# CS - 330

Final Project

David France

david.france@snhu.edu

Southern New Hampshire University



In my scene I have a table, cheese board, eggs, and a bowl, represented by a plane, cubes, spheres, and a cylinder respectively. Starting with the tabletop, it’s the base of the scene, so is naturally a plane. To program, I could have used the CreatePlane() function from ShapeGenerator.cpp, but that would have been needlessly complicated as it’s just two triangles, so I manually created the mesh. The plane also lent itself to the use of a texture – I was able to use a picture of the actual table to add realism to the base of the scene.

The cutting board, cheese block, and cheese slice combined to form the complex object of a cheese board at the center of the scene. Here, I implemented my own function to draw multiple cubes with one piece of code. Rather than map out the vertices for each cube, I created a function – createCube() – to take the X, Y, and Z dimensions to create the size that I was looking for, then translated each one to align them. Like with the tabletop, I was able to add textures to provide realism. This is particularly evident in the cutting board, which looks just like the original.

The eggs and bowl were the most difficult to render realistically and place within the scene. Following the advice of one of the tutorial videos, I kept it simple by rendering them as spheres and a hollow cylinder respectively. I took pictures of the real-life eggs to get the texture right, then rotated the eggs to show the best side. For the bowl, I modified the cylinder.cpp file’s render() function to only render the bottom of the cylinder, but not the top. This provided an acceptable representation of the bowl from the picture. The texture on the bowl didn’t render like the original, but I was happy with the pinwheel pattern and the way the colors on the edge lined up, so decided to keep it. As far as placing the objects within the scene, I spent a fair amount of time making small adjustments to size and placement so they would closely match the original.

Navigating the scene is simple using WASD directional keys, combined with mouse movement, scroll wheel, and QE keys for vertical movement. The WASD keys allow the operator to move the camera forward, backward, and side to side. The mouse movement allows for looking around the scene. Combined, these controls provide for natural movement of looking where you want to go and moving there. The scroll wheel controls camera speed. This allows both quick movement around the scene and slowing down for closer looks. I found this function very helpful when arranging all my objects on the X, Y, and Z axes. Moving up with E and down with Q makes it much easier to look at the scene from above or below. These were also very helpful in the development of the scene. Finally, I assigned the projection matrix control to the P key so that each time the key is pressed the projection toggles between perspective and orthogonal.

Developing the camera movement was also one of the ways I customized functions in my program. Originally, the camera.h file contained no code for vertical movement and used the scroll wheel for zooming in and out. I added functionality to be able to call ‘UP’ and ‘DOWN’ movements in the processMovement() method for the E and Q keys. I also changed the scroll wheel to affect camera speed instead of zooming. This required setting a minimum speed to prevent the operator from scrolling backwards into the negative, which would have resulted in the W and S keys being reversed. Finally, I created the keyCallback() method to handle the orthogonal/perspective toggle on P key press. I registered this with glfwSetKeyCallback() so it would only count one key press at a time. Putting it in with the rest of the keyboard handling causes problems where it registers many key presses at a time. I was able to reuse this code throughout the course to maintain the desired camera movement across programs.

Two other functions I developed/modified were the createCube() and Cylinder() methods. The createCube() method allows for inputting the VBO and VAO for the given cube, along with height, width, and depth dimensions, to create the shapes, as well as the vertex buffer and vertex attribute objects. This allowed me to make the cutting board, cheese block, and cheese slice easily and set them up to be drawn later in the program. For the cylinder’s render() function, as mentioned earlier, I modified it to only draw the base circle to render the shape as a hollow cylinder. Both of these can be reused easily to pump out additional objects.