ippiFilterMinBorder
ippiFilterMin_32f_C4R	Use ippiFilterMinBorder
ippiFilterMin_8u_AC4R	Use ippiFilterMinBorder
ippiFilterMin_8u_C1R	Use ippiFilterMinBorder
ippiFilterMin_8u_C3R	Use ippiFilterMinBorder
ippiFilterMin_8u_C4R	Use ippiFilterMinBorder
ippiFilterPrewittHoriz_16s_AC4R	Use ippiCopy_16s_C4C1R + ippiFilterPrewittHorizBorder_16s_C1R
ippiFilterPrewittHoriz_16s_C1R	ippiFilterPrewittHorizBorder_16s_C1R
ippiFilterPrewittHoriz_16s_C3R	Use ippiCopy_16s_C3C1R + ippiFilterPrewittHorizBorder_16s_C1R
ippiFilterPrewittHoriz_16s_C4R	Use ippiCopy_16s_C4C1R + ippiFilterPrewittHorizBorder_16s_C1R
ippiFilterPrewittHoriz_32f_AC4R	Use ippiCopy_32f_C4C1R + ippiFilterPrewittHorizBorder_32f_C1R
ippiFilterPrewittHoriz_32f_C1R	ippiFilterPrewittHorizBorder_32f_C1R
ippiFilterPrewittHoriz_32f_C3R	Use ippiCopy_32f_C3C1R + ippiFilterPrewittHorizBorder_32f_C1R
ippiFilterPrewittHoriz_32f_C4R	Use ippiCopy_32f_C4C1R + ippiFilterPrewittHorizBorder_32f_C1R
ippiFilterPrewittHoriz_8u_AC4R	Use ippiCopy_8u_C4C1R + ippiFilterPrewittHorizBorder_8u16s_C1R
ippiFilterPrewittHoriz_8u_C1R	ippiFilterPrewittHorizBorder_8u16s_C1R
ippiFilterPrewittHoriz_8u_C3R	Use ippiCopy_8u_C3C1R + ippiFilterPrewittHorizBorder_8u16s_C1R
ippiFilterPrewittHoriz_8u_C4R	Use ippiCopy_8u_C4C1R + ippiFilterPrewittHorizBorder_8u16s_C1R
ippiFilterPrewittVert_16s_AC4R	Use ippiCopy_16s_C4C1R + ippiFilterPrewittVertBorder_16s_C1R
ippiFilterPrewittVert_16s_C1R	ippiFilterPrewittVertBorder_16s_C1R

ippiFilterPrewittVert_16s_C3R	Use ippiCopy_16s_C3C1R + ippiFilterPrewittVertBorder_16s_C1R
ippiFilterPrewittVert_16s_C4R	Use ippiCopy_16s_C4C1R + ippiFilterPrewittVertBorder_16s_C1R
ippiFilterPrewittVert_32f_AC4R	Use ippiCopy_32f_C4C1R + ippiFilterPrewittVertBorder_32f_C1R
ippiFilterPrewittVert_32f_C1R	ippiFilterPrewittVertBorder_32f_C1R
ippiFilterPrewittVert_32f_C3R	Use ippiCopy_32f_C3C1R + ippiFilterPrewittVertBorder_32f_C1R
ippiFilterPrewittVert_32f_C4R	Use ippiCopy_32f_C4C1R + ippiFilterPrewittVertBorder_32f_C1R
ippiFilterPrewittVert_8u_AC4R	Use ippiCopy_8u_C4C1R + ippiFilterPrewittVertBorder_8u16s_C1R
ippiFilterPrewittVert_8u_C1R	ippiFilterPrewittVertBorder_8u16s_C1R
ippiFilterPrewittVert_8u_C3R	Use ippiCopy_8u_C3C1R + ippiFilterPrewittVertBorder_8u16s_C1R
ippiFilterPrewittVert_8u_C4R	Use ippiCopy_8u_C4C1R + ippiFilterPrewittVertBorder_8u16s_C1R
ippiFilterRobertsDown_16s_AC4R	Use ippiCopy_16s_C4C1R + ippiFilterRobertsDownBorder_16s_C1R
ippiFilterRobertsDown_16s_C1R	ippiFilterRobertsDownBorder_16s_C1R
ippiFilterRobertsDown_16s_C3R	Use ippiCopy_16s_C3C1R + ippiFilterRobertsDownBorder_16s_C1R
ippiFilterRobertsDown_32f_AC4R	Use ippiCopy_32f_C4C1R + ippiFilterRobertsDownBorder_32f_C1R
ippiFilterRobertsDown_32f_C1R	ippiFilterRobertsDownBorder_32f_C1R
ippiFilterRobertsDown_32f_C3R	Use ippiCopy_32f_C3C1R + ippiFilterRobertsDownBorder_32f_C1R
ippiFilterRobertsDown_8u_AC4R	Use ippiCopy_8u_C4C1R + ippiFilterRobertsDownBorder_8u16s_C1R
ippiFilterRobertsDown_8u_C1R	ippiFilterRobertsDownBorder_8u16s_C1R
ippiFilterRobertsDown_8u_C3R	Use ippiCopy_8u_C3C1R + ippiFilterRobertsDownBorder_8u16s_C1R
ippiFilterRobertsUp_16s_AC4R	Use ippiCopy_16s_C4C1R + ippiFilterRobertsUpBorder_16s_C1R
ippiFilterRobertsUp_16s_C1R	ippiFilterRobertsUpBorder_16s_C1R

ippiFilterRobertsUp_16s_C3R	Use ippiCopy_16s_C3C1R + ippiFilterRobertsUpBorder_16s_C1R
ippiFilterRobertsUp_32f_AC4R	Use ippiCopy_32f_C4C1R + ippiFilterRobertsUpBorder_32f_C1R
ippiFilterRobertsUp_32f_C1R	ippiFilterRobertsUpBorder_32f_C1R
ippiFilterRobertsUp_32f_C3R	Use ippiCopy_32f_C3C1R + ippiFilterRobertsUpBorder_32f_C1R
ippiFilterRobertsUp_8u_AC4R	Use ippiCopy_8u_C4C1R + ippiFilterRobertsUpBorder_8u16s_C1R
ippiFilterRobertsUp_8u_C1R	ippiFilterRobertsUpBorder_8u16s_C1R
ippiFilterRobertsUp_8u_C3R	Use ippiCopy_8u_C3C1R + ippiFilterRobertsUpBorder_8u16s_C1R
ippiFilterRoundGetBufSize16s_8u_AC4R	ippiFilterBorderGetSize
ippiFilterRoundGetBufSize16s_8u_C1R	ippiFilterBorderGetSize
ippiFilterRoundGetBufSize16s_8u_C3R	ippiFilterBorderGetSize
ippiFilterRoundGetBufSize16s_8u_C4R	ippiFilterBorderGetSize
ippiFilterRoundGetBufSize32f_16s_AC4R	ippiFilterBorderGetSize
ippiFilterRoundGetBufSize32f_16s_C1R	ippiFilterBorderGetSize
ippiFilterRoundGetBufSize32f_16s_C3R	ippiFilterBorderGetSize
ippiFilterRoundGetBufSize32f_16s_C4R	ippiFilterBorderGetSize
ippiFilterRoundGetBufSize32f_16u_AC4R	ippiFilterBorderGetSize
ippiFilterRoundGetBufSize32f_16u_C1R	ippiFilterBorderGetSize
ippiFilterRoundGetBufSize32f_16u_C3R	ippiFilterBorderGetSize
ippiFilterRoundGetBufSize32f_16u_C4R	ippiFilterBorderGetSize
ippiFilterRoundGetBufSize32f_8u_AC4R	ippiFilterBorderGetSize
ippiFilterRoundGetBufSize32f_8u_C1R	ippiFilterBorderGetSize
ippiFilterRoundGetBufSize32f_8u_C3R	ippiFilterBorderGetSize
ippiFilterRoundGetBufSize32f_8u_C4R	ippiFilterBorderGetSize
ippiFilterRoundGetBufSize32s_16s_AC4R	ippiFilterBorderGetSize
ippiFilterRoundGetBufSize32s_16s_C1R	ippiFilterBorderGetSize
ippiFilterRoundGetBufSize32s_16s_C3R	ippiFilterBorderGetSize
ippiFilterRoundGetBufSize32s_16s_C4R	ippiFilterBorderGetSize
ippiFilterRoundGetBufSize32s_16u_AC4R	ippiFilterBorderGetSize

ippiFilterRoundGetBufSize32s_16u_C1R	ippiFilterBorderGetSize
ippiFilterRoundGetBufSize32s_16u_C3R	ippiFilterBorderGetSize
ippiFilterRoundGetBufSize32s_16u_C4R	ippiFilterBorderGetSize
ippiFilterRow_64f_C1R	ippiFilter_64f_C1R
ippiFilterRow32f_16s_AC4R	ippiFilterBorderInit_32f + ippiFilterBorder_16s_C4R (with filter.height=1)
ippiFilterRow32f_16s_C1R	ippiFilterBorderInit_32f + ippiFilterBorder_16s_C1R (with filter.height=1)
ippiFilterRow32f_16s_C3R	ippiFilterBorderInit_32f + ippiFilterBorder_16s_C3R (with filter.height=1)
ippiFilterRow32f_16s_C4R	ippiFilterBorderInit_32f + ippiFilterBorder_16s_C4R (with filter.height=1)
ippiFilterRow32f_16u_AC4R	ippiFilterBorderInit_32f + ippiFilterBorder_16u_C4R (with filter.height=1)
ippiFilterRow32f_16u_C1R	ippiFilterBorderInit_32f + ippiFilterBorder_16u_C1R (with filter.height=1)
ippiFilterRow32f_16u_C3R	ippiFilterBorderInit_32f + ippiFilterBorder_16u_C3R (with filter.height=1)
ippiFilterRow32f_16u_C4R	ippiFilterBorderInit_32f + ippiFilterBorder_16u_C4R (with filter.height=1)
ippiFilterRow32f_8u_AC4R	ippiFilterBorderInit_32f + ippiFilterBorder_8u_C4R (with filter.height=1)
ippiFilterRow32f_8u_C1R	ippiFilterBorderInit_32f + ippiFilterBorder_8u_C1R (with filter.height=1)
ippiFilterRow32f_8u_C3R	ippiFilterBorderInit_32f + ippiFilterBorder_8u_C3R (with filter.height=1)
ippiFilterRow32f_8u_C4R	ippiFilterBorderInit_32f + ippiFilterBorder_8u_C4R (with filter.height=1)
ippiFilterRow_16s_AC4R	ippiFilterBorderInit_16s + ippiFilterBorder_16s_C4R (with filter.height=1)
ippiFilterRow_16s_C1R	ippiFilterBorderInit_16s + ippiFilterBorder_16s_C1R (with filter.height=1)
ippiFilterRow_16s_C3R	ippiFilterBorderInit_16s + ippiFilterBorder_16s_C3R (with filter.height=1)
ippiFilterRow_16s_C4R	ippiFilterBorderInit_16s + ippiFilterBorder_16s_C4R (with filter.height=1)

ippiFilterRow_16u_AC4R	ippiFilterBorderInit_16s + ippiFilterBorder_16u_C4R (with filter.height=1)
ippiFilterRow_16u_C1R	ippiFilterBorderInit_16s + ippiFilterBorder_16u_C1R (with filter.height=1)
ippiFilterRow_16u_C3R	ippiFilterBorderInit_16s + ippiFilterBorder_16u_C3R (with filter.height=1)
ippiFilterRow_16u_C4R	ippiFilterBorderInit_16s + ippiFilterBorder_16u_C4R (with filter.height=1)
ippiFilterRow_32f_AC4R	ippiFilterBorderInit_32f + ippiFilterBorder_32f_C4R (with filter.height=1)
ippiFilterRow_32f_C1R	ippiFilterBorderInit_32f + ippiFilterBorder_32f_C1R (with filter.height=1)
ippiFilterRow_32f_C3R	ippiFilterBorderInit_32f + ippiFilterBorder_32f_C3R (with filter.height=1)
ippiFilterRow_32f_C4R	ippiFilterBorderInit_32f + ippiFilterBorder_32f_C4R (with filter.height=1)
ippiFilterRow_8u_AC4R	ippiFilterBorderInit_16s + ippiFilterBorder_8u_C4R (with filter.height=1)
ippiFilterRow_8u_C1R	ippiFilterBorderInit_16s + ippiFilterBorder_8u_C1R (with filter.height=1)
ippiFilterRow_8u_C3R	ippiFilterBorderInit_16s + ippiFilterBorder_8u_C3R (with filter.height=1)
ippiFilterRow_8u_C4R	ippiFilterBorderInit_16s + ippiFilterBorder_8u_C4R (with filter.height=1)
ippiFilterScharrHoriz_32f_C1R	ippiFilterScharrHorizMaskBorder_32f_C1R
ippiFilterScharrHoriz_8s16s_C1R	ippiConvert_8s16s + ippiFilterScharrHorizMaskBorder_16s_C1R
ippiFilterScharrHoriz_8u16s_C1R	ippiFilterScharrHorizMaskBorder_8u16s_C1R
ippiFilterScharrVert_32f_C1R	ippiFilterScharrVertMaskBorder_32f_C1R
ippiFilterScharrVert_8s16s_C1R	ippiConvert_8s16s + ippiFilterScharrVertMaskBorder_16s_C1R
ippiFilterScharrVert_8u16s_C1R	ippiFilterScharrVertMaskBorder_8u16s_C1R
ippiFilterSharpen_16s_AC4R	ippiFilterSharpenBorder_16s_AC4R
ippiFilterSharpen_16s_C1R	ippiFilterSharpenBorder_16s_C1R
ippiFilterSharpen_16s_C3R	ippiFilterSharpenBorder_16s_C3R
ippiFilterSharpen_16s_C4R	ippiFilterSharpenBorder_16s_C4R

ippiFilterSharpen_32f_AC4R	ippiFilterSharpenBorder_32f_AC4R
ippiFilterSharpen_32f_C1R	ippiFilterSharpenBorder_32f_C1R
ippiFilterSharpen_32f_C3R	ippiFilterSharpenBorder_32f_C3R
ippiFilterSharpen_32f_C4R	ippiFilterSharpenBorder_32f_C4R
ippiFilterSharpen_8u_AC4R	ippiFilterSharpenBorder_8u_AC4R
ippiFilterSharpen_8u_C1R	ippiFilterSharpenBorder_8u_C1R
ippiFilterSharpen_8u_C3R	ippiFilterSharpenBorder_8u_C3R
ippiFilterSharpen_8u_C4R	ippiFilterSharpenBorder_8u_C4R
ippiFilterSobelCross_32f_C1R	ippiFilterSobelCrossBorder_32f_C1R
ippiFilterSobelCross_8s16s_C1R	ippiConvert_8s32f + ippiFilterSobelCrossBorder_32f_C1R
ippiFilterSobelCross_8u16s_C1R	ippiFilterSobelCrossBorder_8u16s_C1R
ippiFilterSobelHorizGetBufferSize_32f_C1R	ippiFilterSobelHorizBorderGetBufferSize
ippiFilterSobelHorizGetBufferSize_8u16s_C1R	ippiFilterSobelHorizBorderGetBufferSize
ippiFilterSobelHorizMask_32f_C1R	ippiFilterSobelHorizBorder_32f_C1R
ippiFilterSobelHorizSecond_32f_C1R	ippiFilterSobelHorizSecondBorder_32f_C1R
ippiFilterSobelHorizSecond_8s16s_C1R	ippiConvert_8s32f + ippiFilterSobelHorizSecondBorder_32f_C1R
ippiFilterSobelHorizSecond_8u16s_C1R	ippiFilterSobelHorizSecondBorder_8u16s_C1R
ippiFilterSobelHoriz_16s_AC4R	Use ippiCopy_16s_C4C1R + ippiFilterSobelHorizBorder_16s_C1R
ippiFilterSobelHoriz_16s_C1R	ippiFilterSobelHorizBorder_16s_C1R
ippiFilterSobelHoriz_16s_C3R	Use ippiCopy_16s_C3C1R + ippiFilterSobelHorizBorder_16s_C1R
ippiFilterSobelHoriz_16s_C4R	Use ippiCopy_16s_C4C1R + ippiFilterSobelHorizBorder_16s_C1R
ippiFilterSobelHoriz_32f_AC4R	Use ippiCopy_32f_C4C1R + ippiFilterSobelHorizBorder_32f_C1R
ippiFilterSobelHoriz_32f_C1R	ippiFilterSobelHorizBorder_32f_C1R
ippiFilterSobelHoriz_32f_C3R	Use ippiCopy_32f_C3C1R + ippiFilterSobelHorizBorder_32f_C1R
ippiFilterSobelHoriz_32f_C4R	Use ippiCopy_32f_C4C1R + ippiFilterSobelHorizBorder_32f_C1R

ippiFilterSobelHoriz_8s16s_C1R	ippiConvert_8s16s + ippiFilterSobelHorizBorder_16s_C1R
ippiFilterSobelHoriz_8u16s_C1R	ippiFilterSobelHorizBorder_8u16s_C1R
ippiFilterSobelHoriz_8u_AC4R	Use ippiCopy_8u_C4C1R + ippiFilterSobelHorizBorder_8u16s_C1R
ippiFilterSobelHoriz_8u_C1R	ippiFilterSobelHorizBorder_8u16s_C1R
ippiFilterSobelHoriz_8u_C3R	Use ippiCopy_8u_C3C1R + ippiFilterSobelHorizBorder_8u16s_C1R
ippiFilterSobelHoriz_8u_C4R	Use ippiCopy_8u_C4C1R + ippiFilterSobelHorizBorder_8u16s_C1R
ippiFilterSobelVertGetBufferSize_32f_C1R	ippiFilterSobelVertBorderGetBufferSize
ippiFilterSobelVertGetBufferSize_8u16s_C1R	ippiFilterSobelVertBorderGetBufferSize
ippiFilterSobelVertMask_32f_C1R	ippiFilterSobelVertBorder_32f_C1R
ippiFilterSobelVertSecond_32f_C1R	ippiFilterSobelVertSecondBorder_32f_C1R
ippiFilterSobelVertSecond_8s16s_C1R	ippiConvert_8s32f + ippiFilterSobelVertSecondBorder_32f_C1R
ippiFilterSobelVertSecond_8u16s_C1R	ippiFilterSobelVertSecondBorder_8u16s_C1R
ippiFilterSobelVert_16s_AC4R	Use ippiCopy_16s_C4C1R + ippiFilterSobelVertBorder_16s_C1R
ippiFilterSobelVert_16s_C1R	ippiFilterSobelVertBorder_16s_C1R
ippiFilterSobelVert_16s_C3R	Use ippiCopy_16s_C3C1R + ippiFilterSobelVertBorder_16s_C1R
ippiFilterSobelVert_16s_C4R	Use ippiCopy_16s_C4C1R + ippiFilterSobelVertBorder_16s_C1R
ippiFilterSobelVert_32f_AC4R	Use ippiCopy_32f_C4C1R + ippiFilterSobelVertBorder_32f_C1R
ippiFilterSobelVert_32f_C1R	ippiFilterSobelVertBorder_32f_C1R
ippiFilterSobelVert_32f_C3R	Use ippiCopy_32f_C3C1R + ippiFilterSobelVertBorder_32f_C1R
ippiFilterSobelVert_32f_C4R	Use ippiCopy_32f_C4C1R + ippiFilterSobelVertBorder_32f_C1R
ippiFilterSobelVert_8s16s_C1R	ippiConvert_8s16s + ippiFilterSobelVertBorder_16s_C1R
ippiFilterSobelVert_8u16s_C1R	ippiFilterSobelVertBorder_8u16s_C1R

