**Final Project**

Drew Townsend

Southern New Hampshire University

CS-330: Computer Graphics and Visualization

Professor Rodriguez

August 15, 2021

When finding my 2D scene, I wanted to find objects that required multiple shapes to fit the need of this class. The scene I ended up going with had a marble tabletop, a book, a Rubik’s cube, a candle, and a perfume bottle. Each of these objects provided a unique challenge where I had to provide textures, lighting, and locations for each. I even went so far as to create my own textures in Photoshop to add more personalization to the project.

Development at times was challenging, however utilizing the libraries we were given saved me a lot of time. For example, it was relatively easy to make the app listen to hardware input, like pressing certain keys or moving the mouse around. I was able to setup a scene where the user could actually move around and view from different angles. Adding the textures allowed the user to actually know what they were looking at. Adding the lighting and normals of each object was also a great way to make the scene more realistic. I implemented the Phong lighting model, introduced various lighting techniques like ambiance, diffusion, and specular lighting.

In terms of controls, I implemented what is essentially a listener for keyboard and mouse input. For the keyboard, users can use the WASD keys to move forward, backward, and side-to-side in the scene. Users can also use the E and Q keys to move up and down, respectively, in the scene. And lastly with the keyboard, users can use the P key to change the projection of the scene from perspective projection to orthogonal projection. As for mouse input, the scene mimics the X and Y axis of the mouse’s movement on the scene. For example, if you move the mouse left, the scene shifts to the right as if you’ve turned your head to the left. The X axis affects side-to-side movement, and the Y axis affects up and down movement in the scene. Lastly, users can use the scroll wheel on the mouse to increase and slow down their movement speed. This is not to be confused with mouse sensitivity.

The controls were relatively simple to implement. Within the URender function, we set our camera’s axis’s. Instead of hardcoding floats into this matrix, we implement float variables, where the value of the floats can be increased and decreased. I then increase and decrease these values based on the keyboard keys and the mouse input.

I’ve added multiple functions that allow for reuse of a lot of code, rather than having to copy and paste code, and potentially make mistakes. The UInitialize function simply initializes many of the necessary libraries and creates the window that OpenGL will render items in. The UResizeWindow function resizes the render area based on the new height and width of the screen. The UProcessInput, UMouseScrollCallback, and mouse\_callback functions essentially listen and perform tasks based on the input. The UCreateMesh function is where I generate the locations of the objects, their normals, and texture coordinates. I also initialize and bind the varies VAO’s and VBO’s, which sends data to the GPU. The UCreateShaderProgram initializes and compiles my shaders, which actually renders the data along with colors, textures, etc… UCreateTexture loads the textures required to make the objects look more realistic. flipImageVertically literally flips the Y axis values of images, which is sometimes necessary when using the STB library. I also have a function inside of my fragment shader, which contains all of the logic that generates my point lights, which are set up on both sides of my scene. I can easily set up multiple point lights without having to increase the size of my shader file.