**Final Project Reflection**

Sean Bruyere

Southern New Hampshire University

CS 330: Comp Graphics and Visualization

Professor Jeff Phillips

February 19, 2023

Although challenging at moments, this project has become one of my most favorite and rewarding in my time at the university. In selecting the items for this project, I wanted to find things that I felt would be achievable and still present enough of a challenge to push my learning. The first item I searched for was my complex object. Here, I found my son’s crayon piggy bank. To build this object using OpenGL, I started with trying to develop the cylinder. I at first struggled to develop the necessary code to render the shape, so I shifted to understand how to create a basic circle since the cylinder is simply a stack of circles in 3d space. Working in the a, y, z axis model, I used basic trigonometry to get the shape, setting the x and z coordinates to the cosine and sine vertices based on the provided angle and shape. I then looped through the number of vertices I wanted to generate the circle. The more vertices I provided, the more of a circle shape I would get because the shape is made of up triangles. With this, and a few other shapes I took advantage of the mutability of vector lists in C++. This helped my not have to write out each point individually. Once I solved this issue, I moved to building my cylinder. Since, I already knew how to calculate x and z, I set y to the height of my cylinder. I used the 0, 0, 0 point of the field as the center point of the cylinder and changed the value of y to negative or positive based on the value of the iteration in the loop. I would simply adjust the position of the cylinder when using the model view projection in my main loop. Once again, I took advantage of vector lists and looping to render the shape. I also researched how to use OpenGL GL\_TRIANGLE\_STRIP constant to help. Instead of needing three vertices, I can simply add the next vertices to get a triangle which worked well with how I implemented the array. Next, to get the cone shape on top of the cylinder, I simply reused the cylinder method I created and reduced the radius on top of the shape to get the aesthetic I was looking for.

Since this is an eight week course, I did not really organize my code files as well as I would like. I instead focused on quick convenience over a true object-oriented model. To start, I created header files to do most of the work in lieu of headers to encapsulate with cpp files to calculate the shapes. This made it simpler for me to add everything to main and created less overall clutter for my files. All my shape code was created under a single Shape.h file. I also created a header file for my individual objects in SceneObjects.h. Here, I could create each individual object in my scene using the shape file. In the lighting portion of the class, I also used this file to take advantage of the individual material shininess. This helped clean up my main file and allowed me to simply create the scene object in the main application. Outside of my complex shapes the others were relatively easy to create and render in my scene as they were simply two planes (one for my surface another for a sheet of paper), a sphere for the basketball, a cube for a Minecraft block, and a rectangular cube for the book. Overall, each of these objects provided me with enough experience to move forward with my learning and experiment further with OpenGL.

An important aspect of the project to give it a more personal experience is providing a means to navigate the scene. The input device it is designed for is the basic keyboard and mouse. To start, I had coded for basic 3D movement with the keyboard. I focused on up, down, left, right, and forward/backward movement. To get this functionality I first created a method to bind certain keys to certain actions and provided the main window while loop with an event listener. To move forward or backward the “W” key and “S” keys are pressed. Left and right are the “A” and “D” keys, while up and down are the “Q” and “E” keys. Each time any of these events are called, a calculation is made on the virtual camera position based on the camera movement speed and the forward position of the camera called the cameraFront. It should be noted that the cross product is needed to calculate the left and right movements of the camera. Next, I provided a function to look around the screen and adjust the movement speed using the mouse. In this part of the lesson, I learned about yaw (left/right) and pitch (up/down) to take advantage of all around movement in the scene. To get this to work, I had to implement the glm vec3 attribute and provide trigonometric calculations on each vertex of the camera front. For the x position, I needed the cosine of the yaw times the cosine of the pitch. For the y position it was the sine of the pitch and for the z it was sine of the yaw times the cosine of the pitch. All of this would adjust and automatically calculate as the mouse is moved in the scene. These basic ideas could be implemented regardless of input device. For instance, it would not be difficult to provide the same calculations to get a joystick to work on the scene.

I largely followed the learnOpenGL tutorials to learn how to get my application to function, but the entire project was something I had typed out and pieced together myself. Many moments were full of frustration, but I powered through and learned a ton on the way. Some of the more custom functionality in my program is the separation of each shape types and the building of the objects in another header file. This was simply a way to keep all of my code in one space and to declutter my main application. Each method is easy to update, especially in the Scene class. If I wanted to create another shape, I simply create a new object method and assemble all of my shapes to create that object. To also help simplify things, I created separate methods to load my textures in a list to implement into my shapes and created methods for my lighting sources. All of this helped my organize the code and helped me when I wanted to create edits or experiment with shapes. Despite this, I was not successful with all my creations. I do want to point out a method I left in my Shapes header but did not implement in my project. I had enormous difficulty when it came to the scene lighting portion of the lesson. I was simply not happy with my results and figured it had to do with how I was calculating my normal vectors on my shapes, especially with the more complex ones such as the cone or sphere. I know that the cross product of two edges of a triangle results in the normal of the triangle. I built a function to help me with this, but I spent way too much time trying to get it to work. Perhaps someday in the future I will try again.

In conclusion, I am sad to see the end of this course as it really helped pushed my understanding of programming in general. At times it was frustrating, especially with the lighting portion, but I really value the experience I gained with this class more than any other I have taken so far. The code I created is far from perfect but provides enough reusability and ease to create more complex projects.