## **Design Decisions**

### **Justification of Development Choices for the 3D Scene**

In my 3D scene development, I selected objects that are both realistic and recognizable in daily life, including:

* A **table** as the foundational plane to anchor all other objects.
* A **coffee cup** with a detailed handle to demonstrate **geometric modeling** using cylinders and tori.
* A **pencil** with a cone-shaped tip to highlight the use of scaling transformations for tapering.
* A **notebook** with pages and binder rings for added realism using box and torus meshes.
* A **keyboard** with aligned buttons, showcasing precise positioning through loops.

These objects were chosen because they:

1. Represent a **workspace environment**, relatable to many users.
2. Incorporate various shapes and transformations like scaling, rotation, and translation to showcase the use of **basic geometric primitives**.
3. Require textures (e.g., wood, stainless, notebook cover), demonstrating the integration of **texture mapping** and material properties.

The functionality was achieved through:

* **Custom materials** (e.g., wood, glass, metal) to simulate realistic object surfaces using ambient, diffuse, and specular reflections.
* **Lighting** with four light sources positioned strategically to illuminate all objects without casting shadows or creating unlit regions.
* **Texture mapping** using images like rusticwood.jpg for the table and notebook.jpg for the notebook cover.
* Modularized **transformation functions** (SetTransformations) to scale, rotate, and position objects efficiently.

### **Programming for Required Functionality**

The required functionality for the 3D scene includes:

1. **Dynamic Camera Control**: Implemented through user input (keyboard and mouse) to allow navigation, zooming, and switching between views.
2. **Custom Shaders**: Vertex and fragment shaders were used to calculate lighting and texture mapping dynamically.
3. **Modular Scene Rendering**: Functions like RenderScene, RenderNotebook, RenderCoffeeCup, and others were developed for individual objects to simplify rendering.

### **Navigation of the 3D Scene**

Users can navigate the 3D scene using the keyboard and mouse:

* **Keyboard Controls**:
  + **W/S**: Move the camera forward and backward.
  + **A/D**: Pan the camera left and right.
  + **Q/E**: Pan the camera up and down.
  + **1/2/3**: Switch between **orthographic views** (front, side, and top).
  + **4**: Return to the **perspective view**.
  + **ESC**: Exit the application.
* **Mouse Input**:
  + Mouse movement controls the **camera orientation** based on X and Y offsets.
  + This is achieved through the Mouse\_Position\_Callback function, which processes mouse events to update camera direction using offsets.

The **Camera class** implements the logic for movement and orientation, ensuring smooth navigation.

### **Virtual Camera Control Setup**

The virtual camera is set up using:

* The **GLFW library** for capturing input events.
* The **Camera class** to manage the camera's position, orientation, and zoom:

**g\_pCamera->ProcessKeyboard(FORWARD, gDeltaTime);**  
**g\_pCamera->ProcessMouseMovement(xOffset, yOffset);**

The camera supports both **perspective and orthographic projections**:

* **Perspective View**: Provides depth perception for realism.
* **Orthographic Views**: Display accurate dimensions for a front, side, or top view.

### **Custom Functions for Modularity and Reusability**

The program is structured using modular functions to ensure code reusability and maintainability. Key examples include:

1. **SetTransformations**:
   1. Scales, rotates, and positions an object in the scene using transformation matrices.
   2. **Reusability**: This function is used for all objects, such as the table, coffee cup, and notebook.
2. **RenderScene**:
   1. Manages the rendering of all scene objects by calling specific render functions.
   2. **Reusability**: It simplifies the main rendering loop and allows easy addition of new objects.
3. **Custom Object Render Functions**:
   1. Functions like RenderCoffeeCup, RenderNotebook, and RenderKeyboard encapsulate the logic for rendering individual objects.
   2. **Reusability**: These functions are modular, meaning any object can be updated independently without affecting others.
4. **LoadSceneTextures and DefineObjectMaterials**:
   1. These functions centralize the loading of textures and materials, ensuring consistent usage across the program.

### **Conclusion**

The modular structure, dynamic camera control, and careful selection of objects result in an organized, functional 3D scene that is easy to navigate. The use of reusable functions and classes allows for maintainable and scalable development.