UOW AUSTRALIA

SCIT

School of Computing & Information Technology

CSCI376 – Multicore and GPU Programming
Spring 2020

Assignment 3

Due on Friday, 13th November 2020 at 17:00

Parallel Image Processing

For this assignment, a test image "peppers.bmp" has been provided.

You will find that the examples in Lab 9 on parallel image processing will be very useful for this assignment. You are allowed to use code from the labs in your assignment.

Note that if you use the code from the lab on image processing, the red, green, and blue values range from 0 to 255 (unsigned char) on the host and 0.0 to 1.0 (float) on the device.

Task 1 (Basic image manipulation)

Write a parallel OpenCL program using image objects to flip an image. Your kernel should accept an input image object and three output image objects. For the respective output images, flip the input image as follows:

a) Flip the image in the horizontal dimension (i.e. left become right, and vice versa)

b) Flip the image in the vertical dimension (i.e. top becomes bottom, and vice versa)

c) Flip the image in both horizontal and vertical dimensions

Output the resulting images into three output image files, named: "Task1a.bmp", "Task1b.bmp" and "Task1c.bmp".

(3 marks)

Task 2 (Luminance image)

Write a parallel OpenCL program to convert the RGB values (i.e. red, green and blue colour channels) of an image to luminance values (this approach is used to convert a colour image into a greyscale image).

For each pixel, calculate:

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

Save the luminance image into a 24-bit BMP file. To do this, set all RGB values of each pixel to the luminance value (i.e. red = luminance, green = luminance, blue = luminance).

Name the output image file "Task2.bmp".

(2 marks)

Task 3 (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 weights for a 7x7 windows:

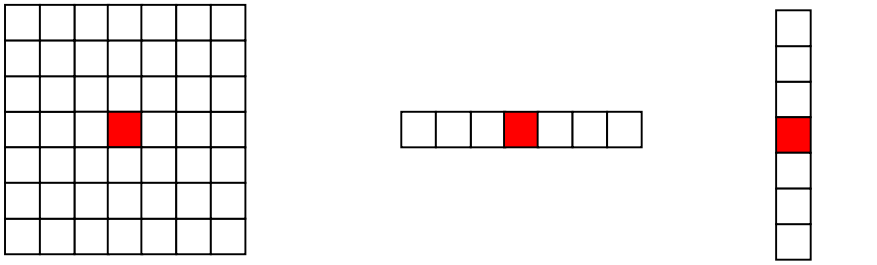
0.000036	0.000363	0.001446	0.002291	0.001446	0.000363	0.000036
0.000363	0.003676	0.014662	0.023226	0.014662	0.003676	0.000363
0.001446	0.014662	0.058488	0.092651	0.058488	0.014662	0.001446
0.002291	0.023226	0.092651	0.146768	0.092651	0.023226	0.002291
0.001446	0.014662	0.058488	0.092651	0.058488	0.014662	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 a filtered image using Gaussian blurring based on the 7x7 window weights provided above.

(3 marks)

b) Instead of using the 7x7 window (the naïve approach), an alternate approach is to run the filter in 2 passes. The first pass will perform blurring in the horizontal direction; the result will then undergo a second pass to blur it in the vertical direction (enqueue the kernel twice to perform blurring in each direction). The result will be similar to the single pass approach (i.e. in Task3a), but the amount of computation will be different.

For example, using a 7x7 window approach, each pixel will have to perform a weighted sum on 49 pixels. In the 2-pass approach, each pixel will have to perform a weighted sum on 7 pixels in each pass, processing a total of 14 pixels. This is illustrated below:



7x7 window horizontal pass vertical pass

Your task is to implement the parallel 2-pass approach. For this, use the following weights for the horizontal pass as well as the vertical pass:

0.00598	0.060626	0.241843	0.383103	0.241843	0.060626	0.00598

(3 marks)

c) The parallel image processing examples provided in Lab 9 are 2-dimensional solutions (i.e. the kernel is enqueued in 2-dimensions). Write an OpenCL program to perform Gaussian

blurring using the 7x7 window previously provided in Task 3a, as a 1-dimensional solution (i.e. enqueue the kernel in 1-dimension) where every pixel is processed by 1 work-item.