ippiFilterSobelVert_8u_AC4R	Use ippiCopy_8u_C4C1R + ippiFilterSobelVertBorder_8u16s_C1R
ippiFilterSobelVert_8u_C1R	ippiFilterSobelVertBorder_8u16s_C1R
ippiFilterSobelVert_8u_C3R	Use ippiCopy_8u_C3C1R + ippiFilterSobelVertBorder_8u16s_C1R
ippiFilterSobelVert_8u_C4R	Use ippiCopy_8u_C4C1R + ippiFilterSobelVertBorder_8u16s_C1R
ippiFilter_16s_AC4R	ippiFilterBorderInit_16s + ippiFilterBorder_16s_C4R
ippiFilter_16s_C1R	ippiFilterBorderInit_16s + ippiFilterBorder_16s_C1R
ippiFilter_16s_C3R	ippiFilterBorderInit_16s + ippiFilterBorder_16s_C3R
ippiFilter_16s_C4R	ippiFilterBorderInit_16s + ippiFilterBorder_16s_C4R
ippiFilter_16u_AC4R	ippiFilterBorderInit_16s + ippiFilterBorder_16u_C4R
ippiFilter_16u_C1R	ippiFilterBorderInit_16s + ippiFilterBorder_16u_C1R
ippiFilter_16u_C3R	ippiFilterBorderInit_16s + ippiFilterBorder_16u_C3R
ippiFilter_16u_C4R	ippiFilterBorderInit_16s + ippiFilterBorder_16u_C4R
ippiFilter_32f_AC4R	ippiFilterBorderInit_32f + ippiFilterBorder_32f_C4R
ippiFilter_32f_C1R	ippiFilterBorderInit_32f + ippiFilterBorder_32f_C1R
ippiFilter_32f_C3R	ippiFilterBorderInit_32f + ippiFilterBorder_32f_C3R
ippiFilter_32f_C4R	ippiFilterBorderInit_32f + ippiFilterBorder_32f_C4R
ippiFilter_8u_AC4R	ippiFilterBorderInit_16s + ippiFilterBorder_8u_C4R
ippiFilter_8u_C1R	ippiFilterBorderInit_16s + ippiFilterBorder_8u_C1R
ippiFilter_8u_C3R	ippiFilterBorderInit_16s + ippiFilterBorder_8u_C3R

ippiFilter_8u_C4R	ippiFilterBorderInit_16s + ippiFilterBorder_8u_C4R
ippiFilter_Round16s_8u_AC4R	ippiFilterBorderInit_16s + ippiFilterBorder_8u_C4R
ippiFilter_Round16s_8u_C1R	ippiFilterBorderInit_16s + ippiFilterBorder_8u_C1R
ippiFilter_Round16s_8u_C3R	ippiFilterBorderInit_16s + ippiFilterBorder_8u_C3R
ippiFilter_Round16s_8u_C4R	ippiFilterBorderInit_16s + ippiFilterBorder_8u_C4R
ippiFilter_Round32f_16s_AC4R	ippiFilterBorderInit_32f + ippiFilterBorder_16s_C4R
ippiFilter_Round32f_16s_C1R	ippiFilterBorderInit_32f + ippiFilterBorder_16s_C1R
ippiFilter_Round32f_16s_C3R	ippiFilterBorderInit_32f + ippiFilterBorder_16s_C3R
ippiFilter_Round32f_16s_C4R	ippiFilterBorderInit_32f + ippiFilterBorder_16s_C4R
ippiFilter_Round32f_16u_AC4R	ippiFilterBorderInit_32f + ippiFilterBorder_16u_C4R
ippiFilter_Round32f_16u_C1R	ippiFilterBorderInit_32f + ippiFilterBorder_16u_C1R
ippiFilter_Round32f_16u_C3R	ippiFilterBorderInit_32f + ippiFilterBorder_16u_C3R
ippiFilter_Round32f_16u_C4R	ippiFilterBorderInit_32f + ippiFilterBorder_16u_C4R
ippiFilter_Round32f_8u_AC4R	ippiFilterBorderInit_32f + ippiFilterBorder_8u_C4R
ippiFilter_Round32f_8u_C1R	ippiFilterBorderInit_32f + ippiFilterBorder_8u_C1R
ippiFilter_Round32f_8u_C3R	ippiFilterBorderInit_32f + ippiFilterBorder_8u_C3R
ippiFilter_Round32f_8u_C4R	ippiFilterBorderInit_32f + ippiFilterBorder_8u_C4R
ippiFilter_Round32s_16s_AC4R	ippiFilterBorderInit_16s + ippiFilterBorder_16s_C4R
ippiFilter_Round32s_16s_C1R	ippiFilterBorderInit_16s + ippiFilterBorder_16s_C1R

ippiFilter_Round32s_16s_C3R	ippiFilterBorderInit_16s + ippiFilterBorder_16s_C3R
ippiFilter_Round32s_16s_C4R	ippiFilterBorderInit_16s + ippiFilterBorder_16s_C4R
ippiFilter_Round32s_16u_AC4R	ippiFilterBorderInit_16s + ippiFilterBorder_16u_C4R
ippiFilter_Round32s_16u_C1R	ippiFilterBorderInit_16s + ippiFilterBorder_16u_C1R
ippiFilter_Round32s_16u_C3R	ippiFilterBorderInit_16s + ippiFilterBorder_16u_C3R
ippiFilter_Round32s_16u_C4R	ippiFilterBorderInit_16s + ippiFilterBorder_16u_C4R
ippiGetCentralMoment_64s	Use Moments_64f
ippiGetHuMoments_64s	Use Moments_64f
ippiGetNormalizedCentralMoment_64s	Use Moments_64f
ippiGetNormalizedSpatialMoment_64s	Use Moments_64f
ippiGetRotateBound	N/A
ippiGetRotateQuad	N/A
ippiGetShearBound	N/A
ippiGetShearQuad	N/A
ippiGetSpatialMoment_64s	Use Moments_64f
ippiHistogramEven_16s_AC4R	ippiHistogram_16s_C4R
ippiHistogramEven_16s_C1R	ippiHistogram_16s_C1R
ippiHistogramEven_16s_C3R	ippiHistogram_16s_C3R
ippiHistogramEven_16s_C4R	ippiHistogram_16s_C4R
ippiHistogramEven_16u_AC4R	ippiHistogram_16u_C4R
ippiHistogramEven_16u_C1R	ippiHistogram_16u_C1R
ippiHistogramEven_16u_C3R	ippiHistogram_16u_C3R
ippiHistogramEven_16u_C4R	ippiHistogram_16u_C4R
ippiHistogramEven_8u_AC4R	ippiHistogram_8u_C4R
ippiHistogramEven_8u_C1R	ippiHistogram_8u_C1R
ippiHistogramEven_8u_C3R	ippiHistogram_8u_C3R
ippiHistogramEven_8u_C4R	ippiHistogram_8u_C4R

ippiHistogramRange_16s_AC4R	ippiHistogram_16s_C4R
ippiHistogramRange_16s_C1R	ippiHistogram_16s_C1R
ippiHistogramRange_16s_C3R	ippiHistogram_16s_C3R
ippiHistogramRange_16s_C4R	ippiHistogram_16s_C4R
ippiHistogramRange_16u_AC4R	ippiHistogram_16u_C4R
ippiHistogramRange_16u_C1R	ippiHistogram_16u_C1R
ippiHistogramRange_16u_C3R	ippiHistogram_16u_C3R
ippiHistogramRange_16u_C4R	ippiHistogram_16u_C4R
ippiHistogramRange_32f_AC4R	ippiHistogram_32f_C4R
ippiHistogramRange_32f_C1R	ippiHistogram_32f_C1R
ippiHistogramRange_32f_C3R	ippiHistogram_32f_C3R
ippiHistogramRange_32f_C4R	ippiHistogram_32f_C4R
ippiHistogramRange_8u_AC4R	ippiHistogram_8u_C4R
ippiHistogramRange_8u_C1R	ippiHistogram_8u_C1R
ippiHistogramRange_8u_C3R	ippiHistogram_8u_C3R
ippiHistogramRange_8u_C4R	ippiHistogram_8u_C4R
ippiImageJaehne_32s_AC4R	ippiImageJaehne_32f_C1R + ippiConvert_32f32s
ippiImageJaehne_32s_C1R	ippiImageJaehne_32f_C1R + ippiConvert_32f32s
ippiImageJaehne_32s_C3R	ippiImageJaehne_32f_C3R + ippiConvert_32f32s
ippiImageJaehne_32s_C4R	ippiImageJaehne_32f_C4R + ippiConvert_32f32s
ippiImageJaehne_8s_AC4R	ippiImageJaehne_16s_C4R + ippiConvert_16s8s
ippiImageJaehne_8s_C1R	ippiImageJaehne_16s_C1R + ippiConvert_16s8s
ippiImageJaehne_8s_C3R	ippiImageJaehne_16s_C3R + ippiConvert_16s8s
ippiImageJaehne_8s_C4R	ippiImageJaehne_16s_C4R + ippiConvert_16s8s
ippiImageRamp_32s_AC4R	ippiImageRamp_32f_C1R + ippiConvert_32f32s

ippiImageRamp_32s_C1R	ippiImageJaehne_32f_C1R + ippiConvert_32f32s
ippiImageRamp_32s_C3R	ippiImageJaehne_32f_C3R + ippiConvert_32f32s
ippiImageRamp_32s_C4R	ippiImageJaehne_32f_C4R + ippiConvert_32f32s
ippiImageRamp_8s_AC4R	ippiImageRamp_16s_C4R + ippiConvert_16s8s
ippiImageRamp_8s_C1R	ippiImageRamp_16s_C1R + ippiConvert_16s8s
ippiImageRamp_8s_C3R	ippiImageRamp_16s_C3R + ippiConvert_16s8s
ippiImageRamp_8s_C4R	ippiImageJaehne_16s_C4R + ippiConvert_16s8s
ippiLBPImage3x3_32f8u_C1R	ippiLBPImageMode3x3_32f8u_C1R
ippiLBPImage3x3_8u_C1R	ippiLBPImageMode3x3_8u_C1R
ippiLBPImage5x5_32f16u_C1R	ippiLBPImageMode5x5_32f16u_C1R
ippiLBPImage5x5_32f8u_C1R	ippiLBPImageMode5x5_32f8u_C1R
ippiLBPImage5x5_8u16u_C1R	ippiLBPImageMode5x5_8u16u_C1R
ippiLBPImage5x5_8u_C1R	ippiLBPImageMode5x5_8u_C1R
ippiLUTPaletteSwap_16u_C3A0C4R	N/A
ippiLUTPaletteSwap_8u_C3A0C4R	N/A
ippiLUTPalette_16u_C3A0C4R	N/A
ippiLUTPalette_8u_C3A0C4R	N/A
ippiLUT_Cubic	Use ippiLUT
ippiLUT_Linear	Use ippiLUT
ippiMagnitudePack_16s_C1RSfs	ippiConvert_16s32f + ippiMagnitudePack_32f
ippiMagnitudePack_16s_C3RSfs	ippiConvert_16s32f + ippiMagnitudePack_32f
ippiMagnitudePack_32s_C1RSfs	ippiConvert_32s32f + ippiMagnitudePack_32f
ippiMagnitudePack_32s_C3RSfs	ippiConvert_32s32f + ippiMagnitudePack_32f
ippiMagnitude_16sc16s_C1RSfs	ippiConvert_16s32f + ippiMagnitude_32f
ippiMagnitude_16sc16s_C3RSfs	ippiConvert_16s32f + ippiMagnitude_32f
ippiMagnitude_16uc16u_C1RSfs	ippiConvert_16u32f + ippiMagnitude_32f

ippiMagnitude_16uc16u_C3RSfs	ippiConvert_16u32f + ippiMagnitude_32f
ippiMagnitude_32sc32s_C1RSfs	ippiConvert_32s32f + ippiMagnitude_32f
ippiMagnitude_32sc32s_C3RSfs	ippiConvert_32s32f + ippiMagnitude_32f
ippiMean_16s_AC4R	ippiMean_16s_C4R
ippiMean_16u_AC4R	ippiMean_16u_C4R
ippiMean_32f_AC4R	ippiMean_32f_C4R
ippiMean_8u_AC4R	ippiMean_8u_C4R
ippiMomentFree_64f	ippiFree
ippiMomentFree_64s	Use Moments_64f
ippiMomentGetStateSize_64s	Use Moments_64f
ippiMomentInitAlloc_64f	ippiMomentsGetSize + ippiMalloc + ippiMomentsInit
ippiMomentInitAlloc_64s	Use Moments_64f
ippiMomentInit_64s	Use Moments_64f
ippiMoments64s_16u_AC4R	Use Moments_64f
ippiMoments64s_16u_C1R	Use Moments_64f
ippiMoments64s_16u_C3R	Use Moments_64f
ippiMoments64s_8u_AC4R	Use Moments_64f
ippiMoments64s_8u_C1R	Use Moments_64f
ippiMoments64s_8u_C3R	Use Moments_64f
ippiMulC_16sc_AC4IRSfs	N/A
ippiMulC_16sc_AC4RSfs	N/A
ippiMulC_16sc_C1IRSfs	N/A
ippiMulC_16sc_C1RSfs	N/A
ippiMulC_16sc_C3IRSfs	N/A
ippiMulC_16sc_C3RSfs	N/A
ippiMulC_32fc_AC4IR	N/A
ippiMulC_32fc_AC4R	N/A
ippiMulC_32fc_C1IR	N/A
ippiMulC_32fc_C1R	N/A
ippiMulC_32fc_C3IR	N/A

ippiMulC_32fc_C3R	N/A
ippiMulC_32sc_AC4IRSfs	N/A
ippiMulC_32sc_AC4RSfs	N/A
ippiMulC_32sc_C1IRSfs	N/A
ippiMulC_32sc_C1RSfs	N/A
ippiMulC_32sc_C3IRSfs	N/A
ippiMulC_32sc_C3RSfs	N/A
ippiMulPack_16s_AC4IRSfs	ippiConvert_16s32f + ippiMulPack_32f
ippiMulPack_16s_AC4RSfs	ippiConvert_16s32f + ippiMulPack_32f
ippiMulPack_16s_C1IRSfs	ippiConvert_16s32f + ippiMulPack_32f
ippiMulPack_16s_C1RSfs	ippiConvert_16s32f + ippiMulPack_32f
ippiMulPack_16s_C3IRSfs	ippiConvert_16s32f + ippiMulPack_32f
ippiMulPack_16s_C3RSfs	ippiConvert_16s32f + ippiMulPack_32f
ippiMulPack_16s_C4IRSfs	ippiConvert_16s32f + ippiMulPack_32f
ippiMulPack_16s_C4RSfs	ippiConvert_16s32f + ippiMulPack_32f
ippiMulPack_32s_AC4IRSfs	ippiConvert_16s32f + ippiMulPack_32f
ippiMulPack_32s_AC4RSfs	ippiConvert_16s32f + ippiMulPack_32f
ippiMulPack_32s_C1IRSfs	ippiConvert_16s32f + ippiMulPack_32f
ippiMulPack_32s_C1RSfs	ippiConvert_16s32f + ippiMulPack_32f
ippiMulPack_32s_C3IRSfs	ippiConvert_16s32f + ippiMulPack_32f
ippiMulPack_32s_C3RSfs	ippiConvert_16s32f + ippiMulPack_32f
ippiMulPack_32s_C4IRSfs	ippiConvert_16s32f + ippiMulPack_32f
ippiMulPack_32s_C4RSfs	ippiConvert_16s32f + ippiMulPack_32f
ippiMul_16sc_AC4IRSfs	N/A
ippiMul_16sc_AC4RSfs	N/A
ippiMul_16sc_C1IRSfs	N/A
ippiMul_16sc_C1RSfs	N/A
ippiMul_16sc_C3IRSfs	N/A
ippiMul_16sc_C3RSfs	N/A
ippiMul_32fc_AC4IR	N/A
ippiMul_32fc_AC4R	N/A

ippiMul_32fc_C1IR	N/A
ippiMul_32fc_C1R	N/A
ippiMul_32fc_C3IR	N/A
ippiMul_32fc_C3R	N/A
ippiMul_32sc_AC4IRSfs	N/A
ippiMul_32sc_AC4RSfs	N/A
ippiMul_32sc_C1IRSfs	N/A
ippiMul_32sc_C1RSfs	N/A
ippiMul_32sc_C3IRSfs	N/A
ippiMul_32sc_C3RSfs	N/A
ippiNormDiff_Inf_16s_AC4R	ippiNormDiff_Inf_16s_C4R
ippiNormDiff_Inf_16u_AC4R	ippiNormDiff_Inf_16u_C4R
ippiNormDiff_Inf_32f_AC4R	ippiNormDiff_Inf_32f_C4R
ippiNormDiff_Inf_8u_AC4R	ippiNormDiff_Inf_8u_C4R
ippiNormDiff_L1_16s_AC4R	ippiNormDiff_L1_16s_C4R
ippiNormDiff_L1_16u_AC4R	ippiNormDiff_L1_16u_C4R
ippiNormDiff_L1_32f_AC4R	ippiNormDiff_L1_32f_C4R
ippiNormDiff_L1_8u_AC4R	ippiNormDiff_L1_8u_C4R
ippiNormDiff_L2_16s_AC4R	ippiNormDiff_L2_16s_C4R
ippiNormDiff_L2_16u_AC4R	ippiNormDiff_L2_16u_C4R
ippiNormDiff_L2_32f_AC4R	ippiNormDiff_L2_32f_C4R
ippiNormDiff_L2_8u_AC4R	ippiNormDiff_L2_8u_C4R
ippiNormRel_Inf_16s_AC4R	ippiNormRel_Inf_16s_C4R
ippiNormRel_Inf_16u_AC4R	ippiNormRel_Inf_16u_C4R
ippiNormRel_Inf_32f_AC4R	ippiNormRel_Inf_32f_C4R
ippiNormRel_Inf_8u_AC4R	ippiNormRel_Inf_8u_C4R
ippiNormRel_L1_16s_AC4R	ippiNormRel_L1_16s_C4R
ippiNormRel_L1_16u_AC4R	ippiNormRel_L1_16u_C4R
ippiNormRel_L1_32f_AC4R	ippiNormRel_L1_32f_C4R
ippiNormRel_L1_8u_AC4R	ippiNormRel_L1_8u_C4R
ippiNormRel_L2_16s_AC4R	ippiNormRel_L2_16s_C4R

ippiNormRel_L2_16u_AC4R	ippiNormRel_L2_16u_C4R
ippiNormRel_L2_32f_AC4R	ippiNormRel_L2_32f_C4R
ippiNormRel_L2_8u_AC4R	ippiNormRel_L2_8u_C4R
ippiNorm_Inf_16s_AC4R	ippiNorm_Inf_16s_C4R
ippiNorm_Inf_16u_AC4R	ippiNorm_Inf_16u_C4R
ippiNorm_Inf_32f_AC4R	ippiNorm_Inf_32f_C4R
ippiNorm_Inf_32s_C1R	ippiConvert_32s32f + ippiNorm_Inf_32f
ippiNorm_Inf_8u_AC4R	ippiNorm_Inf_8u_C4R
ippiNorm_L1_16s_AC4R	ippiNorm_L1_16s_C4R
ippiNorm_L1_16u_AC4R	ippiNorm_L1_16u_C4R
ippiNorm_L1_32f_AC4R	ippiNorm_L1_32f_C4R
ippiNorm_L1_8u_AC4R	ippiNorm_L1_8u_C4R
ippiNorm_L2_16s_AC4R	ippiNorm_L2_16s_C4R
ippiNorm_L2_16u_AC4R	ippiNorm_L2_16u_C4R
ippiNorm_L2_32f_AC4R	ippiNorm_L2_32f_C4R
ippiNorm_L2_8u_AC4R	ippiNorm_L2_8u_C4R
ippiPackToCplxExtend_32s32sc_C1R	ippiConvert_32s32f + ippiPackToCplxExtend_32f
ippiPhasePack_16s_C1RSfs	ippiConvert_16s32f + ippiPhasePack_32f
ippiPhasePack_16s_C3RSfs	ippiConvert_16s32f + ippiPhasePack_32f
ippiPhasePack_32s_C1RSfs	ippiConvert_32s32f + ippiPhasePack_32f
ippiPhasePack_32s_C3RSfs	ippiConvert_32s32f + ippiPhasePack_32f
ippiPhase_16sc16s_C1RSfs	ippiConvert_16s32f + ippiPhase_32f
ippiPhase_16sc16s_C3RSfs	ippiConvert_16s32f + ippiPhase_32f
ippiPhase_16uc16u_C1RSfs	ippiConvert_16u32f + ippiPhase_32f
ippiPhase_16uc16u_C3RSfs	ippiConvert_16u32f + ippiPhase_32f
ippiPhase_32sc32s_C1RSfs	ippiConvert_32s32f + ippiPhase_32f
ippiPhase_32sc32s_C3RSfs	ippiConvert_32s32f + ippiPhase_32f
ippiPolarToCart_16sc_C1R	ippiConvert_16s32f + ippiPolarToCart_32fc_C1R
ippiPolarToCart_16sc_C3R	ippiConvert_16s32f + ippiPolarToCart_32fc_C3R

