3D Workspace Rendering Design Decisions

Noah M. Figueroa

Department of Computer Science, Southern New Hampshire University

CS 330 – Comp Graphic and Visualization

Professor Holbert

February 20, 2025

Development Choices

The decision to replicate a 2D workspace in a 3D environment stemmed from the desire to showcase foundational concepts utilizing the OpenGL graphics API, while also presenting a challenge. I chose this workspace because it allowed me to replicate various objects using basic 3D shapes. The desk was created with a box. The drawer was created with a plane and boxes. The magazine in the drawer was created with a plane. The pens and the mouse were modeled using cylinders. The keyboard and the books were created using boxes. The book ends were constructed from various cylinders. The cup’s body was created with a cylinder, while the dish, handle, and lip of the cup were made using tori. The monitor was built with a box for the screen, and cylinders for the stand and legs. The computer and legs of the desk were created using boxes. The desired outcome was produced by performing various transformations on the shapes such as scaling and rotating. Texturing the objects provided the intended visual appearance of each object. Shaders further enhanced the appearance of objects by controlling the way in which light interacted with their surfaces.

Navigation Within the Scene

The ability to navigate the 3D scene is provided through control of the virtual camera. The camera can be moved up and down using the ‘Q’ and ‘E’ keys respectively. The zoom in and out can be controlled using ‘W’ and ‘S’. The left and right pan are controlled using ‘A’ and ‘D’. The speed of the camera is controlled using the scroll wheel while the ability to look around is controlled by moving the mouse. Using the “Mouse_Position_Callback” method, I was able to record the initial mouse position as well as any subsequent positions to accurately calculate input. This method was automatically called every time the mouse moved. Using the

"ProcessKeyboardEvents" method for keyboard input, every keystroke calls the "ProcessKeyboard" method which proceeds to move the camera in the desired direction. The "ProcessKeyboardEvents" method is continuously called from the main while loop to refresh the rendering display.

Custom Functions and Modularity

Custom functions were implemented in the program to keep the code clean, modular, and easier to maintain. One key function is "SetTransformations". This function sets the transformation values in the buffer, ensuring everything is properly prepared for rendering. It's highly reusable and was called every time a shape was drawn, reducing redundancy. By using this function, I saved 16 lines of code for each of the 44 calls, resulting in a total savings of 704 lines. Another important function is "SetShaderMaterial". This function takes a material tag from a custom object material, passes the unique values to the shader, and ensures that the appropriate material is applied during rendering. With only 14 lines of code, this function was called 44 times, saving 616 lines in total. Similarly, the "SetShaderTexture" function manages texture application. It takes a texture tag from a custom texture, locates the texture, and applies it for the rendering process. This function, while efficient at just 14 lines long, saved a total of 308 lines after being called 44 times. These functions not only streamline the code but also enhance its reusability, helping to reduce redundancy and increase maintainability.