# Using OpenCL™ 2.0 Read-Write Images

## Tutorial

**Intel ® SDK for OpenCL™ Applications - Tutorial**

# Legal Information

By using this document, in addition to any agreements you have with Intel, you accept the terms set forth below. You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein. 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. A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS. 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 processor numbers are not a measure of performance. Processor numbers differentiate features within each processor family, not across different processor families. Go to: http://www.intel.com/products/processor_number/. Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. Intel, Intel logo, Intel Core, VTune, Xeon are trademarks of Intel Corporation in the U.S. and other countries. * Other names and brands may be claimed as the property

# Introduction

Prior to OpenCL™ 2.0, there was no ability to read and write to an image within the same kernel. Images could always be declared as a “CL_MEM_READ_WRITE”, but once the image was passed to the kernel, it had to be either “__read_only” or “__write_only”.

```
input1 = clCreateImage(
oclobjects.context,
CL_MEM_READ_WRITE|CL_MEM_COPY_HOST_PTR,
&format,
&desc,
&input_data1[0],
&err );
SAMPLE_CHECK_ERRORS( err );
```

**Code 1 Image buffer could be created with CL_MEM_READ_WRITE**

```
__kernel void Alpha( __read_write image2d_t inputImage1, __read_only image2d_t
inputImage2, uint width, uint height, float alpha, float beta, int gamma )
```

**Code 2 OpenCL 2.0 introduced the ability to read and write to images in Kernels**

The addition, while intuitive, comes with a few caveats that are discussed in the next section.

# The value of Read-Write Images

While Image convolution is not as effective with the new Read-Write images functionality, any image processing technique that needs be done in place may benefit from the Read-Write images. One example of a process that could be used effectively is image composition.

In OpenCL 1.2 and earlier, images were qualified with the “__read_only” and __write_only” qualifiers. In the OpenCL 2.0, images can be qualified with a “__read_write” qualifier, and copy the output to the input buffer. This reduces the number of resources that are needed.

Since OpenCL 1.2 images are either read_only or write_image. Performing an in-place modifications of an image requires treating the image as a buffer and operating on the buffer (see cl_khr_image2d_from_buffer: https://software.intel.com/en-us/articles/using-image2d-from-buffer-extension.

The current solution is to treat the images as buffers, and manipulate the buffers. Treating 2d images as buffers many not be a free operation and prevents clamping and filtering abilities available in read_images from being used. As a result, it may be more desirable to use read_write qualified images.

## Overview of the Sample

The sample takes two windows bitmap images “input1.bmp” and “input2.bmp” and puts them into an image buffer. These images are then composited based on the value of the alpha, a weight factor in the equation of the calculated pixel, which can be passed in as an option.

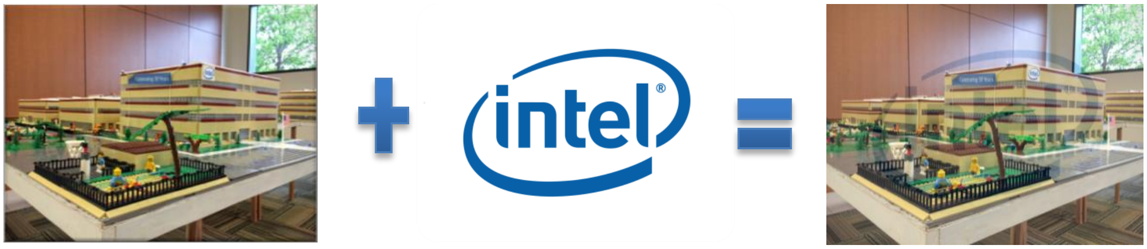


**Figure 1 Using Alpha value 0.84089642**

The images have to be either 24/32-bit images. The output is a 24-bit image. The images have to be of the same size. The images were also of the Format ARGB, so when loading that fact was taken into consideration.

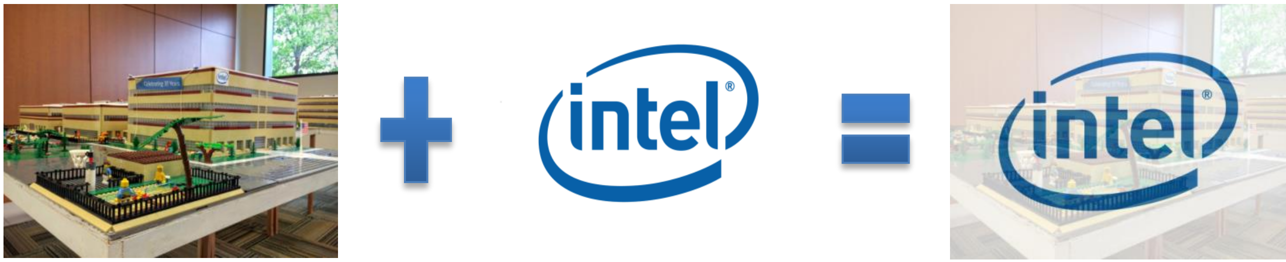


**Figure2. Using Alpha value of 0.32453**

The ARGB is converted to RGBA. Changing the value of the beta value causes a significant change in the output.

