USING INTEL® INTEGRATED PERFORMANCE PRIMITIVES IN

OS X® APPLICATION

# Introduction

The Intel’s Integrated Performance Primitives (Intel® IPP) building blocks are highly optimized using Intel® Streaming SIMD Extensions (Intel® SSE) and Intel® Advanced Vector Extensions (Intel® AVX) instruction sets which makes your application perform faster [1]. This white paper showcases compatibility of Intel® IPP with OS X®. Support for image and video processing operations on two-dimensional signals is available in Intel® IPP libraries. We have developed a sample application to demonstrate integration and use of IPP APIs in OS X® applications. This demo application (ImageResizeIppDemo) implements image resizing capability using IPP image processing libraries. It implements threaded and non-threaded approaches for different image resize filters. In this white paper, we will discuss how to use Xcode® on OS X® to create an application that uses IPP APIs. We will concentrate on how to use IPP Image Resizing API in your OS X® application.

# Intel® IPP Image Resize

The Intel® IPP reference manual [2] describes following approaches for image resizing:

1. Full image resizing:

This approach is only single threaded approach. One can apply this approach when source and destination images are fully accessible in memory during resizing.

1. Tiled image resizing with one prior initialization:

This approach can be implemented using multiple threads for better performance. In this approach source and destination images are not fully accessible in the memory during resizing. Image is split into multiple sections called tiles, all tiles are initialized in the memory and then each tile is resized separately.

1. Tiled image resizing with prior initialization of each tile:

This approach can be used when there are restrictions on amount of memory available for image processing or image is too large. To use this approach, image is required to be divided into multiple tiles. Each source tile can be initialized and processed as independent image so that available amount of memory for processing is utilized without any issues.

For more details on these approaches please refer to the Intel® IPP reference manual [2].

The ImageResizeIppDemo application implements the first two resizing approaches.

Older versions of IPP did not support interpolation filters for image resizing. They supported Square Pixel resizing technique. In IPP 7.1, Square pixel resizing functions are deprecated and the developer is provided with new implementations of following interpolation filters [2]:

1. Nearest Neighbor Resize: In this resizing technique, each pixel in the destination image is filled by copying the value of one pixel at the nearest corresponding location in the source image.
2. Linear Resize: In this resizing technique, the values for destination image pixels are determined by weighing 4 surrounding pixels of corresponding pixel in the source image.
3. Cubic Resize: In this resizing technique, the value for destination image pixels are determined by weighing 16 surrounding pixels of corresponding pixel in the source image.
4. Lanczos Resize: In this resizing technique, samples are weighted via a windowed sine function.
5. Super-sampling: This resizing technique has a limitation that, it can be used only for down sampling or downscaling of images.

In the above list, as we go from top to bottom, quality of resized image increases but image-resizing speed decreases.

The quality of the resized image can be improved further with anti-aliasing support for above resize filters. IPP 7.1 doesn’t support anti-aliasing for these new interpolation filters.

IPP provides resize functions for each interpolation filter type:

* ippiResizeNearest\_<mod>
* ippiResizeLinear\_<mod>
* ippiResizeCubic\_<mod>
* ippiResizeLanczos\_<mod>
* ippiResizeSuper\_<mod>

Here, the value for <mod> depends upon the source image format (i.e. pixels per component, number of channels, etc.).

Before calling any of the above resize functions, we need to make sure everything required for resize functions is setup. Following is the required setup [2]:

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.

IPP also provides very handy memory management and copy functions:

1. ippiMalloc\_<mod>: Allocates memory aligned to 32-byte boundary.
2. ippiFree: Frees memory allocated by the function iipiMalloc.
3. ippiCopy\_<mod>: Copies pixel values between two buffers. Different modes of ippiCopy function also provide ability to divide images into separate planes/channels OR combine separate image planes/channels into a single image.

These memory management functions can be used to prepare input and output memory buffers, working buffer required by resize functions, etc.

For more details about different IPP Image Processing functions please refer to IPP Image Processing user guide.

# Integrating IPP with Xcode®

Install and setup Intel® IPP latest version that comes with the Intel C++ and Fortran compilers distribution packages. Please refer to the IPP Home page to find installers and the installation guide [1]. Generally, on OS X®, IPP gets installed inside ‘/opt/intel/composer\_xe\_<version>/ipp’. The folder ‘/opt/intel/ipp’ is a symbolic link to the ‘/opt/intel/composer\_xe\_<latest\_version>/ipp’.

To develop your own OS X® application, you will need to download and install the latest version of Xcode® on your system. Once you have Xcode ready for development, create a new project and link IPP libraries to your project.

Following are the steps to setup a new IPP project in Xcode 4.x:

1. To create a new Xcode project, select File -> New -> Project menu item.
2. Choose ‘Cocoa Application’ from the provided options and hit ‘Next’ button.
3. Choose options for your new project on the next page and hit ‘Next’ button.
4. Choose a location to save your new project and hit ‘Create’ button. This will create and open your new project in Xcode.
5. Now add required functionality to your project.
6. Import <ipp.h> header file in all the classes, which call IPP functions or use IPP data-types.
7. To build and run your application successfully, link against IPP libraries.

Following are the instructions to link IPP libraries statically or dynamically to your Xcode project:

