Shivcharan T
205001100
CSE - B

SSN COLLEGE OF ENGINEERING, KALAVAKKAM
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
UCS1712 – GRAPHICS AND MULTIMEDIA LAB

Lab Exercise 1 : Study of Basic Output Primitives in C++ using OpenGL

a) To create an output window using OPENGL and to draw the following basic output primitives – POINTS, LINES, LINE_STRIP, LINE_LOOP, TRIANGLES, QUADS, QUAD_STRIP, POLYGON.
b) To create an output window and draw a checkerboard using OpenGL.
c) To create an output window and draw a house using POINTS,LINES,TRAINGLES and QUADS/POLYGON.

Aim:

Study of basic output primitives in c++ using openGL.

Algorithm:

1. Include necessary header files.
2. Create an initialization function (**myInit**) to set up OpenGL settings.
3. Define a function (**drawShape**) to draw various shapes with vertices and labels.
4. Create a function (**myDisplay**) for rendering:
    - Clear the color buffer.
    - Draw points, lines, quads, triangles, and polygons using **drawShape**.
    - Label points and lines with their coordinates using **drawShape**.
5. In the **main** function:
    - Initialize GLUT and set display mode.
    - Create a window, set the display function to **myDisplay**, and initialize OpenGL settings.
    - Enter the GLUT main loop.

Code:

a.cpp:

```
// #include <GLUT/glut.h>   //in clg system
#include <GL/glut.h> //my laptop
#include <stdio.h>
#include <cstring>
#include <iostream>
using namespace std;
void myInit()
{
    glClearColor(0.0, 0.0, 0.0, 0.0); // used for glClear: sets bitplane window
    glPointSize(10);
    glMatrixMode(GL_PROJECTION); // applies the matrix operations to corresponding
stack
    /*
    GL_MODELVIEW    -   modelview matrix stack.
    GL_PROJECTION   -   projection
```

```
    GL_TEXTURE      -   texture
    GL_COLOR        -   color
    */
    glLoadIdentity();                    // replaces the current matrix with the
identity matrix
    gluOrtho2D(0.0, 640.0, 0.0, 480.0); // sets up a 2D orthographic viewing region
    // for (0,0) to be at the center of the screen, put it like (-320,320,-240,240)
}

//*************************version 1*************************:

// void myDisplay()
// {
//     glClear(GL_COLOR_BUFFER_BIT);
//     glBegin(GL_POINTS);
//     glVertex2d(150, 100);
//     glEnd();

//     glBegin(GL_LINES);
//     glVertex2d(150, 150);
//     glVertex2d(150, 200);
//     glEnd();

//     glBegin(GL_QUADS);
//     glColor3f(0.0f, 1.0f, 0.0f);
//     glVertex2d(300, 300);
//     glVertex2d(300, 350);
//     glVertex2d(350, 350);
//     glVertex2d(350, 300);
//     glVertex2d(300, 300);
//     glEnd();

//     glBegin(GL_TRIANGLES); // Each set of 3 vertices form a triangle
//     glColor3f(0.0f, 0.0f, 1.0f);
//     glVertex2d(400, 400);
//     glVertex2d(400, 450);
//     glVertex2d(450, 450);
//     glVertex2d(400, 400);
//     glEnd();

//     glBegin(GL_POLYGON);         // These vertices form a closed polygon
//     glColor3f(1.0f, 1.0f, 0.0f); // Yellow
//     glVertex2d(200, 200);
//     glVertex2d(200, 220);
//     glVertex2d(220, 240);
//     glVertex2d(240, 200);
//     glVertex2d(230, 250);
//     glVertex2d(250, 250);
//     glVertex2d(200, 200);
//     glEnd();

//     glFlush();
// }

//*************************version 2*************************:
```

```
// Function to convert GLenum value to string
string GLenumToString(GLenum mode)
{
    switch (mode)
    {
    case GL_POINTS:
        return "GL_POINTS";
    case GL_LINES:
        return "GL_LINES";
    case GL_LINE_STRIP:
        return "GL_LINE_STRIP";
    case GL_LINE_LOOP:
        return "GL_LINE_LOOP";
    case GL_TRIANGLES:
        return "GL_TRIANGLES";
    case GL_TRIANGLE_STRIP:
        return "GL_TRIANGLE_STRIP";
    case GL_TRIANGLE_FAN:
        return "GL_TRIANGLE_FAN";
    case GL_QUADS:
        return "GL_QUADS";
    case GL_QUAD_STRIP:
        return "GL_QUAD_STRIP";
    case GL_POLYGON:
        return "GL_POLYGON";
    default:
        return "Unknown";
    }
    return "";
}

// Common function to draw shapes
void drawShape(GLenum mode, double vertices[], int numVertices)
{
    // glVertex2d-2d-d=double
    glBegin(mode);
    cout << "Drawing mode: " << GLenumToString(mode) << " (Value: " << mode <<
")\n";
    for (int i = 0; i < numVertices; i += 2)
    {
        // glVertex2d(vertices[i], vertices[i + 1]);

        double x = vertices[i];
        double y = vertices[i + 1];
        cout << "Vertex (" << x << ", " << y << ")" << endl; // Print coordinates
        glVertex2d(x, y);
    }
    glEnd();
    cout << endl;
}

// Function to draw text at a given position
void drawText(double x, double y, int a = 0, int b = 0)
{
```

```
    char text[20];
    snprintf(text, 20, "(%0.0f,%0.0f)", x, y);
    glRasterPos2d(x + a, y + b);
    for (int i = 0; text[i] != '\0'; i++)
        glutBitmapCharacter(GLUT_BITMAP_HELVETICA_10, text[i]);
    memset(text, 0, sizeof(text));
}

void myDisplay()
{
    glClear(GL_COLOR_BUFFER_BIT);

    double pointVertices[] = {150, 100};
    double lineVertices[] = {150, 150, 150, 200};
    double quadVertices[] = {300, 300, 300, 350, 350, 350, 350, 300, 300, 300};
    double triangleVertices[] = {400, 400, 400, 450, 450, 450, 400, 400};
    double polygonVertices[] = {200, 200, 200, 220, 220, 240, 240, 200, 230, 250,
250, 250, 200, 200};

    // Draw points
    drawShape(GL_POINTS,       pointVertices,       sizeof(pointVertices)       /
sizeof(pointVertices[0]));
    drawText(pointVertices[0], pointVertices[1], 10, -5);

    // Draw lines
    drawShape(GL_LINES,        lineVertices,        sizeof(lineVertices)        /
sizeof(lineVertices[0]));
    drawText(lineVertices[0], lineVertices[1], 10, -5);
    drawText(lineVertices[2], lineVertices[3], -55,-5);

    // Draw quads
    glBegin(GL_QUADS);
    glColor3f(0.0f, 1.0f, 0.0f);
    drawShape(GL_QUADS,        quadVertices,        sizeof(quadVertices)        /
sizeof(quadVertices[0]));
    glEnd();

    // Draw triangles
    glColor3f(0.0f, 0.0f, 1.0f);
    drawShape(GL_TRIANGLES,    triangleVertices,    sizeof(triangleVertices)    /
sizeof(triangleVertices[0]));

    // Draw polygon
    glBegin(GL_POLYGON);
    glColor3f(1.0f, 1.0f, 0.0f); // Yellow
    drawShape(GL_POLYGON,      polygonVertices,     sizeof(polygonVertices)     /
sizeof(polygonVertices[0]));
    glEnd();

    glFlush();
}
int main(int argc, char *argv[])
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
```

```
    glutInitWindowSize(640, 480);
    glutCreateWindow("1-a");
    glutDisplayFunc(myDisplay);
    myInit();
    glutMainLoop();
    return 1;
}
```

