Lab Report

Course Code: CSE 422

Course Title: Computer Graphics Lab

Report No: 10

Title: Drawing different 3D objects using OpenGL primitives with 3D transformations (translation, rotation, and scaling).

Submitted To:

Syada Tasmia Alvi

Senior Lecturer

Department of Computer Science and Engineering (C.S.E)

Submitted by:

Name: Mahmudul Hasan Piash

ID: 221-15-5606

Section: 61_D1

Department of Computer Science and Engineering (C.S.E)

Date of submission: 15th December, 2025

Title

Drawing different 3D objects using OpenGL primitives with 3D transformations (translation, rotation, and scaling).

Introduction

In this project, multiple 3D objects have been created using OpenGL primitive shapes such as cubes, pyramids, tetrahedrons, and spheres. The program demonstrates the application of 3D Transformations including translation, rotation, and scaling to manipulate these objects interactively. Multiple helper functions were written to draw reusable 3D objects. This project shows how fundamental 3D geometric shapes can be rendered and transformed in real-time using OpenGL.

Contents

The following functions and shapes have been used in this project:

1. drawCubeFromPoints(x1, y1, z1, x2, y2, z2)

    - Draws a cube using two diagonal corner points.
    - Uses GL_QUADS to draw all six faces with different colors.
    - Each face has 4 vertices specified in counter-clockwise order.

2. drawPyramid(x1, y1, z1, x2, y2, z2)

    - Draws a pyramid with a rectangular base and triangular sides.
    - Uses GL_TRIANGLES for all four triangular faces and the base.
    - The apex is calculated as the center point at the top.

3. drawTetrahedron(float size)

    - Draws a tetrahedron with four triangular faces.
    - Uses GL_TRIANGLES for all four faces with different colors.
    - Each face is a triangle connecting three vertices.

4. drawSphere(float radius, int slices, int stacks)

    - Draws a sphere using latitude and longitude segments.
    - Uses GL_QUAD_STRIP to create smooth curved surfaces.
    - Radius, slices, and stacks control the sphere's detail level.

5. 3D Transformations Used

- Translation: Positioning objects using glTranslatef() with keyboard controls (W/A/S/D/Q/E).
- Rotation: Rotating objects around X, Y, and Z axes using glRotatef() with keyboard controls (X/x, Y/y, Z/z).
- Scaling: Resizing objects uniformly using glScalef() with keyboard controls (+/-).