## Static linking of IPP libraries

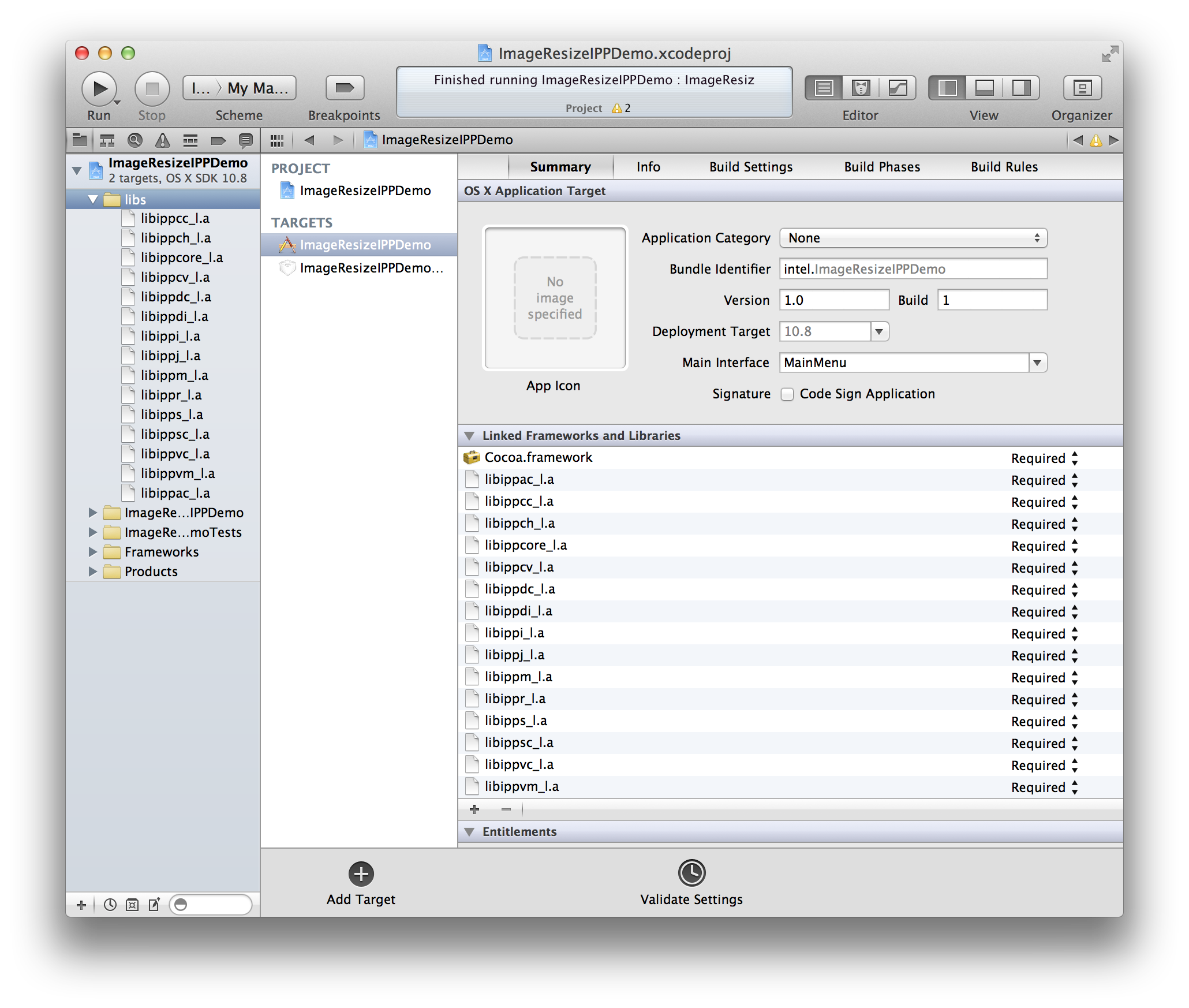
1. In Xcode project, under the project root, create a group named ‘libs’. (Optional)
2. Select ‘libs’ group and open the context menu. Select ‘Add file to <project\_name>’ context menu item. It will open a sheet with add file panel. Browse to the location where IPP is installed, then select required static libraries and click on ‘Add’ button. Screenshot 1 shows the Xcode 4 project in which static IPP libraries are added.
3. Now go to your project’s build settings. Search for ‘Library Search Paths’ and add a non-recursive search path for ipp/libs to it.

(e.g. ‘/opt/intel/ipp/lib’ )

1. In project build settings, search for ‘Header Search Paths’ and add recursive search path to ipp/include to it.

(e.g. ‘/opt/intel/ipp/include’)

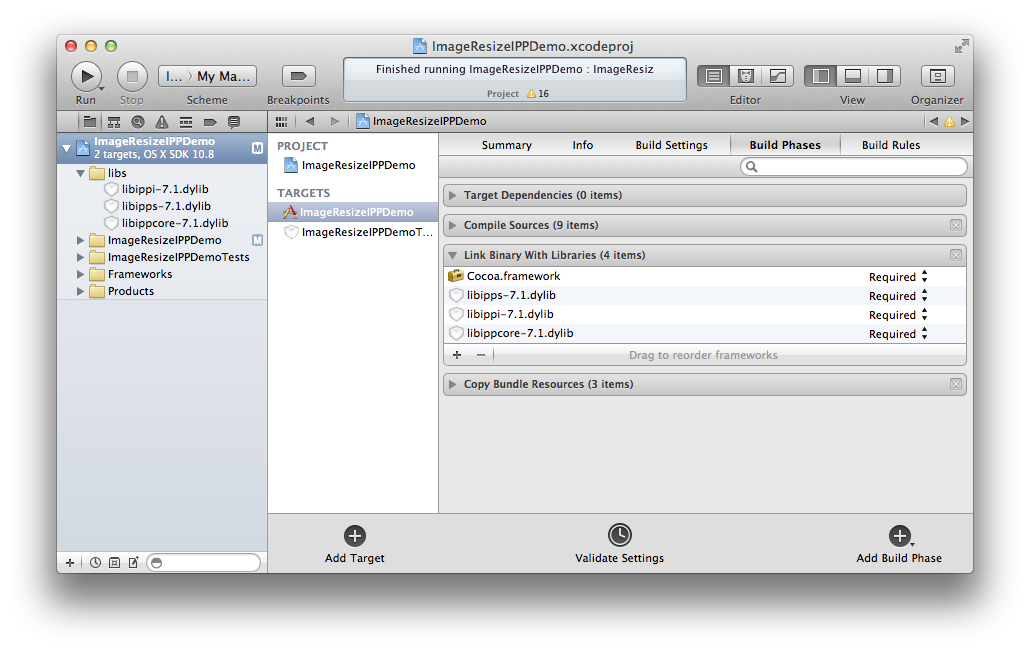
1. Your IPP project should now statically link against IPP libraries without any issues.



Screenshot

## Dynamic linking of IPP libraries

1. If you are using Xcode 4, then go to your project’s ‘Build Phases’.
2. Expand ‘Link Binary With Libraries’ section. Click on ‘+’ to add IPP dylib files. Following screenshot-2 shows IPP dynamic libraries added to the Demo Application.



