DHAKA INTERNATIONAL UNIVERSITY

Department of Computer Science & Engineering

Lab Report

Course Name : Computer Graphics & Multimedia Lab

Course Code : CSE-406

Report No. : 03

Report Name : Implementing 3D Camera, Projection, Lighting, and Rendering a Table using OpenGL (Python + GLFW).

SUBMITTED BY	SUBMITTED TO
NAME : Md. Soumik Hasan **ROLL : 34** **BATCH : D-72**	**Umme Niraj Mahi** **Lecturer** **Department of CSE** **Dhaka International University**

DATE OF SUBMISSION : 02-12-2025

Objective

The objective of this experiment is to implement an interactive 3D graphics environment using OpenGL where a table-shaped 3D model is rendered with realistic lighting effects. The goals include:

- Implementing a fully functional camera system supporting movement and mouse-based rotation.
- Creating a perspective projection matrix and applying Model–View–Projection (MVP) transformations.
- Designing and rendering a 3D table using cuboids.
- Applying Phong lighting with both point light and spotlight support.
- Understanding OpenGL buffers (VAO, VBO), shaders, object modeling, and the rendering pipeline.
- Gaining practical experience in writing GLSL shaders and handling real-time graphics.

Introduction

Computer Graphics revolves around generating, transforming, and displaying visual content using mathematical models and rendering algorithms. OpenGL is an API that exposes GPU functions to developers for real-time rendering.

In this experiment, OpenGL is used via Python (PyOpenGL) alongside GLFW for creating the window and handling input. The project implements a complete 3D environment including:

- A camera system based on yaw-pitch angles.
- A perspective projection to simulate realistic depth.
- Modeling of a table using several cuboids.
- Phong Reflection Model for realistic lighting (ambient, diffuse, specular).
- Point Light and Spotlight that illuminate the object.

This project demonstrates how real-time 3D graphics systems work internally—how matrices transform objects, how lighting produces realism, and how user input manipulates the scene.

Implementation

Tools & Technologies

- **Python 3**
- **OpenGL (PyOpenGL)** – For rendering the 3D scene
- **GLFW** – For window creation, keyboard/mouse input
- **NumPy** – For matrix and vector operations
- **GLSL** – Shader programming language

This combination allows rapid prototyping while giving full access to GPU rendering.

OpenGL Buffers Used (VAO, VBO)

Vertex Buffer Object (VBO)

Stores vertex attributes such as:

- position
- normals
- color

It is uploaded to GPU memory for fast rendering.

Vertex Array Object (VAO)

Keeps track of which VBO corresponds to which attribute index in the shader.

You specify attribute layout:

- Location 0 → Position
- Location 1 → Normal
- Location 2 → Color

This structure ensures the shader receives correct data for each vertex.

Camera & View System

Camera Position & Orientation

- Maintains vectors: `position`, `front`, `up`, `right`
- Uses yaw and pitch to determine rotation
- Mouse movement updates the yaw/pitch → updates direction vector

Keyboard Movement

Supports:

- WASD - Move forward/left/backward/right
- Mouse - Look around
- Space - Move up
- Shift - Move down
- ESC - Exit

Movement speed scaled by `delta_time` ensures smooth motion independent of FPS.

Projection Matrix (Perspective)

A custom function `create_perspective_matrix()` is used:

- Simulates real-world depth
- Makes closer objects larger and distant objects smaller
- Uses field-of-view (FOV), aspect ratio, near plane, and far plane

This matrix is passed to the shader and applied during rendering.

Modeling the Table (Cuboid Construction)

The table model is built by combining multiple cuboids:

- One large cuboid → Table Top
- Four vertical cuboids → Table Legs

Each cuboid is constructed using:

- Center coordinates
- Size (sx, sy, sz)
- RGB color
- Normals for each face (for lighting)

Every cuboid contains 36 vertices (6 faces × 2 triangles × 3 vertices).

This modular approach makes it easy to create any shape using simple building blocks.