Code

```
#include <GL/glut.h>
#include <iostream>
#include <math.h>

using namespace std;

// Global variables for cube coordinates
float X1, Y1, Z1;
float X2, Y2, Z2;

// Global variables for transformations
float transX = 0, transY = 0, transZ = 0;
float angleX = 0.0f, angleY = 0.0f, angleZ = 0.0f;
float scaleX = 1.0f, scaleY = 1.0f, scaleZ = 1.0f;

// Transformation step values
float stepX = 5, stepY = 5, stepZ = 5;
float Tx = 0.5f, Ty = 0.5f, Tz = 0.5f;
float Sx = 0.1f, Sy = 0.1f, Sz = 0.1f;

// ===== Function to draw a Cube =====
```

```
void drawCubeFromPoints(float x1, float y1, float z1, float x2, float y2, float z2)
{
    glBegin(GL_QUADS);

    // Front face (Red)
    glColor3f(1, 0, 0);
    glVertex3f(x1, y1, z1);
    glVertex3f(x2, y1, z1);
    glVertex3f(x2, y2, z1);
    glVertex3f(x1, y2, z1);

    // Back face (Green)
    glColor3f(0, 1, 0);
    glVertex3f(x1, y1, z2);
    glVertex3f(x1, y2, z2);
    glVertex3f(x2, y2, z2);
    glVertex3f(x2, y1, z2);

    // Left face (Blue)
    glColor3f(0, 0, 1);
    glVertex3f(x1, y1, z1);
    glVertex3f(x1, y1, z2);
    glVertex3f(x1, y2, z2);
    glVertex3f(x1, y2, z1);

    // Right face (Yellow)
    glColor3f(1, 1, 0);
```

```
    glVertex3f(x2, y1, z1);
    glVertex3f(x2, y1, z2);
    glVertex3f(x2, y2, z2);
    glVertex3f(x2, y2, z1);

    // Top face (Cyan)
    glColor3f(0, 1, 1);
    glVertex3f(x1, y2, z1);
    glVertex3f(x1, y2, z2);
    glVertex3f(x2, y2, z2);
    glVertex3f(x2, y2, z1);

    // Bottom face (Magenta)
    glColor3f(1, 0, 1);
    glVertex3f(x1, y1, z1);
    glVertex3f(x2, y1, z1);
    glVertex3f(x2, y1, z2);
    glVertex3f(x1, y1, z2);

    glEnd();
}

// ===== Function to draw a Pyramid =====
void drawPyramid(float x1, float y1, float z1, float x2, float y2, float z2)
{
    float centerX = (x1 + x2) / 2.0f;
    float centerY = y2;
```

```
float centerZ = (z1 + z2) / 2.0f;

glBegin(GL_TRIANGLES);

// Front face
glColor3f(1, 0.5, 0);
glVertex3f(x1, y1, z1);
glVertex3f(x2, y1, z1);
glVertex3f(centerX, centerY, centerZ);

// Right face
glColor3f(1, 0, 1);
glVertex3f(x2, y1, z1);
glVertex3f(x2, y1, z2);
glVertex3f(centerX, centerY, centerZ);

// Back face
glColor3f(0, 1, 1);
glVertex3f(x2, y1, z2);
glVertex3f(x1, y1, z2);
glVertex3f(centerX, centerY, centerZ);

// Left face
glColor3f(0.5, 1, 0);
glVertex3f(x1, y1, z2);
glVertex3f(x1, y1, z1);
glVertex3f(centerX, centerY, centerZ);
```

```
    // Base
    glColor3f(0.5, 0.5, 0.5);
    glVertex3f(x1, y1, z1);
    glVertex3f(x2, y1, z1);
    glVertex3f(x2, y1, z2);

    glVertex3f(x1, y1, z1);
    glVertex3f(x2, y1, z2);
    glVertex3f(x1, y1, z2);

    glEnd();
}

// ===== Function to draw a Tetrahedron =====
void drawTetrahedron(float size)
{
    glBegin(GL_TRIANGLES);

    // Face 1
    glColor3f(1, 0, 0);
    glVertex3f(0, 0, 0);
    glVertex3f(size, 0, 0);
    glVertex3f(size/2, size, 0);

    // Face 2
    glColor3f(0, 1, 0);
```

```
    glVertex3f(0, 0, 0);
    glVertex3f(size, 0, 0);
    glVertex3f(size/2, size/2, size);

    // Face 3
    glColor3f(0, 0, 1);
    glVertex3f(size, 0, 0);
    glVertex3f(size/2, size, 0);
    glVertex3f(size/2, size/2, size);

    // Face 4
    glColor3f(1, 1, 0);
    glVertex3f(0, 0, 0);
    glVertex3f(size/2, size, 0);
    glVertex3f(size/2, size/2, size);

    glEnd();
}

// ===== Function to draw a Sphere =====
void drawSphere(float radius, int slices, int stacks)
{
    for(int i = 0; i < stacks; i++)
    {
        float lat0 = M_PI * (-0.5 + (float)i / stacks);
        float lat1 = M_PI * (-0.5 + (float)(i+1) / stacks);
```

```
        glBegin(GL_QUAD_STRIP);

        for(int j = 0; j <= slices; j++)
        {
            float lng = 2 * M_PI * (float)j / slices;

            float x0 = cos(lat0) * cos(lng);
            float y0 = sin(lat0);
            float z0 = cos(lat0) * sin(lng);

            float x1 = cos(lat1) * cos(lng);
            float y1 = sin(lat1);
            float z1 = cos(lat1) * sin(lng);

            glColor3f(0.2f + (i * 0.7f / stacks), 0.3f, 0.9f - (i * 0.5f / stacks));
            glVertex3f(x0 * radius, y0 * radius, z0 * radius);
            glVertex3f(x1 * radius, y1 * radius, z1 * radius);
        }

        glEnd();
    }
}

// ===== Display Function =====
void display()
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