Screenshot

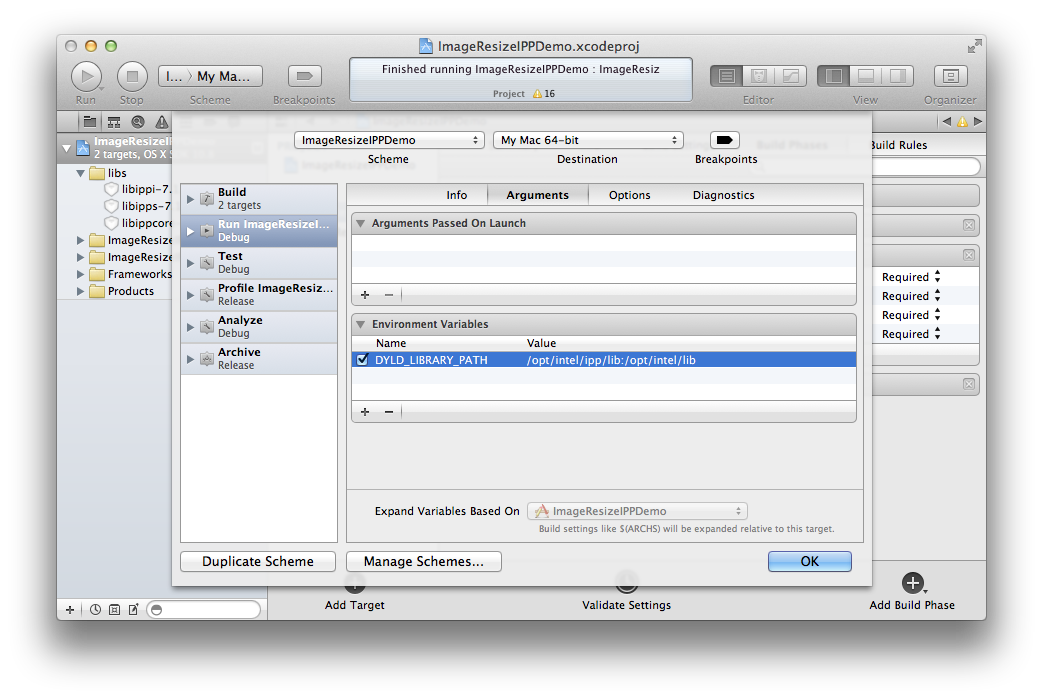
1. Now go to your project’s build settings. Search for ‘Library Search Paths’ and add a non-recursive search path for ipp/libs to it.

(e.g. ‘/opt/intel/ipp/lib’ )

1. In project build settings, search for ‘Header Search Paths’ and add recursive search path to ipp/include to it.

(e.g. ‘/opt/intel/ipp/include’)

1. Last step is to select Xcode’s ‘Product -> Edit Scheme’ menu. Add a new Environment Variable ‘DYLD\_LIBRARY\_PATH’ with value ‘/opt/intel/ipp/lib:/opt/intel/lib’ as shown in the following screenshot-3. Note that even if you set this environment variable from terminal, you will still need to set it again in Xcode.



Screenshot

If you are using an older version of Xcode then please refer to the [Link IPP Libraries to your OS X® project](http://software.intel.com/en-us/articles/intel-integrated-performance-primitives-for-mac-os-how-to-link-to-the-intel-ipp-for-mac-os-x-libraries-in-application) page.

# Using IPP API with Objective-c

IPP function can be called directly from your cocoa objective-c classes. To use IPP functions or data types, objective-c classes are required to include header <ipp.h>. To understand this with example, please download and refer to the <ImageResizeIppDemo> application.

ImageResizeIppDemo application dynamically links IPP libraries to the project. Before running this application from Xcode, make sure that in your project schema, the environment variable ‘DYLD\_LIBRARY\_PATH’ is set to the value ‘/opt/intel/ipp/lib:/opt/intel/lib’.

To run this application by double clicking on the built product (<build\_dir>/Products/Release/ImageResizeIPPDemo.app), user will need to take following steps:

1. Put “setenv DYLD\_LIBRARY\_PATH /opt/intel/ipp/lib:/opt/intel/lib” in file /etc/launchd.conf
2. Open terminal and type following command and hit enter

“launchctl setenv DYLD\_LIBRARY\_PATH /opt/intel/ipp/lib:/opt/intel/lib”

Please refer to the remaining sections of this paper to get more details about the ImageResizeIppDemo application features, implementation and usage.

# Features of IPP Demo Application

The IPP Resize demo application for OS X® has the following features:

* This application is developed using Xcode 4 in Objective-c.
* Application accepts Image to be resized, scaling factor and interpolation filter from user. Using scaling factor and interpolation filter, it resizes the source image using IPP image resize capability and renders the output to the screen.
* For each interpolation filter it has two different implementations, threaded and non-threaded. It uses Apple’s Grand Central Dispatch (GCD) technique for threaded implementation.
* Application allows user to resize original image multiple times without requiring reloading the image.
* Application also allows user to save the image in any image format.

# Implementation Details of ImageResizeIppDemo Application

The ImageResizeIppDemo application reads input image in the CGImage format, but IPP resize functions work on image byte data. Hence, we need logic for conversion from CGImage to image byte data and from Image byte data to CGImage.

Here is how this application implements image format conversions using Apple’s Core Graphics API [4][5]:

### CGImage to Image byte data buffer

- (const void \*)getGCImageData

{

CFDataRef imageData =

CGDataProviderCopyData(CGImageGetDataProvider(inputImage));

NSData \*theNSData = (\_\_bridge NSData\*)imageData;

const void\* ImageBytes = [theNSData bytes];

CFRelease(imageData);

return ImageBytes;

}

### Image byte data buffer to CGImage

- (void)createGCImageFromPixelData:(const void \*)img

stride:(int)stride

width:(int)width

height:(int)height

{

size\_t size = stride \* height;

CGDataProviderRef dataProvider =

CGDataProviderCreateWithData(NULL, img, size, NULL);

CGImageRef inputImage = CGImageCreate( width, height,

self.bitsPerComponent,

self.bitsPerPixel, stride, self.colorSpaceRef,

self.bitmapInfo, dataProvider,

self.decodeArray, self.shouldInterpolate,

self.renderingIntent);

CGDataProviderRelease(dataProvider);

}

This demo application has a class IPPImageScaler, which provides interface for implementation of IPP image resizing with different IPP filters. As IPP image-processing libraries have different versions of API functions depending on image formats (i.e. depending on bits per component value and number of channels in the image.), IPPImageScaler has been further sub-classed into three different classes; IPP8uImageScaler, IPP16uImageScaler, IPP32fImageScaler which override IPP image resize implementations provided by IPPImageScaler.

The IPP API provides memory allocation functions that allocate aligned memory for data of required type. There are different versions of IPP memory allocation functions as well depending on image formats. This demo application implements a function, which prepares memory buffers for source and output images and keeps it ready to be used in IPP resize functions.

Following is an implementation of ‘preparation’ function for 8 bit image formats.