ippiPolarToCart_32f32fc_P2C1R	Use ippsCplxToReal_32fc + ippsPolarToCart_32fc with loop by row
ippiPolarToCart_32sc_C1R	ippiConvert_32s32f + ippiPolarToCart_32fc_C1R
ippiPolarToCart_32sc_C3R	ippiConvert_32s32f + ippiPolarToCart_32fc_C3R
ippiRShiftC_8s_AC4IR	Convert to other data type
ippiRShiftC_8s_AC4R	Convert to other data type
ippiRShiftC_8s_C1IR	Convert to other data type
ippiRShiftC_8s_C1R	Convert to other data type
ippiRShiftC_8s_C3IR	Convert to other data type
ippiRShiftC_8s_C3R	Convert to other data type
ippiRShiftC_8s_C4IR	Convert to other data type
ippiRShiftC_8s_C4R	Convert to other data type
ippiRemap_16s_P3R	Use ippiRemap_XX_C1R for each plane
ippiRemap_16s_P4R	Use ippiRemap_XX_C1R for each plane
ippiRemap_16u_P3R	Use ippiRemap_XX_C1R for each plane
ippiRemap_16u_P4R	Use ippiRemap_XX_C1R for each plane
ippiRemap_32f_P3R	Use ippiRemap_XX_C1R for each plane
ippiRemap_32f_P4R	Use ippiRemap_XX_C1R for each plane
ippiRemap_64f_P3R	Use ippiRemap_XX_C1R for each plane
ippiRemap_64f_P4R	Use ippiRemap_XX_C1R for each plane
ippiRemap_8u_P3R	Use ippiRemap_XX_C1R for each plane
ippiRemap_8u_P4R	Use ippiRemap_XX_C1R for each plane
ippiResampleRowGetBorderWidth_32f	N/A
ippiResampleRowGetSize_32f	N/A
ippiResampleRowInit_32f	N/A
ippiResampleRowReplicateBorder_32f_C1R	N/A
ippiResampleRow_32f_C1	N/A
ippiResizeGetBufSize	N/A
ippiResizeGetBufSize_64f	N/A
ippiResizeSqrPixel_16s_AC4R	N/A

ippiResizeSqrPixel_16s_C1R	N/A
ippiResizeSqrPixel_16s_C3R	N/A
ippiResizeSqrPixel_16s_C4R	N/A
ippiResizeSqrPixel_16s_P3R	Use C1R flavor for each plane
ippiResizeSqrPixel_16s_P4R	Use C1R flavor for each plane
ippiResizeSqrPixel_16u_AC4R	N/A
ippiResizeSqrPixel_16u_C1R	N/A
ippiResizeSqrPixel_16u_C3R	N/A
ippiResizeSqrPixel_16u_C4R	N/A
ippiResizeSqrPixel_16u_P3R	Use C1R flavor for each plane
ippiResizeSqrPixel_16u_P4R	Use C1R flavor for each plane
ippiResizeSqrPixel_32f_AC4R	N/A
ippiResizeSqrPixel_32f_C1R	N/A
ippiResizeSqrPixel_32f_C3R	N/A
ippiResizeSqrPixel_32f_C4R	N/A
ippiResizeSqrPixel_32f_P3R	Use C1R flavor for each plane
ippiResizeSqrPixel_32f_P4R	Use C1R flavor for each plane
ippiResizeSqrPixel_64f_AC4R	N/A
ippiResizeSqrPixel_64f_C1R	N/A
ippiResizeSqrPixel_64f_C3R	N/A
ippiResizeSqrPixel_64f_C4R	N/A
ippiResizeSqrPixel_64f_P3R	Use C1R flavor for each plane
ippiResizeSqrPixel_64f_P4R	Use C1R flavor for each plane
ippiResizeSqrPixel_8u_AC4R	N/A
ippiResizeSqrPixel_8u_C1R	N/A
ippiResizeSqrPixel_8u_C3R	N/A
ippiResizeSqrPixel_8u_C4R	N/A
ippiResizeSqrPixel_8u_P3R	Use C1R flavor for each plane
ippiResizeSqrPixel_8u_P4R	Use C1R flavor for each plane
ippiResizeYUV420GetBufSize	N/A
ippiResizeYUV420_8u_P2R	N/A

ippiResizeYUV422_8u_C2R	N/A
ippiRotateCenter_16u_AC4R	See the RotateCenter.c example.
ippiRotateCenter_16u_C1R	See the RotateCenter.c example.
ippiRotateCenter_16u_C3R	See the RotateCenter.c example.
ippiRotateCenter_16u_C4R	See the RotateCenter.c example.
ippiRotateCenter_16u_P3R	See the RotateCenter.c example.
ippiRotateCenter_16u_P4R	See the RotateCenter.c example.
ippiRotateCenter_32f_AC4R	See the RotateCenter.c example.
ippiRotateCenter_32f_C1R	See the RotateCenter.c example.
ippiRotateCenter_32f_C3R	See the RotateCenter.c example.
ippiRotateCenter_32f_C4R	See the RotateCenter.c example.
ippiRotateCenter_32f_P3R	See the RotateCenter.c example.
ippiRotateCenter_32f_P4R	See the RotateCenter.c example.
ippiRotateCenter_64f_AC4R	See the RotateCenter.c example.
ippiRotateCenter_64f_C1R	See the RotateCenter.c example.
ippiRotateCenter_64f_C3R	See the RotateCenter.c example.
ippiRotateCenter_64f_C4R	See the RotateCenter.c example.
ippiRotateCenter_64f_P3R	See the RotateCenter.c example.
ippiRotateCenter_64f_P4R	See the RotateCenter.c example.
ippiRotateCenter_8u_AC4R	See the RotateCenter.c example.
ippiRotateCenter_8u_C1R	See the RotateCenter.c example.
ippiRotateCenter_8u_C3R	See the RotateCenter.c example.
ippiRotateCenter_8u_C4R	See the RotateCenter.c example.
ippiRotateCenter_8u_P3R	See the RotateCenter.c example.
ippiRotateCenter_8u_P4R	See the RotateCenter.c example.
ippiRotate_16u_AC4R	See the Rotate.c example.
ippiRotate_16u_C1R	See the Rotate.c example.
ippiRotate_16u_C3R	See the Rotate.c example.
ippiRotate_16u_C4R	See the Rotate.c example.
ippiRotate_16u_P3R	See the Rotate.c example.
ippiRotate_16u_P4R	See the Rotate.c example.

ippiRotate_32f_AC4R	See the Rotate.c example.
ippiRotate_32f_C1R	See the Rotate.c example.
ippiRotate_32f_C3R	See the Rotate.c example.
ippiRotate_32f_C4R	See the Rotate.c example.
ippiRotate_32f_P3R	See the Rotate.c example.
ippiRotate_32f_P4R	See the Rotate.c example.
ippiRotate_64f_AC4R	See the Rotate.c example.
ippiRotate_64f_C1R	See the Rotate.c example.
ippiRotate_64f_C3R	See the Rotate.c example.
ippiRotate_64f_C4R	See the Rotate.c example.
ippiRotate_64f_P3R	See the Rotate.c example.
ippiRotate_64f_P4R	See the Rotate.c example.
ippiRotate_8u_AC4R	See the Rotate.c example.
ippiRotate_8u_C1R	See the Rotate.c example.
ippiRotate_8u_C3R	See the Rotate.c example.
ippiRotate_8u_C4R	See the Rotate.c example.
ippiRotate_8u_P3R	See the Rotate.c example.
ippiRotate_8u_P4R	See the Rotate.c example.
ippiSSIM_32f_C1R	N/A
ippiShear_16u_AC4R	N/A
ippiShear_16u_C1R	N/A
ippiShear_16u_C3R	N/A
ippiShear_16u_C4R	N/A
ippiShear_16u_P3R	Use C1R flavor for each plane
ippiShear_16u_P4R	Use C1R flavor for each plane
ippiShear_32f_AC4R	N/A
ippiShear_32f_C1R	N/A
ippiShear_32f_C3R	N/A
ippiShear_32f_C4R	N/A
ippiShear_32f_P3R	Use C1R flavor for each plane
ippiShear_32f_P4R	Use C1R flavor for each plane

ippiShear_64f_AC4R	N/A
ippiShear_64f_C1R	N/A
ippiShear_64f_C3R	N/A
ippiShear_64f_C4R	N/A
ippiShear_64f_P3R	Use C1R flavor for each plane
ippiShear_64f_P4R	Use C1R flavor for each plane
ippiShear_8u_AC4R	N/A
ippiShear_8u_C1R	N/A
ippiShear_8u_C3R	N/A
ippiShear_8u_C4R	N/A
ippiShear_8u_P3R	Use C1R flavor for each plane
ippiShear_8u_P4R	Use C1R flavor for each plane
ippiSqrDistanceFull_Norm_16u32f_AC4R	ippiCopy_16u_C4C1R + ippiSqrDistanceNorm_16u32f_C1R + algType = ippiROIValid | ippiNorm
ippiSqrDistanceFull_Norm_16u32f_C1R	ippiSqrDistanceNorm_16u32f_C1R + algType = ippiROIValid | ippiNorm
ippiSqrDistanceFull_Norm_16u32f_C3R	ippiCopy_16u_C3C1R + ippiSqrDistanceNorm_16u32f_C1R + algType = ippiROIValid | ippiNorm
ippiSqrDistanceFull_Norm_16u32f_C4R	ippiCopy_16u_C4C1R + ippiSqrDistanceNorm_16u32f_C1R + algType = ippiROIValid | ippiNorm
ippiSqrDistanceFull_Norm_32f_AC4R	ippiCopy_32f_C4C1R + ippiSqrDistanceNorm_32f_C1R + algType = ippiROIFull | ippiNorm
ippiSqrDistanceFull_Norm_32f_C1R	ippiSqrDistanceNorm_32f_C1R + algType = ippiROIFull | ippiNorm
ippiSqrDistanceFull_Norm_32f_C3R	ippiCopy_32f_C3C1R + ippiSqrDistanceNorm_32f_C1R + algType = ippiROIFull | ippiNorm
ippiSqrDistanceFull_Norm_32f_C4R	ippiCopy_32f_C4C1R + ippiSqrDistanceNorm_32f_C1R + algType = ippiROIFull | ippiNorm
ippiSqrDistanceFull_Norm_8s32f_AC4R	ippiConvert_8s32f + ippiCopy_32f_C4C1R + ippiSqrDistanceNorm_32f_C1R + algType = ippiROIFull | ippiNorm

ippiSqrDistanceFull_Norm_8s32f_C1R	ippiConvert_8s32f + ippiSqrDistanceNorm_32f_C1R + algType = ippiROIFull | ippiNorm
ippiSqrDistanceFull_Norm_8s32f_C3R	ippiConvert_8s32f + ippiCopy_32f_C3C1R + ippiSqrDistanceNorm_32f_C1R + algType = ippiROIFull | ippiNorm
ippiSqrDistanceFull_Norm_8s32f_C4R	ippiConvert_8s32f + ippiCopy_32f_C4C1R + ippiSqrDistanceNorm_32f_C1R + algType = ippiROIFull | ippiNorm
ippiSqrDistanceFull_Norm_8u32f_AC4R	ippiCopy_8u_C4C1R + ippiSqrDistanceNorm_8u32f_C1R + algType = ippiROIFull | ippiNorm
ippiSqrDistanceFull_Norm_8u32f_C1R	ippiSqrDistanceNorm_8u32f_C1R + algType = ippiROIFull | ippiNorm
ippiSqrDistanceFull_Norm_8u32f_C3R	ippiCopy_8u_C3C1R + ippiSqrDistanceNorm_8u32f_C1R + algType = ippiROIFull | ippiNorm
ippiSqrDistanceFull_Norm_8u32f_C4R	ippiCopy_8u_C4C1R + ippiSqrDistanceNorm_8u32f_C1R + algType = ippiROISame | ippiNorm
ippiSqrDistanceFull_Norm_8u_AC4RSfs	ippiCopy_8u_C4C1R + ippiSqrDistanceNorm_8u_C1RSfs + algType = ippiROIFull | ippiNorm
ippiSqrDistanceFull_Norm_8u_C1RSfs	ippiSqrDistanceNorm_8u_C1RSfs + algType = ippiROIFull | ippiNorm
ippiSqrDistanceFull_Norm_8u_C3RSfs	ippiCopy_8u_C3C1R + ippiSqrDistanceNorm_8u_C1RSfs + algType = ippiROIFull | ippiNorm
ippiSqrDistanceFull_Norm_8u_C4RSfs	ippiCopy_8u_C4C1R + ippiSqrDistanceNorm_8u_C1RSfs + algType = ippiROIValid | ippiNorm
ippiSqrDistanceSame_Norm_16u32f_AC4R	ippiCopy_16u_C4C1R + ippiSqrDistanceNorm_16u32f_C1R + algType = ippiROISame | ippiNorm
ippiSqrDistanceSame_Norm_16u32f_C1R	ippiSqrDistanceNorm_16u32f_C1R + algType = ippiROISame | ippiNorm
ippiSqrDistanceSame_Norm_16u32f_C3R	ippiCopy_16u_C3C1R + ippiSqrDistanceNorm_16u32f_C1R + algType = ippiROISame | ippiNorm

ippiSqrDistanceSame_Norm_16u32f_C4R	ippiCopy_16u_C4C1R + ippiSqrDistanceNorm_16u32f_C1R + algType = ippiROISame | ippiNorm
ippiSqrDistanceSame_Norm_32f_AC4R	ippiCopy_32f_C4C1R + ippiSqrDistanceNorm_32f_C1R + algType = ippiROISame | ippiNorm
ippiSqrDistanceSame_Norm_32f_C1R	ippiSqrDistanceNorm_32f_C1R + algType = ippiROISame | ippiNorm
ippiSqrDistanceSame_Norm_32f_C3R	ippiCopy_32f_C3C1R + ippiSqrDistanceNorm_32f_C1R + algType = ippiROISame | ippiNorm
ippiSqrDistanceSame_Norm_32f_C4R	ippiCopy_32f_C4C1R + ippiSqrDistanceNorm_32f_C1R + algType = ippiROISame | ippiNorm
ippiSqrDistanceSame_Norm_8s32f_AC4R	ippiConvert_8s32f + ippiCopy_32f_C4C1R + ippiSqrDistanceNorm_32f_C1R + algType = ippiROISame | ippiNorm
ippiSqrDistanceSame_Norm_8s32f_C1R	ippiConvert_8s32f + ippiSqrDistanceNorm_32f_C1R + algType = ippiROISame | ippiNorm
ippiSqrDistanceSame_Norm_8s32f_C3R	ippiConvert_8s32f + ippiCopy_32f_C3C1R + ippiSqrDistanceNorm_32f_C1R + algType = ippiROISame | ippiNorm
ippiSqrDistanceSame_Norm_8s32f_C4R	ippiConvert_8s32f + ippiCopy_32f_C4C1R + ippiSqrDistanceNorm_32f_C1R + algType = ippiROISame | ippiNorm
ippiSqrDistanceSame_Norm_8u32f_AC4R	ippiCopy_8u_C4C1R + ippiSqrDistanceNorm_8u32f_C1R + algType = ippiROISame | ippiNorm
ippiSqrDistanceSame_Norm_8u32f_C1R	ippiSqrDistanceNorm_8u32f_C1R + algType = ippiROISame | ippiNorm
ippiSqrDistanceSame_Norm_8u32f_C3R	ippiCopy_8u_C3C1R + ippiSqrDistanceNorm_8u32f_C1R + algType = ippiROISame | ippiNorm
ippiSqrDistanceSame_Norm_8u32f_C4R	ippiCopy_8u_C4C1R + ippiSqrDistanceNorm_8u32f_C1R + algType = ippiROISame | ippiNorm
ippiSqrDistanceSame_Norm_8u_AC4RSfs	ippiCopy_8u_C4C1R + ippiSqrDistanceNorm_8u_C1RSfs + algType = ippiROISame | ippiNorm

ippiSqrDistanceSame_Norm_8u_C1RSfs	ippiSqrDistanceNorm_8u_C1RSfs + algType = ippiROISame | ippiNorm
ippiSqrDistanceSame_Norm_8u_C3RSfs	ippiCopy_8u_C3C1R + ippiSqrDistanceNorm_8u_C1RSfs + algType = ippiROISame | ippiNorm
ippiSqrDistanceSame_Norm_8u_C4RSfs	ippiCopy_8u_C4C1R + ippiSqrDistanceNorm_8u_C1RSfs + algType = ippiROISame | ippiNorm
ippiSqrDistanceValid_Norm_16u32f_AC4R	ippiCopy_16u_C4C1R + ippiSqrDistanceNorm_16u32f_C1R + algType = ippiROIValid | ippiNorm
ippiSqrDistanceValid_Norm_16u32f_C1R	ippiSqrDistanceNorm_16u32f_C1R + algType = ippiROIValid | ippiNorm
ippiSqrDistanceValid_Norm_16u32f_C3R	ippiCopy_16u_C3C1R + ippiSqrDistanceNorm_16u32f_C1R + algType = ippiROIValid | ippiNorm
ippiSqrDistanceValid_Norm_16u32f_C4R	ippiCopy_16u_C4C1R + ippiSqrDistanceNorm_16u32f_C1R + algType = ippiROIValid | ippiNorm
ippiSqrDistanceValid_Norm_32f_AC4R	ippiCopy_32f_C4C1R + ippiSqrDistanceNorm_32f_C1R + algType = ippiROIValid | ippiNorm
ippiSqrDistanceValid_Norm_32f_C1R	ippiSqrDistanceNorm_32f_C1R + algType = ippiROIValid | ippiNorm
ippiSqrDistanceValid_Norm_32f_C3R	ippiCopy_32f_C3C1R + ippiSqrDistanceNorm_32f_C1R + algType = ippiROIValid | ippiNorm
ippiSqrDistanceValid_Norm_32f_C4R	ippiCopy_32f_C4C1R + ippiSqrDistanceNorm_32f_C1R + algType = ippiROIValid | ippiNorm
ippiSqrDistanceValid_Norm_8s32f_AC4R	ippiConvert_8s32f + ippiCopy_32f_C4C1R + ippiSqrDistanceNorm_32f_C1R + algType = ippiROIValid | ippiNorm
ippiSqrDistanceValid_Norm_8s32f_C1R	ippiConvert_8s32f + ippiSqrDistanceNorm_32f_C1R + algType = ippiROIValid | ippiNorm
ippiSqrDistanceValid_Norm_8s32f_C3R	ippiConvert_8s32f + ippiCopy_32f_C3C1R + ippiSqrDistanceNorm_32f_C1R + algType = ippiROIValid | ippiNorm