```
glLoadIdentity();

gluLookAt(20, 20, 30,
        0, 0, 0,
        0, 1, 0);

// Apply transformations
glTranslatef(transX, transY, transZ);
glRotatef(angleX, 1, 0, 0);
glRotatef(angleY, 0, 1, 0);
glRotatef(angleZ, 0, 0, 1);

// ===== Draw Cube =====
glPushMatrix();
glScalef(scaleX, scaleY, scaleZ);
drawCubeFromPoints(X1, Y1, Z1, X2, Y2, Z2);
glPopMatrix();

// ===== Draw Pyramid =====
glPushMatrix();
glTranslatef(8, 0, 0);
glScalef(scaleX, scaleY, scaleZ);
drawPyramid(X1, Y1, Z1, X2, Y2, Z2);
glPopMatrix();

// ===== Draw Tetrahedron =====
glPushMatrix();
```

```
    glTranslatef(-8, 0, 0);
    glScalef(scaleX, scaleY, scaleZ);
    drawTetrahedron(3.0f);
    glPopMatrix();

    // ===== Draw Sphere =====
    glPushMatrix();
    glTranslatef(0, 0, 8);
    drawSphere(2.0f, 20, 20);
    glPopMatrix();

    glutSwapBuffers();
}

// ===== Keyboard Function =====
void keyboard(unsigned char key, int x, int y)
{
    switch(key)
    {
        // Rotation controls
        case 'X': angleX += stepX; break;
        case 'x': angleX -= stepX; break;
        case 'Y': angleY += stepY; break;
        case 'y': angleY -= stepY; break;
        case 'Z': angleZ += stepZ; break;
        case 'z': angleZ -= stepZ; break;
```

```
// Translation controls
case 'w': transY += Ty; break;
case 's': transY -= Ty; break;
case 'd': transX += Tx; break;
case 'a': transX -= Tx; break;
case 'q': transZ += Tz; break;
case 'e': transZ -= Tz; break;

// Scaling controls
case '+':
    scaleX += Sx;
    scaleY += Sy;
    scaleZ += Sz;
    break;
case '-':
    scaleX -= Sx;
    scaleY -= Sy;
    scaleZ -= Sz;
    if(scaleX < 0.1f) scaleX = 0.1f;
    if(scaleY < 0.1f) scaleY = 0.1f;
    if(scaleZ < 0.1f) scaleZ = 0.1f;
    break;

// Reset
case 'r':
    transX = 0; transY = 0; transZ = 0;
    angleX = 0; angleY = 0; angleZ = 0;
```

```
            scaleX = 1.0f; scaleY = 1.0f; scaleZ = 1.0f;
            break;

        case 27: exit(0); // ESC to exit
    }

    glutPostRedisplay();
}

// ===== Reshape Function =====
void reshape(int w, int h)
{
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(60, (float)w / (float)h, 1.0, 100.0);
    glMatrixMode(GL_MODELVIEW);
}

// ===== Initialization Function =====
void init()
{
    glEnable(GL_DEPTH_TEST);
    glClearColor(0.1f, 0.1f, 0.2f, 1.0f);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
```

```
    gluPerspective(60, 1.0, 1.0, 100.0);

    glMatrixMode(GL_MODELVIEW);
}

// ===== Main Function =====
int main(int argc, char** argv)
{
    cout << "=========================================\n";
    cout << "     3D Objects with Transformations\n";
    cout << "=========================================\n\n";

    cout << "Enter first point (x1 y1 z1): ";
    cin >> X1 >> Y1 >> Z1;

    cout << "Enter second point (x2 y2 z2): ";
    cin >> X2 >> Y2 >> Z2;

    cout << "\n========== CONTROLS ==========\n";
    cout << "Rotation:    X/x, Y/y, Z/z\n";
    cout << "Translation: W/A/S/D/Q/E\n";
    cout << "Scale:       +/-\n";
    cout << "Reset:       R\n";
    cout << "Exit:        ESC\n";
    cout << "==============================\n\n";

    glutInit(&argc, argv);
```

```
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(800, 600);
    glutCreateWindow("3D Objects - Cube, Pyramid, Tetrahedron, Sphere");

    init();

