CSCI 444/544 - Lab 1
Friday, January 22, 2021
LAB IS DUE BY **FRIDAY January 29 11:59 PM**!!

This lab will verify your installation is set up correctly and serve as the starting point for A1. The provided code is set to work on the lab machines out of the box. If you are working on a lab machine, continue to Step 0A. If you are working on your personal machine, then continue to Step 0B.

Regardless of which route you go, make sure in CLion that you set the Run > Edit Configurations > Working Directory to be the folder containing the main.cpp file.

Step 0A – Lab Machines

This is the simpler wrote to go since the set up is done for you. Included in this package are two folders: include/ and lib/. You will want to create folders on your Z: drive to copy these contents to. Make two folders:

- Z:/CSCI444/include
- Z:/CSCI444/lib

And copy the include/ folder to Z:/CSCI444/include and the lib/ folder to Z:/CSCI444/lib. This will house the necessary libraries in one place for all your labs and assignments. And the CMakeLists.txt file will reference these locations.

Open CLion, run the project and you should see the program start up! Continue to Step 1.

Step 0B – Personal Machines

This could be the longer process and YMMV. Here are the libraries you'll need to install:

- A compiler that includes OpenGL
  - Windows: MinGW (not Visual Studio) http://www.mingw.org/
  - Mac: XCode
  - Linux: Whatever gives you g++ and OpenGL
- GLFW https://www.glfw.org/
- GLEW http://glew.sourceforge.net/
- glm https://glm.g-truc.net/0.9.9/index.html
- CSCI441 https://github.com/jpaoneMines/csci441
- FreeType https://www.freetype.org/

Then, in the CMakeLists.txt file you'll need to edit three fields:

- Change `includeDirectories()` to the location where all your includes are located (if not in your MinGW include/)
- Change `target_link_directories()` to the location where all your libraries are located (if not in your MinGW lib/)
- Comment/uncomment the Windows/Mac `target_link_libraries()` as appropriate. Linux – you may need to edit these.
- You may need to edit the library to link against for freetype depending on your build. If `freetyped` cannot be found, edit it to be `freetype`.

Run the project and you should see the program start up if everything was installed properly. Additionally, navigate to `cmake-build-debug/src` and you will see a `libstbimage.a` file. Copy this to your MinGW lib/ folder.

Step 1 – Let's Add Some Color

Run the lab. You should see lots of output in your terminal. Information is displayed about the version of OpenGL that's running. It should be 4.1 or higher. You'll then see all the limits for various features – these will make more sense as we move on.

Then is the output of the two shader programs. These should all say success. The first shader, colorPassThrough, you will be editing in this lab. The other, freetypeColoredText, you won't need to touch but is there for reference to display the text in your window.

In the window that pops up, there should be text displaying the FPS and you'll see a yellow square.

Time to investigate the code. Open `main.cpp`. Here is an explanation of the high level functions and then you can deduce what you'll need to change for each of the steps:

- `setupGLFW()` – creates our OpenGL Context and the window that we will be drawing in. Registers all of our callbacks
- `setupOpenGL()` – does OpenGL settings and initialized GLEW. Prints OpenGL context info
- `setupShaders()` – loads and compiles the shader programs along with printing shader info. Gets the uniform and attribute locations for the shader program.
- `setupBuffers()` – creates VAOs/VBOs and transfers data to the GPU
- `setupFonts()` – creates the font glyph library

For this lab, and the majority of assignments, you will only need to ever edit `setupShaders()` and `setupBuffers()`. For future labs/assignments, you will need to edit some of the callback methods, primarily the `key_callback`.

Now to investigate the shaders. We will need to edit `shaders/colorPassThrough.v.glsl` and `shaders/colorPassThrough.f.glsl`. If we look at the fragment shader first, it is hard coding the output for all fragments to be yellow (1,1,0). We want to allow each vertex to have a different color and then set the fragment to be the interpolated color. Let's add this to our shaders. The steps to perform are:

1. Add the color information as an attribute to the vertex shader
2. Pass this value through to a varying out of the vertex shader
3. Read the varying as input to the fragment shader
4. Set the fragment shader output to the value of the varying

Each step corresponds to one line of code, so those four lines are all that's needed in the shaders. Back to the OpenGL side and `main.cpp`. For these steps working with the attribute, check out how each step is done for the position and mimic that structure.

