7-1 Final Project: 3D Scene Reflection

CS-330-R3322 Comp Graphic and Visualization 24EW3

Elizabeth Robles

Southern New Hampshire University

February 24th, 2024

**Development Choices**

The last time I took this course, I was not anticipating how difficult and time consuming it would be to learn the concepts of modern OpenGL. Because of this, I chose objects for my final project that were each very complex and ultimately, unachievable for me. This time around, after spending a lot of time learning and watching tutorials, I decided to choose objects that I knew I could create, such as the cube-shaped sponge, the pyramid-shaped quartz paperweight, and cylindrical and sphere-shaped makeup item, as well as a plane for my objects to rest on. I was able to program for the required functionality by calculating the vertices for my objects, creating shaders, mapping textures to my objects, adding lighting and a camera to view the scene from different angles.

For the cube-shaped sponge, I initially created a regular cube, like the one in LearnOpenGL; to make it more rectangular and flatter, I simply adjusted the scale of the cube along the X, Y, and Z axis. I then mimicked the texture by finding a free image online of a purple sponge and added a filter to it to make it better match my image. I followed a similar process when creating the quartz pyramid and wood tile plane (I took a separate photo of just my wooden tiles to incorporate into the scene). The sphere and cylindrical-shaped object were a bit more complex for me to figure out. I used vertex and index calculations for both the sphere and cylinder, I made texture mapping easier on myself by creating one JPEG image that incorporated a metal portion for the lid and a pink/glass portion for the body of the cylinder and UV mapped the top 25% of the image to the upper portion of the object. I lastly combined the vertices and indices of both shapes by adjusting the sphere to sit on top of the cylinder, so that they could be rendered together.

**Navigation**

For a user to navigate my 3D scene in a multitude of ways, I used LearnOpenGL’s camera.h header file to facilitate camera movements in my 3D scene. For example, I’m able to move my camera around by using the ProcessKeyboard method in the camera.h file. I’m able to change the cameras orientation using the ProcessMouseMovement method and zoom in and out of the scene by using the ProcessMouseScroll method. In my app.cpp file, I set up the camera to view the scene at a certain position using the camera object:

Camera camera(glm::vec3(0.0f, 1.f, 3.0f));

Based on mouse movement, the function glfwSetCursorPosCallback is used to calculate the offset from the mouse position and passes those values to camera.ProcessMouseMovement, which adjusts the cameras yaw and pitch, which allows for the camera to look around the scene. WASD, QE, P, and Esc keyboard inputs are set up to allow camera movement to move back and forward, left and right, between orthographic and perspective projection, lift and lower the camera, and escape from the scene. With this camera implementation, a user can fully explore the scene in whichever perspective they’d like to.

**Custom Functions**

There are a couple of functions and classes throughout my project that make my code more modular, organized, and reusable. For example, a simple representation of this is in the App class, where window creation, initialization, and the main rendering loop occurs. This class could be used in pretty much any OpenGL project that requires a window setup. In my mesh class, the geometry of my pyramid, cube, plane, and sphere/cylinder object can be rendered because of the vertex buffer object, element buffer object, and vertex array object, along with drawing operations. This class can be reused to create pretty much any geometrical object I wish to create. In my shader class, loading, compiling, and linking shaders are used to render with OpenGL. The shader class also includes functions to set shader uniforms, such as transformation matrices and colors that are important for rendering my scene with lighting, textures, colors, etc. This could be used in any OpenGL project that requires shaders.