**Final Project Reflection**

**Development Choices**

In this 3D scene, I chose to model a collection of everyday objects, including a coffee cup, a notebook, pencils in a holder, and a lamp with glasses. These objects were chosen to reflect a typical workspace environment, allowing for a variety of shapes and textures to demonstrate the 3D rendering techniques learned throughout the course. The coffee cup, for example, uses both cylindrical and toroidal shapes, which required me to program two different 3D shapes using basic primitives. Additionally, the notebook provides an opportunity to experiment with box shapes and texture mapping to simulate leather. The glasses required finer alignment and rotations, and the lamp adds vertical variety to the scene, making it more visually interesting.

When programming the required functionality, I ensured that each object followed consistent transformations for scaling, rotation, and translation. This ensured that all objects were correctly placed relative to one another. For example, the pencils had to be precisely placed inside the pencil holder, and the lenses of the glasses had to be aligned with a connecting nose bridge. Implementing different textures, such as stainless steel for the lamp post and reflective materials for the lamp shade, helped enhance the scene’s realism.

**User Navigation**

The 3D scene is set up to be navigable through standard camera controls using a virtual camera. The user can control the scene's perspective using a combination of keyboard and mouse inputs. Specifically, the WASD keys allow for movement along the horizontal plane, while the QE keys enable vertical movement. The mouse allows for pitch and yaw adjustments, which enable the user to rotate the camera and view the scene from different angles. The camera’s perspective can be switched between orthographic and perspective views, providing flexibility for users who prefer different viewing modes.

By enabling these navigation controls, users can explore the 3D workspace, moving closer to objects or viewing them from different heights. This functionality was programmed to ensure smooth navigation and an intuitive experience for the user, making it easy to observe the detailed textures and lighting effects applied to the objects in the scene.

**Custom Functions for Modularity**

To make the code more organized and modular, several custom functions were created. For example, the SetTransformations function applies scaling, rotation, and translation to the objects. This function ensures that the transformations are uniform across different objects, making it reusable for all items in the scene. By passing specific parameters for scale, rotation, and position, this function allows for flexible adjustments without duplicating code.

Another custom function is the SetShaderColor, which standardizes how colors are applied to objects. This function enables the reuse of code when assigning color values to different objects. By separating these functionalities into modular functions, the code becomes easier to maintain and extend, which is particularly useful if more objects or features need to be added in the future.

Finally, the texture-loading function is another key example of modular programming. This function abstracts the process of loading textures from files, making it easy to apply textures to different objects. Rather than duplicating texture-loading code for each object, the function can be reused, ensuring that all textures are loaded consistently and efficiently.