3D Interactive Object Viewer with Transformations
Project Report

1. Project Objective and Methodology

Objective:
Develop an interactive 3D viewer application that allows users to manipulate a 3D object (cube) through transformations (translation, rotation, scaling) while demonstrating core computer graphics concepts, including:

- Perspective projection
- Lighting and material properties
- Texture mapping
- Depth buffering
- User interaction

Methodology:

1. **Core Framework**:
   - Used **Python** with **PyOpenGL** for OpenGL bindings and **Pygame** for window management and input handling.
   - Implemented a modular structure separating geometry definitions, transformations, rendering, and user input.
2. **Key Features**:
   - **Transformations**: Applied matrix operations for translation, rotation, and scaling.
   - **Lighting**: Configured ambient, diffuse, and specular lighting for realism.
   - **Textures**: Integrated texture mapping with fallback to procedural patterns.
   - **User Interaction**: Enabled mouse/keyboard controls for real-time manipulation.
3. **Workflow**:
   - Defined vertex data for a cube.
   - Initialized OpenGL context with Pygame.
   - Implemented rendering pipeline with perspective projection.
   - Added lighting and material properties.
   - Integrated texture loading and error handling.
   - Developed event loops for user input.

---

2. Implementation Details

Libraries Used:

Library	**Purpose**
Pygame	Window management, input handling
PyOpenGL	OpenGL bindings for 3D rendering
OpenGL.GLU	Utilities (e.g., perspective projection)

Code Structure:

1. **Geometry Definitions**:
   - **Vertices**: 8 points defining cube corners.
   - **Edges**: 12 line pairs for wireframe rendering.
   - **Surfaces**: 6 quad faces for solid rendering.
2. **Transformation Variables**:

```
translate = [0, 0, -5]  # Initial position
rotate = [0, 0, 0]      # Rotation angles (x, y, z)
scale = 1.0             # Uniform scaling factor
```

3. **Rendering Pipeline**:
   - **Projection**: gluPerspective(45, 800/600, 0.1, 100.0)
   - **Lighting**: Configured GL_LIGHT0 with ambient, diffuse, and specular components.
   - **Materials**: Set material properties for texture/color modes.
4. **Texture Handling**:
   - **Fallback**: Generated checkerboard pattern if texture file is missing.
   - **Mipmapping**: Used gluBuild2DMipmaps for smoother texture scaling.
5. **User Input**:
   - **Mouse**: Drag to rotate, wheel to zoom.
   - **Keyboard**: Arrow keys for translation, X/Y/Z for rotation, +/- for scaling.

---

3. Challenges and Solutions

Challenge 1: Texture Visibility Issues

- **Problem**: Textures appeared dark or disappeared during rotations.
- **Solution**:

```
glMaterialfv(GL_FRONT, GL_AMBIENT, (1.0, 1.0, 1.0, 1.0))
glMaterialfv(GL_FRONT, GL_DIFFUSE, (1.0, 1.0, 1.0, 1.0))
```

  - Increased ambient light intensity to 0.7.

Challenge 2: Transformation Order

- **Problem**: Incorrect translation/rotation order caused erratic movement.
- **Solution**: Enforced transformation sequence:

```
glTranslatef(*translate)
glRotatef(rotate[0], 1, 0, 0)  # X-axis
glRotatef(rotate[1], 0, 1, 0)  # Y-axis
glRotatef(rotate[2], 0, 0, 1)  # Z-axis
glScalef(scale, scale, scale)
```

Challenge 3: Depth Artifacts

- **Problem**: Surfaces flickered due to incorrect depth sorting.
- **Solution**: Enabled depth testing and buffering:

```
glEnable(GL_DEPTH_TEST)
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
```

Challenge 4: Input Responsiveness

- **Problem**: Mouse rotation felt sluggish.
- **Solution**: Scaled mouse delta values for smoother rotation:

```
rotate[0] += dy * 0.5  # Scale mouse movement
rotate[1] += dx * 0.5
```

Challenge 5: Texture Loading Failures

- **Problem**: Missing textures crashed the application.
- **Solution**: Added fallback to procedural checkerboard texture:

```
 except pygame.error:
    texture_surface = create_checkerboard_texture()
```

4. Conclusion

The project successfully demonstrates core computer graphics concepts through an interactive 3D viewer. Key achievements include:

- Robust transformation handling with mouse/keyboard input.
- Realistic lighting and texture rendering.
- Error-resilient texture loading.
- Optimized rendering pipeline at 60 FPS.

Future enhancements could include:

- Loading complex 3D models (OBJ files).
- Implementing Phong shading.
- Adding multiple light sources.
- Supporting texture blending modes.

This implementation serves as a foundational framework for more advanced 3D graphics applications.