**OpenGL Interactive Scene Implementation Report**

**1. Introduction** This report provides an in-depth explanation of an interactive 3D scene created using OpenGL. The scene incorporates multiple rendering techniques such as Blinn-Phong lighting, texture mapping, normal mapping, and environment mapping. Additionally, it features a torus model loaded from an OBJ file and a procedurally generated wooden crate. Users can navigate the scene using a first-person camera and interact with objects using keyboard controls. This project was both a technical challenge and an opportunity to deepen my understanding of modern OpenGL rendering techniques.

**2. Scene Overview** The scene is set inside a room with textured stone walls and a stone-tiled floor. It includes two primary objects: a reflective torus and a wooden crate. The torus model is loaded using the ASSIMP library, while the crate is created procedurally as a textured cube. The scene is illuminated by a directional light and a point light, both of which contribute to realistic shading effects. To enhance realism, normal mapping is applied to the walls and floor, making the surface appear more detailed than it actually is.

**3. Graphics Techniques Implemented**

### 3.1 Blinn-Phong Lighting Model

Lighting is one of the most important aspects of realistic rendering. In this project, the Blinn-Phong lighting model is used to simulate light interactions on different surfaces. The model consists of three main components:

* **Ambient Lighting**: A constant light that simulates indirect illumination.
* **Diffuse Lighting**: Light that depends on the angle between the light source and the surface normal.
* **Specular Lighting**: Highlights on shiny surfaces based on the viewer's position.

Blinn-Phong is an improvement over the classic Phong shading model, as it uses a halfway vector to calculate specular reflections, reducing computational cost and improving performance.

### 3.2 Lighting Sources

The scene contains two types of lights:

* **Directional Light**: This light simulates sunlight or a far-away light source. It has a fixed direction and affects all objects in the scene uniformly.
* **Point Light**: A local light source with attenuation, meaning its intensity decreases with distance. The point light in the scene is dynamic and can move when toggled using the spacebar.

To implement these lights, the vertex and fragment shaders process light calculations per pixel, ensuring a smooth and realistic effect.

### 3.3 Texture Mapping and Normal Mapping

Texture mapping is used to apply images onto 3D surfaces, giving them a realistic appearance. In this project, the stone walls, floor, and wooden crate use texture mapping to add detail.

Normal mapping is a more advanced technique that enhances surface details without increasing geometry complexity. Instead of modifying the actual shape of an object, normal maps store surface variations in a texture, affecting how light interacts with the surface. This method makes the stone walls and floor appear more rugged and uneven without adding extra vertices.

The implementation involved:

* Loading texture images using the stb\_image library.
* Generating texture coordinates in the shader.
* Using a normal map to perturb surface normals and modify lighting calculations.

### 3.4 Environment Mapping

One of the most visually striking elements in the scene is the reflective torus, which utilizes environment mapping. This technique creates a mirror-like effect by sampling a cubemap texture. The cubemap represents the surrounding environment and is used to determine reflected colors based on the view direction.

The environment mapping effect is adjustable:

* Pressing the + and - keys increases or decreases reflection intensity.

This feature was implemented using a combination of:

* A cubemap texture.
* Reflection vector calculations in the fragment shader.
* Dynamic intensity adjustment through uniform variables in OpenGL.

### 3.5 Camera System

The scene uses a first-person roaming camera controlled via keyboard and mouse. The movement mechanics include:

* **W/S**: Move forward/backward.
* **A/D**: Strafe left/right.
* **Q/E**: Move up/down.
* **Mouse Drag (Right Click)**: Adjust camera orientation.

The camera is implemented using a view matrix that updates based on user input. Mouse movements alter the direction vector, allowing smooth-looking transitions.

**4. Object Interactions** The project includes several interactive features to make the scene more dynamic:

* **Torus Rotation**: The left and right arrow keys adjust the rotation speed of the torus.
* **Environment Mapping Control**: Pressing + or - changes reflection intensity.
* **Point Light Animation**: Pressing the spacebar toggles point light movement.

These interactions were achieved by modifying shader uniform values and updating object transformations based on input events.

**5. Challenges and Learning Experience** This project was an excellent learning experience, but it also came with several challenges:

* **Shader Debugging**: One of the most difficult parts was debugging shaders, especially normal mapping. Incorrect normal transformations initially caused lighting artifacts.
* **Camera Control**: Implementing smooth and intuitive first-person camera movement required quaternion-based rotations and careful handling of mouse input.
* **Performance Optimization**: Using multiple light sources and normal mapping increased computational cost. To optimize performance, I minimized texture fetches and reduced unnecessary calculations in fragment shaders.

Despite these challenges, I gained valuable insights into OpenGL’s rendering pipeline, matrix transformations, and shader programming.

**6. Conclusion and Future Improvements** This project successfully demonstrates advanced OpenGL rendering techniques, interactive features, and real-time object manipulation. However, there are several areas for improvement:

* **Shadow Mapping**: Adding shadows would significantly enhance realism.
* **Physics-Based Interactions**: Implementing collision detection and response would improve interactivity.
* **More Dynamic Scene Elements**: Adding animated objects like moving doors or interactive buttons would make the scene more engaging.

Overall, this project provided a solid foundation in OpenGL programming, and I am excited to continue exploring more advanced techniques in future projects.