## Using the Sample SDK

The SDK demonstrates how to use image composition with Read write images. Use the following command-line options to control this sample:

| Options | Description |
|---|---|
| -h, --help | Show this text and exit |
| -p, --platform number-or-string | Select platform, devices of which are used |
| -t, --type all \| cpu \| gpu \| acc \| default \| <OpenCL constant for device type> | Select the device by type on which the OpenCL Kernel is executed |
| -d, --device number-or-string | Select the device on which all stuff is executed |
| -i, --infile 24/32-bit .bmp file | Base name of the first .bmp file to read. Default is **input1.bmp** |
| -j, --infile 24/32-bit .bmp file | Base name of the second .bmp file to read Default |

| | |
|---|---|
| | is **input2.bmp** |
| -o, --outfile 24/32-bit .bmp file | Base name of the output to write to. Default is **output.bmp** for OCL1.2 and **20_output.bmp** for OCL2.0 |
| -a, --alpha floating point value between 0 and 1 | Non-zero positive value that determines how much the two images will blend in composition. Default **alpha** is 0.84089642. Default **beta** value is 0.15950358. |

The sample SDK has a number of default values that allow the application to be able to run without any user input. The user will be able to use their input .bmp files. The files have to be either 24/32 bmp files as well. The alpha value is used to determine how much prominence image one will have over image 2 as such:

```
calculatedPixel = ((currentPixelImage1 * alpha) + (currentPixeImage2 * beta)
+ gamma);
```

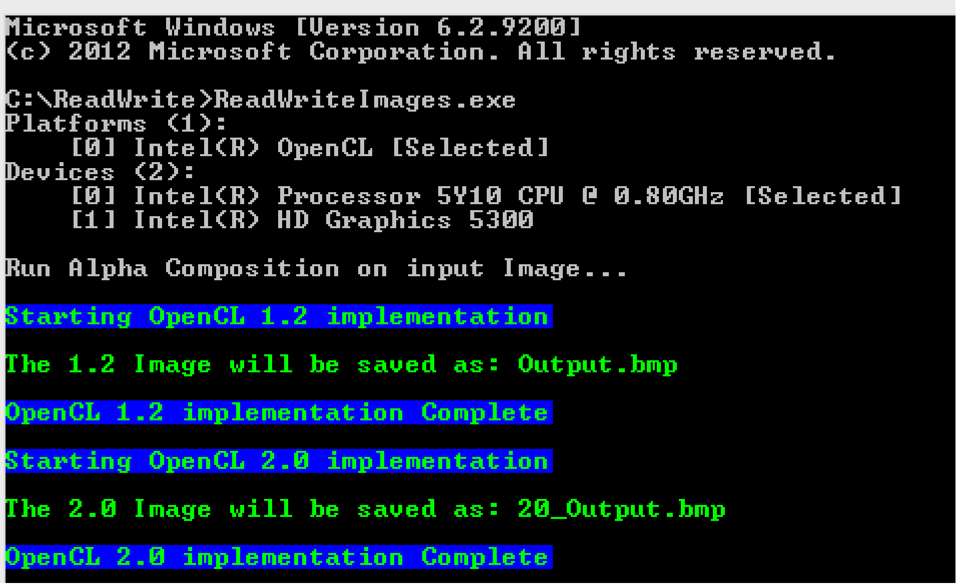
The beta value is determined by subtracting the value of the alpha from 1.

```
float beta = 1 – alpha;
```

These two values determine the weighted distribution of images 1 to image 2.

The gamma value can be used to brighten each of the pixels. The default value is 0. But user can brighten the overall composited image.

## Example Run of Program



```
Microsoft Windows [Version 6.2.9200]
(c) 2012 Microsoft Corporation. All rights reserved.

C:\ReadWrite>ReadWriteImages.exe
Platforms (1):
    [0] Intel(R) OpenCL [Selected]
Devices (2):
    [0] Intel(R) Processor 5Y10 CPU @ 0.80GHz [Selected]
    [1] Intel(R) HD Graphics 5300

Run Alpha Composition on input Image...

Starting OpenCL 1.2 implementation

The 1.2 Image will be saved as: Output.bmp

OpenCL 1.2 implementation Complete

Starting OpenCL 2.0 implementation

The 2.0 Image will be saved as: 20_Output.bmp

OpenCL 2.0 implementation Complete
```

Figure 3 Program running on OCL2.0 Device

## Limitations of Read-Write Images

Barriers cannot be used with images that require synchronization across different workgroups. Image convolution requires synchronizing all threads. Convolution with respect to images usually involves a mathematical operation on two matrices that results in the creation of a third matrix. An example of an image convolution is using Gaussian blur. Other examples are image sharpening, edge detection, and embossing.

