Thomas Strogen  
Final Project  
CS-330  
Professor Gray  
12/14/2024

The initial approach I had planned for this project was ultimately shut down, which turned out to be a blessing in disguise. This allowed me to focus on a simpler, more streamlined design without becoming overly fixated on perfecting individual components, which could have caused misalignment or distortion elsewhere. The weekly instructional videos were invaluable for understanding and implementing functionality, while the OpenGL sample provided an excellent reference when I encountered uncertainties or unexpected outcomes. From that point, it became an iterative process of determining the placement and behavior of elements, which proved to be both manageable and effective.

Navigating the 3D scene is designed to give full control over how the world is explored. The keyboard serves as a steering mechanism: pressing W moves forward, S pulls back, and A and D allow strafing left and right, almost like sliding along a flat plane. But that’s not all—Q and E are included for vertical movement, enabling movement above or below the action, covering all axes of motion.

The mouse adds another layer of control. It functions as a tool for orientation: moving it side to side adjusts the camera’s yaw—the ability to look left or right. Moving it up or down controls pitch, enabling a view of the sky or the ground. Additionally, the scroll wheel acts as a precision dial, allowing for zooming in or out or even controlling navigation speed.

The camera itself offers flexibility by allowing switching between two types of projections. Pressing P activates a perspective view, where objects further away appear smaller, just like in real life. Pressing O switches to an orthographic view, ideal for technical exploration, where size remains consistent regardless of distance.

This setup isn’t just about movement; it empowers interaction with and observation of every corner of the scene. By blending precise mouse control with intuitive keyboard shortcuts, it becomes possible to seamlessly explore, adjust, and focus within the world.

In a well-structured 3D application, modular functions keep everything streamlined and manageable. For example, the LoadSceneTextures() function is designed to handle the loading of textures needed for a scene, such as wood, linen, and checkerboard patterns. Each texture is loaded into memory and mapped to a specific slot, ready for use. Adding a new texture is straightforward—one additional line of code loads it, and the function handles the rest, including binding the textures with BindGLTextures().

Another example is SetupSceneLights(), which configures the lighting for the scene. This function handles both directional light for ambient illumination and point lights for localized effects. Each light is defined with ambient, diffuse, and specular properties. Adjustments—such as changing brightness, repositioning a light, or adding new ones—can all be made here without affecting other parts of the code.

The DefineObjectMaterials() function organizes materials for the objects in the scene, assigning properties like color, reflectivity, and shininess to materials such as ceramic, steel, and cloth. This modular approach makes it simple to add new materials—whether it’s a metallic finish or a soft velvet texture—by extending the list within this function.

For interactivity, the ProcessKeyboardEvents() function links keyboard inputs to specific actions, such as camera movement or toggling between perspective and orthographic views. Adding new controls is straightforward, as each keybinding and its logic is defined here, ensuring consistent and organized input handling.

Each function focuses on a specific aspect of the application, from managing textures and lighting to defining materials and handling input. This approach keeps the codebase modular, efficient, and easy to expand or modify as needed.