CSCI 4110 Laboratory Two

Basic OpenGL

Introduction

This laboratory should largely be review from your previous graphics course. It will make sure that everyone recalls the basics and all the software is installed correctly. In this laboratory you will display a cube from data stored in constant arrays in the program and a sphere that is read in from an obj file. We will use simple shaders that do basic diffuse lighting. We will examine more sophisticated shaders in future laboratories. The code for the program that we will use for this laboratory is available on Blackboard. Take some time to examine this code, it will be used as a skeleton for future laboratories and will be useful for the assignments.

The Cube Display

In the blackboard folder for this laboratory you will find a Files.zip file, this file contains all the files that you will need for this laboratory. Download this file and unzip it. The various files will need to go in different locations, which will be described below.

Now create a new Visual Studio project. I called mine viewer, but you can use any name that you like. In the following you will just need to replace viewer with the name of your project if you used a different name. Assuming that you called your project viewer, you will find another viewer folder within this project (the one that contains the .vcxproj file. This is where you should place all of the program source files (the shader files go in a different folder). Next add these files to your project. This can be done by right clicking either Header Files or Source Files and selecting the Add item.

If you try to compile the program now you will notice errors. You need to provide the include file location for the libraries that we are using, which are GLFW, GLEW and GLM. While you are doing that you can also provide the library locations for GLFW and GLEW. For Additional Dependencies under Linker you need to add glew.lib and glew32.lib (you may also need to add opengl32.lib and glu32.lib). The mechanics for doing this is described in the first laboratory. Once you have done this the program should compile and run. Remember to copy the glfw3.dll and glew32.dll files to the folder with the executable for your program. The output that you should get is shown in figure 1.

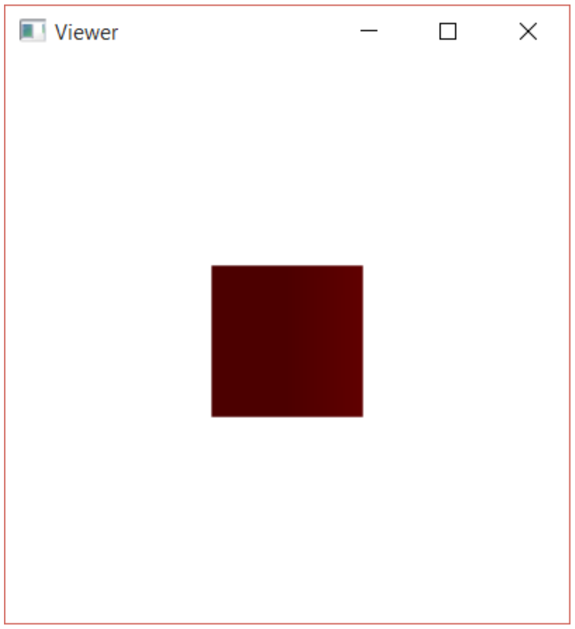

Figure 1 Cube display

Now let's examine the program code. We will use the same program code for both the cube and sphere display, thus we have two init procedures that create the data for the draw commands. We will use #ifdef statements to select the version of init that we will use. Near the top of the program you will find the following two statements:

```
#define CUBE
//#define SPHERE
```

Only one of these statements can be active at a time, initially it is the one for CUBE. When we want to draw the sphere we will reverse the comments.

The cube's init procedure starts with three constant arrays that contain the vertex, normal and index information for the cube. This information is then copied into the vertex array object and the element array buffer. Finally the data in the vertex array object is mapped to the attributes for the vertex shader.

The next procedure, framebufferSizeCallback is called whenever the size of the window changes. It sets the viewport and computes the projection matrix.

The display procedure is called each time the display needs to be updated and draws the data stored in the vertex array object. It starts by computing the view transformation matrix. Next the view transformation matrix and the projection matrix are sent to the vertex shader. This is followed by binding the vertex array object and then issuing the draw command.

The key_callback procedure handles interaction with the user through the keyboard. This function is called each time a key on the keyboard is pressed. The first parameter is the value of the key that was pressed. The user can use the WASD keys to move around the object is a spherical motion. This procedure uses a switch statement to determine if one of the WASD keys was pressed and if it has modifies the corresponding angle. The two angles are used to compute the new eye position and the display is updated.

The final procedure is the main procedure. This procedure starts by initializing GLFW and then initializing GLEW. Once this has been done the change and keyboard callbacks are set. Next we use our shader library to compile the shaders and link the program. Once this has been done the dumpShader procedure is called to print the uniforms and attributes for the shaders. This is useful for debugging our programs. Once the init procedure is called and the initial eye position is established and a loop similar to the first laboratory is used to display the object and process user input.

Sphere Display

To display the sphere all we need to do is change the init procedure that is used. We can do this by exchanging the comment indicator on the two define statements so SPHERE is now defined. After doing this recompile the program and run it. Make sure that the sphere.obj file is in the same folder as the executable for the program. The result should be similar to that shown in figure 2.

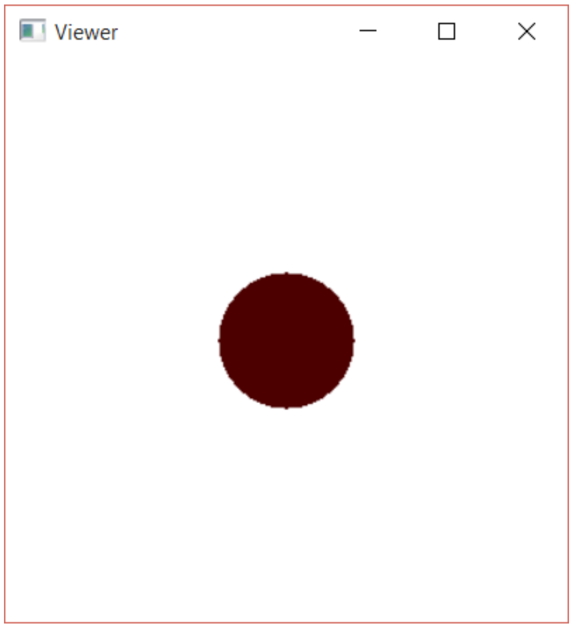

Figure 2 Sphere display

The only difference from the first program is the init procedure, so we only need to examine it. This procedure is more complicated since we are reading the geometry from a file. We have a sphere.obj file that was created using Blender. When we read a file like this we don't know the number of vertices or triangles, so we can't use statically declared arrays as we did in the previous init procedure. Instead we need to use pointers and allocate the storage after the model has been read. We use the tiny obj package to read obj files, the source code is included in the zip file and should already be compiled with your program.

The init procedure starts by calling the LoadObject procedure to read the obj file. This is a relatively simple obj file that consists of a single shape, so processing it is fairly easy. We start by retrieving the number of vertex coordinates and storing it in the variable nv. We then allocate enough storage to hold the vertices using the following statement:

```
vertices = new GLfloat[nv];
```

After we have done this we can copy the vertex coordinates into the vertices array. We do the same thing with the normal vectors and the triangle indices. The rest of the code is basically the same as what we had before, with one exception. With the cube sizeof(vertices) and sizeof(normals) gave us the number of bytes in the array. If we use these now we will only get the number of bytes in a pointer, not what we want. Instead we multiply the size of a float by the number of vertex and normal vector coordinates.

Laboratory Exercise

As you can see from the two figures the fragment program doesn't do a very good job of lighting either the cube or the sphere. Make some simple changes to the fragment program that improves the lighting. Show your results to the TA before the end of the laboratory session.