**Final Project:**

**Reflection**

Jake Sheehan

October 24, 2021

Southern New Hampshire University

Computer Graphics and Visualization

CS-330-T1031

Dr. Diesch

A roll of toilet paper next to a cup of coffee

Description automatically generated with low confidence

Figure 1. This is the image I was attempting to render.

The 3D Scene

For this project, I rendered a 3D scene of my choosing. I took a photograph of carefully selected objects that were placed on a dining room table. These objects included a book, a pen, a cup, and headphones. I chose these specific objects because they could be constructed from simple shapes that can be rendered in OpenGL. I wanted to keep the shapes simple so that I could focus on developing the rendering engine and the input controls. The rendering engine uses OpenGL to interact with the graphics card and render graphics primitives to a window on the screen. In this case, I chose to use triangle primitives, the common choice for modern graphics programming.

Complex objects are constructed by combining small triangles, each of which is rendered using three vertices. Each vertex contains data for position coordinates, normal coordinates, and texture coordinates. The position coordinates simply describe the x, y, and z position of the vertex in 3D space. The normal coordinates describe a unit vector that points perpendicular away from the surface. This unit vector is used to calculate the angle between the surface of the triangle and the position of the light source to determine the strength of the light hitting the surface. The texture coordinates are used to sample an image that will be mapped onto the triangle. A list of indices is used to determine which order the lines of the triangle will be drawn by grouping together three vertices at a time. All this data in stored inside a vertex array object (VAO) which can be bound to a buffer and drawn to the screen.

Navigation

Once the 3D scene is rendered, the user can navigate the scene using the mouse and keyboard. The WASD keys are mapped to movements that go forward, back, left, and right. When the user presses the QE keys, the camera will move up and down. The mouse is mapped to controls that turn the camera to look left, right, up, and down. The scroll wheel will speed up or slow down the movements. To do this, I used the GLFW library, which provides built-in callback functions that are triggered when input events are detected. It is important to note that there is no real “camera”. When the WASD keys are used, there is no camera to move. Instead, the world itself moves, giving the viewer the illusion that they are moving. For example, when the viewer presses W to move forward, everything in the scene in translated to move in the positive direction on the z-axis. This makes every object appear to move closer to the screen.

Modular Organization

During the development of this software, I was careful to use a modular design. Since much of the code is used multiple times, this helped me avoid writing the same code many times. Instead, I encapsulated reusable code in class objects. Each object that is rendered on the screen was encapsulated in the *Model* class. The *Model* class loads Wavefront objects from Blender. Each model can consist of multiple meshes, which are encapsulated in the *Mesh* class. A mesh represents a single shape, and a model represents the combination of meshes that make up a complex object. To ensure that there are no name collisions, each class has its own namespace. The input callback functions, setup function calls, and shaders also have their own namespaces with all actions encapsulated in functions.