Shader Programming (GLSL)

The program uses two shaders:

1. **Vertex Shader Highlights**
    - Applies Model, View, Projection transformations
    - Computes world position and normals
    - Passes vertex color and transformed normals to fragment shader
2. **Fragment Shader Highlights**

    Implements Phong lighting:

    - Ambient
    - Diffuse
    - Specular

Supports:

1. **Point Light**
2. **Spotlight with smooth edges**
3. **Attenuation** for realistic light fall-off

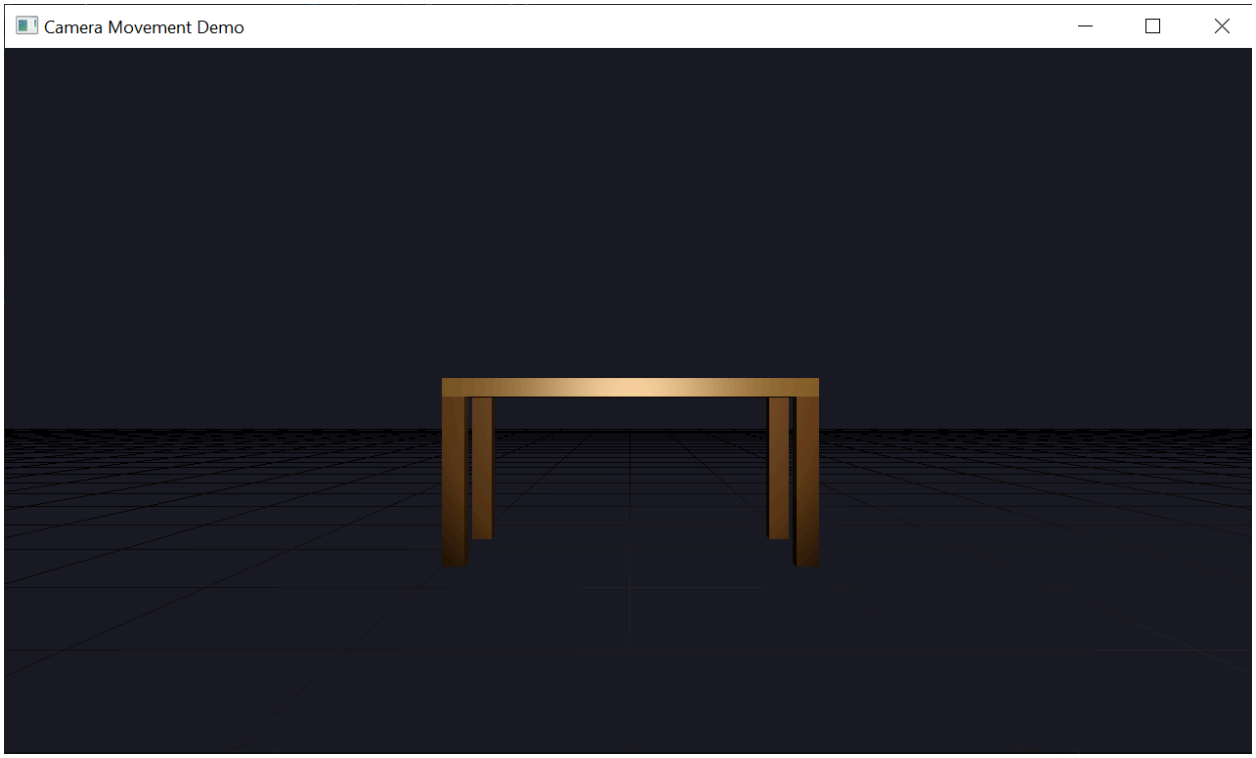
If no light is active, the object displays raw vertex colors.

