Luke Peters

Professor Jeff Phillips

CS – 330

18 February 2023

7-1 Project Two

As I look back on the journey of developing my final project, I am amazed at the progress I made in just six weeks. From creating simple text-based console applications to rendering fully immersive 3-dimensional scenes with light, my class accomplished an incredible feat that showcases the power of computers. In this reflection paper, I want to discuss the various elements of my application that made this journey possible, focusing on the algorithms and methods that I developed myself. Although my final project does not perfectly match my original concept, I intentionally changed it as I learned new techniques and discovered more appealing textures, some of which I created from scratch using Adobe Illustrator and Photoshop. While there are differences, the overall essence of the scene remains the same, and I hope it can be appreciated, nonetheless.

From the beginning, my aim was to make the program modular to facilitate the reuse of its components for future builds. I achieved this by separating the algorithms for shape building from the methods used to render the scene. Moreover, to make the application user-friendly, I consolidated all scene-creating items into their own class so that the user can focus solely on adding shapes and texturing them without having to scroll through hundreds of lines of code. To do this, I refactored the GLMesh struct to take in all the properties of the shapes, and the mesh object gets passed to the ShapeBuilder class to be constructed. Once the shape is constructed, textured, scaled, and transformed in the world, the mesh object is added to a vector called “scene.” The Render method loops through the vector, extracting each shape and drawing it on the screen. This setup helped me save code and reuse algorithms easily. For instance, when creating a new cylinder shape, I just needed to make a new mesh object with the required properties (size, rotation, texture, etc.). The entire scene can be built by the user in one class (“SceneBuilder”) without having to make changes to any other class or file.

The navigation component of the application contains a variety of functions that the user can use to manipulate what is shown on-screen. The standard motion of the camera uses the WASD keys for basic movement, Q and E for up and down movement, and the mouse to change the view of the camera. These keys/motions use the camera.h predefined motions to adjust the camera’s position within the scene. Additionally, to add some “testing” functionality to the application, the user can use the opposite end of the keyboard to adjust the spotlight’s position within the scene. The keys IJKL move the light around within the x and z planes, while U and O move it up and down. The light can revolve around the scene with the left and right alt keys, while the directional light from above can be turned on and off with the left and right brackets. Finally, the view perspective can change with V and B keys, and the shapes can be shown in wire-frame mode with the left and right arrows. Since the camera and light positions are determined by vectors for location, modifying these vectors ultimately modifies the object's location. As the user presses each key, the value at the specified location within the vector is adjusted, either by the camera.h file or within the main source code, and the object appears to move.

The most interesting part of my application is the "ShapeBuilder" class, which contains shape-building methods that I developed from scratch using minimal external assistance. One of these methods is the cylinder function, which employs mathematical calculations to determine a point along the edge of a circle based on its radius. By computing the sine and cosine of 2π times the section of the circle being drawn, multiplied by the radius and the starting coordinate, the method can locate the necessary point in 2-dimensional space. The method then draws a fan of triangles that start at the center origin and radiate outward.