

CAP 5705 – Assignment 2

Due October 8, 2021

OpenGL Viewer

In this assignment you will write a simple OpenGL application, in the modern pipeline, to read and display a triangular mesh. You will learn how to prepare and send information to the GPU and write simple code to program the GPU. To give you a starting point, a C++ source file is available at <https://www.cise.ufl.edu/class/cap5705fa21/files/helloTriangle.cpp> that displays 2 triangles whose vertices are provided in a vertex array (i.e., “separate triangles” data structure):

```
// set up vertex data (and buffer(s)) and configure vertex attributes
//
float vertices[] = {
    -0.5f, -0.25f, 0.0f, // left
     0.5f, -0.75f, 0.0f, // right
     0.0f,  0.5f, 0.0f, // top

    -0.5f, -0.5f, 0.0f, // left
     0.5f, -0.5f, 0.0f, // right
     0.0f, -1.0f, 0.0f, // bottom
};
unsigned int numVertices = sizeof(vertices)/3;
```

We communicate the vertex information (presently just geometry in above) to the GPU that executes a little program called **vertex shader** on each vertex. The GPU executes this program in a massively parallel fashion on all vertices (as concurrently as possible). We can change this little program to accomplish our objectives while benefitting from the massive parallelism without ANY parallel programming! Furthermore, once the GPU rasterizes (i.e., turns our triangles to fragments), we can also program a little code that gets executed for each fragment, aka **fragment shader**. To program these shaders, we need to pick up a small language called GLSL which is very similar to C in syntax. While GLSL is simple and easy to understand, it gets executed on the GPU, so you don’t have direct access to the variables in your program. Therefore, we have to send the data (e.g., attributes for each vertex) from our C/C++ program to the GPU so that we can access it from within GLSL:

- On the CPU side, inside C/C++, we have to prepare the data using Vertex Buffer Objects (VBOs) that contains the actual data (e.g., vertex geometry or other attributes) and Vertex Array Objects (VAOs) to be sent to GPU. VAO provides meta-information describing the layout of the data (i.e., how to interpret the stream of bytes in VBO). For example VAO informs the GPU to take 3 chunks from the stream, with each chunk being 4 bytes, and interpret it as the 3 floating point variables describing the **x**, **y**, **z** of a vertex.
- On the GPU side, inside GLSL, a shader program has **in** variables declared to facilitate receiving the data and **out** variables to send the output for the rest of the pipeline. This process can be tricky and hard to debug! As long as we start with simple data (e.g., just geometry) and then slowly add more attributes and test/debug things *at each step* along the way, we should be able to tackle this challenge painlessly (ok, not as much pain ;-).

Chapter 17 of our textbook (Marschner & Shirley) gives an introduction to graphics hardware. For a quick introduction to GLSL for this assignment, see <https://learnopengl.com/Getting-started/Shader>. Main debugging tool you have is `glGetError()` to retrieve error codes. Use it extensively throughout your program! If OpenGL 4.3 or higher is supported on your system, you can look into the debug output extension within `GLFW`. Of course you may also find debugger extensions for your IDE helpful (e.g., Nsight, CodeXL, RenderDoc).

- Change the Vertex Buffer Objects (VBOs) and Vertex Array Objects (VAOs) to associate a color for each vertex. Then change the vertex shader and fragment shader codes to determine the color of each fragment based on the colors associated to each vertex. **20pts**
- Add the functionality to read vertex geometry and connectivity from an `.obj` file and generate a “separate triangles” data structure in the `vertices` array. For this part you may want to only build `x`, `y`, `z` attributes (but welcome to try color if you wish). A number of meshes are available at <https://www.cise.ufl.edu/class/cap5705fa21/files/data>. It is highly recommended to start with `cube.obj` to debug your code before trying other meshes. **30pts**
- Currently the vertex and fragment shader codes are provided in a string `const char*`. Add the functionality to read these codes from two separate files (e.g., `source.vs` for vertex shader and `source.fs` for fragment shader). **20pts**
- Make simple changes to vertex shader and fragment shader and observe the effect on the rendered image. The objective of this part is to get a feel for accessing/changing variables in GLSL, so as long as the change you make has a visible effect on the output, it will do. Simple examples: flip coordinates, some transformation, change colors, ... **10pts**
- Bonus: Read up about Element Buffer Objects (EBOs) to implement the “indexed triangle” data structure we learned in class. **+20pts**
- Along with the source code and makefile (or a README for compiling instructions), submit a report (a PDF file) that describes and documents *each functionality* you implement. **20pts**
- This is an individual effort assignment. You are welcome to discuss ideas/problems with classmates but the source code and report you submit **MUST** be your own work. You need to clearly acknowledge sources (ideas, solutions, websites) that you use.

Submission Guidelines
Submit to E-learning site a single file as a <code>.zip</code> or a <code>.tar.gz</code> bundle that contains all the files to be submitted. Include the report PDF and source codes for your programs in the submission bundle. Please include a ‘README’ file that clearly explains how to run and test the program. Also include a ‘Makefile’ that compiles and links the program from the source files.
Late submissions are penalized by 20% of the grade for each day (up to 3) past the due date.