Reflection

Mohamed Aziz Zaghdoudi

## Justification of Development Choices

For the development of my 3D scene, I selected objects that create a familiar and visually interesting environment. Specifically, I included: Lamp Post: The lamp post serves as a vertical element that adds height to the scene and provides a strong visual focal point. The lighting effects on the lamp post emphasize the interplay between light and shadow. Bench: The bench complements the lamp post and grounds the scene, providing a horizontal visual element. Together, they create a simple but effective street-side environment. Brick Wall: The brick wall acts as a background, adding depth and texture to the scene without overwhelming the user’s attention.

These choices were made to create a balanced and clean layout that is easy to navigate and visually pleasing.

### Programming for Required Functionality

To achieve the required functionality, I used OpenGL and GLSL shaders to render objects with appropriate materials and lighting. Objects were modeled as primitive shapes (Boxes, planes, and spheres) and assembled to form the lamp post and bench. Lighting: I implemented multiple light sources, including point lights and a spotlight for the lamp. This enhances realism by creating highlights and shadows. Projection Modes: I provided both orthographic and perspective views, allowing for versatile scene navigation and testing.

## Scene Navigation

Users can navigate the 3D scene interactively: Keyboard Controls: W, A, S, D move the camera forward, backward, left, or right. Q, E move the camera up and down. Arrow keys allow for rotation around the scene (pan and tilt). Mouse Input: Moving the mouse adjusts the camera's view direction using yaw and pitch angles. This simulates looking around in the 3D space. Zoom Functionality: The scroll wheel is used to adjust the field of view (FOV) in perspective mode or to scale the view in orthographic mode.

## Virtual Camera Setup

The virtual camera is implemented using a custom Camera class. The camera position, target, and up vector are managed using glm::lookAt() to calculate the view matrix. Two projection modes are supported: Perspective Projection, which provides a realistic 3D view, and Orthographic Projection, which ensures no perspective distortion, useful for technical views.

Input Devices: Keyboard for positional movements and rotations. Mouse for directional control (yaw and pitch). Scroll Wheel for zoom functionality. The Camera class maintains clean separation between camera movement and input handling, ensuring modularity.

## Custom Functions for Modularity

To make the code modular and organized, I created reusable custom functions that abstract repetitive logic and simplify scene management:

SetupSceneLights(): This function initializes all lighting parameters, including directional, point, and spotlight configurations. It reduces redundancy by centralizing light setup in one function. Reusability: I can call this function whenever I need to reset or update the lighting configuration.

DefineObjectMaterials(): This function sets up the materials for all objects in the scene, such as diffuse, specular, and shininess properties. Reusability: Adding new materials simply requires appending them to this function without modifying other parts of the code.

ProcessKeyboardEvents(): Handles all keyboard input for moving the camera and toggling projection modes. Reusability: This function can be extended to include additional input features without modifying the main loop.

RenderScene(): Combines the rendering of all objects into a single function, making the main loop cleaner and easier to manage. It iterates through a list of objects and applies the appropriate transformation and rendering logic.

## Conclusion

The design of my 3D scene, including the selection of objects, navigation system, camera controls, and custom functions, was guided by principles of clarity, simplicity, and modularity. Users can navigate the environment intuitively using keyboard and mouse inputs, and the code remains clean and reusable through well-organized custom functions.