ippiSqrDistanceValid_Norm_8s32f_C4R	ippiConvert_8s32f + ippiCopy_32f_C4C1R + ippiSqrDistanceNorm_32f_C1R + algType = ippiROIValid | ippiNorm
ippiSqrDistanceValid_Norm_8u32f_AC4R	ippiCopy_8u_C4C1R + ippiSqrDistanceNorm_8u32f_C1R + algType = ippiROIValid | ippiNorm
ippiSqrDistanceValid_Norm_8u32f_C1R	ippiSqrDistanceNorm_8u32f_C1R + algType = ippiROIValid | ippiNorm
ippiSqrDistanceValid_Norm_8u32f_C3R	ippiCopy_8u_C3C1R + ippiSqrDistanceNorm_8u32f_C1R + algType = ippiROIValid | ippiNorm
ippiSqrDistanceValid_Norm_8u32f_C4R	ippiCopy_8u_C4C1R + ippiSqrDistanceNorm_8u32f_C1R + algType = ippiROISame | ippiNorm
ippiSqrDistanceValid_Norm_8u_AC4RSfs	ippiCopy_8u_C4C1R + ippiSqrDistanceNorm_8u_C1RSfs + algType = ippiROIValid | ippiNorm
ippiSqrDistanceValid_Norm_8u_C1RSfs	ippiSqrDistanceNorm_8u_C1RSfs + algType = ippiROIValid | ippiNorm
ippiSqrDistanceValid_Norm_8u_C3RSfs	ippiCopy_8u_C3C1R + ippiSqrDistanceNorm_8u_C1RSfs + algType = ippiROIValid | ippiNorm
ippiSqrDistanceValid_Norm_8u_C4RSfs	ippiCopy_8u_C4C1R + ippiSqrDistanceNorm_8u_C1RSfs + algType = ippiROIValid | ippiNorm
ippiSubC_16sc_AC4IRSfs	N/A
ippiSubC_16sc_AC4RSfs	N/A
ippiSubC_16sc_C1IRSfs	N/A
ippiSubC_16sc_C1RSfs	N/A
ippiSubC_16sc_C3IRSfs	N/A
ippiSubC_16sc_C3RSfs	N/A
ippiSubC_32fc_AC4IR	N/A
ippiSubC_32fc_AC4R	N/A
ippiSubC_32fc_C1IR	N/A
ippiSubC_32fc_C1R	N/A
ippiSubC_32fc_C3IR	N/A
ippiSubC_32fc_C3R	N/A

ippiSubC_32sc_AC4IRSfs	N/A
ippiSubC_32sc_AC4RSfs	N/A
ippiSubC_32sc_C1IRSfs	N/A
ippiSubC_32sc_C1RSfs	N/A
ippiSubC_32sc_C3IRSfs	N/A
ippiSubC_32sc_C3RSfs	N/A
ippiSub_16sc_AC4IRSfs	N/A
ippiSub_16sc_AC4RSfs	N/A
ippiSub_16sc_C1IRSfs	N/A
ippiSub_16sc_C1RSfs	N/A
ippiSub_16sc_C3IRSfs	N/A
ippiSub_16sc_C3RSfs	N/A
ippiSub_32fc_AC4IR	N/A
ippiSub_32fc_AC4R	N/A
ippiSub_32fc_C1IR	N/A
ippiSub_32fc_C1R	N/A
ippiSub_32fc_C3IR	N/A
ippiSub_32fc_C3R	N/A
ippiSub_32sc_AC4IRSfs	N/A
ippiSub_32sc_AC4RSfs	N/A
ippiSub_32sc_C1IRSfs	N/A
ippiSub_32sc_C1RSfs	N/A
ippiSub_32sc_C3IRSfs	N/A
ippiSub_32sc_C3RSfs	N/A
ippiSum_16s_AC4R	ippiSum_16s_C4R
ippiSum_16u_AC4R	ippiSum_16u_C4R
ippiSum_32f_AC4R	ippiSum_32f_C4R
ippiSum_8u_AC4R	ippiSum_8u_C4R
ippiSuperSamplingGetBufSize	N/A
ippiSuperSampling_16s_AC4R	N/A
ippiSuperSampling_16s_C1R	N/A

ippiSuperSampling_16s_C3R	N/A
ippiSuperSampling_16s_C4R	N/A
ippiSuperSampling_16s_P3R	Use C1R flavor for each plane
ippiSuperSampling_16s_P4R	Use C1R flavor for each plane
ippiSuperSampling_16u_AC4R	N/A
ippiSuperSampling_16u_C1R	N/A
ippiSuperSampling_16u_C3R	N/A
ippiSuperSampling_16u_C4R	N/A
ippiSuperSampling_16u_P3R	Use C1R flavor for each plane
ippiSuperSampling_16u_P4R	Use C1R flavor for each plane
ippiSuperSampling_32f_AC4R	N/A
ippiSuperSampling_32f_C1R	N/A
ippiSuperSampling_32f_C3R	N/A
ippiSuperSampling_32f_C4R	N/A
ippiSuperSampling_32f_P3R	Use C1R flavor for each plane
ippiSuperSampling_32f_P4R	Use C1R flavor for each plane
ippiSuperSampling_8u_AC4R	N/A
ippiSuperSampling_8u_C1R	N/A
ippiSuperSampling_8u_C3R	N/A
ippiSuperSampling_8u_C4R	N/A
ippiSuperSampling_8u_P3R	Use C1R flavor for each plane
ippiSuperSampling_8u_P4R	Use C1R flavor for each plane
ippiWTFwdFree_32f_C1R	ippiFree
ippiWTFwdFree_32f_C3R	ippiFree
ippiWTFwdGetBufSize_C1R	N/A
ippiWTFwdGetBufSize_C3R	N/A
ippiWTFwdInitAlloc_32f_C1R	ippiWTFwdGetSize + ippiMalloc + ippiWTFwdInit
ippiWTFwdInitAlloc_32f_C3R	ippiWTFwdGetSize + ippiMalloc + ippiWTFwdInit
ippiWTInvFree_32f_C1R	ippiFree

ippiWTInvFree_32f_C3R	ippiFree
ippiWTInvGetBufSize_C1R	N/A
ippiWTInvGetBufSize_C3R	N/A
ippiWTInvInitAlloc_32f_C1R	ippiWTInvGetSize + ippiMalloc + ippiWTInvInit
ippiWTInvInitAlloc_32f_C3R	ippiWTInvGetSize + ippiMalloc + ippiWTInvInit
ippiWarpAffineBack_16u_AC4R	Use new **WarpAffine`** functions
ippiWarpAffineBack_16u_C1R	Use new **WarpAffine`** functions
ippiWarpAffineBack_16u_C3R	Use new **WarpAffine`** functions
ippiWarpAffineBack_16u_C4R	Use new **WarpAffine`** functions
ippiWarpAffineBack_16u_P3R	Use new **WarpAffine`** functions
ippiWarpAffineBack_16u_P4R	Use new **WarpAffine`** functions
ippiWarpAffineBack_32f_AC4R	Use new **WarpAffine`** functions
ippiWarpAffineBack_32f_C1R	Use new **WarpAffine`** functions
ippiWarpAffineBack_32f_C3R	Use new **WarpAffine`** functions
ippiWarpAffineBack_32f_C4R	Use new **WarpAffine`** functions
ippiWarpAffineBack_32f_P3R	Use new **WarpAffine`** functions
ippiWarpAffineBack_32f_P4R	Use new **WarpAffine`** functions
ippiWarpAffineBack_8u_AC4R	Use new **WarpAffine`** functions
ippiWarpAffineBack_8u_C1R	Use new **WarpAffine`** functions
ippiWarpAffineBack_8u_C3R	Use new **WarpAffine`** functions
ippiWarpAffineBack_8u_C4R	Use new **WarpAffine`** functions
ippiWarpAffineBack_8u_P3R	Use new **WarpAffine`** functions
ippiWarpAffineBack_8u_P4R	Use new **WarpAffine`** functions
ippiWarpAffineQuad_16u_AC4R	Use new WarpQuad functions.
ippiWarpAffineQuad_16u_C1R	Use new WarpQuad functions.
ippiWarpAffineQuad_16u_C3R	Use new WarpQuad functions.
ippiWarpAffineQuad_16u_C4R	Use new WarpQuad functions.
ippiWarpAffineQuad_16u_P3R	Use new WarpQuad functions.
ippiWarpAffineQuad_16u_P4R	Use new WarpQuad functions.

ippiWarpAffineQuad_32f_AC4R	Use new WarpQuad functions.
ippiWarpAffineQuad_32f_C1R	Use new WarpQuad functions.
ippiWarpAffineQuad_32f_C3R	Use new WarpQuad functions.
ippiWarpAffineQuad_32f_C4R	Use new WarpQuad functions.
ippiWarpAffineQuad_32f_P3R	Use new WarpQuad functions.
ippiWarpAffineQuad_32f_P4R	Use new WarpQuad functions.
ippiWarpAffineQuad_8u_AC4R	Use new WarpQuad functions.
ippiWarpAffineQuad_8u_C1R	Use new WarpQuad functions.
ippiWarpAffineQuad_8u_C3R	Use new WarpQuad functions.
ippiWarpAffineQuad_8u_C4R	Use new WarpQuad functions.
ippiWarpAffineQuad_8u_P3R	Use new WarpQuad functions.
ippiWarpAffineQuad_8u_P4R	Use new WarpQuad functions.
ippiWarpAffine_16u_AC4R	Use new **WarpAffine`** functions
ippiWarpAffine_16u_C1R	Use new **WarpAffine`** functions
ippiWarpAffine_16u_C3R	Use new **WarpAffine`** functions
ippiWarpAffine_16u_C4R	Use new **WarpAffine`** functions
ippiWarpAffine_16u_P3R	Use new **WarpAffine`** functions
ippiWarpAffine_16u_P4R	Use new **WarpAffine`** functions
ippiWarpAffine_32f_AC4R	Use new **WarpAffine`** functions
ippiWarpAffine_32f_C1R	Use new **WarpAffine`** functions
ippiWarpAffine_32f_C3R	Use new **WarpAffine`** functions
ippiWarpAffine_32f_C4R	Use new **WarpAffine`** functions
ippiWarpAffine_32f_P3R	Use new **WarpAffine`** functions
ippiWarpAffine_32f_P4R	Use new **WarpAffine`** functions
ippiWarpAffine_64f_AC4R	Use new **WarpAffine`** functions
ippiWarpAffine_64f_C1R	Use new **WarpAffine`** functions
ippiWarpAffine_64f_C3R	Use new **WarpAffine`** functions
ippiWarpAffine_64f_C4R	Use new **WarpAffine`** functions
ippiWarpAffine_64f_P3R	Use new **WarpAffine`** functions
ippiWarpAffine_64f_P4R	Use new **WarpAffine`** functions
ippiWarpAffine_8u_AC4R	Use new **WarpAffine`** functions

ippiWarpAffine_8u_C1R	Use new **WarpAffine`** functions
ippiWarpAffine_8u_C3R	Use new **WarpAffine`** functions
ippiWarpAffine_8u_C4R	Use new **WarpAffine`** functions
ippiWarpAffine_8u_P3R	Use new **WarpAffine`** functions
ippiWarpAffine_8u_P4R	Use new **WarpAffine`** functions
ippiWarpBilinearBack_16u_AC4R	N/A
ippiWarpBilinearBack_16u_P3R	Use C1R flavor for each plane
ippiWarpBilinearBack_16u_P4R	Use C1R flavor for each plane
ippiWarpBilinearBack_32f_AC4R	N/A
ippiWarpBilinearBack_32f_P3R	Use C1R flavor for each plane
ippiWarpBilinearBack_32f_P4R	Use C1R flavor for each plane
ippiWarpBilinearBack_8u_AC4R	N/A
ippiWarpBilinearBack_8u_P3R	Use C1R flavor for each planeN/A
ippiWarpBilinearBack_8u_P4R	Use C1R flavor for each plane
ippiWarpBilinearQuad_16u_AC4R	Use new `WarpQuad` functions.
ippiWarpBilinearQuad_16u_P3R	Use new `WarpQuad` functions.
ippiWarpBilinearQuad_16u_P4R	Use new `WarpQuad` functions.
ippiWarpBilinearQuad_32f_AC4R	Use new `WarpQuad` functions.
ippiWarpBilinearQuad_32f_P3R	Use new `WarpQuad` functions.
ippiWarpBilinearQuad_32f_P4R	Use new `WarpQuad` functions.
ippiWarpBilinearQuad_8u_AC4R	Use new `WarpQuad` functions.
ippiWarpBilinearQuad_8u_P3R	Use new `WarpQuad` functions.
ippiWarpBilinearQuad_8u_P4R	Use new `WarpQuad` functions.
ippiWarpBilinear_16u_AC4R	N/A
ippiWarpBilinear_16u_P3R	Use C1R flavor for each plane
ippiWarpBilinear_16u_P4R	Use C1R flavor for each plane
ippiWarpBilinear_32f_AC4R	N/A
ippiWarpBilinear_32f_P3R	Use C1R flavor for each plane
ippiWarpBilinear_32f_P4R	Use C1R flavor for each plane
ippiWarpBilinear_8u_AC4R	N/A
ippiWarpBilinear_8u_P3R	Use C1R flavor for each plane

ippiWarpBilinear_8u_P4R	Use C1R flavor for each plane
ippiWarpPerspectiveBack_16u_AC4R	N/A
ippiWarpPerspectiveBack_16u_C1R	N/A
ippiWarpPerspectiveBack_16u_C3R	N/A
ippiWarpPerspectiveBack_16u_C4R	N/A
ippiWarpPerspectiveBack_16u_P3R	Use C1R flavor for each plane
ippiWarpPerspectiveBack_16u_P4R	Use C1R flavor for each plane
ippiWarpPerspectiveBack_32f_AC4R	N/A
ippiWarpPerspectiveBack_32f_C1R	N/A
ippiWarpPerspectiveBack_32f_C3R	N/A
ippiWarpPerspectiveBack_32f_C4R	N/A
ippiWarpPerspectiveBack_32f_P3R	Use C1R flavor for each plane
ippiWarpPerspectiveBack_32f_P4R	Use C1R flavor for each plane
ippiWarpPerspectiveBack_8u_AC4R	N/A
ippiWarpPerspectiveBack_8u_C1R	N/A
ippiWarpPerspectiveBack_8u_C3R	N/A
ippiWarpPerspectiveBack_8u_C4R	N/A
ippiWarpPerspectiveBack_8u_P3R	Use C1R flavor for each plane
ippiWarpPerspectiveBack_8u_P4R	Use C1R flavor for each plane
ippiWarpPerspectiveQuad_16u_AC4R	Use new WarpQuad functions.
ippiWarpPerspectiveQuad_16u_C1R	Use new WarpQuad functions.
ippiWarpPerspectiveQuad_16u_C3R	Use new WarpQuad functions.
ippiWarpPerspectiveQuad_16u_C4R	Use new WarpQuad functions.
ippiWarpPerspectiveQuad_16u_P3R	Use new WarpQuad functions.
ippiWarpPerspectiveQuad_16u_P4R	Use new WarpQuad functions.
ippiWarpPerspectiveQuad_32f_AC4R	Use new WarpQuad functions.
ippiWarpPerspectiveQuad_32f_C1R	Use new WarpQuad functions.
ippiWarpPerspectiveQuad_32f_C3R	Use new WarpQuad functions.
ippiWarpPerspectiveQuad_32f_C4R	Use new WarpQuad functions.
ippiWarpPerspectiveQuad_32f_P3R	Use new WarpQuad functions.
ippiWarpPerspectiveQuad_32f_P4R	Use new WarpQuad functions.

ippiWarpPerspectiveQuad_8u_AC4R	Use new WarpQuad functions.
ippiWarpPerspectiveQuad_8u_C1R	Use new WarpQuad functions.
ippiWarpPerspectiveQuad_8u_C3R	Use new WarpQuad functions.
ippiWarpPerspectiveQuad_8u_C4R	Use new WarpQuad functions.
ippiWarpPerspectiveQuad_8u_P3R	Use new WarpQuad functions.
ippiWarpPerspectiveQuad_8u_P4R	Use new WarpQuad functions.
ippiWarpPerspective_16u_AC4R	Use new WarpPerspective functions.
ippiWarpPerspective_16u_C1R	Use new WarpPerspective functions.
ippiWarpPerspective_16u_C3R	Use new WarpPerspective functions.
ippiWarpPerspective_16u_C4R	Use new WarpPerspective functions.
ippiWarpPerspective_16u_P3R	Use new WarpPerspective functions.
ippiWarpPerspective_16u_P4R	Use new WarpPerspective functions.
ippiWarpPerspective_32f_AC4R	Use new WarpPerspective functions.
ippiWarpPerspective_32f_C1R	Use new WarpPerspective functions.
ippiWarpPerspective_32f_C3R	Use new WarpPerspective functions.
ippiWarpPerspective_32f_C4R	Use new WarpPerspective functions.
ippiWarpPerspective_32f_P3R	Use new WarpPerspective functions.
ippiWarpPerspective_32f_P4R	Use new WarpPerspective functions.
ippiWarpPerspective_8u_AC4R	Use new WarpPerspective functions.
ippiWarpPerspective_8u_C1R	Use new WarpPerspective functions.
ippiWarpPerspective_8u_C3R	Use new WarpPerspective functions.
ippiWarpPerspective_8u_C4R	Use new WarpPerspective functions.
ippiWarpPerspective_8u_P3R	Use new WarpPerspective functions.
ippiWarpPerspective_8u_P4R	Use new WarpPerspective functions.

ippj.h:

Removed from 9.0	Substitution or Workaround
ippiAdd128_JPEG_16s8u_C1R	N/A
ippiBGR555ToYCbCr411LS_MCU_16u16s_C3P3R	N/A
ippiBGR555ToYCbCr422LS_MCU_16u16s_C3P3R	N/A
ippiBGR555ToYCbCr444LS_MCU_16u16s_C3P3R	N/A

ippiBGR555ToYCbCr_JPEG_16u8u_C3P3R	N/A
ippiBGR565ToYCbCr411LS_MCU_16u16s_C3P3R	N/A
ippiBGR565ToYCbCr422LS_MCU_16u16s_C3P3R	N/A
ippiBGR565ToYCbCr444LS_MCU_16u16s_C3P3R	N/A
ippiBGR565ToYCbCr_JPEG_16u8u_C3P3R	N/A
ippiBGRToYCbCr411LS_MCU_8u16s_C3P3R	N/A
ippiBGRToYCbCr411_JPEG_8u_C3P3R	N/A
ippiBGRToYCbCr411_JPEG_8u_C4P3R	N/A
ippiBGRToYCbCr422LS_MCU_8u16s_C3P3R	N/A
ippiBGRToYCbCr422_JPEG_8u_C3P3R	N/A
ippiBGRToYCbCr422_JPEG_8u_C4P3R	N/A
ippiBGRToYCbCr444LS_MCU_8u16s_C3P3R	N/A
ippiBGRToYCbCr_JPEG_8u_C3P3R	N/A
ippiBGRToYCbCr_JPEG_8u_C4P3R	N/A
ippiBGRToY_JPEG_8u_C3C1R	N/A
ippiCMYKToYCCK411LS_MCU_8u16s_C4P4R	N/A
ippiCMYKToYCCK422LS_MCU_8u16s_C4P4R	N/A
ippiCMYKToYCCK444LS_MCU_8u16s_C4P4R	N/A
ippiCMYKToYCCK_JPEG_8u_C4P4R	N/A
ippiCMYKToYCCK_JPEG_8u_P4R	N/A
ippiDCTQuantFwd8x8LS_JPEG_16u16s_C1R	N/A
ippiDCTQuantFwd8x8LS_JPEG_8u16s_C1R	N/A
ippiDCTQuantFwd8x8_JPEG_16s_C1	N/A
ippiDCTQuantFwd8x8_JPEG_16s_C1I	N/A
ippiDCTQuantInv8x8LS_1x1_JPEG_16s8u_C1R	N/A
ippiDCTQuantInv8x8LS_2x2_JPEG_16s8u_C1R	N/A
ippiDCTQuantInv8x8LS_4x4_JPEG_16s8u_C1R	N/A
ippiDCTQuantInv8x8LS_JPEG_16s16u_C1R	N/A
ippiDCTQuantInv8x8LS_JPEG_16s8u_C1R	N/A
ippiDCTQuantInv8x8To2x2LS_JPEG_16s8u_C1R	N/A
ippiDCTQuantInv8x8To4x4LS_JPEG_16s8u_C1R	N/A

