CS-330: Computational Graphics and Visualization

Professor Wabara

Trice Bonomo

April 22, 2022

Project Reflection

I developed a 3D scene featuring a mortar, pestle, ramekin, and vase staged on a wooden table. For the final scene, I decided against including the pestle. I was not confident in my ability to render a cylinder within a cylinder in a way that was aesthetically pleasing, so I dropped the shape from my scene.

For my primitive shape selections, I chose shapes that closely resembled the intended object. With a wider variety of shapes to choose from, or with the option of choosing less than four unique primitive shapes, I might have made different choices. However, given the project requirements and shape restrictions, each decision was logical.

To build my scene, I decided to wrestle with the optimized codebase provided in the course tutorials. I did this for two reasons: to get practice working with unfamiliar code and to avoid rewriting the same code lines week to week. Using the given codebase saved tons of time because I didn't have to rewrite the OpenGL methods week to week, but it was also the source of numerous headaches. Since I made this decision in the first few weeks of the course, I had no choice but the keep at it as the concepts became more complex. While this created a rather painful coding experience, I learned a great deal along the way.

One of the most frustrating parts of my coding choices was dealing with the lighting. OpenGL's fragment shader and shader programs were some of the least comprehensible code included in the tutorials. The syntax is unlike anything else I've ever used, and it took me weeks to figure out how those sections of code worked. While working on my final scene, I finally figured out that I could create lamp objects the same way I created meshes and texture objects. Once I understood that I just had to figure out where to stage the lights and how to make the scene aesthetically pleasing. I assigned the different types of light to individual light sources and added a third lamp. The third lamp is the primary reason the vase looks like glass, and I felt it lent additional dimension to the scene.

Setting up the camera controls for my scene was pretty straightforward, except for the orthogonal view and the movement speed. Importing the Camera.h file made setting the WASD bindings easy. By adding a couple of lines of code to Camera.h, I was also able to get Q&E to work. Figuring out how to get the mouse wheel to work took time, and I never did figure out the orthogonal view. My code will shift to an ortho perspective, but it doesn't render properly. If I use glfwPollEvents(), the screen flickers, and toggling 'P' only works intermittently. If you don't press the button until a few seconds after the program starts running, and you don't press it quickly, it works. Otherwise, the screen flickers incessantly, and 'P' only works if you hit it in time with the polling cycle. I ended up switching to glfwWaitEvents() to deal with this issue, but it makes my camera movement choppy for the first few seconds any key is pressed. It is the one part of my program that I am not happy with.

The custom functions I created for the program include my DrawCylinder() and DrawTorus() methods. DrawCylinder() I wrote from the ground up, while DrawTorus is a hybrid of a sourced code solution and my work. Since my image involves a total of five cylinders, I knew I had to write a method for creating them. Creating a method to generate vertice and indices arrays meant I didn't have to write all the points for five cylinders. The torus method, though functional, is not reusable. It will only generate shapes at the same x,y, and z coordinate location. I did not bother fixing the logic because my scene only required one torus and instead focused on adjusting the lighting.

I am aware that using vectors in DrawTorus(), arrays in DrawCylinder(), and then pulling in code to draw spheres doesn't align with standard coding practices. However, these decisions make sense scholastically. Coding DrawCylinder() challenged me to create code from scratch, and DrawTorus() helped me practice repurposing someone else's code to suit my needs. Importing the sphere files forced me to troubleshoot environment issues and practice code integration. Though this program is not entirely modular, and I failed to figure out how to get the orthogonal view to work correctly, I am incredibly proud of it.