b.cpp:

```
// #include <GLUT/glut.h>   //in clg system
#include <GL/glut.h> //my laptop

int windowWidth = 800;
int windowHeight = 800;
void myInit()
{
    glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
    // glPointSize(10);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0, windowWidth, 0.0, windowHeight);

    // why?
    // glMatrixMode(GL_MODELVIEW);
}
void drawCheckerboard()
{
    // glVertex2i-2i-i=int

    int rows = 8;
    int cols = 8;
    int squareSize = windowWidth / cols;

    glColor3f(1.0f, 1.0f, 1.0f);
    glBegin(GL_QUADS);

    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < cols; j++)
        {
            ((i + j) & 1) ? glColor3f(0.0f, 0.0f, 0.0f) : glColor3f(1.0f, 1.0f,
1.0f);
            glVertex2i(j * squareSize, i * squareSize);
            glVertex2i((j + 1) * squareSize, i * squareSize);
            glVertex2i((j + 1) * squareSize, (i + 1) * squareSize);
            glVertex2i(j * squareSize, (i + 1) * squareSize);
        }
    }

    glEnd();
    glFlush();
}
```

```
void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    drawCheckerboard();
}
int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(windowWidth, windowHeight);
    glutCreateWindow("1-b");
    glutDisplayFunc(display);
    myInit(); // glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
    glutMainLoop();
    return 1;
}
```

c.cpp:

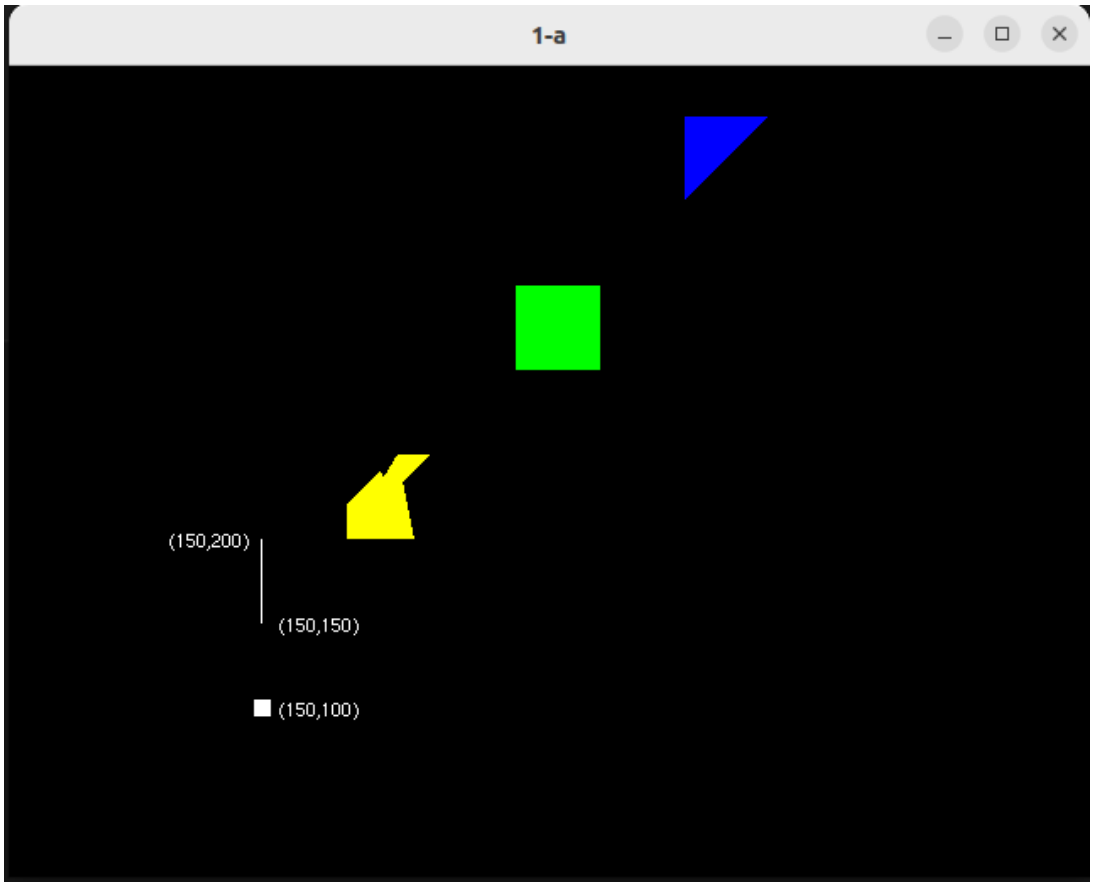
```
// #include <GLUT/glut.h>   //in clg system
#include <GL/glut.h> //my laptop

int windowWidth = 800;
int windowHeight = 600;

void myInit()
{
    glClearColor(1.0f, 1.0f, 1.0f, 0.0f); //White BG
    // glPointSize(10);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0, windowWidth, 0.0, windowHeight);

    // why?
    // glMatrixMode(GL_MODELVIEW);
}
void drawHouse()
{
    // Clear the screen
    // Draw the house using different primitive shapes
    // Draw the base of the house using a quad
    glColor3f(0.59f, 0.85f, 0.71f); // Gray color
    glBegin(GL_QUADS);
    glVertex2i(100, 100);
    glVertex2i(500, 100);
    glVertex2i(500, 400);
    glVertex2i(100, 400);
    glEnd();

    // Draw the roof using triangles
    glColor3f(1.0f, 0.0f, 0.0f); // Red color
    glBegin(GL_TRIANGLES);
    glVertex2i(100, 400);
```