ippiDCTQuantInv8x8_JPEG_16s_C1	N/A
ippiDCTQuantInv8x8_JPEG_16s_C1I	N/A
ippiDecodeCBProgrAttach_JPEG2K_32s_C1R	N/A
ippiDecodeCBProgrFree_JPEG2K	N/A
ippiDecodeCBProgrGetCurBitPlaneNum_JPEG2K	N/A
ippiDecodeCBProgrGetPassCounter_JPEG2K	N/A
ippiDecodeCBProgrGetStateSize_JPEG2K	N/A
ippiDecodeCBProgrInitAlloc_JPEG2K	N/A
ippiDecodeCBProgrInit_JPEG2K	N/A
ippiDecodeCBProgrSetPassCounter_JPEG2K	N/A
ippiDecodeCBProgrStep_JPEG2K	N/A
ippiDecodeCodeBlock_JPEG2K_1u32s_C1R	N/A
ippiDecodeGetBufSize_JPEG2K	N/A
ippiDecodeHuffman8x8_ACFirst_JPEG_1u16s_C1	N/A
ippiDecodeHuffman8x8_ACRefine_JPEG_1u16s_C1	N/A
ippiDecodeHuffman8x8_DCFirst_JPEG_1u16s_C1	N/A
ippiDecodeHuffman8x8_DCRefine_JPEG_1u16s_C1	N/A
ippiDecodeHuffman8x8_Direct_JPEG_1u16s_C1	N/A
ippiDecodeHuffman8x8_JPEG_1u16s_C1	N/A
ippiDecodeHuffmanOne_JPEG_1u16s_C1	N/A
ippiDecodeHuffmanRow_JPEG_1u16s_C1P4	N/A
ippiDecodeHuffmanSpecFree_JPEG_8u	N/A
ippiDecodeHuffmanSpecGetBufSize_JPEG_8u	N/A
ippiDecodeHuffmanSpecInitAlloc_JPEG_8u	N/A
ippiDecodeHuffmanSpecInit_JPEG_8u	N/A
ippiDecodeHuffmanStateFree_JPEG_8u	N/A
ippiDecodeHuffmanStateGetBufSize_JPEG_8u	N/A
ippiDecodeHuffmanStateInitAlloc_JPEG_8u	N/A

ippiDecodeHuffmanStateInit_JPEG_8u	N/A
ippiDiffPredFirstRow_JPEG_16s_C1	N/A
ippiDiffPredRow_JPEG_16s_C1	N/A
ippiEncodeFree_JPEG2K	N/A
ippiEncodeGetDist_JPEG2K	N/A
ippiEncodeGetRate_JPEG2K	N/A
ippiEncodeGetTermPassLen_JPEG2K	N/A
ippiEncodeHuffman8x8_ACFirst_JPEG_16s1u_C1	N/A
ippiEncodeHuffman8x8_ACRefine_JPEG_16s1u_C1	N/A
ippiEncodeHuffman8x8_DCFirst_JPEG_16s1u_C1	N/A
ippiEncodeHuffman8x8_DCRefine_JPEG_16s1u_C1	N/A
ippiEncodeHuffman8x8_Direct_JPEG_16s1u_C1	N/A
ippiEncodeHuffman8x8_JPEG_16s1u_C1	N/A
ippiEncodeHuffmanOne_JPEG_16s1u_C1	N/A
ippiEncodeHuffmanRawTableInit_JPEG_8u	N/A
ippiEncodeHuffmanRow_JPEG_16s1u_P4C1	N/A
ippiEncodeHuffmanSpecFree_JPEG_8u	N/A
ippiEncodeHuffmanSpecGetBufSize_JPEG_8u	N/A
ippiEncodeHuffmanSpecInitAlloc_JPEG_8u	N/A
ippiEncodeHuffmanSpecInit_JPEG_8u	N/A
ippiEncodeHuffmanStateFree_JPEG_8u	N/A
ippiEncodeHuffmanStateGetBufSize_JPEG_8u	N/A
ippiEncodeHuffmanStateInitAlloc_JPEG_8u	N/A
ippiEncodeHuffmanStateInit_JPEG_8u	N/A
ippiEncodeInitAlloc_JPEG2K	N/A
ippiEncodeLoadCodeBlock_JPEG2K_32s_C1R	N/A
ippiEncodeStoreBits_JPEG2K_1u	N/A
ippiFLCDecode4x4_JPEGXR_1u16s_C1IR	N/A

ippiFLCDecode4x4_JPEGXR_1u32s_C1IR	N/A
ippiFLCEncode4x4_JPEGXR_16s1u_C1R	N/A
ippiFLCEncode4x4_JPEGXR_32s1u_C1R	N/A
ippiGetHuffmanStatistics8x8_ACFirst_JPEG_16s_C1	N/A
ippiGetHuffmanStatistics8x8_ACRefine_JPEG_16s_C1	N/A
ippiGetHuffmanStatistics8x8_DCFirst_JPEG_16s_C1	N/A
ippiGetHuffmanStatistics8x8_JPEG_16s_C1	N/A
ippiGetHuffmanStatisticsOne_JPEG_16s_C1	N/A
ippiICTFwd_JPEG2K_16s_P3IR	N/A
ippiICTFwd_JPEG2K_32f_C3P3R	N/A
ippiICTFwd_JPEG2K_32f_P3IR	N/A
ippiICTFwd_JPEG2K_32s_P3IR	N/A
ippiICTInv_JPEG2K_16s_P3IR	N/A
ippiICTInv_JPEG2K_32f_P3C3R	N/A
ippiICTInv_JPEG2K_32f_P3IR	N/A
ippiICTInv_JPEG2K_32s_P3IR	N/A
ippiJoin422LS_MCU_16s8u_P3C2R	N/A
ippiJoinRow_TIFF_8u16u_C1	N/A
ippiPCTFwd16x16_JPEGXR_16s_C1IR	N/A
ippiPCTFwd16x16_JPEGXR_32f_C1IR	N/A
ippiPCTFwd16x16_JPEGXR_32s_C1IR	N/A
ippiPCTFwd8x16_JPEGXR_16s_C1IR	N/A
ippiPCTFwd8x16_JPEGXR_32f_C1IR	N/A
ippiPCTFwd8x16_JPEGXR_32s_C1IR	N/A
ippiPCTFwd8x8_JPEGXR_16s_C1IR	N/A
ippiPCTFwd8x8_JPEGXR_32f_C1IR	N/A
ippiPCTFwd8x8_JPEGXR_32s_C1IR	N/A
ippiPCTFwd_JPEGXR_16s_C1IR	N/A
ippiPCTFwd_JPEGXR_32f_C1IR	N/A

ippiPCTFwd_JPEGXR_32s_C1IR	N/A
ippiPCTInv16x16_JPEGXR_16s_C1IR	N/A
ippiPCTInv16x16_JPEGXR_32f_C1IR	N/A
ippiPCTInv16x16_JPEGXR_32s_C1IR	N/A
ippiPCTInv8x16_JPEGXR_16s_C1IR	N/A
ippiPCTInv8x16_JPEGXR_32f_C1IR	N/A
ippiPCTInv8x16_JPEGXR_32s_C1IR	N/A
ippiPCTInv8x8_JPEGXR_16s_C1IR	N/A
ippiPCTInv8x8_JPEGXR_32f_C1IR	N/A
ippiPCTInv8x8_JPEGXR_32s_C1IR	N/A
ippiPCTInv_JPEGXR_16s_C1IR	N/A
ippiPCTInv_JPEGXR_32f_C1IR	N/A
ippiPCTInv_JPEGXR_32s_C1IR	N/A
ippiPackBitsRow_TIFF_8u_C1	N/A
ippiQuantFwd8x8_JPEG_16s_C1I	N/A
ippiQuantFwdRawTableInit_JPEG_8u	N/A
ippiQuantFwdTableInit_JPEG_8u16u	N/A
ippiQuantInv8x8_JPEG_16s_C1I	N/A
ippiQuantInvTableInit_JPEG_8u16u	N/A
ippiRCTFwd_JPEG2K_16s_C3P3R	N/A
ippiRCTFwd_JPEG2K_16s_P3IR	N/A
ippiRCTFwd_JPEG2K_32s_C3P3R	N/A
ippiRCTFwd_JPEG2K_32s_P3IR	N/A
ippiRCTInv_JPEG2K_16s_P3C3R	N/A
ippiRCTInv_JPEG2K_16s_P3IR	N/A
ippiRCTInv_JPEG2K_32s_P3C3R	N/A
ippiRCTInv_JPEG2K_32s_P3IR	N/A
ippiRGB555ToYCbCr_JPEG_16u8u_C3P3R	N/A
ippiRGB565ToYCbCr_JPEG_16u8u_C3P3R	N/A
ippiRGBToYCbCr411LS_MCU_8u16s_C3P3R	N/A
ippiRGBToYCbCr411_JPEG_8u_C3P3R	N/A

ippiRGBToYCbCr411_JPEG_8u_C4P3R	N/A
ippiRGBToYCbCr422LS_MCU_8u16s_C3P3R	N/A
ippiRGBToYCbCr422_JPEG_8u_C3P3R	N/A
ippiRGBToYCbCr422_JPEG_8u_C4P3R	N/A
ippiRGBToYCbCr444LS_MCU_8u16s_C3P3R	N/A
ippiRGBToYCbCr_JPEG_8u_C3P3R	N/A
ippiRGBToYCbCr_JPEG_8u_C4P3R	N/A
ippiRGBToYCbCr_JPEG_8u_P3R	N/A
ippiRGBToY_JPEG_8u_C3C1R	N/A
ippiRGBToY_JPEG_8u_P3C1R	N/A
ippiReconstructPredFirstRow_JPEG_16s_C1	N/A
ippiReconstructPredRow_JPEG_16s_C1	N/A
ippiSampleDown411LS_MCU_8u16s_C3P3R	N/A
ippiSampleDown422LS_MCU_8u16s_C3P3R	N/A
ippiSampleDown444LS_MCU_8u16s_C3P3R	N/A
ippiSampleDownH2V1_JPEG_8u_C1R	N/A
ippiSampleDownH2V2_JPEG_8u_C1R	N/A
ippiSampleDownRowH2V1_Box_JPEG_8u_C1	N/A
ippiSampleDownRowH2V2_Box_JPEG_8u_C1	N/A
ippiSampleUp411LS_MCU_16s8u_P3C3R	N/A
ippiSampleUp422LS_MCU_16s8u_P3C3R	N/A
ippiSampleUp444LS_MCU_16s8u_P3C3R	N/A
ippiSampleUpH2V1_JPEG_8u_C1R	N/A
ippiSampleUpH2V2_JPEG_8u_C1R	N/A
ippiSampleUpRowH2V1_Triangle_JPEG_8u_C1	N/A
ippiSampleUpRowH2V2_Triangle_JPEG_8u_C1	N/A
ippiSplit422LS_MCU_8u16s_C2P3R	N/A
ippiSplitRow_TIFF_16u8u_C1	N/A
ippiSub128_JPEG_8u16s_C1R	N/A
ippiUnpackBitsRow_TIFF_8u_C1	N/A
ippiVLCAdapt_JPEGXR	N/A

ippiVLCDecode4x4_JPEGXR_1u16s_C1R	N/A
ippiVLCDecode4x4_JPEGXR_1u32s_C1R	N/A
ippiVLCDecodeDC420_JPEGXR_1u32s	N/A
ippiVLCDecodeDC422_JPEGXR_1u32s	N/A
ippiVLCDecodeDC444_JPEGXR_1u32s	N/A
ippiVLCEncode4x4Flex_JPEGXR_16s1u_C1R	N/A
ippiVLCEncode4x4Flex_JPEGXR_32s1u_C1R	N/A
ippiVLCEncode4x4_JPEGXR_16s1u_C1R	N/A
ippiVLCEncode4x4_JPEGXR_32s1u_C1R	N/A
ippiVLCEncodeDC420_JPEGXR_32s1u	N/A
ippiVLCEncodeDC422_JPEGXR_32s1u	N/A
ippiVLCEncodeDC444_JPEGXR_32s1u	N/A
ippiVLCGetStateSize_JPEGXR	N/A
ippiVLCInit_JPEGXR	N/A
ippiVLCScan4x4DC_JPEGXR_32s	N/A
ippiVLCScanReset_JPEGXR	N/A
ippiVLCScanSet_JPEGXR	N/A
ippiWTFwdColLift_B53_JPEG2K_16s_C1	N/A
ippiWTFwdColLift_B53_JPEG2K_32s_C1	N/A
ippiWTFwdColLift_D97_JPEG2K_16s_C1	N/A
ippiWTFwdColLift_D97_JPEG2K_32f_C1	N/A
ippiWTFwdColLift_D97_JPEG2K_32s_C1	N/A
ippiWTFwdCol_B53_JPEG2K_16s_C1R	N/A
ippiWTFwdCol_B53_JPEG2K_32s_C1R	N/A
ippiWTFwdCol_D97_JPEG2K_32f_C1R	N/A
ippiWTFwdRow_B53_JPEG2K_16s_C1R	N/A
ippiWTFwdRow_B53_JPEG2K_32s_C1R	N/A
ippiWTFwdRow_D97_JPEG2K_16s_C1R	N/A
ippiWTFwdRow_D97_JPEG2K_32f_C1R	N/A
ippiWTFwdRow_D97_JPEG2K_32s_C1R	N/A
ippiWTFwd_B53_JPEG2K_16s_C1IR	N/A

ippiWTFwd_B53_JPEG2K_16s_C1R	N/A
ippiWTFwd_B53_JPEG2K_32s_C1IR	N/A
ippiWTFwd_B53_JPEG2K_32s_C1R	N/A
ippiWTFwd_D97_JPEG2K_16s_C1IR	N/A
ippiWTFwd_D97_JPEG2K_16s_C1R	N/A
ippiWTFwd_D97_JPEG2K_32s_C1IR	N/A
ippiWTFwd_D97_JPEG2K_32s_C1R	N/A
ippiWTGetBufSize_B53_JPEG2K_16s_C1IR	N/A
ippiWTGetBufSize_B53_JPEG2K_16s_C1R	N/A
ippiWTGetBufSize_B53_JPEG2K_32s_C1IR	N/A
ippiWTGetBufSize_B53_JPEG2K_32s_C1R	N/A
ippiWTGetBufSize_D97_JPEG2K_16s_C1IR	N/A
ippiWTGetBufSize_D97_JPEG2K_16s_C1R	N/A
ippiWTGetBufSize_D97_JPEG2K_32s_C1IR	N/A
ippiWTGetBufSize_D97_JPEG2K_32s_C1R	N/A
ippiWTInvColLift_B53_JPEG2K_16s_C1	N/A
ippiWTInvColLift_B53_JPEG2K_32s_C1	N/A
ippiWTInvColLift_D97_JPEG2K_16s_C1	N/A
ippiWTInvColLift_D97_JPEG2K_32f_C1	N/A
ippiWTInvColLift_D97_JPEG2K_32s_C1	N/A
ippiWTInvCol_B53_JPEG2K_16s_C1R	N/A
ippiWTInvCol_B53_JPEG2K_32s_C1R	N/A
ippiWTInvCol_D97_JPEG2K_32f_C1R	N/A
ippiWTInvRow_B53_JPEG2K_16s_C1R	N/A
ippiWTInvRow_B53_JPEG2K_32s_C1R	N/A
ippiWTInvRow_D97_JPEG2K_16s_C1R	N/A
ippiWTInvRow_D97_JPEG2K_32f_C1R	N/A
ippiWTInvRow_D97_JPEG2K_32s_C1R	N/A
ippiWTInv_B53_JPEG2K_16s_C1IR	N/A
ippiWTInv_B53_JPEG2K_16s_C1R	N/A
ippiWTInv_B53_JPEG2K_32s_C1IR	N/A

ippiWTInv_B53_JPEG2K_32s_C1R	N/A
ippiWTInv_D97_JPEG2K_16s_C1IR	N/A
ippiWTInv_D97_JPEG2K_16s_C1R	N/A
ippiWTInv_D97_JPEG2K_32s_C1IR	N/A
ippiWTInv_D97_JPEG2K_32s_C1R	N/A
ippiYCCK411ToCMYKLS_MCU_16s8u_P4C4R	N/A
ippiYCCK422ToCMYKLS_MCU_16s8u_P4C4R	N/A
ippiYCCK444ToCMYKLS_MCU_16s8u_P4C4R	N/A
ippiYCCKToCMYK_JPEG_8u_P4C4R	N/A
ippiYCCKToCMYK_JPEG_8u_P4R	N/A
ippiYCbCr411ToBGR555LS_MCU_16s16u_P3C3R	N/A
ippiYCbCr411ToBGR565LS_MCU_16s16u_P3C3R	N/A
ippiYCbCr411ToBGRLS_MCU_16s8u_P3C3R	N/A
ippiYCbCr411ToBGR_JPEG_8u_P3C3R	N/A
ippiYCbCr411ToBGR_JPEG_8u_P3C4R	N/A
ippiYCbCr411ToRGBLS_MCU_16s8u_P3C3R	N/A
ippiYCbCr411ToRGB_JPEG_8u_P3C3R	N/A
ippiYCbCr411ToRGB_JPEG_8u_P3C4R	N/A
ippiYCbCr422ToBGR555LS_MCU_16s16u_P3C3R	N/A
ippiYCbCr422ToBGR565LS_MCU_16s16u_P3C3R	N/A
ippiYCbCr422ToBGRLS_MCU_16s8u_P3C3R	N/A
ippiYCbCr422ToBGR_JPEG_8u_P3C3R	N/A
ippiYCbCr422ToBGR_JPEG_8u_P3C4R	N/A
ippiYCbCr422ToRGBLS_MCU_16s8u_P3C3R	N/A
ippiYCbCr422ToRGB_JPEG_8u_C2C3R	N/A
ippiYCbCr422ToRGB_JPEG_8u_P3C3R	N/A
ippiYCbCr422ToRGB_JPEG_8u_P3C4R	N/A
ippiYCbCr444ToBGR555LS_MCU_16s16u_P3C3R	N/A
ippiYCbCr444ToBGR565LS_MCU_16s16u_P3C3R	N/A
ippiYCbCr444ToBGRLS_MCU_16s8u_P3C3R	N/A
ippiYCbCr444ToRGBLS_MCU_16s8u_P3C3R	N/A

ippiYCbCrToBGR555_JPEG_8u16u_P3C3R	N/A
ippiYCbCrToBGR565_JPEG_8u16u_P3C3R	N/A
ippiYCbCrToBGR_JPEG_8u_P3C3R	N/A
ippiYCbCrToBGR_JPEG_8u_P3C4R	N/A
ippiYCbCrToRGB555_JPEG_8u16u_P3C3R	N/A
ippiYCbCrToRGB565_JPEG_8u16u_P3C3R	N/A
ippiYCbCrToRGB_JPEG_8u_P3C3R	N/A
ippiYCbCrToRGB_JPEG_8u_P3C4R	N/A
ippiYCbCrToRGB_JPEG_8u_P3R	N/A
ippjGetLibVersion	N/A

ippvc.h:

