Heaven’s Light is Our Guide

Rajshahi University of Engineering & Technology

Department of Computer Science & Engineering

Assignment Report

Course Code: CSE 4201

Course Title: Computer Graphics and Animations

Submitted T0:

Md. Sozib Hossain

Lecturer

Department of Computer Science & Engineering

Submitted by:

S M Golam Rifat

Roll: 1903113

Department of Computer Science & Engineering

OpenGL Assignment: 3D Interactive Scene

Introduction

This project implements a 3D graphics application using OpenGL and GLUT, fulfilling the requirements of rendering multiple 3D objects, applying transformations, lighting, textures, and camera controls. The scene includes a cube, sphere, and pyramid, each with distinct colors and textures, positioned on a textured ground plane under a simple sky. The application supports user interaction for object manipulation and camera movement, along with automatic object animations, demonstrating core concepts of computer graphics such as rendering, shading, and real-time interaction.

The primary goal was to create an interactive 3D environment that showcases OpenGL's capabilities while meeting the assignment's specifications: setting up an OpenGL environment, rendering 3D objects, implementing transformations, adding lighting and shading, applying textures, enabling camera control, and incorporating bonus features like animations and a background environment.

Implementation

The project utilizes OpenGL with GLUT to create an interactive 3D scene featuring a cube, sphere, and pyramid. Below is an explanation of the key OpenGL techniques implemented, supported by relevant code snippets from the application.

1. OpenGL Environment Setup

I successfully installed GLUT in my environment. GLUT handles window creation and event management. The main function initializes a double-buffered window with depth testing, and the init function configures the rendering environment.

2. 3D Object Creation

Three objects are rendered using immediate mode (***glBegin/glEnd***) instead of VAOs/VBOs due to simplicity. For example, the cube is drawn with textured quads:

```
void drawCube() {
    glEnable(GL_TEXTURE_2D);
    glBindTexture(GL_TEXTURE_2D, cubeTexture);
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
    glColor3f(1.0f, 0.0f, 0.0f);
    glBegin(GL_QUADS);

    glNormal3f(0.0f, 0.0f, 1.0f);
    glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f, -1.0f, 1.0f);
    glTexCoord2f(1.0f, 0.0f); glVertex3f(1.0f, -1.0f, 1.0f);
    glTexCoord2f(1.0f, 1.0f); glVertex3f(1.0f, 1.0f, 1.0f);
    glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f, 1.0f, 1.0f);

    glEnd();
    glDisable(GL_TEXTURE_2D);
```

```
}
```

3.Transformations

Transformations are applied in the render function using ***glPushMatrix*** and ***glPopMatrix*** to isolate changes per object:

```
glPushMatrix();
glTranslatef(cubeX, cubeY, cubeZ);
glRotatef(cubeRotX, 1.0f, 0.0f, 0.0f);
glRotatef(cubeRotY, 0.0f, 1.0f, 0.0f);
glScalef(cubeScale, cubeScale, cubeScale);
drawCube();
glPopMatrix();
```

Translation, rotation, and scaling are controlled via global variables updated by **user input.**

4.Lighting and Shading

A point light source is set up with ambient, diffuse, and specular components, using smooth shading:

```
void setupLighting() {
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    float ambient[] = {0.2f, 0.2f, 0.2f, 1.0f};
    float diffuse[] = {1.0f, 1.0f, 1.0f, 1.0f};
    float specular[] = {1.0f, 1.0f, 1.0f, 1.0f};
    glLightfv(GL_LIGHT0, GL_AMBIENT, ambient);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuse);
    glLightfv(GL_LIGHT0, GL_SPECULAR, specular);
    glLightfv(GL_LIGHT0, GL_POSITION, lightPos);
    glShadeModel(GL_SMOOTH);
    glEnable(GL_NORMALIZE);
    glEnable(GL_COLOR_MATERIAL);
    glColorMaterial(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE);
}
```

The light position is updated in render to remain fixed relative to the camera.

5.Textures and Colors

The cube uses a self-generated chessboard texture:

```
void generateCubeTexture() {
    glEnable(GL_TEXTURE_2D);
    glGenTextures(1, &cubeTexture);
    glBindTexture(GL_TEXTURE_2D, cubeTexture);
    const int width = 32, height = 32;
    unsigned char textureData[width][height][3];
    for (int i = 0; i < width; i++) {
        for (int j = 0; j < height; j++) {
            if ((i / 4 + j / 4) % 2 == 0) {
                textureData[i][j][0] = 0;
                textureData[i][j][1] = 0;
                textureData[i][j][2] = 0;
            } else {
                textureData[i][j][0] = 255;
                textureData[i][j][1] = 255;
```

```
                    textureData[i][j][2] = 255;
                }
            }
        }
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB,
GL_UNSIGNED_BYTE, textureData);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
}
```

The sphere and pyramid use gradient coloring (blue and orange, respectively).

