In developing the 3D scene for the final project, I aimed to translate a selected 2D reference image into a three-dimensional environment using OpenGL, C++, and GLSL shaders. This document details the key design decisions made throughout the process, including object selection, texture application, lighting configuration, navigation controls, and code organization. The result is a 3D scene that meets the stated project requirements while maintaining clarity, visual coherence, and technical correctness.

**Object Selection and Representation**

My first step was to identify the core elements of the chosen 2D reference image and approximate them using low-polygon 3D primitives. After analyzing the image, I decided on four main objects that would closely represent the image’s components: a floor (plane), a cube, a sphere, and a lamp constructed from a cylinder and cone. These choices ensured variety and met the requirement that one object be composed of two or more primitive shapes. The plane serves as the ground plane, the cube and sphere act as decorative or structural elements, and the lamp (cylinder plus cone) provides a recognizable multi-shape object that stands out in the scene.

Maintaining a low polygon count was essential to keep modeling simple and efficient. The provided shape-loading utilities (e.g., ShapeMeshes) offered basic primitives with manageable triangle counts well under 1,000. Scaling, translating, and rotating these shapes allowed me to position them thoughtfully around the center of the scene. The cube was placed near the origin, the sphere slightly offset, and the lamp positioned to the left, ensuring a balanced layout that approximates the composition seen in the 2D image.

**Texture Application and Selection**

Two of the scene’s objects are textured to meet the requirements. For the floor, I used a high-resolution pavers.jpg texture to simulate a realistic tiled surface. This texture was scaled within the shader to repeat seamlessly and avoid visible tiling patterns. For the cube, I applied a breadcrust.jpg texture, chosen for its subtle pattern that adds visual interest without overwhelming the scene.

The lamp and sphere also feature textures (gold-seamless-texture.jpg and circular-brushed-gold-texture.jpg, respectively) to enhance their metallic appearance. By using these textures, the objects gain a reflective, polished look that helps them stand out. The background (using drywall.jpg) provides a neutral, off-white surface, eliminating the black void and contributing to a more cohesive environment. All textures were sourced from royalty-free image repositories, meeting the project’s sourcing guidelines.

**Lighting Configuration**

Proper lighting is critical for a polished visualization. I implemented a global ambient light to ensure no object is completely in shadow, providing a base level of illumination. Additionally, a directional light simulates sunlight at a slight angle, producing soft shadows and highlighting object contours. A point light with a warmer tone is placed near the lamp object, adding localized variation and depth.

All components of the Phong shading model—ambient, diffuse, and specular—are utilized. Ambient lighting brightens the scene globally, diffuse lighting emphasizes object surfaces facing the light sources, and specular highlights impart a subtle reflective quality to metallic surfaces. By adjusting these parameters, I achieved a visually balanced scene that does not appear overly harsh or washed out.

**Camera Placement and Navigation**

To provide an effective viewing angle, the camera is positioned slightly above and away from the center of the scene. This vantage point allows the user to see all four objects clearly and appreciate the textures and lighting. The code supports both perspective and orthographic projections, toggled via a user input. Perspective projection offers depth and realism, while orthographic projection can be used for more technical views.

Navigation controls enable the user to move along the X, Y, and Z axes using WASD (horizontal plane) and QE (vertical movement) keys. Mouse movement adjusts the camera’s pitch and yaw to look around, and the mouse scroll wheel changes movement speed. These features meet the project’s requirements for nuanced camera controls, allowing the user to explore the scene fully.

**Code Organization and Best Practices**

The code is broken down into logical, reusable functions. Separate classes for shaders, shapes, and scene management ensure modularity and readability. The SceneManager sets up materials, textures, and transformations, while ShaderManager handles uniform updates for lights, materials, and camera matrices. Each step—loading textures, defining materials, placing objects, and configuring lights—is achieved through well-named functions, reducing complexity and improving maintenance.

Every function and material definition includes comments explaining its purpose. The scene preparation code is brief and clearly annotated, making it easy for others to understand the workflow. Adjusting parameters like light intensity or texture scaling is straightforward, thanks to encapsulated logic in setter functions.

**Conclusion**

Throughout this project, I carefully considered object selection, texture application, lighting conditions, camera positioning, and navigation controls to create a 3D scene that is both visually pleasing and functional. The chosen objects reflect the client’s 2D reference image as closely as possible within the constraints of simple primitive shapes. The textures and lighting enhance realism and depth, while the camera and navigation controls allow the user to interact with and explore the scene fully. By adhering to best practices in coding, commenting, and organization, the final project meets all stated requirements and can be easily understood, maintained, and extended in the future.

A screenshot of a computer

Description automatically generated