UOW AUSTRALIA

SCIT

School of Computing & Information Technology

CSCI376 – Multicore and GPU Programming
Spring 2018

Assignment 3

Due on Wednesday, 17th October 2018 at 23:55

Part 1: Stencil Parallel Pattern - Gaussian Blurring

Gaussian blurring is a commonly used technique to image processing and graphics to create a smooth blurring effect using a Gaussian function. The weights of the filter depend on the size of the Gaussian filter window. The following are example weights for 3x3, 5x5 and 7x7 windows:

0.077847	0.123317	0.077847
0.123317	0.195346	0.123317
0.077847	0.123317	0.077847

0.003765	0.015019	0.023792	0.015019	0.003765
0.015019	0.059912	0.094907	0.059912	0.015019
0.023792	0.094907	0.150342	0.094907	0.023792
0.015019	0.059912	0.094907	0.059912	0.015019
0.003765	0.015019	0.023792	0.015019	0.003765

0.000036	0.000363	0.001446	0.002291	0.001446	0.000363	0.000036
0.000363	0.003676	0.014662	0.023226	0.014662	0.003676	0.000363
0.001446	0.014662	0.058488	0.092651	0.058488	0.014662	0.001446
0.002291	0.023226	0.092651	0.146768	0.092651	0.023226	0.002291
0.001446	0.014662	0.058488	0.092651	0.058488	0.014662	0.001446
0.000363	0.003676	0.014662	0.023226	0.014662	0.003676	0.000363
0.000036	0.000363	0.001446	0.002291	0.001446	0.000363	0.000036

a) Write an OpenCL program that accepts a colour image and outputs the results of a filtered image using Gaussian blurring. Your kernel (use a single kernel function) should be able to handle different window sizes based on the weights provided above.

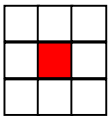
(2 marks)

b) Modify your program to display all the available devices on a computer (if a device is supported by more than one platform, any platform will do), and allow the user to select which of the devices to run the kernel on.

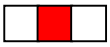
(2 marks)

c) Instead of using the 3x3, 5x5 and 7x7 windows (the naïve approach), an alternate approach is to run the filter in 2 passes. The first pass will be in the horizontal direction; The result will then undergo a second pass in the vertical direction (enqueue the kernel twice to perform the blur in different directions). The result will be the same as the single window approach, but the amount of computation will be different.

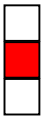
For example, using a 3x3 window directly, each pixel will have to perform a weighted sum on 9 pixels. In the 2-pass approach, each pixel will have to perform a weighted sum on 3 pixels in each pass, processing a total of 6 pixels. This is illustrated below:



3x3 window



horizontal pass



vertical pass

This obviously scales with window size, e.g., in a 7x7 window, the weighted sum for each pixel has to deal with 49 pixels, whereas the 2-pass approach deals with 14 pixels.

Your task is to implement the parallel 2-pass approach. For this, use the following weights for both the horizontal and vertical passes of the respective 3x3, 5x5 and 7x7 cases:

		0.27901	0.44198	0.27901		
	0.06136	0.24477	0.38774	0.24477	0.06136	
0.00598	0.060626	0.241843	0.383103	0.241843	0.060626	0.00598

(4 marks)

d) Compare the Gaussian blurring kernel functions that you wrote in 1(a) and 1(c) by profiling the kernel execution times. Also, compare the kernel execution times based on the different window sizes, and whether it was run on a multicore CPU or GPU. Plot a graph showing the kernel execution times versus window sizes for the naïve approach and the 2-pass approach (this can be done on the same graph) using a multicore CPU, and another graph for execution on a GPU.

To get more reliable results, run the kernel at least 1000 times for each case and calculate the average time. Submit your graphs as pdf files.

(2 marks)

Part 2: Reduction Parallel Pattern

This part has to do with parallel reduction.

a) Write a normal program to calculate the average luminance value of a colour image (i.e. the average of all intensity values in an image).