Removed from 9.0	Substitution or Workaround
ippiAdd8x8HP_16s8u_C1RS	N/A
ippiAdd8x8_16s8u_C1IRS	N/A
ippiAddBackPredPB_H263_8u_C1R	N/A
ippiAddC8x8_16s8u_C1IR	N/A
ippiAverage16x16_8u_C1IR	N/A
ippiAverage16x16_8u_C1R	N/A
ippiAverage8x8_8u_C1IR	N/A
ippiAverage8x8_8u_C1R	N/A
ippiBiDirWeightBlockImplicit_H264_8u_P2P1R	N/A
ippiBiDirWeightBlockImplicit_H264_8u_P3P1R	N/A
ippiBiDirWeightBlock_H264_8u_C2R	N/A
ippiBiDirWeightBlock_H264_8u_P2P1R	N/A
ippiBiDirWeightBlock_H264_8u_P3P1R	N/A
ippiBidirWeightImplicit_H264_16u_P2P1R	N/A
ippiBidirWeight_H264_16u_P2P1R	N/A
ippiBidir_H264_16u_P2P1R	N/A
ippiCABACEncodeBinBypass_H264	N/A

ippiCABACEncodeBin_H264	N/A
ippiCABACEncodeResidualBlock_H264_16s	N/A
ippiCABACEncodeResidualBlock_H264_32s	N/A
ippiCABACFree_H264	N/A
ippiCABACGetContexts_H264	N/A
ippiCABACGetSize_H264	N/A
ippiCABACGetStreamSize_H264	N/A
ippiCABACInitAlloc_H264	N/A
ippiCABACInit_H264	N/A
ippiCABACSetStream_H264	N/A
ippiCABACTerminateSlice_H264	N/A
ippiCalcGlobalMV_MPEG4	N/A
ippiCbYCr422ToYCbCr420_Rotate_8u_C2P3R	N/A
ippiCbYCr422ToYCbCr420_Rotate_8u_P3R	N/A
ippiChangeSpriteBrightness_MPEG4_8u_C1IR	N/A
ippiCopy16x16HP_8u_C1R	N/A
ippiCopy16x16QP_MPEG4_8u_C1R	N/A
ippiCopy16x16_8u_C1R	N/A
ippiCopy16x8HP_8u_C1R	N/A
ippiCopy16x8QP_MPEG4_8u_C1R	N/A
ippiCopy8x4HP_8u_C1R	N/A
ippiCopy8x8HP_8u_C1R	N/A
ippiCopy8x8QP_MPEG4_8u_C1R	N/A
ippiCopy8x8_8u_C1R	N/A
ippiCountZeros8x8_16s_C1	N/A
ippiCreateRLEncodeTable	N/A
ippiDCT2x4x8Frw_16s_C1I	N/A
ippiDCT2x4x8Inv_16s_C1I	N/A
ippiDCT8x4x2To4x4Inv_DV_16s_C1I	N/A
ippiDCT8x8Fwd_8u16s_C2P2	N/A
ippiDCT8x8InvOrSet_16s8u_P2C2	N/A

ippiDCT8x8Inv_AANTransposed_16s8u_C1R	N/A
ippiDCT8x8Inv_AANTransposed_16s8u_P2C2R	N/A
ippiDCT8x8Inv_AANTransposed_16s_C1R	N/A
ippiDCT8x8Inv_AANTransposed_16s_P2C2R	N/A
ippiDecodeCAVLCChroma422DcCoeffs_H264_1u16s	N/A
ippiDecodeCAVLCChroma422DcCoeffs_H264_1u32s	N/A
ippiDecodeCAVLCChromaDcCoeffs_H264_1u16s	N/A
ippiDecodeCAVLCChromaDcCoeffs_H264_1u32s	N/A
ippiDecodeCAVLCCoeffsIdxs_H264_1u16s	N/A
ippiDecodeCAVLCCoeffs_H264_1u16s	N/A
ippiDecodeCAVLCCoeffs_H264_1u32s	N/A
ippiDecodeChromaBlock_AVS_1u16s	N/A
ippiDecodeCoeffsInterRVLCBack_MPEG4_1u16s	N/A
ippiDecodeCoeffsInter_H261_1u16s	N/A
ippiDecodeCoeffsInter_H263_1u16s	N/A
ippiDecodeCoeffsInter_MPEG4_1u16s	N/A
ippiDecodeCoeffsIntraRVLCBack_MPEG4_1u16s	N/A
ippiDecodeCoeffsIntra_H261_1u16s	N/A
ippiDecodeCoeffsIntra_H263_1u16s	N/A
ippiDecodeCoeffsIntra_MPEG4_1u16s	N/A
ippiDecodeDCIntra_H263_1u16s	N/A
ippiDecodeDCIntra_MPEG4_1u16s	N/A
ippiDecodeExpGolombOne_H264_1u16s	N/A
ippiDecodeExpGolombOne_H264_1u32s	N/A
ippiDecodeHuffmanOne_1u32s	N/A
ippiDecodeHuffmanPair_1u16s	N/A
ippiDecodeLumaBlockInter_AVS_1u16s	N/A
ippiDecodeLumaBlockIntra_AVS_1u16s	N/A
ippiDeinterlaceBlendFree_8u_C1	N/A

ippiDeinterlaceBlendFree_8u_C2	N/A
ippiDeinterlaceBlendGetSize_8u_C1	N/A
ippiDeinterlaceBlendGetSize_8u_C2	N/A
ippiDeinterlaceBlendInitAlloc_8u_C1	N/A
ippiDeinterlaceBlendInitAlloc_8u_C2	N/A
ippiDeinterlaceBlendInit_8u_C1	N/A
ippiDeinterlaceBlendInit_8u_C2	N/A
ippiDeinterlaceBlend_8u_C1	N/A
ippiDeinterlaceBlend_8u_C2	N/A
ippiDeinterlaceEdgeDetect_8u_C1R	N/A
ippiDeinterlaceFilterTriangle_8u_C1R	N/A
ippiDeinterlaceFilterTriangle_8u_C2R	N/A
ippiDeinterlaceMedianThreshold_8u_C1R	N/A
ippiDeinterlaceMotionAdaptive_8u_C1	N/A
ippiDequantTransformResidualAndAdd_H264_16s_C1I	N/A
ippiDequantTransformResidual_H264_16s_C1I	N/A
ippiDequantTransformResidual_SISP_H264_16s_C1I	N/A
ippiDisassembleChroma420Intra_AVS_16s8u_C1R	N/A
ippiDisassembleLumaIntra_AVS_16s8u_C1R	N/A
ippiDownsampleFour16x16_H263_16s_C1R	N/A
ippiDownsampleFour_H263_8u_C1R	N/A
ippiEdgesDetect16x16_16u_C1R	N/A
ippiEdgesDetect16x16_8u_C1R	N/A
ippiEncodeChromaDcCoeffsCAVLC_H264_16s	N/A
ippiEncodeCoeffsCAVLCChromaDC2x2_H264_32s	N/A
ippiEncodeCoeffsCAVLCChromaDC2x4_H264_16s	N/A
ippiEncodeCoeffsCAVLCChromaDC2x4_H264_32s	N/A
ippiEncodeCoeffsCAVLC_H264_16s	N/A
ippiEncodeCoeffsCAVLC_H264_32s	N/A

ippiEncodeCoeffsInter_H261_16s1u	N/A
ippiEncodeCoeffsInter_H263_16s1u	N/A
ippiEncodeCoeffsInter_MPEG4_16s1u	N/A
ippiEncodeCoeffsIntra_H261_16s1u	N/A
ippiEncodeCoeffsIntra_H263_16s1u	N/A
ippiEncodeCoeffsIntra_MPEG4_16s1u	N/A
ippiEncodeDCIntra_H263_16s1u	N/A
ippiEncodeDCIntra_MPEG4_16s1u	N/A
ippiExpandPlane_H264_8u_C1R	N/A
ippiFilter8x8_H261_8u_C1R	N/A
ippiFilterBlockBoundaryHorEdge_H263_8u_C1IR	N/A
ippiFilterBlockBoundaryVerEdge_H263_8u_C1IR	N/A
ippiFilterDeblocking16x16HorEdge_H263_8u_C1IR	N/A
ippiFilterDeblocking16x16VerEdge_H263_8u_C1IR	N/A
ippiFilterDeblocking8x8HorEdge_H263_8u_C1IR	N/A
ippiFilterDeblocking8x8HorEdge_MPEG4_8u_C1IR	N/A
ippiFilterDeblocking8x8VerEdge_H263_8u_C1IR	N/A
ippiFilterDeblocking8x8VerEdge_MPEG4_8u_C1IR	N/A
ippiFilterDeblockingChroma422HorEdge_H264_16u_C1IR	N/A
ippiFilterDeblockingChroma422HorEdge_H264_8u_C1IR	N/A
ippiFilterDeblockingChroma422VerEdge_H264_16u_C1IR	N/A
ippiFilterDeblockingChroma422VerEdge_H264_8u_C1IR	N/A
ippiFilterDeblockingChromaHorEdge_H264_16u_C1IR	N/A

ippiFilterDeblockingChromaVerEdgeMBAFF_H264_16u_C1IR	N/A
ippiFilterDeblockingChromaVerEdge_H264_16u_C1IR	N/A
ippiFilterDeblockingChroma_HorEdge_AVS_8u_C1IR	N/A
ippiFilterDeblockingChroma_HorEdge_H264_8u_C1IR	N/A
ippiFilterDeblockingChroma_HorEdge_H264_8u_C2I	N/A
ippiFilterDeblockingChroma_HorEdge_H264_8u_C2IR	N/A
ippiFilterDeblockingChroma_HorEdge_VC1_8u_C1IR	N/A
ippiFilterDeblockingChroma_HorEdge_VC1_8u_C2IR	N/A
ippiFilterDeblockingChroma_VerEdge_AVS_8u_C1IR	N/A
ippiFilterDeblockingChroma_VerEdge_H264_8u_C1IR	N/A
ippiFilterDeblockingChroma_VerEdge_H264_8u_C2I	N/A
ippiFilterDeblockingChroma_VerEdge_H264_8u_C2IR	N/A
ippiFilterDeblockingChroma_VerEdge_MBAFF_H264_8u_C1IR	N/A
ippiFilterDeblockingChroma_VerEdge_VC1_8u_C1IR	N/A
ippiFilterDeblockingChroma_VerEdge_VC1_8u_C2IR	N/A
ippiFilterDeblockingLumaHorEdge_H264_16u_C1IR	N/A
ippiFilterDeblockingLumaVerEdgeMBAFF_H264_16u_C1IR	N/A
ippiFilterDeblockingLumaVerEdge_H264_16u_C1IR	N/A
ippiFilterDeblockingLuma_HorEdge_AVS_8u_C1IR	N/A

ippiFilterDeblockingLuma_HorEdge_H264_8u_C1IR	N/A
ippiFilterDeblockingLuma_HorEdge_VC1_8u_C1IR	N/A
ippiFilterDeblockingLuma_VerEdge_AVS_8u_C1IR	N/A
ippiFilterDeblockingLuma_VerEdge_H264_8u_C1IR	N/A
ippiFilterDeblockingLuma_VerEdge_MBAFF_H264_8u_C1IR	N/A
ippiFilterDeblockingLuma_VerEdge_VC1_8u_C1IR	N/A
ippiFilterDenoiseAdaptiveFree_8u_C1	N/A
ippiFilterDenoiseAdaptiveInitAlloc_8u_C1	N/A
ippiFilterDenoiseAdaptive_8u_C1R	N/A
ippiFilterDenoiseCASTInit	N/A
ippiFilterDenoiseCASTYUV422_8u_C2R	N/A
ippiFilterDenoiseCAST_8u_C1R	N/A
ippiFilterDenoiseMosquitoFree_8u_C1	N/A
ippiFilterDenoiseMosquitoInitAlloc_8u_C1	N/A
ippiFilterDenoiseMosquito_8u_C1R	N/A
ippiFilterDenoiseSmooth_8u_C1R	N/A
ippiFilterDeringingSmooth8x8_MPEG4_8u_C1R	N/A
ippiFilterDeringingThreshold_MPEG4_8u_P3R	N/A
ippiFrameFieldSAD16x16_16s32s_C1R	N/A
ippiFrameFieldSAD16x16_8u32s_C1R	N/A
ippiFreeHuffmanTable_DV_32u	N/A
ippiGenScaleLevel4x4_H264_8u16s_C1	N/A
ippiGenScaleLevel8x8_H264_8u16s_D2	N/A
ippiGetDiff16x16B_8u16s_C1	N/A
ippiGetDiff16x16_8u16s_C1	N/A
ippiGetDiff16x8B_8u16s_C1	N/A
ippiGetDiff16x8_8u16s_C1	N/A

ippiGetDiff8x16B_8u16s_C1	N/A
ippiGetDiff8x16_8u16s_C1	N/A
ippiGetDiff8x4B_8u16s_C1	N/A
ippiGetDiff8x4B_8u16s_C2P2	N/A
ippiGetDiff8x4_8u16s_C1	N/A
ippiGetDiff8x4_8u16s_C2P2	N/A
ippiGetDiff8x8B_8u16s_C1	N/A
ippiGetDiff8x8B_8u16s_C2P2	N/A
ippiGetDiff8x8_8u16s_C1	N/A
ippiGetDiff8x8_8u16s_C2P2	N/A
ippiHadamard8x8Sum_VC1_16s	N/A
ippiHadamard8x8Sum_VC1_8u16s	N/A
ippiHuffmanDecodeSegmentOnePass_DV_8u16s	N/A
ippiHuffmanDecodeSegment_DV100_8u16s	N/A
ippiHuffmanDecodeSegment_DV_8u16s	N/A
ippiHuffmanRunLevelTableInitAlloc_32s	N/A
ippiHuffmanTableFree_32s	N/A
ippiHuffmanTableInitAlloc_32s	N/A
ippiInitAllocHuffmanTable_DV_32u	N/A
ippiInterpolateAverage16x16_8u_C1IR	N/A
ippiInterpolateAverage16x8_8u_C1IR	N/A
ippiInterpolateAverage8x4_8u_C1IR	N/A
ippiInterpolateAverage8x8_8u_C1IR	N/A
ippiInterpolateBlock_H264_8u_P2P1R	N/A
ippiInterpolateBlock_H264_8u_P3P1R	N/A
ippiInterpolateChromaBlock_H264_16u_P2R	N/A
ippiInterpolateChromaBlock_H264_8u_C2C2R	N/A
ippiInterpolateChromaBlock_H264_8u_C2P2R	N/A
ippiInterpolateChromaBlock_H264_8u_P2R	N/A
ippiInterpolateChromaBottom_H264_16u_C1R	N/A
ippiInterpolateChromaBottom_H264_8u_C1R	N/A

ippiInterpolateChromaTop_H264_16u_C1R	N/A
ippiInterpolateChromaTop_H264_8u_C1R	N/A
ippiInterpolateChroma_H264_16u_C1R	N/A
ippiInterpolateChroma_H264_8u_C1R	N/A
ippiInterpolateChroma_H264_8u_C2P2R	N/A
ippiInterpolateICBicubicBlock_VC1_8u_C1R	N/A
ippiInterpolateICBilinearBlock_VC1_8u_C1R	N/A
ippiInterpolateICBilinearBlock_VC1_8u_C2R	N/A
ippiInterpolateLumaBlock_AVS_8u_P1R	N/A
ippiInterpolateLumaBlock_H264_16u_P1R	N/A
ippiInterpolateLumaBlock_H264_8u_P1R	N/A
ippiInterpolateLumaBottom_H264_16u_C1R	N/A
ippiInterpolateLumaBottom_H264_8u_C1R	N/A
ippiInterpolateLumaTop_H264_16u_C1R	N/A
ippiInterpolateLumaTop_H264_8u_C1R	N/A
ippiInterpolateLuma_H264_16u_C1R	N/A
ippiInterpolateLuma_H264_8u_C1R	N/A
ippiInterpolateQPBicubic_VC1_8u_C1R	N/A
ippiInterpolateQPBilinear_VC1_8u_C1R	N/A
ippiInterpolateQPBilinear_VC1_8u_C2R	N/A
ippiMC16x16B_8u_C1	N/A
ippiMC16x16_8u_C1	N/A
ippiMC16x4B_8u_C1	N/A
ippiMC16x4_8u_C1	N/A
ippiMC16x8BUV_8u_C1	N/A
ippiMC16x8B_8u_C1	N/A
ippiMC16x8UV_8u_C1	N/A
ippiMC16x8_8u_C1	N/A
ippiMC2x2B_8u_C1	N/A
ippiMC2x2_8u_C1	N/A
ippiMC2x4B_8u_C1	N/A

ippiMC2x4_8u_C1	N/A
ippiMC4x2B_8u_C1	N/A
ippiMC4x2_8u_C1	N/A
ippiMC4x4B_8u_C1	N/A
ippiMC4x4_8u_C1	N/A
ippiMC4x8B_8u_C1	N/A
ippiMC4x8_8u_C1	N/A
ippiMC8x16B_8u_C1	N/A
ippiMC8x16_8u_C1	N/A
ippiMC8x4B_16s8u_P2C2R	N/A
ippiMC8x4B_8u_C1	N/A
ippiMC8x4_16s8u_P2C2R	N/A
ippiMC8x4_8u_C1	N/A
ippiMC8x8B_16s8u_P2C2R	N/A
ippiMC8x8B_8u_C1	N/A
ippiMC8x8_16s8u_P2C2R	N/A
ippiMC8x8_8u_C1	N/A
ippiMeanAbsDev16x16_8u32s_C1R	N/A
ippiMeanAbsDev8x8_8u32s_C1R	N/A
ippiOBMC16x16HP_MPEG4_8u_C1R	N/A
ippiOBMC8x8HP_MPEG4_8u_C1R	N/A
ippiOBMC8x8QP_MPEG4_8u_C1R	N/A
ippiPredictIntraChroma8x8_H264_8u_C1IR	N/A
ippiPredictIntra_16x16_H264_8u_C1IR	N/A
ippiPredictIntra_4x4_H264_8u_C1IR	N/A
ippiPutIntraBlock	N/A
ippiQuantInterGetSize_MPEG4	N/A
ippiQuantInterInit_MPEG4	N/A
ippiQuantInterNonuniform_VC1_16s_C1IR	N/A
ippiQuantInterUniform_VC1_16s_C1IR	N/A
ippiQuantInter_H263_16s_C1I	N/A

ippiQuantInter_MPEG4_16s_C1I	N/A
ippiQuantIntraGetSize_MPEG4	N/A
ippiQuantIntraInit_MPEG4	N/A
ippiQuantIntraNonuniform_VC1_16s_C1IR	N/A
ippiQuantIntraUniform_VC1_16s_C1IR	N/A
ippiQuantIntra_H263_16s_C1I	N/A
ippiQuantIntra_MPEG2_16s_C1I	N/A
ippiQuantIntra_MPEG4_16s_C1I	N/A
ippiQuantInvInterGetSize_MPEG4	N/A
ippiQuantInvInterInit_MPEG4	N/A
ippiQuantInvInterNonuniform_VC1_16s_C1IR	N/A
ippiQuantInvInterNonuniform_VC1_16s_C1R	N/A
ippiQuantInvInterUniform_VC1_16s_C1IR	N/A
ippiQuantInvInterUniform_VC1_16s_C1R	N/A
ippiQuantInvInter_H263_16s_C1I	N/A
ippiQuantInvInter_MPEG4_16s_C1I	N/A
ippiQuantInvIntraGetSize_MPEG4	N/A
ippiQuantInvIntraInit_MPEG4	N/A
ippiQuantInvIntraNonuniform_VC1_16s_C1IR	N/A
ippiQuantInvIntraNonuniform_VC1_16s_C1R	N/A
ippiQuantInvIntraUniform_VC1_16s_C1IR	N/A
ippiQuantInvIntraUniform_VC1_16s_C1R	N/A
ippiQuantInvIntra_H263_16s_C1I	N/A
ippiQuantInvIntra_MPEG2_16s_C1I	N/A
ippiQuantInvIntra_MPEG4_16s_C1I	N/A
ippiQuantInvLuma8x8_H264_32s_C1I	N/A
ippiQuantInv_DV_16s_C1I	N/A
ippiQuantInv_MPEG2_16s_C1I	N/A
ippiQuantLuma8x8Inv_H264_16s_C1I	N/A
ippiQuantLuma8x8_H264_16s_C1	N/A
ippiQuantLuma8x8_H264_32s_C1	N/A

