Mark Meyer

Brian Holbert, Ph.D

CS-330-X6374

August 15, 2021

**FINAL PROJECT: DESIGN DECISIONS**

As I look back on this 3D project to when I selected my objects, I am happy that I went with setting up my own scene because I was able to gauge my workload by picking objects that aligned with my OpenGL skill level--beginner. I used children’s toys for my primitive objects, and my complex object was made of other primitive objects such as cube, a rectangular cube, and a pyramid. In setting up OpenGL, I met several challenges such as libraries clashing, trying to learn concepts by cross-referencing tutorials and documentation but not being able to actually use the libraries the tutorials were using, or learning how to use a camera or shader header from a tutorial but later concepts taking a different approach and adapting my code to make things work just to learn a more complex concept.

I found the most helpful thing you can possible do for yourself in situations like this to modularize the code—I cannot stress this enough. OpenGL requires a great deal of code that will continuously repeat itself the more you add to your scene: objects, textures, coordinates, buffers, arrays, elements, lights, positions, colors, and so on. If there is one thing I learned that after making my first (complex) object, I modularized all my code before going any further, and as a result, once I was able to figure how to calculate coordinates, I added my sphereto the scene in a matter of minutes. There were a few keys factors that really pulled the project together in making things easier to add and texture an object: a mesh struct which defined the vertex array object, vertex buffer object, element buffer object, and the indices for each object; GLSL shader programs which setup ambient, diffuse, and specular lighting which accepts a single texture unit upon each draw call allowing the same shader to texture each object; a function called “UDrawObjectWithTexture” which acquires the uniform location of the specified texture ID, and then draws the indices from the element buffer object using the indices specified by the mesh; a function called “UPassTransformMatrices” which allows you to specify the model position, view, projection, camera position, shader program, light position, and light color for each object before you draw the object; a function called “UPassUniformLampMatrices” which is a much simpler version of the previous function that allows you to specify model position, view, projection, and lamp shader program—ideal for drawing light source objects; destruction functions to destroy shader programs, meshes, textures as well as unbind textures; likewise, I also have a function that binds textures and a function that sets the uniform location of a texture unit; naturally, there is a list of creation functions for creating a mesh, shader programs, textures, and calculating the vertices and indices of a sphere.

The important part here is that creating a scene in OpenGL presents a certain complexity where clean, communicative, and concise code is your best friend and messy, uncommented, un-modularized code is your worst enemy. There are far too many components working together to not be organizing the code into conceptual functions. To give a quick break down of how this program works, here are the operations: OpenGL initializes the GLFW and GLAD context, creating the window and setting the callback function in order to display to and process input from the user—in this case, the keyboard and mouse; then the mesh is created using a vertex array object for the vertex buffer objects, the vertex buffer objects define the vertex, normal, and texture coordinates for the shader program as well as the indices, the element buffer objects define each objects indices, and the vertex attribute pointers send information to the binding points in the shader program which allow the shader to know how to go about lighting and texturing the coordinates being sent; the shader programs are created and the textures are loaded into a variable; the textures are then created by being activated, binded, defined by parameters, RGB channels, and size, and given a texture ID which corresponds with the uniform sampler in the shader program—then finally, the mipmaps are generated for the texture; once all that is complete, the color is cleared from the window and the loop is started which sets the timing, processes user input from the window, renders the objects (where all objects, lighting, positioning, and textures are all defined and drawn), then GLFW polls all events to provide input for the application as well as letting the window system know the program has not locked up; finally, once that is all complete, the program runs the destruction functions to release and de-allocate data and resources, and then the application terminates.

This program is set up to navigate using the “WASD” keys on a keyboard to move forward, reverse, strafe left, and strafe right. The “Q” key is used to move up on the Y axis, and the “E” key is used to move down the Y axis. The mouse wheel can be scrolled to adjust the speed of the camera at small increments, and the movement of the mouse itself may be used to look around in any direction which really gives a freedom to move wherever you want in the scene. If I were to go back to this project and change anything, I would modularize it even more, perhaps moving some of the larger functions and shader programs to their own .h, .txt, or .cpp files. This could really make the project a powerful contender for future OpenGL projects.