Christopher Dewey

SNHU: CS330

Module Seven

October 13, 2022

**Final Project Reflection**



I chose a scene consisting of five objects (six including the plane), because each object would pose different challenges in the creation of their 3D model and applying texture. Ultimately, I selected the tape measure, screwdriver, vernier caliper, and screw in my final scene. The tape measure consists of cylinders and cubes, the screw consists of a partial sphere, and a cylinder, the screwdriver consists of cylinders and pyramids, and the calipers consist of cubes, cylinders, and pyramids.

Some artistic liberties were taken to reduce the complexity of these objects to reduce time spent modeling. I modeled each object, assigned normal coordinates, and assigned texture coordinates by hand. Using graph paper, I sketched each model’s “face” and calculated each vertex location. I experimented with rendering the objects using the draw elements function first, and moved to drawing arrays after encountering some roadblocks. If I could start over, I would stick with using the draw elements function since it would have saved me countless hours modeling and tracing vertices and would reduce the sizes of my vertex arrays.

I went through numerous iterations when creating my project, going down deep “rabbit holes” that ended in more frustration than I am happy to admit, but each of these endeavors helped me make monumental progress understanding OpenGL and better my C++ skills. Aside from the model vertex arrays, I scrapped my entire project numerous times in attempt to troubleshoot problems.

Since OpenGL has a tendency to be syntactically verbose, I often found myself lost in my code looking for an individual line item, so in order to stay organized, I refactored, changed my comment style, utilized header classes, placed my shaders into independent GLSL files, and used Visual Studio’s pragma region functionalities without impacting scope. Even though I repeated writing code multiple times in each iteration, I finally identified the two most impactful areas causing my frustration: shaders and normal coordinates. Once I finally figured out that my normal (and texture) coordinates needed to be set properly before tinkering with my shader source code, pieces finally started to fall together!

By abstracting shaders to their own files and shader actions to their own Shader class, I was given modular control over how I used and reused shaders in my application. Instead of creating each shader individually and applying uniforms using the verbose OpenGL constructs, I could simply call each shader object and reduce clutter in my main cpp file.

Placing texture creation into its own class allows for easier binding and unbinding of textures through the reuse of a custom name that is more memorable than the ones offered in the OpenGL libraries.

Through module six, I linked external libraries locally. In the final project, I linked each library dynamically and included the critical components in my code files, so other users are no longer required to manually link these libraries prior to running the code on their local machines. This adds to the file size of the project, but as long as few relatively small libraries are used, the additional storage space required is well worth the convenience.

In addition to the models and OpenGL functionalities, I established keyboard navigation controls through a manual implementation of a “camera” similar to the example provided in the LearnOpenGL material. Pressing the W, S, A, D, Q, or E keys, a user can move forward and backwards (in Z space), pan left and right (in X space), and move up and down (in Y space), respectively. Once a keyboard input handler (KeyPressHandler) was established to recognize when each key was pressed, the camera position vector could be incremented or decremented accordingly to give the user the illusion of moving through 3D space, but this actually results in the “world” moving around the user in opposite directions.

To enhance user interaction, a mouse position callback (MousePositionHandler) allows the user to “look” around the scene by moving the mouse. This stores yaw and pitch in global variables that also impact keyboard movements, making navigation more intuitive. Each navigation action key will always move the “camera” in the direction that correlates with the current cursor position.

If default navigation speeds are too fast or slow for the user’s liking, the mouse scroll wheel will increase or decrease speed if rolled forward or backward, respectively. When the scroll wheel is moved, a global variable for movement speed is modified and multiplied against each keyboard navigation command.

Lastly, a custom camera orbit function was written to automatically circle the scene. This disables the keyboard and mouse handlers, with the exception of the middle mouse button, which is used to toggle this feature on or off.

Although this course has been the most strenuous in the entirety of my time spent studying computer science, I am proud to have accomplished as much as I have. I do not expect to work in a graphics field in my future, but the skills learned here are applicable in every aspect of programming and are invaluable to my career aspirations!