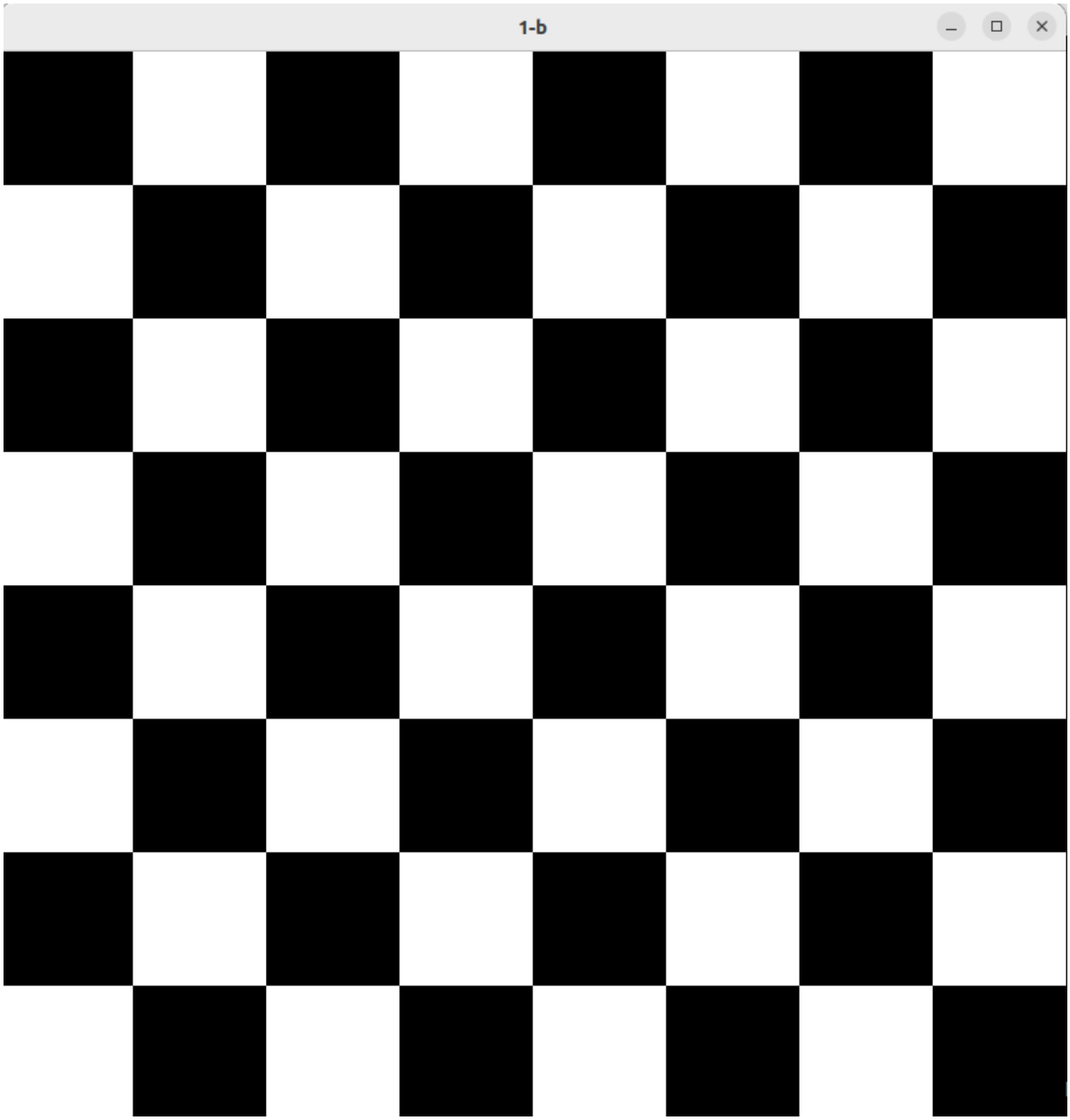
```
    glVertex2i(300, 600);
    glVertex2i(500, 400);
    glEnd();

    // Draw the door using quads
    glColor3f(1.0f, 1.0f, 1.0f); // Blue color
    glBegin(GL_QUADS);
    glVertex2i(250, 100);
    glVertex2i(350, 100);
    glVertex2i(350, 300);
    glVertex2i(250, 300);
    // glEnd();

    glColor3f(0.36f, 0.05f, 0.05f);
    // glBegin(GL_QUADS);
    glVertex2i(250, 100);
    glVertex2i(330, 130);
    glVertex2i(330, 300);
    glVertex2i(250, 300);
    // glEnd();

    // Draw the windows using quads
    glColor3f(0.06f, 0.22f, 0.45f); // Green color
    // glBegin(GL_QUADS);
    glVertex2i(150, 200);
    glVertex2i(200, 200);
    glVertex2i(200, 250);
    glVertex2i(150, 250);
    glVertex2i(400, 200);
    glVertex2i(450, 200);
    glVertex2i(450, 250);
    glVertex2i(400, 250);

    glEnd();
    glFlush();
}
void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    drawHouse();
}
int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(windowWidth, windowHeight);
    glutCreateWindow("Drawing a House using OpenGL");
    glutDisplayFunc(display);
    myInit(); // glClearColor(1.0f, 1.0f, 1.0f, 0.0f); // White background
    glutMainLoop();
    return 0;
}
```

