**Jacob Smith**

**CS330**

**Project Reflection**

Creating this 3D scene involved a series of deliberate choices and programming strategies to build an interactive and visually appealing environment. Here's a detailed explanation of my development approach:

**Development Choices**

**Object Selection**

* **Cylinder**: I chose a cylinder as it's a fundamental geometric shape that's more complex than a simple cube or sphere. Rendering a cylinder effectively demonstrates handling curved surfaces in OpenGL, which is crucial for more advanced 3D modeling.
* **Elongated Pyramid**: This shape adds an architectural element to the scene. It's not just a regular pyramid; its elongation requires different calculations for vertices and normals, showcasing more complex geometry handling.
* **Plane**: The plane serves as a base or ground, adding context to the other objects. It's also a simple geometry that contrasts with the more complex shapes, balancing the scene's complexity.

**Lighting and Textures**

* **Dual Light Sources**: Using two light sources, each positioned differently, creates varied lighting effects, highlighting the shapes' dimensions and textures. This approach demonstrates how light positioning affects the mood and perception of a 3D space.
* **Texture Mapping**: The texture on the cylinder and wood texture on the plane and pyramid were chosen for their visual impact and to illustrate texture mapping techniques. These textures provide a more realistic appearance and demonstrate how textures can enhance the visual depth of objects.

**Shader Programming**

* **Custom Vertex and Fragment Shaders**: Writing these shaders from scratch allowed me to control how the objects are rendered at a granular level. This includes handling transformations, calculating lighting (ambient, diffuse, specular), and applying textures. Custom shaders are essential for fine-tuning the visual output and are a cornerstone of 3D graphics programming.

**User Navigation**

**Keyboard and Mouse Controls**

* **WASD for Movement**: This setup is a standard in 3D navigation, providing an intuitive way for users to move forward, backward, and sideways in the scene.
* **Q/E for Vertical Movement**: These keys add another dimension of movement, allowing the user to move up or down, which is particularly useful for exploring 3D scenes.
* **Mouse for Viewing Direction**: The mouse controls the camera's yaw and pitch, allowing users to look around smoothly. This is crucial for a fully interactive 3D experience.

**Projection Toggle**

* **Perspective vs. Orthographic**: The 'P' key toggles between these two projections. Perspective projection offers a realistic view, while orthographic projection provides a more technical, blueprint-like view. This feature demonstrates the impact of projection types on 3D visualization.

**Custom Functions**

**Geometry Generation**

* **generateCylinderData, generateElongatedPyramidData, generatePlaneData**: These functions are dedicated to creating the vertex and index buffers for each object. They encapsulate the logic for defining the shapes, making the main code cleaner and more readable. This modular approach also allows for easy adjustments and reuse in different contexts.

**Camera and Interaction**

* **processInput**: This function centralizes all the keyboard input handling, making the main loop cleaner and more manageable. It's a key part of making the camera and scene interactive.
* **mouse\_callback, scroll\_callback**: Handling mouse movement and scroll events, these functions update the camera's orientation and speed. They are crucial for creating a responsive and intuitive camera control system.

In summary, each element of this 3D scene, from the objects chosen to the interaction mechanisms, serves a specific purpose. The objects demonstrate both simple and complex geometry handling, the lighting and textures add realism and depth, and the shaders provide fine control over the rendering. The user interaction, facilitated through keyboard and mouse controls, makes the scene dynamic and engaging. All these aspects combined showcase the capabilities and intricacies of OpenGL and GLSL in creating interactive 3D graphics.