ippiQuantWeightBlockInv_DV100_16s_C1I	N/A
ippiQuantWeightBlockInv_DV_16s_C1I	N/A
ippiQuant_MPEG2_16s_C1I	N/A
ippiQuantizeResidual4x4Fwd_H264_16s32s_C1	N/A
ippiQuantizeResidual4x4Fwd_H264_16s_C1	N/A
ippiQuantizeResidual4x4Fwd_H264_32s_C1	N/A
ippiRangeMapping_VC1_8u_C1R	N/A
ippiReconstructChroma422Inter4x4_H264High_16s8u_IP2R	N/A
ippiReconstructChroma422Inter4x4_H264High_32s16u_IP2R	N/A
ippiReconstructChroma422Intra4x4_H264High_16s8u_IP2R	N/A
ippiReconstructChroma422Intra4x4_H264High_32s16u_IP2R	N/A
ippiReconstructChroma422IntraHalf4x4_H264High_16s8u_IP2R	N/A
ippiReconstructChroma422IntraHalf4x4_H264High_32s16u_IP2R	N/A
ippiReconstructChromaInter4x4MB_H264_16s8u_C2R	N/A
ippiReconstructChromaInter4x4MB_H264_16s8u_P2R	N/A
ippiReconstructChromaInter4x4_H264High_32s16u_IP2R	N/A
ippiReconstructChromaInterMB_H264_16s8u_C2R	N/A
ippiReconstructChromaInterMB_H264_16s8u_P2R	N/A
ippiReconstructChromaInter_AVS_16s8u_C1R	N/A
ippiReconstructChromaIntra4x4MB_H264_16s8u_C2R	N/A
ippiReconstructChromaIntra4x4MB_H264_16s8u_P2R	N/A
ippiReconstructChromaIntra4x4_H264High_32s16u_IP2R	N/A

ippiReconstructChromaIntraHalf4x4_H264High_32s16u_IP2R	N/A
ippiReconstructChromaIntraHalfs4x4MB_H264_16s8u_P2R	N/A
ippiReconstructChromaIntraHalfsMB_H264_16s8u_P2R	N/A
ippiReconstructChromaIntraHalves4x4MB_H264_16s8u_P2R	N/A
ippiReconstructChromaIntraHalvesMB_H264_16s8u_C2R	N/A
ippiReconstructChromaIntraHalvesMB_H264_16s8u_P2R	N/A
ippiReconstructChromaIntraMB_H264_16s8u_C2R	N/A
ippiReconstructChromaIntraMB_H264_16s8u_P2R	N/A
ippiReconstructChromaIntra_AVS_16s8u_C1R	N/A
ippiReconstructCoeffsInter_H261_1u16s	N/A
ippiReconstructCoeffsInter_H263_1u16s	N/A
ippiReconstructCoeffsInter_MPEG4_1u16s	N/A
ippiReconstructCoeffsIntra_H261_1u16s	N/A
ippiReconstructCoeffsIntra_H263_1u16s	N/A
ippiReconstructDCTBlockIntra_MPEG1_32s	N/A
ippiReconstructDCTBlockIntra_MPEG2_32s	N/A
ippiReconstructDCTBlock_MPEG1_32s	N/A
ippiReconstructDCTBlock_MPEG2_32s	N/A
ippiReconstructLumaInter4x4MB_H264_16s8u_C1R	N/A
ippiReconstructLumaInter4x4_H264High_32s16u_IP1R	N/A
ippiReconstructLumaInter8x8MB_H264_16s8u_C1R	N/A
ippiReconstructLumaInter8x8_H264High_32s16u_IP1R	N/A
ippiReconstructLumaInterMB_H264_16s8u_C1R	N/A

ippiReconstructLumaInter_AVS_16s8u_C1R	N/A
ippiReconstructLumaIntra16x16MB_H264_16s8u_C1R	N/A
ippiReconstructLumaIntra16x16_H264High_32s16u_IP1R	N/A
ippiReconstructLumaIntra4x4MB_H264_16s8u_C1R	N/A
ippiReconstructLumaIntra4x4_H264High_32s16u_IP1R	N/A
ippiReconstructLumaIntra8x8MB_H264_16s8u_C1R	N/A
ippiReconstructLumaIntra8x8_H264High_32s16u_IP1R	N/A
ippiReconstructLumaIntraHalf4x4MB_H264_16s8u_C1R	N/A
ippiReconstructLumaIntraHalf4x4_H264High_32s16u_IP1R	N/A
ippiReconstructLumaIntraHalf8x8MB_H264_16s8u_C1R	N/A
ippiReconstructLumaIntraHalf8x8_H264High_32s16u_IP1R	N/A
ippiReconstructLumaIntraHalfMB_H264_16s8u_C1R	N/A
ippiReconstructLumaIntraMB_H264_16s8u_C1R	N/A
ippiReconstructLumaIntra_16x16MB_H264_16s8u_C1R	N/A
ippiReconstructLumaIntra_AVS_16s8u_C1R	N/A
ippiResample_H263_8u_P3R	N/A
ippiResizeCCRotate_8u_C2R	N/A
ippiSAD16x16Blocks4x4_16u32u_C1R	N/A
ippiSAD16x16Blocks4x4_8u16u	N/A
ippiSAD16x16Blocks8x8_16u32u_C1R	N/A
ippiSAD16x16Blocks8x8_8u16u	N/A
ippiSAD16x16_16u32s_C1R	N/A
ippiSAD16x16_8u32s	N/A

ippiSAD16x16xNI_8u16u_C1R	N/A
ippiSAD16x16xN_8u16u_C1R	N/A
ippiSAD16x8_8u32s_C1R	N/A
ippiSAD2x2xN_8u16u_C1R	N/A
ippiSAD4x4_16u32s_C1R	N/A
ippiSAD4x4_8u32s	N/A
ippiSAD4x4xNI_8u16u_C1R	N/A
ippiSAD4x4xN_8u16u_C1R	N/A
ippiSAD4x8_8u32s_C1R	N/A
ippiSAD8x16_8u32s_C1R	N/A
ippiSAD8x4_8u32s_C1R	N/A
ippiSAD8x8_16u32s_C1R	N/A
ippiSAD8x8_8u32s_C1R	N/A
ippiSAD8x8_8u32s_C2R	N/A
ippiSAD8x8xNI_8u16u_C1R	N/A
ippiSAD8x8xN_8u16u_C1R	N/A
ippiSAT8x8D_16u32s_C1R	N/A
ippiSAT8x8D_8u32s_C1R	N/A
ippiSATD16x16_8u32s_C1R	N/A
ippiSATD16x8_8u32s_C1R	N/A
ippiSATD4x4_8u32s_C1R	N/A
ippiSATD4x8_8u32s_C1R	N/A
ippiSATD8x16_8u32s_C1R	N/A
ippiSATD8x4_8u32s_C1R	N/A
ippiSATD8x8_8u32s_C1R	N/A
ippiSSD4x4_8u32s_C1R	N/A
ippiSSD8x8_8u32s_C1R	N/A
ippiScanFwd_16s_C1	N/A
ippiScanInv_16s_C1	N/A
ippiSmoothingChroma_HorEdge_VC1_16s8u_C1R	N/A

ippiSmoothingChroma_HorEdge_VC1_16s8u_P2C2R	N/A
ippiSmoothingChroma_VerEdge_VC1_16s8u_C1R	N/A
ippiSmoothingChroma_VerEdge_VC1_16s8u_P2C2R	N/A
ippiSmoothingLuma_HorEdge_VC1_16s8u_C1R	N/A
ippiSmoothingLuma_VerEdge_VC1_16s8u_C1R	N/A
ippiSpatialInterpolation_H263_8u_C1R	N/A
ippiSqrDiff16x16B_8u32s	N/A
ippiSqrDiff16x16_8u32s	N/A
ippiSub16x16_8u16s_C1R	N/A
ippiSub4x4_16u16s_C1R	N/A
ippiSub4x4_8u16s_C1R	N/A
ippiSub8x8_16u16s_C1R	N/A
ippiSub8x8_8u16s_C1R	N/A
ippiSubSAD8x8_8u16s_C1R	N/A
ippiSumsDiff16x16Blocks4x4_16u32s_C1R	N/A
ippiSumsDiff16x16Blocks4x4_8u16s_C1	N/A
ippiSumsDiff8x8Blocks4x4_16u32s_C1R	N/A
ippiSumsDiff8x8Blocks4x4_8u16s_C1	N/A
ippiSumsDiff8x8Blocks4x4_8u16s_C2P2	N/A
ippiTransform4x4Fwd_VC1_16s_C1IR	N/A
ippiTransform4x4Fwd_VC1_16s_C1R	N/A
ippiTransform4x4Inv_VC1_16s_C1IR	N/A
ippiTransform4x4Inv_VC1_16s_C1R	N/A
ippiTransform4x8Fwd_VC1_16s_C1IR	N/A
ippiTransform4x8Fwd_VC1_16s_C1R	N/A
ippiTransform4x8Inv_VC1_16s_C1IR	N/A
ippiTransform4x8Inv_VC1_16s_C1R	N/A
ippiTransform8x4Fwd_VC1_16s_C1IR	N/A
ippiTransform8x4Fwd_VC1_16s_C1R	N/A

ippiTransform8x4Inv_VC1_16s_C1IR	N/A
ippiTransform8x4Inv_VC1_16s_C1R	N/A
ippiTransform8x8Fwd_VC1_16s_C1IR	N/A
ippiTransform8x8Fwd_VC1_16s_C1R	N/A
ippiTransform8x8Inv_VC1_16s_C1IR	N/A
ippiTransform8x8Inv_VC1_16s_C1R	N/A
ippiTransformDequantChromaDC_H264_16s_C1I	N/A
ippiTransformDequantChromaDC_SISP_H264_16s_C1I	N/A
ippiTransformDequantLumaDC_H264_16s_C1I	N/A
ippiTransformFwdLuma8x8_H264_16s32s_C1	N/A
ippiTransformFwdLuma8x8_H264_16s_C1	N/A
ippiTransformInvAddPredLuma8x8_H264_32s16u_C1R	N/A
ippiTransformLuma8x8Fwd_H264_16s_C1I	N/A
ippiTransformLuma8x8InvAddPred_H264_16s8u_C1R	N/A
ippiTransformPrediction_H264_8u16s_C1	N/A
ippiTransformQuant8x8Fwd_AVS_16s_C1	N/A
ippiTransformQuantChromaDC_H264_16s_C1I	N/A
ippiTransformQuantFwd4x4_H264_16s32s_C1	N/A
ippiTransformQuantFwd4x4_H264_16s_C1	N/A
ippiTransformQuantFwdChromaDC2x2_H264_16s_C1I	N/A
ippiTransformQuantFwdChromaDC2x2_H264_32s_C1I	N/A
ippiTransformQuantFwdChromaDC2x4_H264_16s_C1I	N/A
ippiTransformQuantFwdChromaDC2x4_H264_32s_C1I	N/A
ippiTransformQuantFwdLumaDC4x4_H264_16s_C1I	N/A
ippiTransformQuantFwdLumaDC4x4_H264_32s_C1I	N/A

ippiTransformQuantInvAddPred4x4_H264_16s_C1IR	N/A
ippiTransformQuantInvAddPred4x4_H264_32s_C1IR	N/A
ippiTransformQuantInvChromaDC2x2_H264_16s_C1I	N/A
ippiTransformQuantInvChromaDC2x2_H264_32s_C1I	N/A
ippiTransformQuantInvChromaDC2x4_H264_16s_C1I	N/A
ippiTransformQuantInvChromaDC2x4_H264_32s_C1I	N/A
ippiTransformQuantInvLumaDC4x4_H264_16s_C1I	N/A
ippiTransformQuantInvLumaDC4x4_H264_32s_C1I	N/A
ippiTransformQuantLumaDC_H264_16s_C1I	N/A
ippiTransformQuantResidual_H264_16s_C1I	N/A
ippiTransformResidual4x4Fwd_H264_16s_C1	N/A
ippiTransformResidual4x4Fwd_H264_32s_C1	N/A
ippiTransformResidual4x4Inv_H264_16s_C1	N/A
ippiTransformResidual4x4Inv_H264_32s_C1	N/A
ippiUniDirWeightBlock_H264_8u_C1IR	N/A
ippiUniDirWeightBlock_H264_8u_C1R	N/A
ippiUniDirWeightBlock_H264_8u_C2R	N/A
ippiUnidirWeight_H264_16u_IP2P1R	N/A
ippiUpsampleFour8x8_H263_16s_C1R	N/A
ippiUpsampleFour_H263_8u_C1R	N/A
ippiVarMean8x8_16s32s_C1R	N/A
ippiVarMean8x8_8u32s_C1R	N/A
ippiVarMeanDiff16x16_8u32s_C1R	N/A
ippiVarMeanDiff16x8_8u32s_C1R	N/A
ippiVarSum8x8_16s32s_C1R	N/A
ippiVarSum8x8_8u32s_C1R	N/A

ippiVariance16x16_8u32s	N/A
ippiWarpChroma_MPEG4_8u_P2R	N/A
ippiWarpGetSize_MPEG4	N/A
ippiWarpInit_MPEG4	N/A
ippiWarpLuma_MPEG4_8u_C1R	N/A
ippiWeightPrediction_AVS_8u_C1R	N/A
ippiWeightedAverage_H264_8u_C1IR	N/A
ippiYCrCb411ToYCbCr422_16x4x5MB_DV_16s8u_P3C2R	N/A
ippiYCrCb411ToYCbCr422_5MBDV_16s8u_P3C2R	N/A
ippiYCrCb411ToYCbCr422_8x8MB_DV_16s8u_P3C2R	N/A
ippiYCrCb411ToYCbCr422_EdgeDV_16s8u_P3C2R	N/A
ippiYCrCb411ToYCbCr422_ZoomOut2_5MBDV_16s8u_P3C2R	N/A
ippiYCrCb411ToYCbCr422_ZoomOut2_EdgeDV_16s8u_P3C2R	N/A
ippiYCrCb411ToYCbCr422_ZoomOut4_5MBDV_16s8u_P3C2R	N/A
ippiYCrCb411ToYCbCr422_ZoomOut4_EdgeDV_16s8u_P3C2R	N/A
ippiYCrCb411ToYCbCr422_ZoomOut8_5MBDV_16s8u_P3C2R	N/A
ippiYCrCb411ToYCbCr422_ZoomOut8_EdgeDV_16s8u_P3C2R	N/A
ippiYCrCb420ToYCbCr422_5MBDV_16s8u_P3C2R	N/A
ippiYCrCb420ToYCbCr422_8x8x5MB_DV_16s8u_P3C2R	N/A
ippiYCrCb420ToYCbCr422_ZoomOut2_5MBDV_16s8u_P3C2R	N/A
ippiYCrCb420ToYCbCr422_ZoomOut4_5MBDV_16s8u_P3C2R	N/A
ippiYCrCb420ToYCbCr422_ZoomOut8_5MBDV_16s8u_P3C2R	N/A
ippiYCrCb422ToYCbCr422_10HalvesMB16x8_DV100_16s8u_P3C2R	N/A

ippiYCrCb422ToYCbCr422_5MBDV_16s8u_P3C2R	N/A
ippiYCrCb422ToYCbCr422_8x4x5MB_DV_16s8u_P3C2R	N/A
ippiYCrCb422ToYCbCr422_ZoomOut2_5MBDV_16s8u_P3C2R	N/A
ippiYCrCb422ToYCbCr422_ZoomOut4_5MBDV_16s8u_P3C2R	N/A
ippiYCrCb422ToYCbCr422_ZoomOut8_5MBDV_16s8u_P3C2R	N/A
ippvcGetLibVersion	N/A

Sample Code

Rotate.c

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

//   A simple example of performing the rotate operation of image.
//   Nearest neighbot interpolation is used
// implemented with Intel(R) Integrated Performance Primitives (Intel(R) IPP) functions:
//     ippiGetRotateShift
//     ippiGetRotateTransform
//     ippiWarpAffineGetSize
//     ippiWarpAffineNearestInit
//     ippiWarpGetBufferSize
//     ippiWarpAffineNearest_8u_C1R

#include <stdio.h>
#include "ipp.h"

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define check_sts(st)        \
    if ((st) != ippStsNoErr) \
        goto exitLine; /* Go to Exit if Intel(R) IPP function returned status different from
ippStsNoErr */

/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
```

```
check and will return an appropriate status  */

int main(void)
{
    IppStatus status = ippStsNoErr;

    Ipp8u src[4 * 4] = {4, 4, 4, 4, 3, 3, 3, 3, 2, 2, 2, 2, 1, 1, 1, 1};

    Ipp8u dst[6 * 6];

    IppiSize srcSize = {4, 4};
    IppiSize dstSize = {6, 6};

    IppiWarpSpec *pSpec = 0;
    double coeffs[2][3];
    int specSize = 0, initSize = 0, bufSize = 0;
    Ipp8u *pBuffer = 0;
    IppiPoint dstOffset = {0, 0};
    double xShift;
    double yShift;
    double xShiftTmp;
    double yShiftTmp;
    double xCenterSrc = 2.0;
    double yCenterSrc = 2.0;
    double xCenterDst = 3.5;
    double yCenterDst = 2.5;
    double angle = 35.0;

    check_sts(status = ippiGetRotateShift(xCenterSrc, yCenterSrc, angle, &xShift, &yShift));
    /* xShift = -0.79 , yShift = 1.51 */

    check_sts(status = ippiGetRotateShift(1.0, 1.0, angle, &xShiftTmp, &yShiftTmp));
    xShift += xShiftTmp;
    yShift += yShiftTmp;
    /* xShift = -1.18, yShift = 2.26 */

    xShift += xCenterDst - xCenterSrc;
    yShift += yCenterDst - yCenterSrc;
    /* xShift = 0.32, yShift = 2.76 */

    check_sts(status = ippiSet_8u_C1R(0, dst, 6, dstSize));

    check_sts(status = ippiGetRotateTransform(angle, xShift, yShift, coeffs));

    /* Spec and init buffer sizes */
    check_sts(status = ippiWarpAffineGetSize(srcSize, dstSize, ipp8u, coeffs, ippNearest,
ippWarpForward, ippBorderTransp, &specSize, &initSize));

    /* Memory allocation */
    pSpec = (IppiWarpSpec *)ippsMalloc_8u(specSize);

    /* Filter initialization */
    check_sts(status = ippiWarpAffineNearestInit(srcSize, dstSize, ipp8u, coeffs,
ippWarpForward, 1, ippBorderTransp, NULL, 0, pSpec));

    /* work buffer size */
    check_sts(status = ippiWarpGetBufferSize(pSpec, dstSize, &bufSize));
```

```
    pBuffer = ippsMalloc_8u(bufSize);

    /* Image rotation processing */
    check_sts(status = ippiWarpAffineNearest_8u_C1R(src, 4, dst, 6, dstOffset, dstSize, pSpec,
pBuffer));

    /* Result:
     *  0 0 0 0 0 0
     *  0 0 0 0 0 0
     *  0 0 4 3 0 0
     *  0 3 3 2 2 0    dst
     *  0 0 2 1 0 0
     *  0 0 1 0 0 0
     */

    EXIT_MAIN
    ippsFree(pSpec);
    ippsFree(pBuffer);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

