CS330 Final Project

Gary Clark

Southern New Hampshire University

CS330 Final Project

From early on in this course, I had a strong idea of what I wanted to build within the OpenGL environment. The scene I chose was simple, yet containing enough of a challenge to satisfy the criteria with multiple shapes, textures, and lights within the model. While the OpenGL language was completely missing within my skillset, I has confidence that I could learn quickly and the first couple modules reaffirmed that belief. However, the proverbial brick wall soon found me and a variety of events amplified this challenge to an astronomical degree. Through the struggles, I have managed to piece together a model of the scene which was selected at the beginning of the term even if it differs from what I initially envisioned.

The scene of which I was tasked with modeling in OpenGL was fairly simple consisting of a candle, a mug, a book, and some earbuds laid out on a countertop. These items can all be built using very basic primitive meshes and much of the detail on each object can be generated with textures. This scene would require the development of code that could produce a cylinder, a sphere, a rectangular prism, and a torus which would then be called upon to build the objects. Based on my research into OpenGL, it was understood that these shapes would be built using vertices that the program could link together to draw a surface. Each of these vertices needed to not only have x, y, and z positional coordinates, but also coordinates detailing how textures would be mapped and which define the behavior of light reflections. Keeping this in mind, it was paramount that any code built to create a given mesh object be able to also generate these datapoints for each vertex. Once the shape’s vertices are fully defined, one must then tell OpenGL what to do with this data. Many of the shapes used a similar method to draw surfaces from the vertex data, but during development the cylinder object was creating a challenge working in the same manner as the other objects. The other objects in the scene such as the book or the countertop were being draw by OpenGL using indices which serve to tell the program in what order it should connect the points. However, to generate the necessary cylinders for the candle, mug, and earbuds, the method of choice was the Draw Arrays function. This function draws directly from the vertex buffer object and connects the vertices using predefined shapes. To me, this breakthrough made the most sense because it made it possible to keep all the drawing and rendering functions for the meshes in the same class. None of the objects within the scene were overly complicated aside from the mug and for a time the challenge of creating the “inside” of the mug stumped me. I settled on using a texture to mimic the appearance of a hollow mug which does help reduce the complexity of rendering the shape. Once the shapes and scene had been generated, movement controls had to be devised.

Within this project, much of the code around the creation of a virtual camera was borrowed from the various assignments throughout the term. This camera would hold an initial position upon startup which would then be updated based on inputs from the user. A standard WASD control method was utilized with the mouse providing directionality for both the view and the movement. Each keystroke would trigger a function to add or subject from each positional vector which could then be fed into the view location vectors. The view vectors would then form the V in the MVP matrix which defines how OpenGL will project the data into the viewport. The default method of projection is perspective which mimics depth by scaling object down the further away the camera much like in the physical world. However, other view styles can be useful and as such there was an input option added to switch to an orthographic projection style. This style is 2D and like a blueprint or a paper drawing of a cube where the scale of objects is uniform.

As development of the scene progressed, there were many different functions required to generate the necessary data. It became unfeasible to contain it all within a single file and because of this I had decided to break off the code to generate the vertex data for each shape into separate class files. These files would provide the data necessary for the mesh class to draw and render the shapes. The shaders were also split off into a separate file of which the mesh class would have access to. My objective was to minimize the amount of code in the main method while not sacrificing on functionality. By splitting these shape classes off, it allowed the testing of each object independently from the next. In this design structure, the mesh class handles the generation and drawing of all VAO’s, VBO’s, and EBO’s while also containing the structure definitions of vertices and textures. It feeds data into these buffers through the glm::vec3 and glm::vec2 datatypes and then handles the world and clip transformations of the scene objects. The advantage of this structure is clear in that it will accept any vertex data that is presented in this form meaning that formulas for other shapes could be generated and drawn using this single class. Once the method could handle all the vertex data, there were some added functionalities created to alter the way a shape is drawn in the scene. Transformations such as rotation and scaling can be changed during the call to draw a given meshes’ data which further extends the utility of this single class method. There were multiple instances where I was able to create multiple pieces of an object using a single mesh through these transformation vectors which greatly reduces the required resources of the program. In theory, one could draw every cylinder required in the scene from a single mesh by simply scaling rotating and moving the piece around. In this scenario, the computer only must calculate the vertices once, then simply transform them into the world and clip spaces. Through the modularization of the code in this program, it becomes far more portable and powerful giving the user the ability to generate a near endless number of objects or models. This project, while extremely challenging, was also incredibly rewarding in both the knowledge and experience gained.