(2 marks)

d) Compare the performance of the Gaussian blurring kernels that you wrote in Task 3a, Task 3b and Task 3c by profiling the kernel execution times. You will not need an addition program for this part; simply add code for profiling kernel execution time into your previous programs.

Run the programs on the same device and plot a graph/bar chart to compare the kernel execution times of Task 3a, Task 3b and Task 3c. Clearly label your graph/bar chart and provide information about the device type (CPU or GPU) and the device name that you ran the programs on.

To get more reliable results, run the kernel at least 1000 times for each case and calculate the average time. Note that for Task 3b, the total kernel execution time is the time taken for both passes.

Submit your graph/bar chart as a pdf file named "Task3d.pdf".

(2 marks)

Task 4 (Bloom effect)

Bloom effects are commonly used in graphics, movies, video games, etc. This part combines the work from Task 2 and Task 3b. The basic steps to create an image with a bloom effect are illustrated as follows:

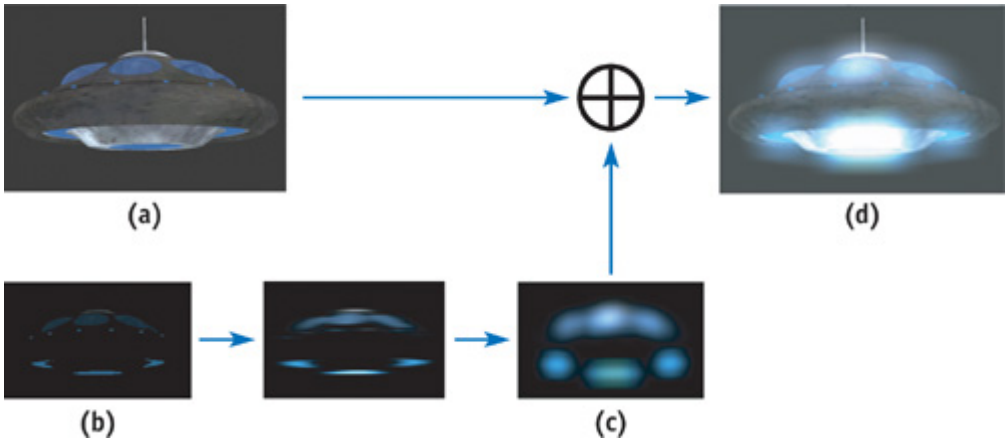


Figure 1: Bloom effect steps.

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[1]. For this assignment, allow the user to input a valid threshold luminance value. Pixels above the threshold luminance value are kept, while pixels below this luminance value are set to black. This step is related to Task 2.

[1] Note that Figure 1(b) shows a down-sampled image (i.e. the image has been shrunk). 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.

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). This step is related to Task 3b.
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). Note that values above the maximum value should be clamped to the maximum value.

Write a parallel program using OpenCL to perform the bloom effect on an input image. For the threshold value (in step 2), allow the user to enter a valid threshold value.

Your program should output the following images:

- "Task4a.bmp" – an image after step 2 (i.e. image showing the glowing pixels)
- "Task4b.bmp" – an image after the horizontal blur pass
- "Task4c.bmp" – an image after the vertical blur pass
- "Task4d.bmp" – the final image with the bloom effect

(5 marks)

Instructions and Assessment

For Task 1, Task 2 and Task 4, include the output image files produced by your programs in your submission.

For Task 3, submit your graph/bar chart as a pdf file.

Organise your solutions into folders, for each task (you may include subfolders as necessary), and put all your **source files** (i.e. files required to compile and run your program, e.g., **.cpp**, **.h** and **.cl** files) **and output files** into the respective folders.

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

You may be asked to demonstrate your program on your computer. 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. Hence, it is better to comment out sections of code that do not work and provide a note for the marker in a readme.txt file.

References

The images were sourced from

- http://http.developer.nvidia.com/GPUGems/gpugems_ch21.html