- (void)preparation

{

dstsize.width = mImage.srcImageSize.width \* self.scalingFactor;

dstsize.height = mImage.srcImageSize.height \* self.scalingFactor;

if( mImage.numChannel == 1 )

{

// --------------------------

// get the image size

// --------------------------

pSrc = ippiMalloc\_8u\_C1(mImage.srcImageSize.width,

mImage.srcImageSize.height, &stridesrc);

pDst = ippiMalloc\_8u\_C1(dstsize.width, dstsize.height, &stridedst);

// ---------------------------

// read the source image

// ---------------------------

ippiCopy\_8u\_C1R(srcImage, mImage.srcImageSize.width \*

mImage.bitsPerPixel/8, pSrc, stridesrc, mImage.srcImageSize);

}

else if( mImage.numChannel == 3 )

{

pSrc = ippiMalloc\_8u\_C3(mImage.srcImageSize.width,

mImage.srcImageSize.height, &stridesrc);

pDst = ippiMalloc\_8u\_C3(dstsize.width, dstsize.height, &stridedst);

ippiCopy\_8u\_C3R(srcImage, mImage.srcImageSize.width \*

mImage.bitsPerPixel/8, pSrc, stridesrc, mImage.srcImageSize);

}

else if ( mImage.numChannel == 4 )

{

pSrc = ippiMalloc\_8u\_C4(mImage.srcImageSize.width,

mImage.srcImageSize.height, &stridesrc);

pDst = ippiMalloc\_8u\_C4(dstsize.width, dstsize.height, &stridedst);

ippiCopy\_8u\_C4R(srcImage, mImage.srcImageSize.width \*

mImage.bitsPerPixel/8, pSrc, stridesrc, mImage.srcImageSize);

}

else

{

NSLog(@"Unsupported color component format");

}

}

Note: In this function ‘srcImage’ is nothing but input image byte data returned by conversion function - (const void \*)getGCImageData mentioned above.

This application implements resize functions with two different approaches. The non-threaded approach, which performs full image resizing using IPP API and the threaded approach, which performs tiled image resize using IPP API.

Lets see how these approaches are implemented in Objective-c. Here, we will show the implementation of resizing with linear interpolation filter.

Following is the function declaration in the interface for linear resize:

- (const void \*)linearResizeWithIsthreadedFlag:(BOOL)isThreaded

stride:(NSInteger \*)outputStride;

### Function parameters:

1. (BOOL)isThreaded Flag

This function accepts a flag to decide whether to execute threaded implementation or non-threaded implementation, or in other words, whether to perform full image resize or tiled image resize.

1. (NSInteger \*)outputStride

Another parameter is Stride, which is an output parameter. Value for the stride is calculated for the output image with new size and it is then used while creating GCImage from output image byte data.

### Return Value:

The function returns resized image byte data in the form of (const void \*). With the above conversion function a CGImage is created from this byte data and rendered to the screen.

Following is the implementation of linearResizeWithIsthreadedFlag function:

- (const void \*)linearResizeWithIsthreadedFlag:(BOOL)isThreaded

stride:(NSInteger \*)outputStride

{

Ipp32s specSize=0, initSize=0, bufSize=0;

IppiResizeSpec\_32f \*pSpec = 0;

Ipp8u \*pInitBuf=0;

//set up spec and working buffers

ippiResizeGetSize\_8u( mImage.srcImageSize, dstsize, ippLinear, 0,

&specSize, &initSize );

pInitBuf = ippsMalloc\_8u( initSize );

pSpec = (IppiResizeSpec\_32f\*)ippsMalloc\_8u( specSize );

/\* Filter initialization \*/

ippiResizeLinearInit\_8u( mImage.srcImageSize, dstsize, pSpec );

IppiPoint dstOffset = {0, 0};

if(!isThreaded)

{

/\* work buffer size \*/

ippiResizeGetBufferSize\_8u( pSpec,dstsize,mImage.numChannel,&bufSize );

Ipp8u \*pOneBuf=ippsMalloc\_8u(bufSize);

if( mImage.numChannel == 1 )

{

ippiResizeLinear\_8u\_C1R(pSrc,stridesrc,pDst,stridedst,dstOffset,

dstsize, ippBorderRepl, 0, pSpec, pOneBuf);

}

else if( mImage.numChannel == 3 )

{

ippiResizeLinear\_8u\_C3R(pSrc,stridesrc,pDst,stridedst,dstOffset,

dstsize, ippBorderRepl, 0, pSpec,pOneBuf);

}

else if ( mImage.numChannel == 4 )

{

ippiResizeLinear\_8u\_C4R(pSrc,stridesrc,pDst,stridedst,dstOffset,

dstsize, ippBorderRepl, 0, pSpec,pOneBuf);

}

else

{

NSLog(@"Unsupported color component format");

ippsFree( pOneBuf);

ippsFree( pSpec );

return nil;

}

ippsFree( pOneBuf);

}

else

{

unsigned int numOfChunks = self.availableCpus, slice, tail;

IppiSize dstTileSize, dstLastTileSize;

Ipp32s bufSize1=0, bufSize2=0;

IppiBorderSize borderSize = {0, 0, 0, 0};

IppiBorderType border = ippBorderRepl;

Ipp8u\* pBuffer = 0;

ippiResizeGetBorderSize\_8u(pSpec, &borderSize);

slice = dstsize.height / numOfChunks;

tail = dstsize.height % numOfChunks;

dstTileSize.width = dstLastTileSize.width = dstsize.width;

dstTileSize.height = slice;

dstLastTileSize.height = slice + tail;

ippiResizeGetBufferSize\_8u(pSpec, dstTileSize, mImage.numChannel,

&bufSize1);

ippiResizeGetBufferSize\_8u(pSpec, dstLastTileSize, mImage.numChannel,

&bufSize2);

pBuffer = ippsMalloc\_8u(bufSize1 \* (numOfChunks - 1) + bufSize2);

dispatch\_queue\_t queue =

dispatch\_get\_global\_queue( DISPATCH\_QUEUE\_PRIORITY\_DEFAULT, 0 );

if (pBuffer)

{

dispatch\_apply( numOfChunks, queue, ^(size\_t count) {

Ipp8u \*pSrcT, \*pDstT;

IppiPoint srcOffset = {0, 0};

IppiPoint dstOffset = {0, 0};

IppiSize srcSizeT = mImage.srcImageSize;

IppiSize dstSizeT = dstTileSize;

dstSizeT.height = slice; dstOffset.y += count \* slice;

if (count == numOfChunks - 1)

dstSizeT = dstLastTileSize;

ippiResizeGetSrcRoi\_8u(pSpec, dstOffset, dstSizeT, &srcOffset,

&srcSizeT);

pSrcT = (Ipp8u\*)((char\*)pSrc + srcOffset.y \* stridesrc);

pDstT = (Ipp8u\*)((char\*)pDst + dstOffset.y \* stridedst);

if( mImage.numChannel == 1 )