RotateCenter.c

```
/*******************************************************************************
 * Copyright 2015 Intel Corporation.
 *
 *
 * This software and the related documents are Intel copyrighted materials, and your use of them
is governed by
 * the express license under which they were provided to you ('License'). Unless the License
provides otherwise,
 * you may not use, modify, copy, publish, distribute, disclose or transmit this software or the
related
 * documents without Intel's prior written permission.
 * This software and the related documents are provided as is, with no express or implied
warranties, other than
 * those that are expressly stated in the License.
 *******************************************************************************/

//   A simple example of performing the rotate operation of image around the center point.
//   Nearest neighbot interpolation is used
// implemented with Intel(R) Integrated Performance Primitives (Intel(R) IPP) functions:
//     ippiGetRotateTransform
//     ippiWarpAffineGetSize
//     ippiWarpAffineNearestInit
//     ippiWarpGetBufferSize
//     ippiWarpAffineNearest_8u_C1R

#include <stdio.h>
#include "ipp.h"

/* Next two defines are created to simplify code reading and understanding */
#define EXIT_MAIN \
    exitLine: /* Label for Exit */
#define check_sts(st)        \
    if ((st) != ippStsNoErr) \
        goto exitLine; /* Go to Exit if Intel(R) IPP function returned status different from
ippStsNoErr */
```

```
/* Results of ippMalloc() are not validated because Intel(R) IPP functions perform bad arguments
check and will return an appropriate status  */

int main(void)
{
    IppStatus status = ippStsNoErr;

    Ipp8u src[4 * 4] = {4, 4, 4, 4, 3, 3, 3, 3, 2, 2, 2, 2, 1, 1, 1, 1};

    Ipp8u dst[6 * 6];

    IppiSize srcSize = {4, 4};
    IppiSize dstSize = {6, 6};

    double angle = -45.0;
    double xCenter = 1.0;
    double yCenter = 4.0;

    IppiWarpSpec *pSpec = 0;
    double coeffs[2][3];
    int specSize = 0, initSize = 0, bufSize = 0;
    Ipp8u *pBuffer = 0;
    IppiPoint dstOffset = {0, 0};

    check_sts(status = ippiSet_8u_C1R(0, dst, 6, dstSize));

    check_sts(status = ippiGetRotateTransform(angle, 0, 0, coeffs));

    coeffs[0][2] = xCenter - coeffs[0][0] * xCenter - coeffs[0][1] * yCenter;
    coeffs[1][2] = yCenter - coeffs[1][0] * xCenter - coeffs[1][1] * yCenter;

    /* Spec and init buffer sizes */
    check_sts(status = ippiWarpAffineGetSize(srcSize, dstSize, ipp8u, coeffs, ippNearest,
ippWarpForward, ippBorderTransp, &specSize, &initSize));

    /* Memory allocation */
    pSpec = (IppiWarpSpec *)ippsMalloc_8u(specSize);

    /* Filter initialization */
    check_sts(status = ippiWarpAffineNearestInit(srcSize, dstSize, ipp8u, coeffs,
ippWarpForward, 1, ippBorderTransp, NULL, 0, pSpec));

    /* work buffer size */
    check_sts(status = ippiWarpGetBufferSize(pSpec, dstSize, &bufSize));

    pBuffer = ippsMalloc_8u(bufSize);

    /* Image rotation processing */
    check_sts(status = ippiWarpAffineNearest_8u_C1R(src, 4, dst, 6, dstOffset, dstSize, pSpec,
pBuffer));

    /* Result:
     *  0 0 0 0 0 0
     *  0 0 0 4 0 0
     *  0 0 2 3 4 0
     *  0 0 1 2 3 0    dst
     *  0 0 0 1 0 0
```

```
     *  0 0 0 0 0 0
     */

    EXIT_MAIN
    ippsFree(pSpec);
    ippsFree(pBuffer);
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

Bibliography for Image Processing

This bibliography provides a list of publications that might be helpful to you in using the image processing subset of Intel IPP. This list is not complete; it serves only as a starting point. The books [Rog85], [Rog90], and [Foley90] are good resources of information on image processing and computer graphics, with mathematical formulas and code examples.

[Aka96]	A.Akansu, M.Smith (editors). *Subband and Wavelet transform. Design and Applications*, Kluwer Academic Publishers, 1996.
[AP922]	*A Fast Precise Implementation of 8x8 Discrete Cosine Transform Using the Intel® Streaming SIMD Extensions and MMX™ Instructions*, Application Note AP922, Intel Corp. Order number 742474, 1999.
[APMF]	*Fast Algorithms for Median Filtering*, Application Note, Intel Corp. Document number 79835, 2001.
[AVS]	GB/T 200090.2-2006. China Standard. Information Technology . *Coding of Audio-Visual Objects - Part 2: Visual* (02/2006).
[Bert01]	M.Bertalmio, A.L.Bertozzi, G.Sapiro. *Navier-Stokes, Fluid Dynamics, and Image and Video Inpainting.* Proc. ICCV 2001, pp.1335-1362, 2001.
[Bor86]	G.Borgefors. *Distance Transformations in Digital Images*. Computer Vision, Graphics, and Image Processing 34, 1986.
[Bou99]	J-Y.Bouguet. *Pyramidal Implementation of the Lucas-Kanade Feature Tracker.* OpenCV Documentation, Microprocessor Research Lab, Intel Corporation, 1999.
[Canny86]	1. Canny. *A Computational Approach to Edge Detection*, IEEE Trans. on Pattern Analysis and Machine Intelligence 8(6), 1986.
[Davis97]	J.Davis and Bobick. *The Representation and Recognition of Action Using Temporal Templates*. MIT Media Lab Technical Report 402, 1997.
[Davis99]	J.Davis and G.Bradski. *Real-Time Motion Template Gradients Using Intel® Computer Vision Library*. IEEE ICCV'99 FRAME-RATE WORKSHOP, 1999.

[Dalal05]	**1.** Dalal and B. Triggs. *Histograms of Oriented Gradients for Human Detection*. INRIA, 2005.
[DICOM]	Digital Imaging and Communications in Medicine (DICOM), published by National Electrical Manufacturers Association, 2003.
[Feig92]	**1.** Feig and S. Winograd. *Fast Algorithms for the Discrete Cosine Transform*, IEEE Trans. Signal Processing, vol. 40, no. 9, pp. 2174-2193, Sep. 1992.
[Felz04]	**1.** Felzenszwalb, D. Hattenlocher. *Distance Transforms of Sampled Functions*. Cornell Computing and Information Science Technical Report TR2004-1963, September 2004.
[Foley90]	James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics — Principles and Practice*, Second Edition. Addison Wesley, 1990.
[Gon93]	R.C. Gonzalez and R.E.Wood. *Digital Image Processing*. Prentice Hall, 1993.
[Hir05]	**1.** Hirakawa, T.W. Parks, *Adaptive Homogeneity-Directed Demosaicing Algorithm*, IEEE Trans. Image Processing, March, 2005.
[IEC61834]	IEC 61834. International Electrochemical Commission. *Specifications of Consumer-Use Digital VCRs using 6.3 mm magnetic tape* (*the "Blue Book" DV specification*).
[IEEE]	*IEEE Standard Specifications for the Implementations of 8X8 Inverse Discrete Cosine Transform*, IEEE #1180 (1997).
[IPL]	*Intel© Image Processing Library Reference Manual.* Intel Corp. Order number 663791, 1999.
[ISO11172]	ISOCD 11172. Information Technology. *Coding of Moving Pictures and Associated Audio for Digital Storage Media at up to about 1.5 Mbit/s* (1993).
[ISO13818]	ISO/IEC 13818. Information Technology. *Coding of Moving Pictures and Associated Audio Information*, (11/94).
[ISO14496]	International Standard ISO/IEC 14496-2. Information Technology. Coding of Audio-Visual Objects - Part 2: Visual.
[ISO14496A]	ISO/IEC 14496-2:1999/Amd.1:2000(E) . Information Technology. *Coding of Audio-Visual Objects.* Part2:Visual. Amendment 1: Visual Extensions (01/00).
[ISO10918]	International Standard ISO/IEC 10918-1, *Digital Compression and Coding of Continuous Tone Still Images*, Appendix A - *Requirements and guidelines*.

[ISO15444]	International Standard ISO/IEC 15444-1, *JPEG 2000 Image coding system*, part 1: Core coding system.
[ISO29199]	International Standard ISO/IEC 29199-2, *JPEG XR Image coding system*, part 2: Image coding specification. (2009-08-15)
[ITUH261]	ITU-T Recommendation H.261. *Line transmission of non-telephone signals. Video codec for audiovisual services at p x 64 kbits* (03/93).
[ITUH263]	ITU-T Recommendation H.263. Series H: AUDIOVISUAL AND MULTIMEDIA SYSTEMS. *Infrastructure of audiovisual services - Coding of moving video. Video coding for low bit rate communication* (02/98).
[ITUH264]	ITU-T Recommendation H.264. Series H: AUDIOVISUAL AND MULTIMEDIA SYSTEMS. *Infrastructure of audiovisual services - Coding of moving video. Advanced video coding for generic audiovisual services*. (ITU-T Rec. H.264 | ISO/IEC 14496-10:2005)(03/05).
[ITUH264_07]	ITU-T Recommendation H.264. Series H: AUDIOVISUAL AND MULTIMEDIA SYSTEMS. *Infrastructure of audiovisual services - Coding of moving video. Advanced video coding for generic audiovisual services*. (ITU-T Rec. H.264 | ISO/IEC 14496-10:2008)(11/07).
[ITU709]	ITU-R Recommendation BT.709, *Basic Parameter Values for the HDTV Standard for the Studio and International Programme Exchange* [formerly CCIR Rec.709] ITU, Geneva, Switzerland, 1990.
[Jae95]	Jaehne, Bernd. *Digital Image Processing,* 3rd Edition, Springer-Verlag, Berlin, 1995.
[Jae97]	Jaehne, Bernd. *Practical Handbook on Image Processing for Scientific Applications*, CRC Press, New York, 1997.
[Jack01]	Jack, Keith. *Video Demystified: a Handbook for the Digital Engineer*, LLH Technology Publishing, 3rd Edition, 2001.
[JVTG050]	JVT-G050. *ITU-T Recommendation and Final Draft International Standard of Joint Video Specification* (ITU-T Rec. H.264 | ISO/IEC 14496-10 AVC) (03/03).
[Kadew01]	P.KadewTraKuPong, R.Bowden. *An Improved Adaptive Background Mixture Model for Real-Time Tracking with Shadow Detection*. Proc. 2nd European Workshop on Advanced Video-Based Surveilolance Systems, 2001.
[Lein02]	R.Leinhart, J.Maydt. An Extended Set of Haar-like Features for Rapid Object Detection. IEEE ICIP, vol.1, pp. 900-903, Sep. 2002.

[Li03]	L.Li, W.Huang, I.Gu, Q.Tian. *Foreground Object Detection from Videos Containing Complex Background.* Proc.ACM Multimedia Conference, Berkley, 2003.
[Lim90]	Jae S.Lim. *Two-Dimensional Signal and Image Processing*, Prentice Hall, Englewood Cliffs, NJ, 1990.
[Lotufo00]	R.Lotufo, A.Falcao. *The Ordered Queue and the Optimality of the Watershed Algorithm*. Mathematical Morphology and its Applications to Image and Signal Processing, vol.18, pp.341-350. Kluwer Academic Publishers. Palo Alto, USA, June 2000.
[Lowe04]	D.G.Lowe. *Distinctive Image Features from Scale-Invariant Keypoints*. International Journal of Computer Vision, vol.60, No.2, pp. 91-110, 2004.
[Malvar03]	H.S.Malvar, G.J.Sullivan. *Transform, Scaling & Color Space Impact of Professional Extensions*, ISO/IEC JTC/SC29/WG11 and ITU-T SG16 Q.6 Document JVT-H031, Geneva, May 2003.
[Matas00]	J.Matas, C.Galambos, J.V.Kittler. *Robust Detection of Lines Using the Progressive Probabilistic Hough Transform*, CVIU 78 1, pp. 119-137, 2000.
[Malvar03-1]	H.S.Malvar, G.J.Sullivan. *YCoCg-R: A Color Space with RGB Reversibility and Low Dynamic Range*, Joint Video Team (JVT) of ISO/IEC MPEG & ITU-T VCEG, Document No.JVT-1014r3, July 2003.
[Meyer92]	F.Meyer. *Color Image Segmentation*, 4th International Conference on Image Processing and its Applications, p.4, Maastricht, April 1993.
[Meyer94]	F.Meyer. *Topographic Distance and Watershed Lines*. Signal Processing, No.38, pp.113-125, 1994.
[Mitchell88]	Don P. Mitchell, Arun N. Netravali. *Reconstruction Filters in Computer Graphics*. Computer Graphics, Volume 22, Number 4, AT&T Bell Laboratories, Murray Hill, New Jersey, August 1988.
[Myler93]	H.Myler, A.Weeks. *Computer Imaging Recipes in C*, Prentice Hall, 1993.
[Otsu79]	**1.** Otsu. *A Threshold Selection Method From Gray Level Histograms*. IEEE Transactions on Systems, Man, and Cybernetics, vol.9, No.1, January 1979, pp. 62-66.
[Puetter05]	R.C.Puetter, T.R.Gosnell, and Amos Yahil. *Digital Image Recosntuction: Deblurring and Denoising*, Annual Review of Astronomy and Astrophysics, 2005.
[Randy97]	Randy Crane. *A Simplified Approach to Image Processing*, Prentice Hall PTR, 1997.

[Rao90]	K.R. Rao and P. Yip. *Discrete Cosine Transform. Algorithms, Advantages, Applications*. Academic Press, Inc, London, 1990.
[Ric72]	W.Richardson. *Bayesian-Based Iterative Method of Image Reconstruction*. Journal of the Optical Society of America, vol.62, No.1, January 1972.
[Ritter96]	G.Ritter, J.Wilson. *Computer Vision. Algorithms in Image Algebra*. CRC Press, 1996.
[Rog85]	David Rogers. *Procedural Elements for Computer Graphics*. McGraw-Hill, 1985.
[Rog90]	David Rogers and J.Alan Adams. *Mathematical Elements for Computer Graphics*. McGraw-Hill, 1990.
[S3TC]	*S3 Texture Compression*. http://en.wikipedia.org/wiki/S3TC
[Sak98]	1. Sakamoto, C. Nakanishi, and T. Hase, *Software pixel interpolation for digital still cameras suitable for a 32-bit MCU*, IEEE Trans. Consumer Electronics, vol. 44, No. 4, November 1998.
[ScrewBender]	1. Powell, J. Rossignac, *ScrewBender: Polyscrew Subdivision for Smoothing Interpolating Motions* , IEEE Computer Graphics and Applications, vol.28, No. 1, January 2008
[Serra82]	J.Serra. *Image Analysis and Mathematical Morphology*. Academic Press, London, 1982.
[Schu92]	Dale A. Schumacher. *General Filtered Image Rescaling,* Graphic Gems III, Academic Press, 1992.
[Schu94]	Dale A. Schumacher. *A comparison of digital halftoning techniques,* Graphic Gems III, Academic Press, 1994, pp. 57-71.
[Sha98]	Tom Shanley. *Pentium Pro and Pentium II System Architecture*. Addison-Wesley, 1998.
[SMPTE314M]	SMPTE 314M-2005 for Television - *Data Structure for DV-Based Audio, Data and Compressed Video - 25 and 50 Mb/s*. The Society of Motion Picture and Television Engineers (09/05).
[SMPTE370M]	SMPTE 370M-2002 for Television - *Data Structure for DV-Based Audio, Data and Compressed Video at 100 Mb/s, 1080/60i, 1080/50i, 720/60p*. The Society of Motion Picture and Television Engineers (07/02).
[SMPTE370M-06]	SMPTE 370M-2006 for Television - *Data Structure for DV-Based Audio, Data and Compressed Video at 100 Mb/s, 1080/60i, 1080/50i, 720/60p, 720/50p*. The Society of Motion Picture and Television Engineers (04/06).

[SMPTE421M]	SMPTE 421M. Final Draft SMPTE Standard - *VC-1 Compressed Video Bitstream Format and Decoding Process* (01/06).
[Telea04]	A.Telea. *An Image Inprinting Technique Based on the Fast Marching Method.* Journal of Graphic Tools, vol.9, No.1, ACM Press, 2004.
[Tho91]	Spencer W. Thomas and Rod G. Bogart. *Color dithering*, Graphic Gems II, Academic Press, 1991, pp. 72-77.
[Ulichney93]	R.Ulichney. *Digital halftoning*. MIT press, 1993.
[Vincent91]	L.Vincent, P.Soille. *Watershed in Digital Spaces: An Efficient Algorithm Based on Immersion Simulation*. IEEE Transactions of Pattern Analysis and Machine Intelligence, vol.3, No.6, June 1991, pp.583-598.
[Vincent93]	L.Vincent. *Morphological Gray Scale Reconstruction in Image Analysis: Applications and Efficient Algorithms*. IEEE Transactions on Image Processing, vol.2, No.2, April 1993.
[Viola01]	P.Viola, M.J.Jones. *Rapid Object Detection using a Boosted Cascade of Simple Features*. Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR 2001), 2001.
[Wang02]	Z.Wang, A.C.Bovik. *A Universal Image Quality Index*. IEEE Signal Processing Letters, vol.9, No.3, March 2002, pp.81-84.
[Wolberg96]	G.Wolberg. *Digital Image Warping*. IEEE Computer Society Press, 1996.
[Zivkovic04]	Z.Zivkovic. *Improved Adaptive Gaussian Mixture Model for Background Subtraction*. International Conference Pattern Recognition, UK, August, 2004.

Glossary

absolute colors	Colors specified by each pixel's coordinates in a color space. Intel® Integrated Performance Primitives for image processing use images with absolute colors.
alpha channel	A color channel, also known as the opacity channel, that can be used in color models; for example, the RGBA model.
arithmetic operation	An operation that adds, subtracts, multiplies, divides, or squares the image pixel values.
color-twist matrix	A matrix used to multiply the pixel components in one color space for determining the components in another color space.

DCT	Acronym for the discrete cosine transform. *See*Discrete Cosine Transforms in "Image Linear Transforms".
dilation	A morphological operation that sets each output pixel to the minimum of the corresponding input pixel and its 8 neighbors.
dyadic operation	An operation that has two input images. It can have other input parameters as well.
element-wise operation	An element-wise operation performs the same operation on each element of a vector, or uses the elements of the same position in multiple vectors as inputs to the operation.
erosion	A morphological operation that sets each output pixel to the maximum of the corresponding input pixel and its 8 neighbors.
four-channel model	A color model that uses four color channels; for example, the RGBA color model.
gray scale image	An image characterized by a single intensity channel so that each intensity value corresponds to a certain shade of gray.
in-place operation	An operation whose output image is one of the input images.
linear filtering	In this document, 2D convolution operations.
linear image transforms	In this document, the discrete cosine transform (DCT).
MMX™ technology	An enhancement to the Intel® architecture aimed at better performance in multimedia and communications applications. The technology uses four additional data types, eight 64-bit MMX registers, and 57 additional instructions implementing the SIMD (single instruction, multiple data) technique.
monadic operation	An operation that has a single input image. It can have other input parameters as well.
morphological operation	An erosion, dilation, or their combinations.
not-in-place operation	An operation whose output is an image other than the input image(s). *See* in-place operation.
pixel depth	The number of bits determining each channel intensity for a single pixel in the image.
pixel-oriented ordering	Storing the image information in such an order that the values of all color channels for each pixel are clustered; for example, RGBRGB… .
planar-oriented ordering	Storing the image information so that all data of one color channel follow all data of another channel, thus forming a separate "plane" for each channel; for example, RRRRRGGGGGBBBBB….

region of interest	A rectangular image region on which an operation acts (or processing occurs).
RGB	Red-green-blue. A three-channel color model that uses red, green, and blue color channels.
RGBA	Red-green-blue-alpha. A four-channel color model that uses red, green, blue, and alpha (or opacity) channels.
ROI	*See* identity matrix.
row-major order	The default storage method for arrays in C. Memory representation is such that the rows of an array are stored contiguously. For example, for the array `a[3][4]`, the element `a[1][0]` immediately follows `a[0][3]`.
saturation	Using saturation arithmetic, when a number exceeds the data-range limit for its data type, it saturates to the upper data-range limit. For example, a signed word greater than `7FFFh` saturates to `7FFFh`. When a number is less than the lower data-range limit, it saturates to the lower data-range. For example, a signed word less than `8000h` saturates to `8000h`.
Intel® Streaming SIMD Extensions	The enhancement to the Intel architecture instruction set for the next generation processors. It incorporates a group of general-purpose floating-point instructions operating on packed data, additional packed integer instructions, together with cacheability control and state management instructions. These instructions significantly improve performance of applications using compute-intensive processing of floating-point and integer data.
three-channel model	A color model that uses three color channels; for example, the RGB color model.

Getting Help and Support

Getting Technical Support

If you did not register your Intel® Software Product during installation, please do so now at the Intel® Software Development Products Registration Center. Registration entitles you to free technical support, product updates and upgrades for the duration of the support term.

For general information about Intel technical support, product updates, user forums, FAQs, tips and tricks and other support questions, please visit the official Support page .

NOTE If your distributor provides technical support for this product, please contact them rather than Intel.

For technical information about the Intel IPP library, including FAQ’s, tips and tricks, and other support information, please visit the Intel© IPP Forum.