    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutKeyboardFunc(keyboard);

    glutMainLoop();

    return 0;
}
```

Explanation

3D Objects:

- The cube is drawn from two diagonal corner points, making it flexible and easy to resize.
- The pyramid is constructed with a rectangular base and four triangular sides meeting at an apex.
- The tetrahedron is a simple shape with four triangular faces, serving as a basic 3D primitive.
- The sphere is generated using trigonometric functions to create latitude and longitude segments.

3D Transformations:

The display() function applies transformations in a specific order: first translation, then rotation around all three axes, and finally scaling. This order is important because matrix multiplication in OpenGL is not commutative—changing the order of transformations produces different results.

Keyboard Controls:

Rotation:

- 'X' / 'x' - rotate in positive/negative direction around X-axis
- 'Y' / 'y' - rotate in positive/negative direction around Y-axis
- 'Z' / 'z' - rotate in positive/negative direction around Z-axis

Translation:

- 'W' - move up (positive Y-axis)
- 'S' - move down (negative Y-axis)
- 'D' - move right (positive X-axis)
- 'A' - move left (negative X-axis)
- 'Q' - move forward (positive Z-axis)
- 'E' - move backward (negative Z-axis)

Scaling:

- '+' - increase size uniformly in all dimensions
- '-' - decrease size uniformly in all dimensions (minimum 0.1)

Other:

- 'R' - reset all transformations
- 'ESC' - exit the program

Input Example:

- (x1, y1, z1) = (-2, -2, -2)