How Everything Works Together (Pipeline Summary)

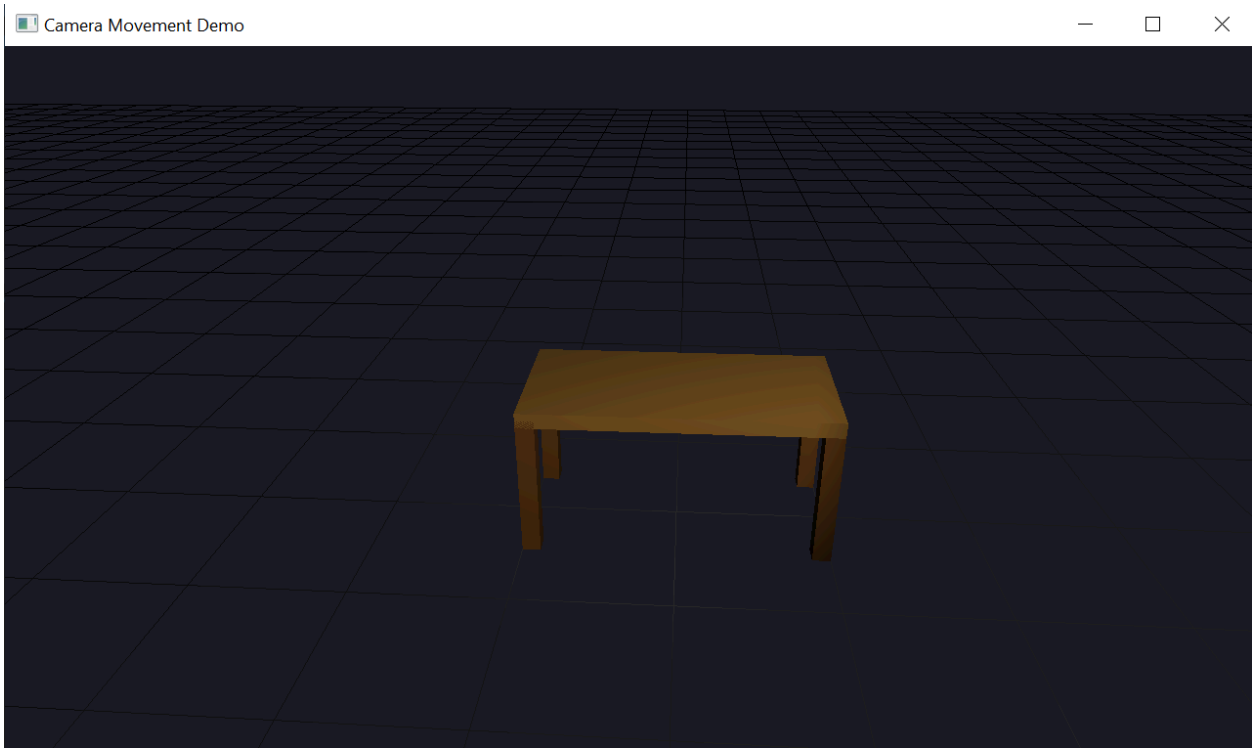
1. Create Window using GLFW
2. Initialize OpenGL
3. Compile Shaders & Activate Program
4. Load Table Vertex Data into VBO, Configure VAO
5. Set up Camera, Projection, Lighting
6. Main Rendering Loop:
    - Clear screen
    - Update camera from input
    - Create model matrix
    - Pass uniforms to shader
    - Draw all cuboids
7. Swap Buffers for real-time animation

This loop runs continuously until the user closes the window.

Output


Camera Movement Demo


Camera Movement Demo



5. Conclusion

In this experiment, I gained a comprehensive understanding of how modern 3D rendering works—from geometry creation to real-time interaction. Implementing the camera movement gave me practical insights into how yaw-pitch angles generate directional vectors, while writing the projection matrix deepened my knowledge of how perspective transforms simulate human vision in 3D graphics. Building the table model taught me how complex objects can be formed from simple cuboids using correct vertex, color, and normal data. The Phong lighting implementation showed how ambient, diffuse, and specular components together create realistic shading and how point and spotlights behave differently. Overall, this project strengthened my understanding of OpenGL's pipeline, shader programming, and the math behind 3D transformations, giving me hands-on experience in creating an interactive and visually accurate 3D scene.