To do this, first convert the RGB values to luminance values (this approach is used to convert a colour image into a greyscale image). For each RGB pixel, calculate:

Luminance = 0.299*R + 0.587*G + 0.114*B

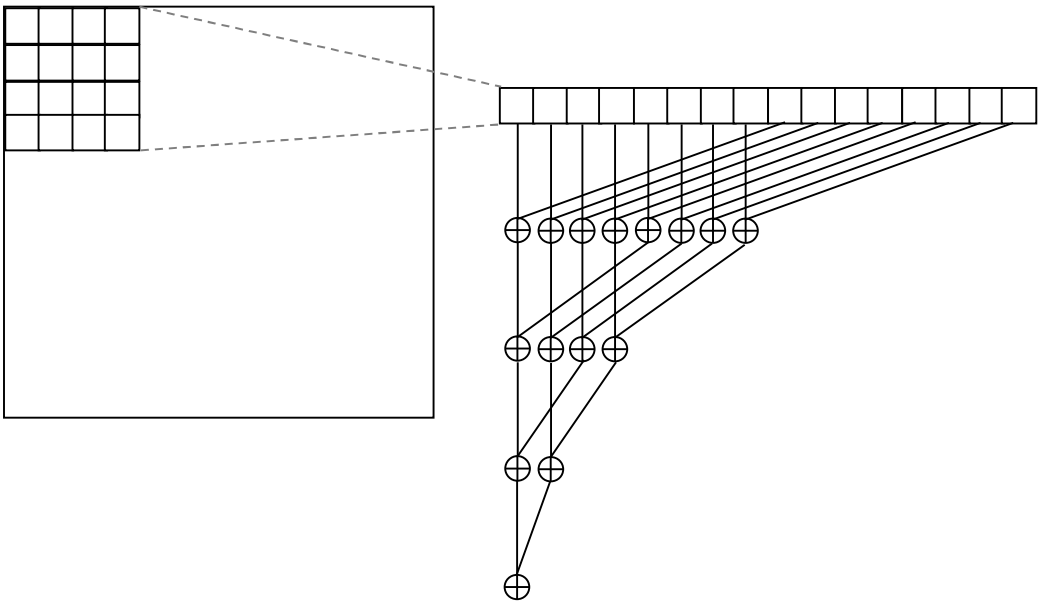
Then, calculate the average luminance of the image by averaging the luminance values of all pixels in the image.

(Note that if you use the example code from the tutorial on image processing, the R, G, and B values range from 0 to 255 on the host (unsigned char), and 0.0 to 1.0 (float) on the device.)

(2 marks)

b) Write an OpenCL program that performs parallel reduction to find the average luminance value of a colour image. Check this result using the code that you wrote in 2(a) by displaying the results for 2(a) and 2(b) on screen. Note that you may get slightly different results due to rounding errors, but the difference should not be overly significant. The parallel reduction pattern is illustrated for a portion of an image below. Note that it should be computed in a work-group using local memory.

(You do not have to use an image memory object when passing image data to the kernel, a buffer memory object is okay)



(3 marks)

Part 3: Bloom Effect

Bloom effects are commonly used in graphics, movies, video games, etc. This part combines the tasks from Part 1 and Part 2. The basic steps to create an image with a bloom effect are illustrated as follows:

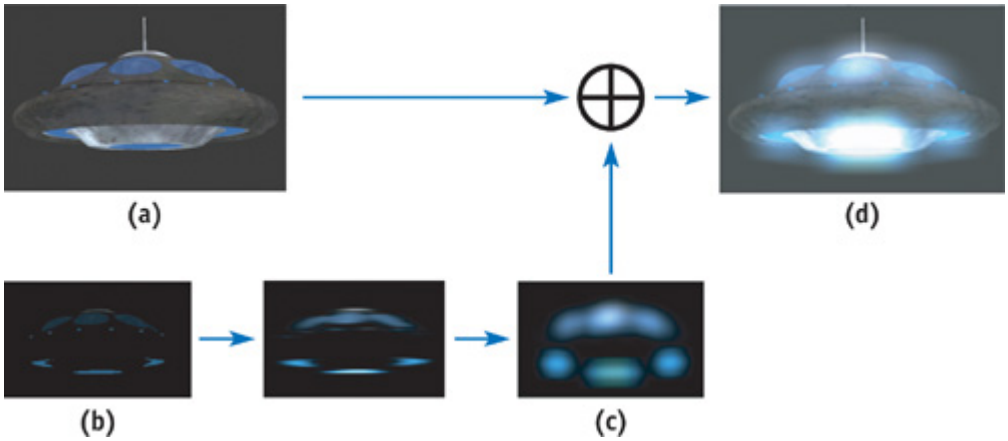


Figure 1: Bloom effect steps.

1. The image in Fig. 1(a) shows the original image
2. The image in Fig. 1(b) shows an image where the glowing pixels are kept, while the rest are set to black. For this assignment, use the average luminance value that you obtained from your program in Part 2 as the default threshold value, i.e. pixels above the average luminance value are kept, while pixels below the average luminance value are set to black
3. The image in Fig. 1(b) undergoes a horizontal blur pass, then a vertical blur pass to obtain the image depicted in Fig. 1(c)
4. Finally, the pixel values in the images shown in Fig. 1(a) and Fig. 1(c) are added together to form the final image shown in Fig. 1(d)

Write a parallel program in OpenCL based on your code in Part 1 and Part 2. Your program is to accept an input image and output an image with the bloom effect. Allow the user to select the size of the Gaussian blur filter from Part 1. For the threshold value (in step 2), use the average luminance value computation from your code from Part 2 as the default threshold value, but also allow the user to enter a different threshold value.

Note that Figure 1(b) shows a down-sampled image (i.e. the image has been shrunk by interpolating values from the original image). It is more efficient to process a down-sampled image during the blurring step because there are fewer pixels to process. This also works well for blurring since referencing the pixels from the smaller image in the final step will also cause blurring (blurring caused by effectively up-sampling back to the original image size. In fact, some approaches simply use down-sampling for blurring.). To make things easier, for this assignment you do not have to perform down-sampling/up-sampling (even though you would have somewhat done this in 2(b)).

(5 marks)

Instructions and Assessment

Organise your solutions into 3 folders (named part1, part2 and part3) and put your **source files** (.c or .cpp, .cl and .h) into the respective folders.

Also submit:

- 6 output images (i.e. 3x3, 5x5 and 7x7 windows using the bunnycity1.bmp and bunnycity2.bmp images) for part 1 in the part1 folder.
- the pdf graphs for part 2 in the part2 folder.
- 6 output images (i.e. 3x3, 5x5 and 7x7 windows with the default threshold using the bunnycity1.bmp and bunnycity2.bmp images) for part 3 in the part3 folder.

Zip the 3 folders into a single file and submit this via Moodle by the due date and time (do **NOT** zip entire visual studio project files as this can be very large). Assignments that are not submitted on Moodle will not be marked.

You will have to demonstrate your working programs during the lab in Week 12. You must be ready for this lab task to be assessed at the start of this lab. Do not try to fix your code during this lab, otherwise late penalties may apply. Your programs will be marked in the lab so it must work on the computers in the lab or you must demonstrate them in the lab on your own laptop.

The assignment must be your own work. If asked, you must be able to explain what you did and how you did it. Marks will be deducted if you cannot correctly explain your code.

NOTE: The marking allocations shown above are merely a guide. Marks will be awarded based on the overall quality of your work. Marks may be deducted for other reasons, e.g. if your code is too messy or inefficient, if you cannot correctly explain your code, etc.

For code that does not compile, does not work or for programs that crash, the most you can get is half the marks.

References

The images were sourced from

- http://http.developer.nvidia.com/GPUGems/gpugems_ch21.html
- http://raytracey.blogspot.com.au/2012/03/real-time-path-traced-stanford-bunny-in.html