Let's use Gaussian blur as an example. A Gaussian filter is a low pass filter that removes high frequency values. The implication of this is to reduce detail and eventually cause a blurring like effect. Applying a Gaussian blur is the same as convolving the image with a Gaussian function that is often called the mask. To effectively show the functionality of Read-Write images, a horizontal and vertical blurring had to be done.

In OpenCL 1.2, this would have to be done in two passes. One kernel would be exclusively used for the horizontal blur, and another does the vertical blur. The result of one of the blurs would be used as the input of the next one depending on which was done first.

```
__kernel void GaussianBlurHorizontalPass( __read_only image2d_t inputImage, __write_only
image2d_t outputImage, __constant float* mask, int maskSize)
{
    int2 currentPosition = (int2)(get_global_id(0), get_global_id(1));
    float4 currentPixel = (float4)(0,0,0,0);
    float4 calculatedPixel = (float4)(0,0,0,0);
    for(int maskIndex = -maskSize; maskIndex < maskSize+1; ++maskIndex)
    {
        currentPixel = read_imagef(inputImage, imageSampler, currentPosition +
(int2)(maskIndex, 0));
        calculatedPixel += currentPixel * mask[maskSize + maskIndex];
    }
    write_imagef(outputImage, currentPosition, calculatedPixel);
}
__kernel void GaussianBlurVerticalPass( __read_only image2d_t inputImage, __write_only
image2d_t outputImage, __constant float* mask, int maskSize)
{
    int2 currentPosition = (int2)(get_global_id(0), get_global_id(1));
    float4 currentPixel = (float4)(0,0,0,0);
    float4 calculatedPixel = (float4)(0,0,0,0);
    for(int maskIndex = -maskSize; maskIndex < maskSize+1; ++maskIndex)
    {
        currentPixel = read_imagef(inputImage, imageSampler, currentPosition + (int2)(0,
maskIndex));
        calculatedPixel += currentPixel * mask[maskSize + maskIndex];
    }
    write_imagef(outputImage, currentPosition, calculatedPixel);
}
```

**Code 3 Gaussian Blur OCL1.2 Kernel**

The idea for the OpenCL 2.0 would be to combine these two kernels into one. Use a barrier to force the completion of each of the horizontal or vertical blurs before the next one begins.

```
__kernel void GaussianBlurDualPass( __read_only image2d_t inputImage, __read_write
image2d_t tempRW, __write_only image2d_t outputImage, __constant float* mask, int
maskSize)
{
    int2 currentPosition = (int2)(get_global_id(0), get_global_id(1));
    float4 currentPixel = (float4)(0,0,0,0);
    float4 calculatedPixel = (float4)(0,0,0,0)
    currentPixel = read_imagef(inputImage, currentPosition);
    for(int maskIndex = -maskSize; maskIndex < maskSize+1; ++maskIndex)
    {
        currentPixel = read_imagef(inputImage, currentPosition + (int2)(maskIndex, 0));
        calculatedPixel += currentPixel * mask[maskSize + maskIndex];
    }
    write_imagef(tempRW, currentPosition, calculatedPixel);
    barrier(CLK_GLOBAL_MEM_FENCE);
    for(int maskIndex = -maskSize; maskIndex < maskSize+1; ++maskIndex)
    {
        currentPixel = read_imagef(tempRW, currentPosition + (int2)(0, maskIndex));
        calculatedPixel += currentPixel * mask[maskSize + maskIndex];
    }
    write_imagef(outputImage, currentPosition, calculatedPixel);
```

Code 4 Gaussian Blur 2.0 Kernel

Barriers were found to be ineffective. Using a barrier does not guarantee that the horizontal blur is completed before the vertical blur begins, assuming you did the horizontal blur first. The implication of this was an inconsistent result in multiple runs. Barriers can be used to synchronize threads within a group. The reason the problem occurs is that edge pixels are read from multiple workgroups, and there is no way to synchronize multiple workgroups. The initial assumption that we can implement a single Gaussian blur using read_write images proved incorrect because the inter-workgroup data dependency cannot be synchronized in OpenCL.

## References

- OpenCL 2.0 Spec: https://www.khronos.org/registry/cl/specs/opencl-2.0.pdf
- OpenCL Programming Guide – A. Munshi et al (Pearson 2012)
- OpenCL Programming by Example – Banfer, Ravishekhar
- Shapiro, L. G. & Stockman, G. C: "Computer Vision", page 137, 150. Prentice Hall, 2001
- https://en.wikipedia.org/wiki/Alpha_compositing
- http://mathnathan.com/2010/07/5opencv/