### **Reflection**

**9. Justify Development Choices for Your 3D Scene** The development of this 3D scene was guided by both aesthetic appeal and functional requirements. The scene includes a rustic well with wooden stands, a bucket, a tree, and a starry background. These objects were selected to evoke a serene outdoor setting, blending natural elements with a hint of structure.

* **Object Selection**:
  + The **well** acts as the focal point, combining stone and wood textures for realism and durability.
  + The **bucket**, a tapered cylinder, was chosen for its practical and aesthetic relevance, complementing the well’s design.
  + The **tree** adds depth and contrast to the scene, with its vibrant leaves texture and bark. It showcases the interaction of different textures and lighting in the scene.
  + The **starry sky** gives the scene a sense of time and place, creating a peaceful nighttime atmosphere.
* **Programming for Functionality**:
  + Textures were loaded using the CreateGLTexture function to ensure each object appears realistic.
  + Materials were defined in DefineObjectMaterials to control reflectiveness, shininess, and colors, enhancing the visual realism of each object.
  + Lighting was implemented in SetupSceneLights to highlight key elements (e.g., well lighting) while maintaining a balanced ambiance.

**10. How Users Navigate the 3D Scene** The user can navigate the 3D scene using keyboard and mouse inputs:

* **Keyboard Controls**:
  + W, A, S, D: Move the camera forward, backward, left, and right.
  + Q, E: Move the camera up and down.
  + K, L, I, M: Adjust the camera's view direction to look left, right, up, or down.
  + O, P: Switch between orthographic and perspective projection modes.
* **Mouse Controls**:
  + Mouse movement dynamically adjusts the camera's yaw and pitch, providing a smooth and intuitive viewing experience.

**Camera Setup**: The virtual camera was configured in the ViewManager class. It uses the Camera object for movement and interaction. The GetViewMatrix function returns the camera's current view matrix, updated with user input. This ensures fluid navigation.

**11. Custom Functions for Modularity** To keep the code organized and modular, several reusable functions were implemented:

* **CreateGLTexture**:
  + Loads and configures textures from image files.
  + Handles different image formats (RGB, RGBA).
  + Generates mipmaps for better texture scaling at various distances.
  + This function ensures new textures can be added without altering the rendering logic.
* **SetTransformations**:
  + Accepts scaling, rotation, and translation parameters to update the transformation matrix.
  + Simplifies object positioning and ensures consistent transformations for all objects.
* **SetShaderMaterial**:
  + Passes material properties (e.g., shininess, color) to the shader.
  + Abstracts the material setup, making it easier to apply different properties to new objects.
* **RenderScene and Render Functions**:
  + Functions like RenderBucket and RenderTrees encapsulate the rendering logic for specific objects.
  + This modular approach allows adding new objects with minimal impact on the main rendering workflow.