# Development Choices and Justification for 3D Scene

## 1. Selected Objects and Design Choices

The objects selected for this 3D scene were chosen to create a realistic and visually engaging environment that effectively demonstrates various 3D graphics concepts. The scene includes the following objects:

* **Lamp**: The lamp serves as a complex object that utilizes two primitive shapes: a box for the base and a cylinder for the stand. This object showcases the ability to build more intricate models using basic shapes.
* **High Block**: Positioned at the center of the scene, this high block replaces the table and functions as a visually prominent structure. This choice demonstrates flexibility in creating both functional and aesthetic forms.
* **Additional Box**: An extra box is placed in the scene to add variety in shape while keeping the geometric structure simple. This object allows for an additional application of textures and colors, contrasting with the other objects.
* **Ball (Sphere)**: This round shape provides diversity in geometry and textures, showcasing both uniform scaling and realistic shading.

These objects were selected to satisfy the requirement of using multiple primitive shapes and different textures. Each shape and object reflects a conscious choice to keep the scene visually balanced while demonstrating functionality like texture application, lighting effects, and interaction.

## 2. Programming for Required Functionality

To program the required functionalities, I implemented several key features in the scene:

* **Texturing**: Two main textures are applied in the scene: one for the lamp and another for the block, with additional textures for variety. This setup meets the requirement of applying at least two unique textures to different objects.
* **Lighting**: Three light sources are implemented, with ambient, diffuse, and specular components, following the Phong shading model. The lights are positioned strategically to illuminate all objects without casting excessive shadows on any part of the scene. One of the lights is colored, meeting the requirement for color diversity in lighting.
* **Camera Control**: The camera allows users to explore the scene interactively, with control over movements such as panning, zooming, and rotating, which are covered in the following section.

## 3. Navigation and Camera Setup

The 3D scene is designed to be navigated interactively, giving users full control of the view. The navigation includes:

* **Mouse Controls**: The mouse is used for looking around the scene, allowing users to pan horizontally and vertically. This input is processed by capturing the change in cursor position and adjusting the camera’s orientation based on the delta in x and y coordinates.
* **Scroll Wheel**: Scrolling adjusts the camera’s zoom level, allowing for closer or more distant views of the objects in the scene. The scroll input increases or decreases the camera’s movement speed, enhancing the user’s control over navigation.
* **Keyboard Controls**: The keyboard supports WASD keys for movement (forward, backward, left, and right) and QE for vertical movement (up and down). This combination of inputs ensures users can navigate the scene comfortably and view the objects from various angles.

This camera setup, which combines mouse and keyboard inputs, gives the user an intuitive way to interact with the 3D environment, emulating real-world exploration.

## 4. Modularity and Custom Functions

To make the code modular and organized, I created specific functions that serve distinct purposes in managing the scene and its components:

* **SetTransformations**: This function is responsible for applying transformations, such as scaling, rotation, and translation, to any object in the scene. By passing in transformation parameters, this function centralizes all transformation-related logic, making it easy to apply and adjust transformations for any object. This approach also improves reusability, as the function can be applied to multiple objects without duplicating code.
* **SetShaderColor and SetShaderTexture**: These functions modularize the process of applying colors and textures to objects. SetShaderColor allows for easy color changes by passing RGBA values, while SetShaderTexture handles texture application by referencing texture tags. This separation simplifies object customization, ensuring that colors and textures are managed independently of transformations or mesh data.
* **DrawSphereMesh, DrawBoxMesh, DrawCylinderMesh**: These functions, part of the ShapeMeshes class, make the code more modular by allowing each shape to be drawn independently. Each function binds the appropriate VAO and uses the pre-loaded indices and vertices, allowing for efficient rendering of each shape with minimal code. This modularity supports flexibility in scene composition and makes adding new shapes straightforward.