{

ippiResizeLinear\_8u\_C1R(pSrcT,stridesrc,pDstT,stridedst,

dstOffset, dstSizeT,border, 0, pSpec,

pBuffer + count \* bufSize1);

}

else if( mImage.numChannel == 3 )

{

ippiResizeLinear\_8u\_C3R(pSrcT,stridesrc,pDstT,stridedst,

dstOffset, dstSizeT,border, 0, pSpec,

pBuffer + count \* bufSize1);

}

else if ( mImage.numChannel == 4 )

{

ippiResizeLinear\_8u\_C4R(pSrcT,stridesrc,pDstT,stridedst,

dstOffset, dstSizeT,border, 0, pSpec,

pBuffer + count \* bufSize1);

}

});

}

ippsFree( pBuffer );

}

ippsFree( pInitBuf );

ippsFree( pSpec );

\*outputStride = (NSInteger)stridedst;

return pDst;

}

Note: All the functions used above that start with ‘ipp’ and the data types that start with ‘IPP’ are from IPP API. Please refer to the IPP User Guide for more details about different IPP image processing functions and data types used in the above implementation.

## Apple’s Grand Central Dispatch (GCD)

Use of Apple’s GCD [6] provides support for concurrent code execution on multicore hardware systems with OS X®. Please refer to the above implementation of linearResizeWithIsthreadedFlag method. For the threaded (tiled image resize approach) implementation, we are using GCD API. We are horizontally dividing image data into number of chunks. Total number of chunks is same as number of currently available CPUs on the system, so that each CPU can resize a chunk in parallel reducing the time required for resizing complete image.

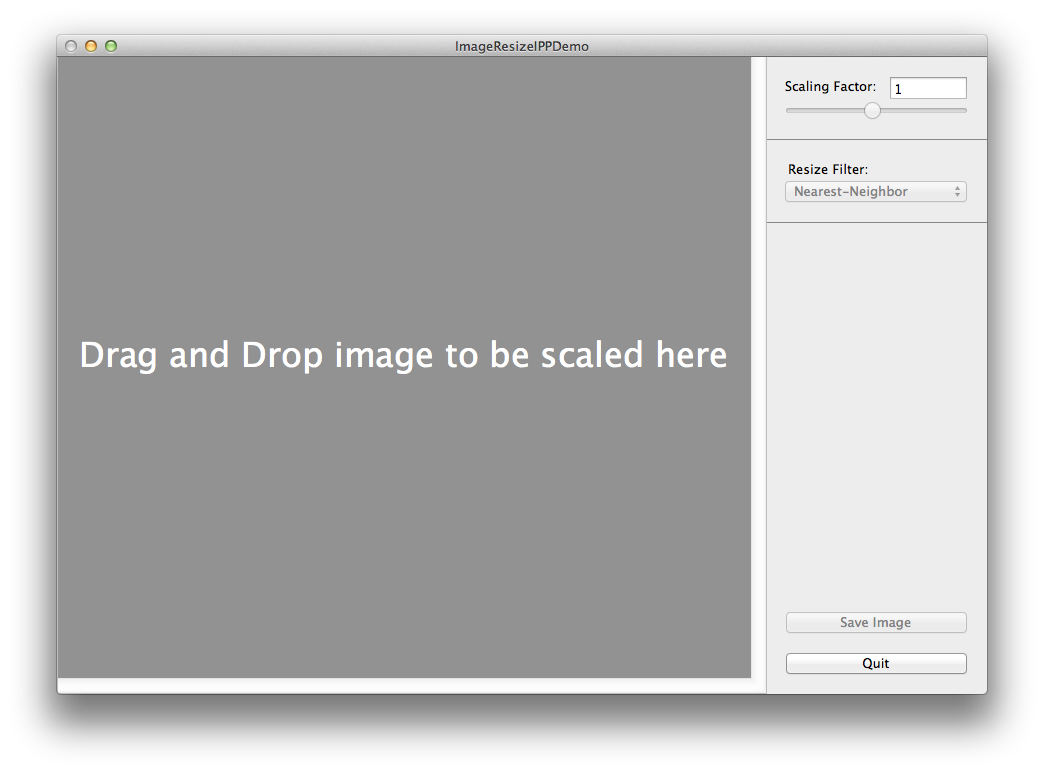
To en-queue image chunks for parallel processing, GCD needs a dispatch\_queue. Application uses one of the global dispatch queues with default priority. To get global dispatch queue we are using GCD API function dispatch\_get\_global\_queue.

dispatch\_apply Function takes three parameters as shown in the above implementation.

1. dispatch\_queue
2. A block of code to be executed in parallel
3. Number of times the provided block of code is to be executed. This is same as number of image data chunks, as, the block of code should be executed for each image data chunk.

# User Interface Details of ImageResizeIppDemo Application

When a user runs the application, following window is shown to the user.

****

Screenshot

The user can drag and drop an image to be resized on the ImageView shown above. This image is represented in the application using Apple’s Core Graphics API.