- (x2, y2, z2) = (2, 2, 2)

Output

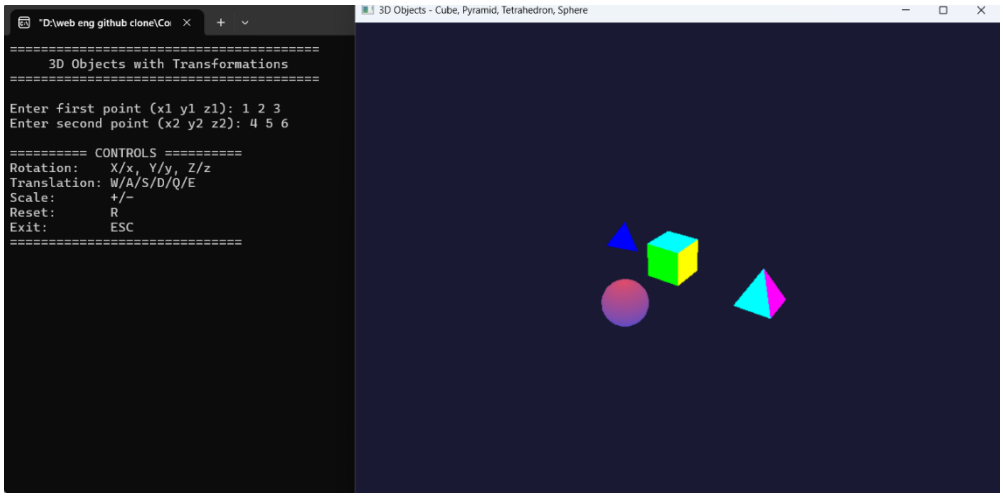

Figure 1: Initial Position with value and also preeing “r”.

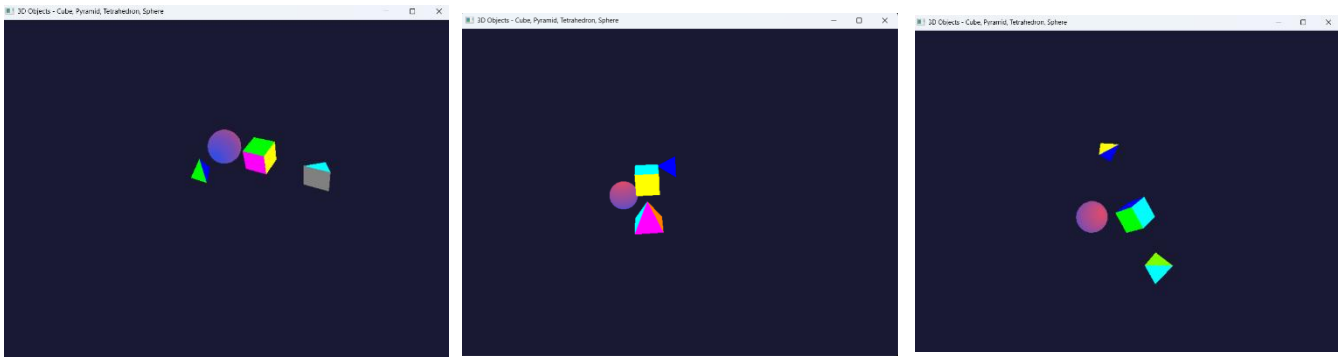

Figure 2,3,4: X-axis, Y-axis, Z-axis moveme nt by pressing “x”, “y” and “z”.

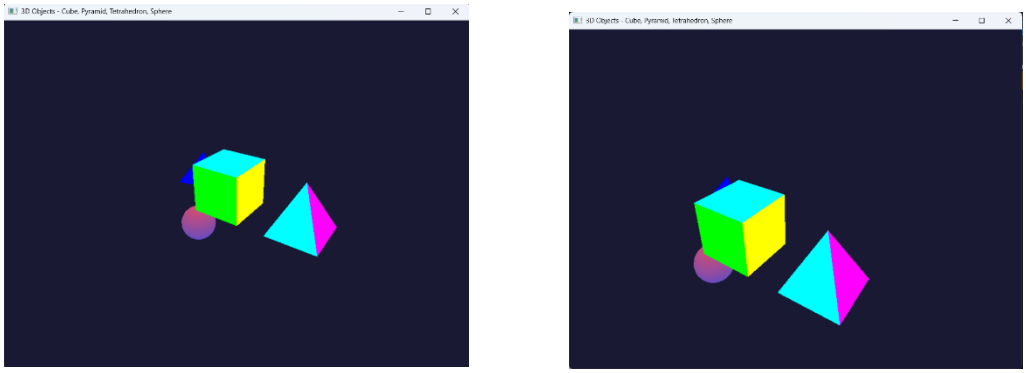

Figure 5,6: Figure Scaling by pressing "+/-“, Translation by pressing "W/D/A/S/Q/E”

Discussion

The key learnings include:

1. 3D Object Rendering: Multiple 3D geometric shapes (cube, pyramid, tetrahedron, sphere) are rendered using different primitive types (GL_QUADS and GL_TRIANGLES).
2. Coordinate Systems: Understanding 3D Cartesian coordinates and how to specify vertices in 3D space using (x, y, z) coordinates.
3. Transformation Matrices: The application of glTranslatef(), glRotatef(), and glScalef() demonstrates how transformation matrices are applied to modify object positions, orientations, and sizes.
4. Matrix Stack: Use of glPushMatrix() and glPopMatrix() allows independent transformation of multiple objects without affecting each other.
5. Viewing Pipeline: The gluLookAt() function positions the camera in 3D space, and gluPerspective() defines the projection matrix for proper 3D perspective rendering.

6. Depth Testing: Enabling GL_DEPTH_TEST ensures proper rendering order of overlapping objects based on their distance from the camera.

This program demonstrates fundamental 3D graphics concepts by rendering four different 3D objects (cube, pyramid, tetrahedron, and sphere) and allowing interactive manipulation through real-time transformations.