Alexander Ray

Prof. Aly

Cs 330

August 20, 2025

**Justify development choices for your 3D scene**.

I chose the specific objects for my 3D scene—two bottles and a dumbbell—to demonstrate a variety of modeling techniques using basic geometric primitives. The first bottle, composed of a prism, a pyramid, and a cylinder, was an exercise in **compound object modeling** and a test of my ability to align and position multiple shapes precisely. The second bottle, using a cylinder and a tapered cylinder, provided an opportunity to work with different shapes while maintaining a cohesive design. Finally, the dumbbell, made from a cylinder and two spheres, allowed me to practice creating a more complex, everyday object and apply distinct textures to its different parts. These choices allowed me to showcase a wide range of skills and ensure that all the required functionality—such as transformations, texturing, and lighting—was fully utilized. To achieve this, I developed a set of modular functions. For instance, the SetTransformations() function is highly reusable, taking scale, rotation, and position as arguments and applying them to any object, regardless of its shape. This approach kept my code clean and prevented me from having to manually write transformation matrices for every single object, thus making the overall program more efficient and easier to debug.

**Explain how a user can navigate your 3D scene**.

A user can navigate my 3D scene using both the keyboard and the mouse to control the virtual camera. The camera's movement is handled through the **keyboard**, allowing the user to move forward, backward, left, and right, as well as to ascend and descend. This provides the user with control over their position within the virtual space. To change the camera's orientation and look around the scene, the user can use the **mouse**. Moving the mouse triggers changes in the camera's pitch and yaw, effectively simulating a "look-around" motion. The camera's behavior is set up to ensure smooth navigation. For example, when using the mouse, a function calculates the new direction vector for the camera based on the mouse's x and y offsets. This decoupled control scheme—position via keyboard and orientation via mouse—provides an intuitive and immersive navigation experience.

**Explain the custom functions in your program that you are using to make your code more modular and organized**.

To make my code more modular and organized, I created several custom functions. The SetTransformations() function, as mentioned earlier, is a key example. It accepts parameters for scale, rotation, and position, calculates the final model matrix, and then passes it to the shader. This function is reusable because it can be called for any object in the scene, eliminating repetitive code. Another crucial function is CreateGLTexture(), which takes a filename and a string tag, loads the image data, creates an OpenGL texture, and stores its ID with the provided tag. This function centralizes the texture loading process, making it easy to add new textures to the scene without cluttering the main rendering loop. The FindTextureSlot() function complements this by allowing me to retrieve the correct texture slot index using a simple string tag, further abstracting the underlying OpenGL details and improving code readability. Finally, SetupSceneLights() consolidates all the lighting setup into a single, well-defined function. It defines the light's properties—position, ambient, diffuse, and specular colors—and passes them to the shader. This makes it simple to adjust the scene's lighting by changing values in one place, rather than searching through the code for each light property. These functions collectively enhance the code's readability, reduce redundancy, and make the program more manageable and scalable for future modifications.