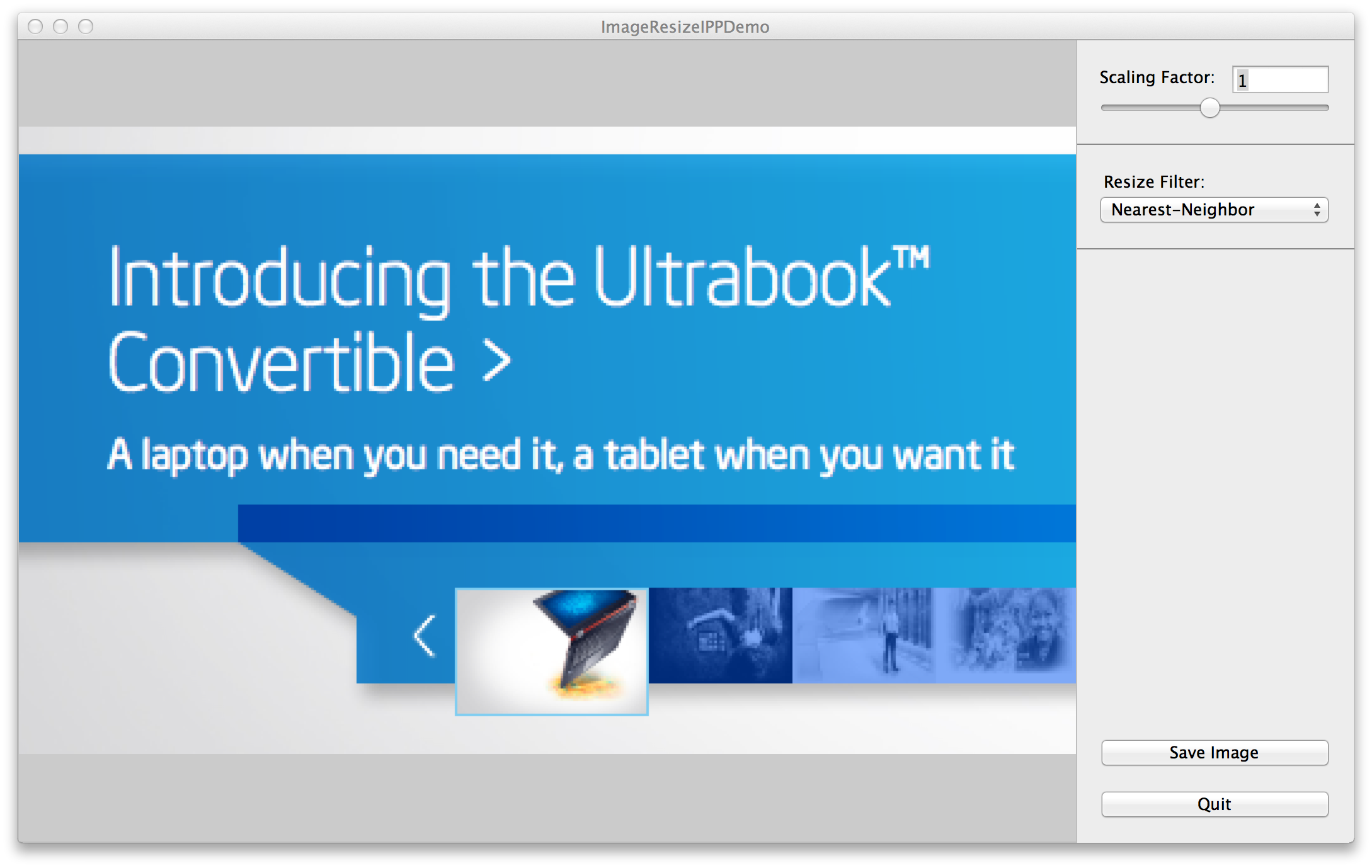
An NSSlider is provided to accept a scaling factor value. The scaling factor value selected by the slider is shown in the Text Field displayed above the slider.

The NSPopUpButton in the user interface allows the user to choose an interpolation filter from available options.

The ‘Save Image’ button allows user to save an image displayed in the imageView with its current dimensions.

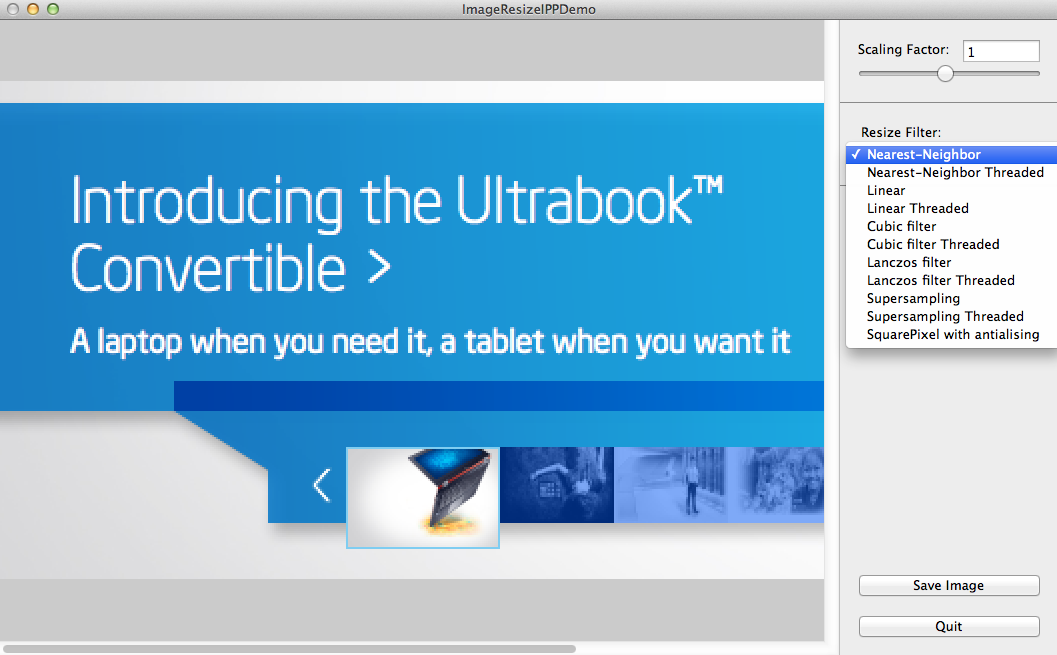
The ‘Quit’ button allows user to close the application window and quit the application. The user cannot use red close button on top left corner of the window to close application window or quit the application.

As you see in Screenshot-4, initially, all controls on the window are disabled when no image is selected for resizing. As soon as the user drops image on the ImageView, all controls are enabled.