6.Camera Control

The camera is implemented with ***gluLookAt***, adjustable via keyboard:

```
void render() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    gluLookAt(eyeX, eyeY, eyeZ, focusX, focusY, focusZ, 0.0f, 1.0f,
0.0f);

}
```

Variables eyeX, eyeY, eyeZ (camera position) and focusX, focusY, focusZ (look-at point) are modified in handleKeyboard.

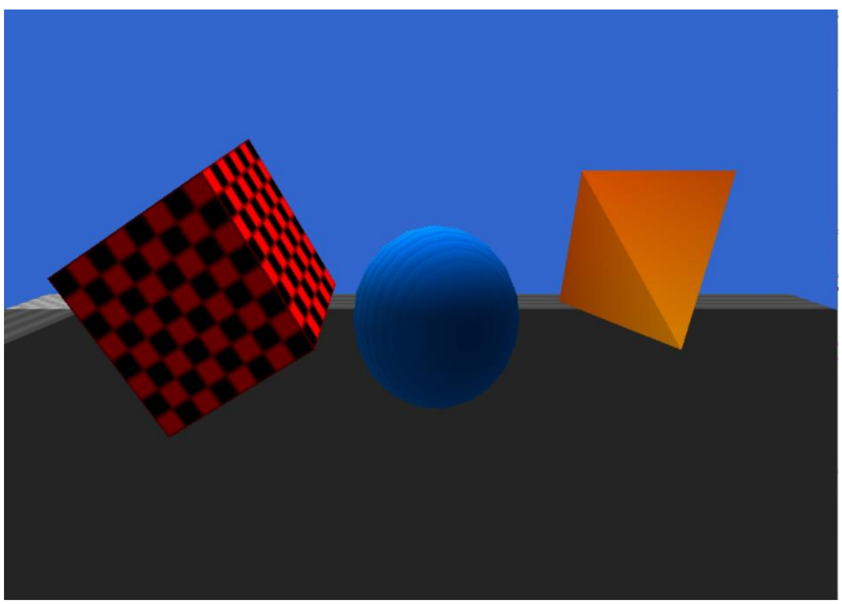

Fig 1. Final Result

7.Bonus Features

Objects animate automatically (cube rotates on X and Y axes, sphere on Y, pyramid on X) using a timer function. A simple sky quad and textured ground provide a background environment.

User Interaction

The application offers extensive keyboard-based controls:

- **Cube**: Move (q, e, w, s, a, d), rotate (r, t), scale (f, g).
- **Sphere**: Move (u, o, i, k, j, l), rotate (p, [), scale (h, n).
- **Pyramid**: Move (1, 3, 2, 5, 4, 6), rotate (7, 8), scale (9, 0).
- **Camera**: Move eye point (z, x, c, v, b, y), pan focus (;, ',], \).
- **Exit**: Press ESC.

These controls allow users to reposition objects, adjust their orientation and size, and explore the scene from different perspectives, enhancing interactivity.

Challenges & Solutions

1. **VAOs/VBOs Not Used**: The assignment required VAOs and VBOs for efficient rendering, but I faced difficulties integrating them with GLUT due to limited modern OpenGL experience. **Solution**: I reverted to immediate mode, acknowledging this as a limitation to address in future iterations by studying buffer objects further.
2. **Texture Generation**: Loading external textures was complex, so I opted for procedural generation. **Solution**: I created chessboard patterns programmatically for the cube and ground, ensuring compliance with the texture requirement.
3. **Camera Movement**: Early camera controls felt unintuitive. **Solution**: I separated eye and focus point adjustments, providing finer control over the view.
4. **Performance**: Animations caused stuttering on slower machines. **Solution**: I set a 16ms timer (~60 FPS) and kept geometry simple to maintain smooth rendering.

Conclusion

This project successfully demonstrates fundamental OpenGL techniques, including object rendering, transformations, lighting, texturing, and user interaction. While it meets most requirements, the absence of VAOs/VBOs highlights an area for improvement. The addition of animations and a basic environment enhances the visual experience.