**Estanislao 7-1 Final Project Reflection**

Tomas J. Estanislao

Southern New Hampshire University

CS 330: Computational Graphics and Visualization

Professor Eisen Montalvo

October 21, 2023

**Estanislao 7-1 Final Project Reflection**

I chose my glass prism souvenir as my main complex object because I thought it would be fun to implement. I thought it would be interesting to try to emulate a glass object with all its reflections and colors— as well as the multiple images within. Because of this choice, it ended up being a great object to apply multiple overlapping textures onto. The vertex colors underneath also match some of the colors on my real-world object. The shapes composing it are a cube, a plane, cylinders, and a hexagonal prism. The rest of the objects in the scene are a candle, book, and ball plush. After creating the prism, it was relatively simple to create cube objects for the book and stand. The cylinder was a bit more challenging because I had to add so many extra vertices to create a circle-like object. I found the vertex coordinates online as:

V(x, y, z) = (radius \* cos(i \* theta), y, radius \* cos(i \* theta)), where i increments from 0 to 19, and theta is equal 2 \* pi \* 20.

I re-used this configuration for each cylinder, torus, and sphere object to define each set of 20 vertices that form a ring. I used element arrays to reduce the number of extraneous vertices. In my indices array I specified which vertices connected to form a triangle. This approach also required me to define extra vertices for wrapping textures—often re-defining vertex 0 with new texture coordinates— as well as if for the bases of my prisms. It was important to get the main dimensions set, and then to adjust each model in the scene—with scale, rotate, and translate for finetuning each transformation. I also added an alpha value to my RGB coordinates to adjust the opacity of certain objects, such as the plate that my hexagonal prism sits on. For lighting, it was important to have 2 overhead lights that mirrored my real-world scene. In addition, to setting the light color to a warm light-yellow.

**The camera moves through the WASDQE keys, respectively controlling: Forward, Left, Backward, Right, Down, and Up. Pressing the P key switches the projection view between perspective and orthographic. The mouse moves the camera angle direction, pitch, and yaw. The scroll wheel increases and decreases movement speed. I had trouble with the P key switching correctly when pressed, although it does work after pressing a few times. I believe it is either a problem with my boolean switch, or the projection matrix updating multiple times within the render loop.**

**The loadTexture() function is a utility function for loading a texture that allows for multiple image formats.** ProcessInput(), mouse\_callback(), and scroll\_callback() all use methods implemented in camera.h to process camera movement on the Camera object. My shaders are compiled with the Shader class implemented in shader.h. I created .vs and .fs files for my normal textures and for my lamp objects. Both are given to the default constructor of the Shader object. Furthermore, the Shader object has utility methods implemented that avoid having to use glGetUniformLocation. If I could make any further changes, it would be implementing a helper function to bind textures and render each container. This could cut down on repeated lines of code in the render loop and modularize my code. I would also like to explore a helper function that deletes the vertex arrays and buffers.

Overall, I am happy with my scene. I decided to use GLAD and follow learnopengl.com as much as possible for reference. I also avoided the SNHU tutorial code, as GLEW was used. However, tutorial 6 was very helpful to implement lighting. OpenGLSample was useful for implementing helper functions.