1. If we look at the structs at the top of the file, the Vertex already has color information associated with it. Keep going down the file to the structs storing our shader locations. We need to add an attribute value for color, in addition to position.
2. Next we'll move to `setupShaders()`, and get the attribute location for the color attribute.
3. Now in `setupBuffers()`, we will enable and point to the color attribute data. Recall the arguments to `glVertexAttribPointer()`:
    a. The location this data is sent to (the color location)
    b. The number of elements in this attribute (RGB → 3)
    c. The data type of each element (floats)
    d. If the value should be normalized (no)
    e. The stride between values (there is a full Vertex in between each starting point)
    f. The offset of the first value (the first color is after the first three floats corresponding to the position)
4. Do Step 3 for both the Cube VBO and the Ground VBO.

At this point, you can build and run. You'll see a nicely colored ground plane and a triangle floating through the air. Let's make that triangle in to a cube.

Step 2 – Cube It

The only change we'll be making is to our cube data at the top of our file. Currently, there are only 3 vertices. A cube is made up of 8 vertices. Add data corresponding to the following 8 points:

Vertex Name	Position	Color
`Bottom, Left, Near`	`(-0.5, -0.5, -0.5)`	`(0, 0, 0)`
`Bottom, Right, Near`	`(0.5, -0.5, -0.5)`	`(0, 1, 0)`
`Top, Right, Near`	`(0.5, 0.5, -0.5)`	`(1, 1, 0)`
`Top, Left, Near`	`(-0.5, 0.5, -0.5)`	`(1, 0, 0)`
`Bottom, Left, Far`	`(-0.5, -0.5, 0.5)`	`(0, 0, 1)`
`Bottom, Right, Far`	`(0.5, -0.5, 0.5)`	`(0, 1, 1)`
`Top, Right, Far`	`(0.5, 0.5, 0.5)`	`(1, 1, 1)`
`Top, Left, Far`	`(-0.5, 0.5, 0.5)`	`(1, 0, 1)`

In the array, these are indexed 0-7 in the order listed above. We now need to state what order to make the triangles in. Add the following indices to correspond to each of the six faces:

Face	Triangle 1	Triangle 2
`Near`	`0, 2, 1`	`0, 3, 2`
`Right`	`1, 2, 5`	`5, 2, 6`
`Top`	`2, 7, 6`	`3, 7, 2`
`Bottom`	`0, 1, 4`	`1, 5, 4`
`Back`	`4, 5, 6`	`4, 6, 7`
`Left`	`0, 4, 3`	`4, 7, 3`

Compile and run. If all the data was entered properly, you should now see a colored cube rotating above the ground.

If the triangles don't form a cube, then the ordering of the indices is not correct and refers to the wrong vertex.

If the colors don't blend smoothly from black to white, then the winding order of the triangle is backwards.

Step 3 – Moving Forward

At this point, your task for Lab1 is done. Your environment is set up correctly and you edited a shader and VAO. Zip up your Lab1 folder and submit it to Canvas.

A1 will continue from this exact point. The full write up is online but the tasks that will need to occur:

- Add normals to the vertex data instead of colors
- Place a light in the scene and set a material
- Apply the Phong Illumination model to the objects in the scene (will need to edit shaders and VBOs)
- Using a shader subroutine (1/27 topic), switch between Phong and Blinn-Phong specular highlights
- Have two shader programs: one that applies Gourad Shading and one that applies Phong Shading
- Allow the user via key presses to switch between Phong/Blinn-Phong specular highlights and Gourad/Phong shading for a total of 4 combinations. Display the current configuration via text.
- Use UBOs (Uniform Buffer Objects – 2/1 topic) to set the uniforms for both your shader programs at once

A1 will be due 2/12.

LAB IS DUE BY **FRIDAY January 29 11:59 PM**!!