**Project Design Decisions**

**I. Development Choices**

In designing the 3D scene, I aimed to replicate the selected 2D image (Figures 1 and 2) into an interactive computer graphics interface (illustrated in Figures 3 and 4), keeping relevant constraints in mind such as time to deliverable, project complexity, and system performance.

|  |  |
| --- | --- |
| **A lamp and candle on a shelf  Description automatically generated** | **A lamp and candle on a table  Description automatically generated** |
| **Figure 1: Front View of 2D Image** | **Figure 2: Side View of 2D Image** |

|  |  |
| --- | --- |
|  | **A computer screen shot of a lamp and candle  Description automatically generated** |
| **Figure 3: Front View of 3D Replication** | **Figure 4: Angled View of 3D Replication** |

To do this, I selected each object from the 2D image and made decisions on the replication and rendering of each object, including the composition of the shape(s) involved, sourcing of similar textures, accurate lighting and reflectivity of each object, and their positions relative to each other. Moreover, each object's rendering and the functionality for texturing, lighting, and camera movements (discussed in Section II) are well modularized, as detailed in Section III. This allows the user to navigate the 3D environment comfortably and with realistic photo approximation.

**II. 3D Scene Navigation**

The user may navigate the virtual scene with the following keyboard commands:

|  |  |
| --- | --- |
| W – zoom in | S – zoom out |
| A – pan left | D – pan right |
| Q – pan up | E – pan down |
| O – orthographic view | P – perspective view |

Additionally, the user may use the mouse to pan around the scene and the scroll wheel to change the speed at which the camera travels across the scene.

**III. Discussion on Code Modularity and Reusability**

In designing the codebase, steps were taken to modularize the creation of objects in the scene and to abstract away the low-level implementation of the shaders for vertex and fragment processing stages of the graphics pipeline. First, each object in the scene was assigned a unique function to render its relevant shapes, textures, materials, color, and position, allowing for code reusability and further expandability of the project. Moreover, the initialization of textures, materials, and lighting are encapsulated as functions and reused efficiently in the codebase’s *SceneManager* class. Vertex drawings for the shape meshes are defined in the *ShapeMeshes* class and are reused extensively in the *SceneManger* class. Overall, the heart of the creative rendering process occurs in the *SceneManager::PrepareScene()* and *SceneManager:: RenderScene()* methods, providing the developer with a bird’s eye view of the layers of implementation and abstraction, allowing for easy comprehensibility for further customization and development.