Design Decisions

**Development Choices**

For this project, I recreated a small personal space in my room, including objects like a blue metallic water bottle, two light bulbs, a Rubik’s cube, and two steel dumbbells, all across the backside of the dark wooden end of my bed​. I picked this setup because it had a variety of surface types including metal, glass, plastic, fabric, and wood, which let me demonstrate the use of different lighting, material, and texturing techniques. When coding the scene, I made sure to set up each object's surface properties carefully using the DefineObjectMaterials() function. I programmed specific material settings such as ambient strength, diffuse color, and specular highlights to match how the real-world objects interact with light. For example, I kept the dumbbells' steel material highly specular and reflective, while the carpet material was given almost no shininess to simulate its soft, dull appearance. I also custom-loaded and bound textures like the blue metallic paint, oak wood, carpet, and dark wood grain to accurately represent the real scene. In RenderScene(), I made use of modular functions like SetShaderMaterial(), SetShaderTexture(), and SetTransformations() to organize the rendering steps cleanly and avoid repetitive code. These design choices made the project more scalable, polished, and easier to debug during development.

**Navigation**

Users can navigate the 3D scene interactively through keyboard and mouse input, which I set up in the ViewManager.cpp file. I configured movement with the W, A, S, and D keys to move the camera forward, backward, left, and right, while Q and E allow moving vertically. The mouse moves the camera’s view direction based on cursor movement, and the scroll wheel adjusts the speed of the camera movement. In the code, I customized the Mouse\_Position\_Callback() and Mouse\_Scroll\_Callback() functions to fine-tune how smoothly the camera rotates and zooms in/out. I also edited ProcessKeyboardEvents() to allow up and down movement on top of the basic forward and sideways movement. Also, I implemented the ability to toggle between perspective view and orthographic view by pressing the 'P' and 'O' keys, which required modifying the PrepareSceneView() function. When switching views, I repositioned and reset the camera’s front vector appropriately to match orthographic projection rules. These edits created a flexible and smooth user experience for exploring the scene from different angles and perspectives.

**Custom Functions**

To keep the code modular and organized, I used a few important reusable functions. DefineObjectMaterials() organizes all material settings for the different objects, letting me easily tweak surface behavior without touching the main render code. SetupSceneLights() groups all light source configurations together, including setting up four lights with proper ambient, diffuse, and specular components, and adding a colored warm light to meet project requirements. I also used reusable functions like SetTransformations(), SetShaderTexture(), and SetShaderMaterial() to separate texture, material, and transformation setup cleanly from actual shape rendering in RenderScene(). This structure made it much easier to debug problems with lighting, object appearance, or navigation during the final polishing phase.