CS4345 General Purpose Computation on GPU (2016/2017 Semester 1)

Programming Assignment 2

Release Date: 15 September 2016, Thursday
Submission Deadline: 9 October 2016, Sunday, 11:59 PM

LEARNING OBJECTIVES

Writing GLSL shaders and the supporting OpenGL application programs to perform some common general-purpose computation on the GPU. After completing the programming assignment, you should have

- learned how to use 32-bit floating-point textures to store the inputs and outputs,
- learned how to use the framebuffer object extension to perform render-to-texture,
- learned how to use the texture rectangle extension to ease texture access,
- learned how to set up the OpenGL projection and draw an appropriately-sized quadrilateral to start the computations on the GPU,
- learned how to implement the odd-even transition sort algorithm on the GPU,
- learned how to use the ping-pong technique,
- understood how to refer to a fragment or texel by their precise positions.

TASK

In this assignment, you are required to complete a given OpenGL **application program** and a GLSL **fragment shader** to sort an input array in non-decreasing order using the **odd-even transition sort** on the GPU.

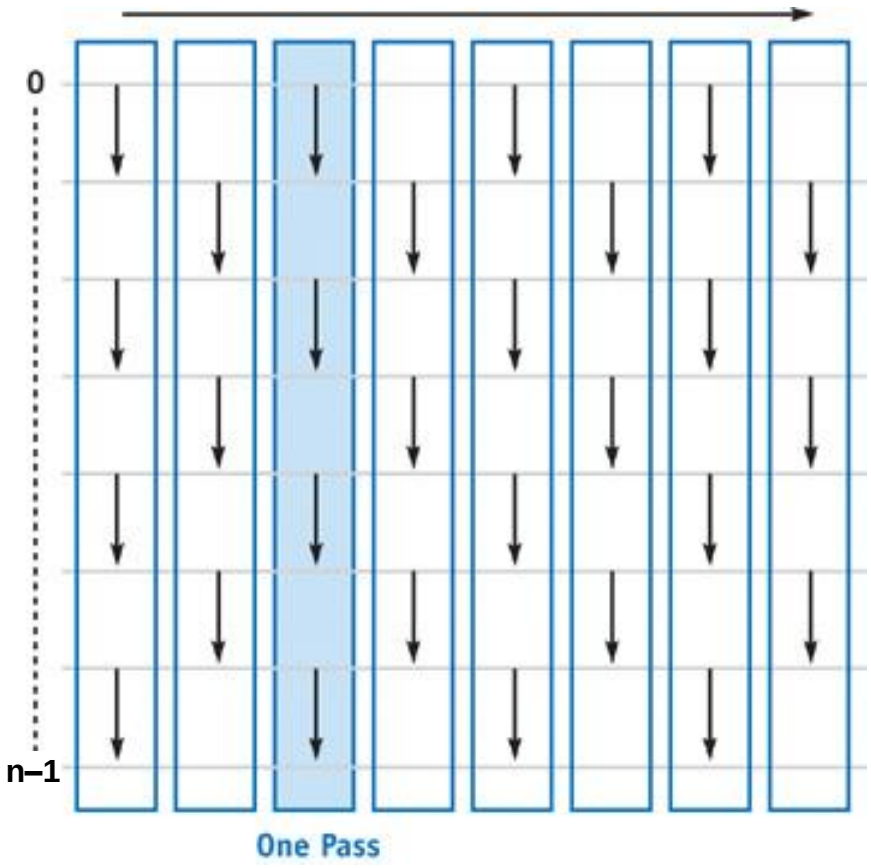
Please download the ZIP file **cs4345_assign2_2016_todo.zip** from the **Assignment** folder in the IVLE Workbin. Your task is to complete the OpenGL application program file **assign2.cpp** and the GLSL fragment shader file **assign2.frag** according to the following requirements.

You should study the given code carefully. A Visual Studio 2008 Solution file is provided for you to build your completed program. You have to complete the program by filling in your code at places marked with the comments "**`WRITE YOUR CODE HERE`**" in both **assign2.cpp** and **assign2.frag**. You are not allowed to modify any other code in assign2.cpp, but you can add new functions in **assign2.frag**.