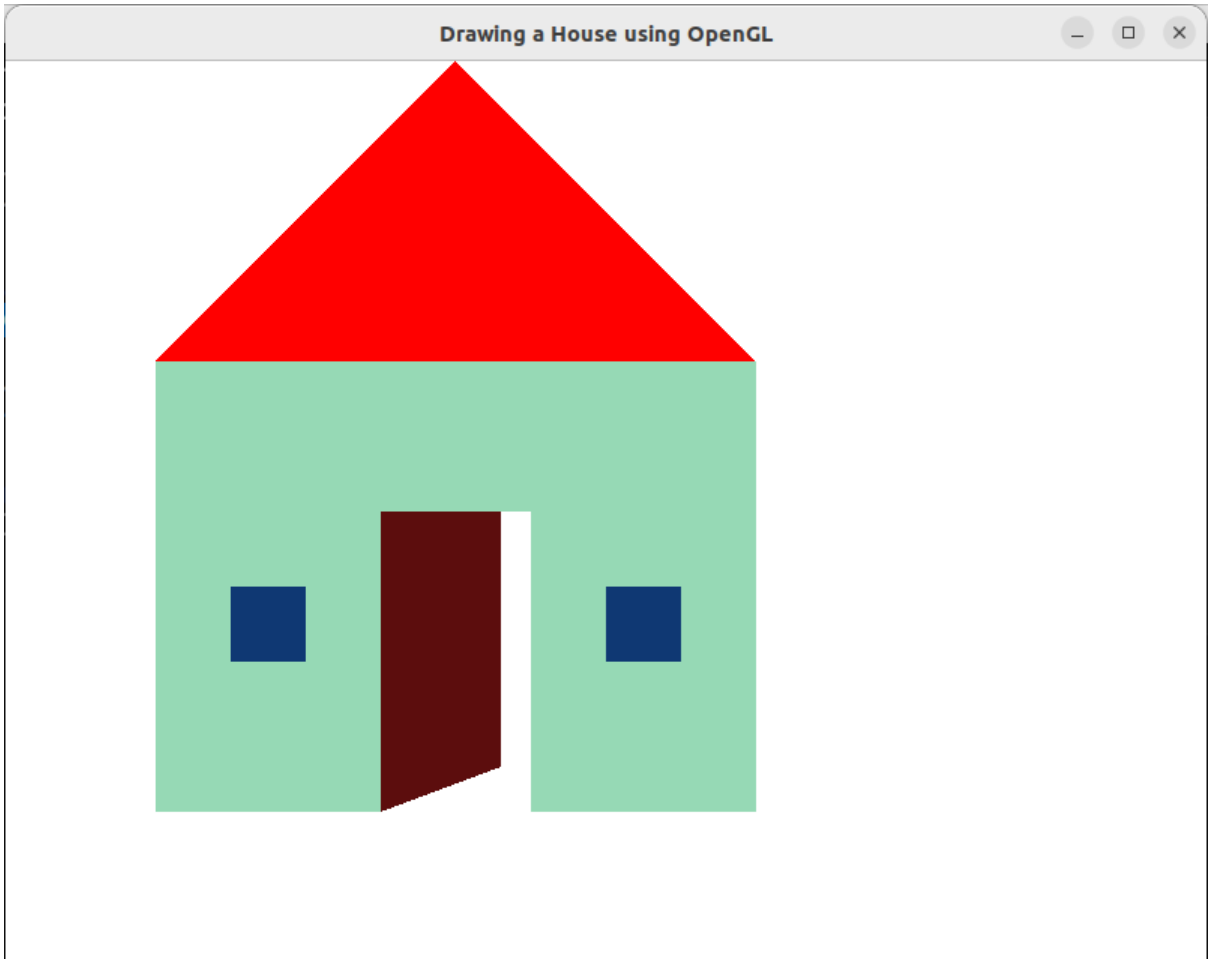
run.sh:

```
g++ a.cpp -lGL -lglut -lGLU
./a.out
```