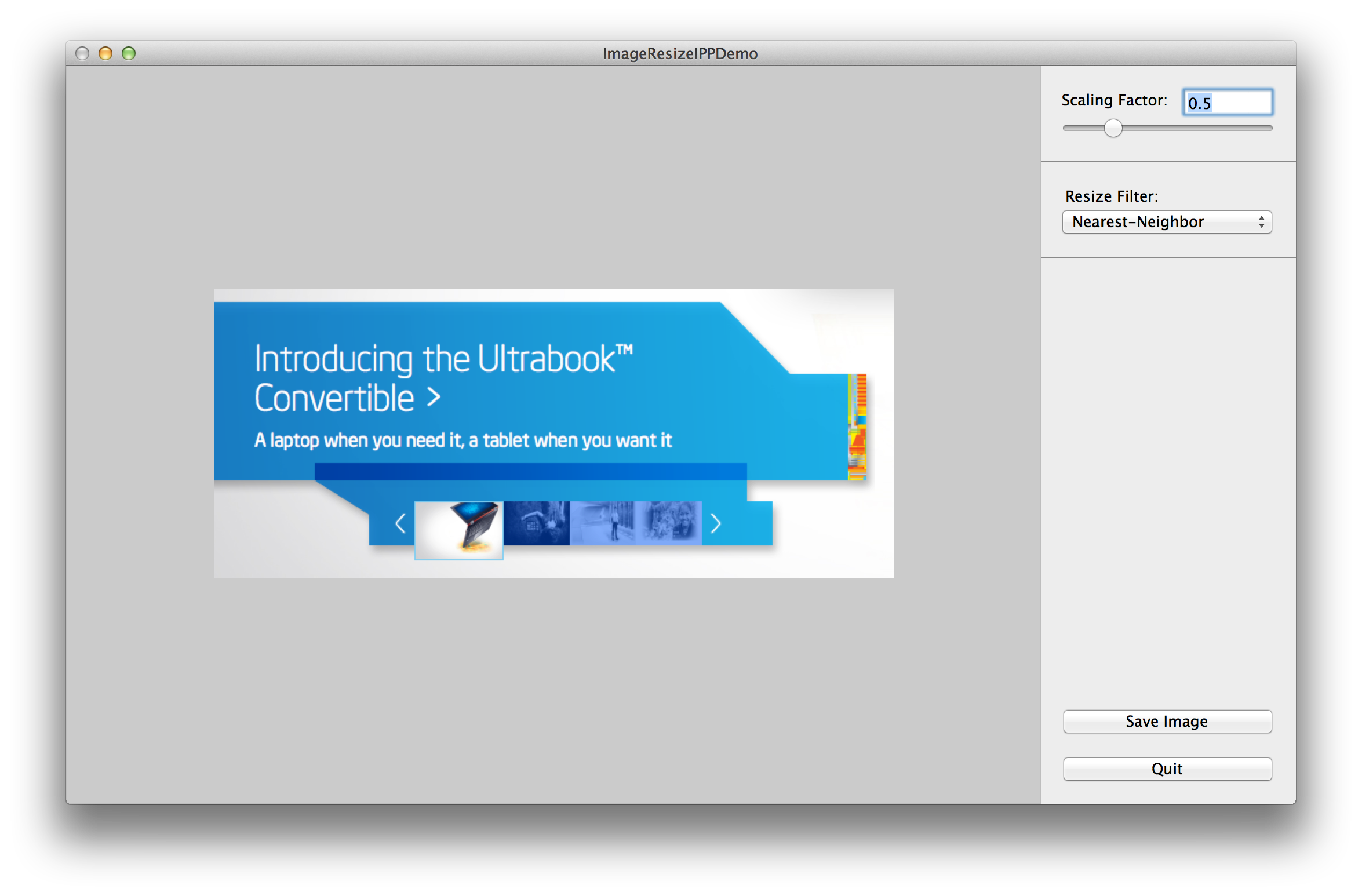
Screenshot

Screenshot-5 shows the window with an image loaded in the ImageView and all controls enabled.

****

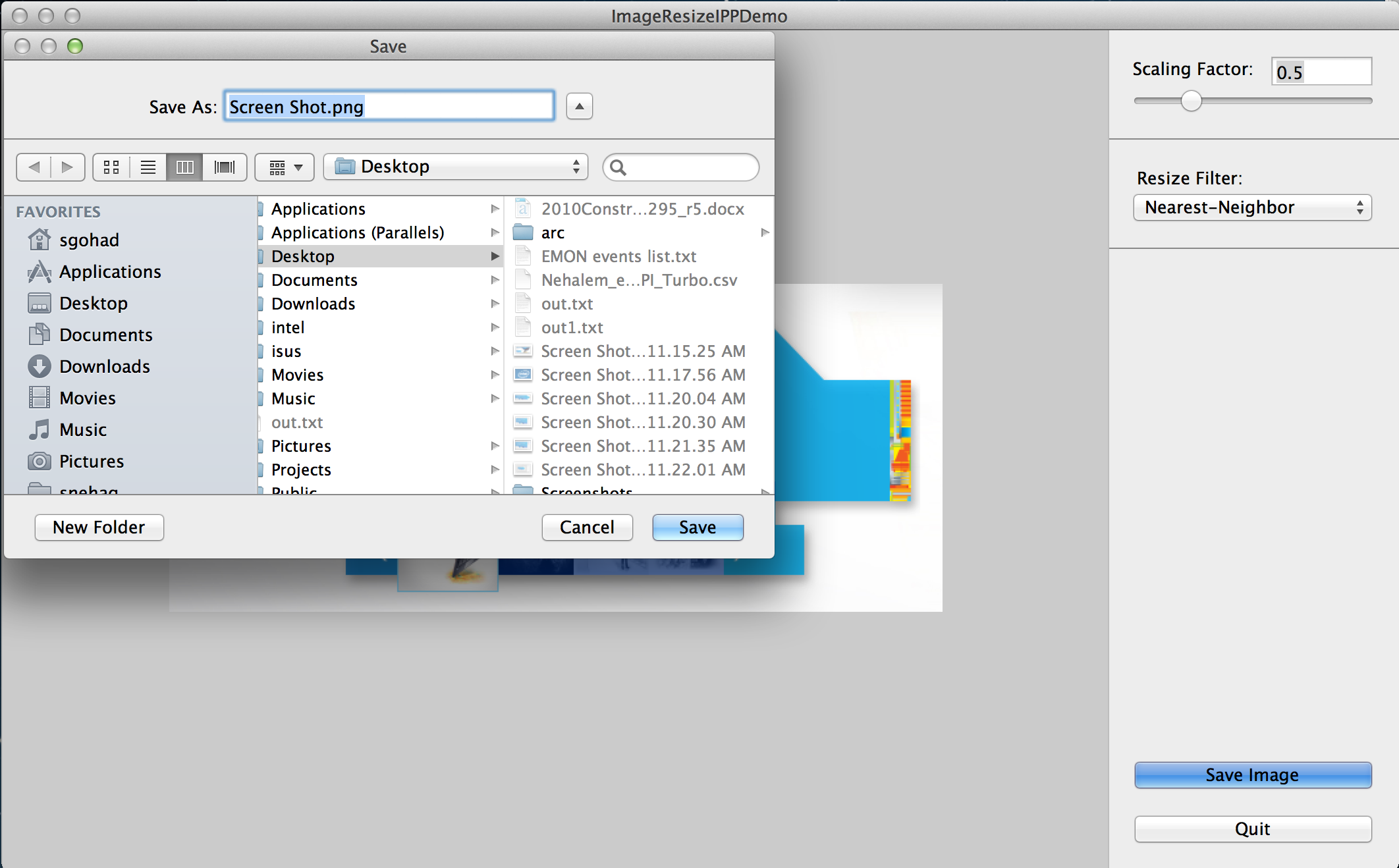
Screenshot

Screenshot-6 shows how a user can select interpolation filter and available interpolation filter options. Whenever the user changes interpolation filter, the resize functionality gets triggered and the image is scaled with the pre-selected scaling factor and the new resize filter.