I have provided a Win32 executable **assign2.exe** of my completed **assign2.cpp**. Therefore, you can use that to test your fragment shader first, and then move on to complete your own **assign2.cpp**.

You are to implement the function **`GPU_OddEvenSort()`** that performs the **odd-even transition sort** to sort the input array in non-decreasing order on the GPU.

The following diagram illustrates how the odd-even transition sort operates on an array of n elements. For any $n \geq 3$, the maximum number of passes to completely sort the input array is n.



The following table shows an example to sort an array of 9 elements in non-decreasing order using odd-even transition sort. The "–" between a pair of numbers indicates that the numbers are to be compared and swapped (if necessary) to produce result for the next pass.

	Array Elements
Input Array	8 – 7 6 – 5 4 – 3 2 – 1 0
After Pass 1	7 8 – 5 6 – 3 4 – 1 2 – 0
After Pass 2	7 – 5 8 – 3 6 – 1 4 – 0 2
After Pass 3	5 7 – 3 8 – 1 6 – 0 4 – 2
After Pass 4	5 – 3 7 – 1 8 – 0 6 – 2 4
After Pass 5	3 5 – 1 7 – 0 8 – 2 6 – 4
After Pass 6	3 – 1 5 – 0 7 – 2 8 – 4 6
After Pass 7	1 3 – 0 5 – 2 7 – 4 8 – 6
After Pass 8	1 – 0 3 – 2 5 – 4 7 – 6 8
After Pass 9	0 1 2 3 4 5 6 7 8

For computation using GLSL, the input **1D array is placed into a 2D texture map** of some given width and height, but it always has exactly the same number of elements as the input array. Since the algorithm requires multiple rendering passes, we can allocate two textures (one as input to your fragment shader and the other as the output buffer), and use the **ping-pong technique** to swap their input/output roles after every pass.

The **input and output textures** are of the **`GL_ALPHA32F_ARB`** internal format. That means that each element in the input array is stored in the alpha channel of a texel in the input texture. In fact, each texel has only the alpha channel. The same arrangement applies to the output texture too.

In your program, you are going to work with 2D textures instead of 1D array. You have to use **texture rectangles** for the textures. A **framebuffer object** is also needed for you to render directly to the output texture. You can assume that the input array can always be contained in a texture. More instructions are provided in the program source file.

In your fragment shader, you must make sure **not to use "dangerous" texture coordinates** to access the texture map.

You should keep your fragment shader as simple and as efficient as possible. **For your convenience, you can assume that the texture width is always an even number** (therefore the number of input elements is even too).

You should try different valid texture sizes to test your program. In most cases, you will notice that the running time of the CPU is much better than that of the GPU. There are a few causes for this, and they are interesting to know.

You should submit only your completed files **assign2.cpp** and **assign2.frag**.

GRADING

The maximum marks for this programming assignment is **100**, and it constitutes **8%** of your total marks for CS4345.

Program **correctness** constitutes **90 marks** while **design/style** constitutes **10 marks**. Note that if your application program or shader cannot be compiled and linked, you get 0 (zero) mark for program correctness.

Good coding style. Comment your code adequately, use meaningful names for functions and variables, and indent your code properly. You must fill in your **name**, **matriculation number**, and **NUS email address** in the **header comment**.

SUBMISSION

For this assignment, you need to **submit only the two completed files**:

- **assign2.cpp**
- **assign2.frag**

You must put them in a ZIP file and name your ZIP file ***<matric_no.>*.zip**. For example, **A0123456X.zip**. All letters in your matric. number must be capitalized.

Submit your ZIP file to the **Assignment 2 Submission** folder in the IVLE Workbin. Before the submission deadline, you may upload your ZIP file as many times as you want to the correct folder. **We will take only your latest submission.** Once you have uploaded a new version to the folder, you **must delete the old versions**. Note that when your file is uploaded to the Workbin folder, the filename may be automatically appended with a number. This is fine, and there is no need to worry about it.

DEADLINE

Late submissions will NOT be accepted. The submission folder in the IVLE Workbin will automatically close at the deadline.

——— End of Document ———