CS 300: Final Project

Reflections

Mary Eakins

In making choices about objects to include in my scene, I wanted all of them to be cohesive and make sense with each other. While I could find several random objects that contained one or more of the primitive shapes, putting them together did not make much sense. Instead I wanted the objects to represent related items, that would reasonably appear together in the real world. I began with simpler shapes like cubes and planes, and then proceeded on to objects that required more complicated mathematical operations. I tried to use several different shapes and made sure that at least one of my objects incorporated two or more shapes. I fulfilled this with the truck and the cone. I knew the truck would be a challenge but breaking it down to ‘simple’ shapes (like cubes and tori) would achieve the effect in an attainable manner. Of course a real truck has a lot more nuanced shapes, but with the tools I had I created something that I think most people would recognize as a truck.

The user may navigate my scene using a combination of keyboard keys and mouse controls. The ‘A’ key moves the camera (viewer) to the left, ‘D’ moves right, ‘W’ is forward, ‘S’ is backward, ‘Q’ is up and ‘E’ for down. Users may push more than 1 key at a time to achieve more nuanced movements. Additionally, the user may switch between orthographic and perspective views by using the ‘P’ key. The mouse will change the camera direction so that the user may swivel in any desired direction when navigating the scene. The mouse cursor controls the orientation of the camera, moving it up, down, right or left. The mouse scroll can change the speed of the camera movement.

I created custom functions to try to simplify my Source.cpp file. I have a generic Mesh struct object that contains all of the variable data necessary to render one object. This includes vertices, texture coordinates, texture file names, and variables to hold Vertex Array Objects, Vertex Buffer Objects, etc. This also includes glm matrices for scale, rotation, and translation. Using this Mesh object allowed me to define all of these variables in one place, instead of having to create 7 or 8 different versions of each. Ultimately this saved me from writing a lot of unnecessary code. My ‘ShapeMaker’ class did the work of creating the vertices. Each method created vertices for a different shape (Cone, Plane, Cube, etc.). These vertices included position vertices, texture vertices, as well as normals. I also had a method to create a VAO and VBO for each object upon its creation, as well as assigning Vertex Attribute Pointers. This is also where I did all of my model transformations, such as rotate, scale and translate. Then in my SceneMaker class, I called off to the various methods in ShapeMaker to build shapes as needed. I could define the transformations here by use of my ‘transformation’ vector (included in my Mesh struct). This was a very helpful feature, because I could tweak the translation and rotation of my various objects all in one place without too much scrolling or clicking back and forth. I also moved my shaders to their own header files to even further declutter my Source.cpp file. I used Booleans to allow for the reuse of certain aspects, such as shaders. For instance, my ‘cubeMap’ Boolean would determine how a texture was made for a certain object, because Cube Maps require a different method of texture creation and application.