Rajshahi University of Engineering and Technology

Department of Computer Science and Engineering

Assignment Report

Course No: CSE 4201

Course Title: Computer Graphics and Animations

Project Title: 3D Interactive Scene using OpenGL

Submitted by:

Raiyan Zannat

Roll: 1903073

Section: B

Submitted to:

Md. Sozib Hossain

Lecturer

Dept of CSE, RUET

Date: 21 March, 2025

Introduction

The "3D Interactive Scene using OpenGL" project is a practical implementation of 3D graphics programming, utilizing OpenGL to render a scene containing a cube, pyramid, and sphere, enhanced with dynamic lighting, shadows, and a skybox background. Developed in C++ with the GLFW library for window management and GLAD for OpenGL function loading, this project demonstrates core graphics techniques such as vertex and fragment shading, texture mapping, and real-time user interaction. The scene includes a textured cube, color-differentiated pyramid and sphere, and a ground plane to display shadows, all set within a skybox environment. This report details the implementation process, user interaction features, and challenges faced, along with their resolutions, providing a comprehensive overview of the development experience.

Implementation

1. Setup OpenGL Environment:

- The environment was initialized using GLFW to establish a window with a modern OpenGL context.
- GLAD was incorporated to dynamically load OpenGL functions, ensuring access to current extensions.
- The setup was configured to support a 3.3 core profile, providing a robust foundation for advanced rendering features.

2. 3D Object Creation:

- The cube and pyramid were designed with predefined vertex data, including positions and normals for accurate rendering.
- The sphere was generated procedurally using a parametric method with adjustable sector and stack divisions for smoothness.
- VAOs and VBOs were utilized to efficiently store and manage vertex data, enhancing rendering performance.

3. Transformations:

- Model matrices were constructed to apply translation, rotation, and scaling to each object independently.
- View and projection matrices were implemented to transform the scene into camera and screen space with perspective.
- The transformation system was designed with flexibility to support dynamic adjustments of object properties.

4. Lighting and Shading:

- A point light source was positioned to illuminate the scene, contributing to realistic lighting effects.
- The Phong shading model was adopted, calculating ambient, diffuse, and specular components per fragment.
- Shadow mapping was integrated using a depth-based approach, projecting shadows onto a ground plane.

5. Textures and Colors:

- A texture was applied to the cube using an external image file, mapped with UV coordinates in the shader.

- The pyramid was assigned vertex colors with a mix of red, green, and blue for visual distinction.
- The sphere was given a uniform purple color, ensuring a clear contrast with other objects.

6. Camera Control:
- A custom view matrix was developed to define the camera's position and orientation within the scene.
- A perspective projection matrix was set up with a specified field of view and clipping planes for proper rendering.
- The camera system was designed to allow dynamic adjustments, ensuring visibility of all scene elements.

7. Bonus Features:
- A background was implemented as a large sphere with an equirectangular environment map, rendered to simulate a background environment.
- Shadow mapping was enhanced with a depth map technique, rendering the scene from the light's perspective.
- An animation effect was added to the cube, enabling continuous rotation around its vertical axis.

User Interaction

The application provides intuitive controls for users to interact with the 3D scene, allowing for both camera navigation and object manipulation:

Camera Control:

- Movement: Users can navigate the scene using the `W` key to move forward, `S` to move backward, `A` to strafe left, `D` to strafe right, `Q` to move up, and `E` to move down. The movement speed is scaled by the frame time to ensure consistent navigation across different hardware.

Rotation:

Mouse movement adjusts the camera's yaw and pitch, with the cursor locked to the window for smooth first-person control. The sensitivity is set to 0.1, and the pitch is clamped between -89° and 89° to prevent the camera from flipping upside down, providing an intuitive way to look around the scene.

Object Selection and Manipulation:

- Selection: Users can select which object to manipulate by pressing `1` for the cube, `2` for the pyramid, or `3` for the sphere, enabling focused control over individual objects.
- Translation: The selected object can be moved using `I` to translate upward, `K` downward, `J` to the left, `L` to the right, `U` backward, and `O` forward, allowing precise positioning within the 3D space.
- Rotation: Object rotation is controlled with the `Up` and `Down` arrow keys for pitch, `Left` and `Right` arrow keys for yaw, `P` for positive roll, and `;` for negative roll, enabling full rotational control to adjust the object's orientation.
- Scaling: The mouse scroll wheel adjusts the selected object's scale, with scrolling up to increase size and down to decrease, constrained to a minimum scale of 0.1 to prevent the object from disappearing.

Exit:

Pressing the `Esc` key closes the application, providing a straightforward way to exit the program.

These controls enable users to explore the scene comprehensively, manipulate objects dynamically, and observe the effects of lighting and shadows in real-time.

Challenges & Solutions

Several challenges were encountered during the development process, each requiring specific solutions:

Rotating Background:

During the development of the 3D scene, the background, designed as a large sphere with an environment map, was initially rotating continuously, creating a distracting effect that reduced the scene's realism. This was caused by an unintended automatic rotation, which was resolved by disabling the feature, ensuring a stable and fixed background.

Texture Overlap:

The cube’s texture was unintentionally applied to the sphere and pyramid during certain rendering conditions. This occurred because the texture sampling logic was not object-specific. A uniform boolean was introduced in the fragment shader to control texture application, ensuring that only the cube uses the texture while other objects render with their assigned vertex colors.

Shadow Visibility:

Shadows were not visible initially, reducing the realism of the lighting effects. This issue stemmed from an improper light position and lack of a surface to receive shadows. The light position was adjusted to a more suitable location, and a ground plane was added at a fixed height below the objects to catch and display the shadows. The shadow calculation was also fine-tuned to minimize artifacts like shadow acne.

Sphere Coloring:

The sphere initially appeared gray, which clashed with the pyramid’s vibrant red, green, and blue colors, making the scene less visually appealing. To address this, the sphere’s vertex colors were set to a uniform purple, providing a distinct and vibrant appearance that complemented the other objects while maintaining visual clarity.

Conclusion

The "3D Interactive Scene using OpenGL" project successfully implements a dynamic 3D environment with a cube, pyramid, and sphere, enhanced by lighting, shadows, and a skybox using OpenGL, GLFW, and GLAD in C++. It demonstrates key graphics techniques like shading, texture mapping, and user interaction, with intuitive controls for camera navigation and object manipulation. Challenges such as skybox positioning, texture overlap, shadow visibility, and sphere coloring were effectively resolved, improving the scene's visual fidelity. Future improvements could include multiple light sources, dynamic skybox textures, or additional object types. Overall, it offers a practical and engaging exploration of OpenGL's capabilities in creating immersive 3D scenes.