Sample I/O:




1-b



Learning Outcomes:

Thus, the following shapes/objects have been created using OpenGL primitive

Shivcharan T
205001100
CSE - B

SSN COLLEGE OF ENGINEERING, KALAVAKKAM
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
UCS1712 – GRAPHICS AND MULTIMEDIA LAB

Lab Exercise 2 : DDA Line Drawing Algorithm in C++ using OpenGL

To plot points that make up the line with endpoints (x0,y0) and (xn,yn) using DDA line drawing algorithm.
Case 1: +ve slope Left to Right line
Case 2: +ve slope Right to Left line
Case 3: -ve slope Left to Right line
Case 4: -ve slope Right to Left line
Each case has two subdivisions
(i) |m|<= 1 (ii) |m|>1
Note that all four cases of line drawing must be given as test cases.

Aim:
To implement DDA line drawing algorithm

Algorithm:

1. Accept two endpoints `P1(x1, y1)` and `P2(x2, y2)` of the line to be drawn.
2. Calculate the differences in x and y coordinates:
   dx = x2 - x1
   dy = y2 – y1
3. Determine the number of steps required for drawing the line. Use the larger of `dx` and `dy` as the number of steps:
   steps = max(abs(dx), abs(dy))
4. Calculate the incremental values for `x` and `y`:
   x_increment = dx / steps
   y_increment = dy / steps
5. Initialize a loop and start drawing the line by repeatedly adding the incremental values to `x` and `y`:
   x = x1
   y = y1
   for i from 1 to steps:
      Plot the point (x, y)
      x = x + x_increment
      y = y + y_increment
6. The loop will draw the line from `P1` to `P2` by plotting points along the line at regular intervals.
7. End the algorithm.

