**Justify Development Choices for the 3D Scene**

For my 3D scene, I chose to create a small tabletop environment featuring a mug placed on a surface. The mug was constructed using a combination of a cylinder for the body and a torus for the handle. These shapes were selected because they replicate common, real-world geometry and demonstrate my understanding of mesh transformations and scaling. The table surface was represented using a plane mesh, which provided a simple and stable base for applying texture mapping and lighting effects.

I implemented two texture maps — a stainless steel texture for the table surface and a drywall texture for the mug. These materials were chosen to demonstrate proper UV mapping, texture scaling, and the use of different material properties in OpenGL. By loading these textures through the LoadSceneTextures() function, I was able to achieve realistic surface detail that enhanced the depth and visual realism of the rendered scene. Additionally, I applied lighting to the scene using two light sources positioned strategically to illuminate the model and create visible reflections and shading across the objects. These lighting choices helped highlight the texture work and gave the model a more natural appearance under dynamic lighting conditions.

Overall, these design and technical choices supported the required functionality by showcasing how object transformations, texture mapping, and lighting work together within an OpenGL rendering pipeline.

**Explain How a User Can Navigate Your 3D Scene**

User navigation in the 3D scene was implemented through the ViewManager class, which handles camera movement and orientation. The camera allows users to explore the scene dynamically, providing a more interactive experience. Input devices, such as the keyboard and mouse, were used to control camera positioning and rotation. For example, standard movement keys (W, A, S, D) allow users to move the camera forward, backward, and side-to-side, while mouse movement adjusts the camera’s pitch and yaw to simulate looking around the environment.

The mouse scroll wheel was used to control the zoom or “movement speed,” allowing for closer inspection of the mug or a wider view of the entire table setup. This navigation setup mirrors standard first-person or free-fly controls used in 3D applications and games, providing intuitive and responsive user interaction. These controls were configured to update the camera’s position vectors in real-time, ensuring smooth transitions and maintaining consistent lighting and object orientation relative to the user’s point of view.

**Explain the Custom Functions Used to Make the Code More Modular and Organized**

To make the code modular and reusable, I implemented several custom functions within the SceneManager class. Functions such as LoadSceneTextures(), SetShaderTexture(), SetShaderMaterial(), and SetTransformations() helped separate responsibilities and reduce redundancy.

The LoadSceneTextures() function handles all texture initialization and binding, ensuring that each image file is correctly loaded into GPU memory and associated with a tag for easy retrieval. This modularity allows additional textures to be added later without modifying other parts of the rendering pipeline. The SetTransformations() function handles object scaling, rotation, and translation in a single place, ensuring consistent model-view transformations across all objects drawn in the scene. Similarly, SetShaderTexture() and SetShaderMaterial() allow for streamlined management of textures and lighting properties without rewriting OpenGL uniform code multiple times.

By designing these reusable functions, the project maintained clean, organized, and readable source code. Each function has a single, well-defined purpose, which aligns with software engineering best practices and supports easier debugging and future enhancements. These coding decisions contributed to the overall structure, clarity, and maintainability of the final 3D scene project.