Screenshot

Screenshot 7 shows the resized (down-sampled) image to the selected scaling factor with selected resize filter. The resize functionality gets triggered when user changes the scaling factor value. The user can also manually edit the scaling factor value in the TextField and slider adjusts its position to the new value, which again triggers image resize.



Screenshot

Screenshot 8 shows how a user can save the resized image. The save panel is opened with pre-populated image file name when user clicks on the ‘Save Image’ button. The user can change the image extension to choose to save image with different image types.

# Image Formats supported in ImageResizeIppDemo Application

This demo application supports the following Image formats:

* Gray scale 8 bit/16 bit/ 32 bit
* Gray scale with alpha channel
* Indexed
* Indexed with alpha channel
* RGB 8 bit/16 bit/ 32 bit
* RGB with alpha channel
* CMYK 8 bit/16 bit/ 32 bit

IPP does not have direct support to handle resizing of 2 channel (e.g. 8 bit Gray scale + Alpha channel) or 5 channel (e.g. CMYK + Alpha channel) images. You can still support these image formats using the following techniques:

### 2 Channel Images

There are two approaches to handle this case. In some cases just using 16 bit IPP functions to resize this type of images works. This approach works well with Nearest-Neighbor interpolation filter. Unfortunately, this is not a great solution, as you can get some strange looking results for some other interpolation filters.

Another approach is to split the image into 2 planes; 1 for Gray and 1 for Alpha channel. Then you can safely run two resizes on each image plane (buffer) separately.

ImageResizeIppDemo application currently implements the first approach to handle 2 channel images. If resizing 8-bit Gray scale with Alpha channel, then it treats that image as 16 bit image while resizing.

### 5 Channel Images

In case of 5 channel images, you can split image in separate planes. For example, you can split the image into 5 one-channel images OR 1 four-channel image and 1 one-channel image. Then you can safely run resizes on separate image buffers and combine resized image buffers back to one when done. ImageResizeIppDemo application doesn’t support scaling of 5 channel images currently.

IPP provides some ippiCopy functions that help in separating images into multiple planes and also combining separate planes into one image.

# Workloads used for testing of ImageResizeIppDemo Application

The following table shows the image types and image formats for which this ImageResizeIppDemo application is tested:

***Note:*** 

✔= Image format available in test data and Application resizes this image correctly.

**−** = Test image not available.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | TIFF | PSD | PNG | GIF | JPG | BMP |
| Gray Scale |  |  |  |  |  |  |  |
|  | 8-bit  No Alpha |   ✔ |   ✔ | **−** | **−** |   ✔ |   ✔ |
|  | 8-bit  kCGImageAlphaLast |   ✔ |   ✔ | **−** | **−** | **−** |   ✔ |
|  | 8-bit  kCGImageAlphaNoneSkipLast | **−** | **−** | **−** | **−** | **−** |   ✔ |
|  | 8-bit  kCGImageAlphaPremultipliedLast |   **−** | **−** | **−** | **−** | **−** |   ✔ |
|  | 16-bit  No Alpha |   ✔ |   ✔ | **−** | **−** | **−** |   ✔ |
| Indexed |  |  |  |  |  |  |  |
|  | 8-bit  No Alpha |   ✔ |   ✔ | **−** |   ✔ | **−** |   ✔ |
|  | 8-bit  kCGImageAlphaLast |   ✔ |   ✔ | **−** |   ✔ | **−** |   ✔ |
| RGB |  |  |  |  |  |  |  |
|  | 8-bit  No Alpha |   ✔ |   ✔ |   ✔ | **−** |   ✔ |   ✔ |
|  | 8-bit  kCGImageAlphaLast |   ✔ |   ✔ |   ✔ | **−** | **−** |   ✔ |
|  | 8-bit  kCGImageAlphaNoneSkipLast |   ✔ |   ✔ |   ✔ | **−** |   ✔ |   ✔ |
|  | 8-bit  kCGImageAlphaPremultipliedLast |   ✔ |   ✔ | **−** | **−** | **−** |   ✔ |
|  | 16-bit  No Alpha |   ✔ |   ✔ | **−** | **−** | **−** |   ✔ |
| CMYK |  |  |  |  |  |  |  |
|  | 8-bit  No Alpha |   ✔ |   ✔ | **−** | **−** |   ✔ |   **−** |
|  | 16-bit  No Alpha |   ✔ |   ✔ | **−** | **−** | **−** |   **−** |

Table

# Summary

Because of the highly optimized implementation of Intel® IPP libraries, the applications using IPP APIs perform faster. Intel® IPP can be integrated in development tools on OS X® to use different IPP APIs in cocoa applications.

# References

1. **Intel® IPP Home Page:** http://software.intel.com/en-us/intel-ipp
2. **Intel® IPP Reference manual:** http://software.intel.com/en-us/articles/intel-integrated-performance-primitives-documentation/
3. **Link IPP Libraries to your OS X® project:** http://software.intel.com/en-us/articles/intel-integrated-performance-primitives-for-mac-os-how-to-link-to-the-intel-ipp-for-mac-os-x-libraries-in-application
4. **Quartz 2D Programming Guide:** https://developer.apple.com/library/mac/#documentation/GraphicsImaging/Conceptual/drawingwithquartz2d/Introduction/Introduction.html
5. **Core Graphics Reference Collection:** https://developer.apple.com/library/mac/#documentation/GraphicsImaging/Reference/CoreGraphicsReferenceCollection/\_index.html
6. **Grand Central Dispatch (GCD):** https://developer.apple.com/library/mac/#documentation/Performance/Reference/GCD\_libdispatch\_Ref/Reference/reference.html

# About the Author

Sneha Gohad is an Application Engineer in Intel’s Developer’s Relations Division within the Software and Services Group at Intel, Santa Clara. Sneha has completed her MS in Computing Studies from Arizona State University with specialization in application development.

# Notices

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS OTHERWISE AGREED IN WRITING BY INTEL, THE INTEL PRODUCTS ARE NOT DESIGNED NOR INTENDED FOR ANY APPLICATION IN WHICH THE FAILURE OF THE INTEL PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document 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.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to: <http://www.intel.com/design/literature.htm>

Intel, the Intel logo, VTune, Cilk and Xeon are trademarks of Intel Corporation in the U.S. and other countries.

\*Other names and brands may be claimed as the property of others

Copyright© 2012 Intel Corporation. All rights reserved.

**Optimization Notice**

Optimization Notice

Intel’s compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2®, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804