**Module 7: Project Reflection**

Jeremiah A. Boothe

Department of Technology, Southern New Hampshire University

CS-330-R3323 Comp Graphic and Visualization

Dr. Brian Holbert

February 23, 2024

# Introduction

Now that the end of the term is coming closer, I fully understand why this class is notoriously known as the most difficult class of the Computer Science program. I have probably written more code in this one class than I have in the rest of the program combined. A lot of which was deleted and rewritten several times as I tried out different ideas and methods to get a sense of how various OpenGL functions work together.

# Choices:

I chose my scene because it is a place where I spend much of my time these days and there are plenty of things to choose from. I figured I would just see how far I could push myself and keep creating objects until time ran out or there was nothing left to create. Unfortunately, I did not get to complete the mouse. Another week or two and that may have been a different story. With every new addition the code grew exponentially, and my efforts to scale that down were quickly undone by every new possibility.

I spent far too much time on the lighting. There were numerous unfinished attempts to expand even further. The checklist of functionality I wanted to implement included shadow casting as the top priority and using imported normal map textures. Most of all, I wanted to capture a mood, as I often work late nights with the glow of the monitor and the highlights of various other light sources. To give it a broader sense of space I figured why not make the whole scene just float in a starry night sky for the time being. My room itself is not nearly as interesting to look at. Mostly just homemade sound dampening boxes filled with Safe ‘n Sound insulation and upholstered in random fabrics. I ran with ideas until they resulted in the desired result or into a temporary dead end.

# **Navigation**

## Keyboard:

Working through the tutorials helped me implement general WASD movement, as well as QE representing down and up respectively. The P key was implemented to switch between projection and orthographic, which I just started to grasp how that can be used for shadow casting and reflections. A bonus feature for this project is the implementation of a rotating pyramid. The L key starts the rotation, the K key stops the rotation. It was another piece from the tutorials that helped me expand the program beyond being a resource intensive photograph. Having the ability to get that light rotation going really brings life to the scene, particularly in close views where you can really see the transition of the light and color change as the rotating light washes out the other colors then moves out of range allowing the colored lights to fill back in. At one point I had altered that to make it infinitely grow larger over time eventually swallowing up the entire scene, which was a lot of fun!

## **Mouse:**

It supports general look around functionality, and scrolling the mouse wheel forward increases your movement speed up to a set point, and rolling backward puts the breaks on that speed. Speaking of speed, the sky box night scene is remarkably close but infinitely far away, you can move toward it eternally and never reach the end. If you switch to Ortho view, the skybox appears in the distance behind the scenes. I have an idea of how to fix that, but it is low on the list compared to other features I would love to see work.

# Custom Functions

At this stage of the process, it has been particularly challenging to find my own way in terms of abstraction and code reuse, but I still made the effort. First off, having a class specifically to call and build each shape populated with its own VAO’s, VBO’s and other general parameters was a great idea. Having the map to pull from and some flexibility with the objects made some things easier. In the current form I have tried to take that a step further with a global *Lighting* singleton that is instantiated with the overall shader program. I also created a separate map to hold pre-compiled shader programs but still have not determined if I should fully commit to that path over having those programs stored with the shapes. I am thinking having the shapes hold pointers to the pre-compiled shader programs might be the way to go. Thus having 1 version of each shader that could be referenced by many, instead of a bunch of duplicates for objects using the same shader.  
 Beyond general abstractions I worked out several functions that work well about 98% of the time although not perfect they got me in the ballpark. One example is the combining functions. These functions allowed me to trade a little bit more front end work for an overall reduction in time when creating various meshes. This allowed me to take a few entries and create a fully iterated representation of the mesh. This brings up another set of functions to write those meshes as comma separated values in text files. Then I could just copy that back into Visual studio and use it as a fully realized mesh via glDrawArrays(). While using the combining function was helpful and could also be used to just create a full mesh from a scaled down set of values upon execution, it was a resource drain. It did make a great utility function.  
 The last most significant expansion I have worked out which is relatively close, but I have not polished it yet, are the normalizing functions I used on the cylinders. It calculates the cross product of two triangles and makes a decent attempt at getting two triangles that ultimately make up a quad to work together and reflect light in the same way. I still had to make a few manual adjustments on some values, but it worked well on both the tapered cylinder and the normal cylinder. The latter still has one face that is a bit off but requires zooming in close and looking for it to even notice.

# Conclusion

I wanted to see how far I could push myself over the term. Unfortunately, I did not get extremely far with the mouse. You were spot on about that being the most difficult to pull off. If I spent another week or two working on nothing other than mouse iterations, I might be able to get a reasonable approximation.

It has been challenging to balance learning a heap load of new material, knock the rust off C++, and try to keep thousands of lines of code and values organized while adding and deleting new code daily. I did not make it nearly as far with modularization as I wanted to and realized that is exactly what led to the first OpenGL based game engines. It gives me a newfound respect for people like John Romero and his team as they worked out how to bring the original Doom to life. It was always impressive but viewing it from the outside to being more on the inside has changed my understanding of what they must have endured to achieve that.