Code:

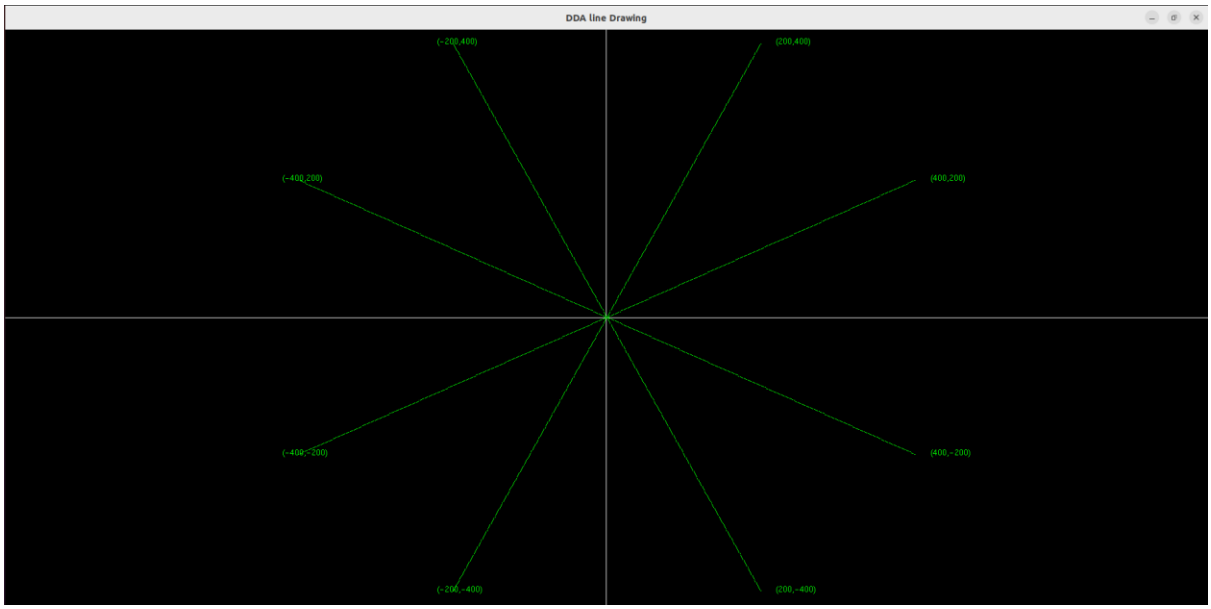
```
#include <stdio.h>
#include <GL/glut.h>
#include <math.h>
#include <cstring>
#define pi 3.142857

void output(int x, int y, const char *string)
{
    glRasterPos2f(x, y);
    int len, i;
    len = (int)strlen(string);
    for (i = 0; i < len; i++)
    {
        glutBitmapCharacter(GLUT_BITMAP_HELVETICA_12, string[i]);
    }
}
void draw_pixel(int x, int y)
{
    glBegin(GL_POINTS);
    glVertex2i(x, y);
    glEnd();
}
void myInit(void)
{
    glClearColor(0.0, 0.0, 0.0, 1.0);
    glColor3f(0.0, 1.0, 0.0); // making picture color green (in RGB mode), as middle
argument is 1.0
    glPointSize(1.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(-780, 780, -420, 420);
}
void drawLineDDA(float x0, float y0, float xn, float yn)
{
    // glClear(GL_COLOR_BUFFER_BIT);
    float dx = xn - x0;
    float dy = yn - y0;
    float steps = fabs(dx) > fabs(dy) ? fabs(dx) : fabs(dy);
    float xIncrement = dx / steps;
    float yIncrement = dy / steps;
    float x = x0;
    float y = y0;
    for (int i = 0; i <= steps; ++i)
    {
        draw_pixel(static_cast<int>(x + 0.5), static_cast<int>(y + 0.5));
        x += xIncrement;
        y += yIncrement;
    }
}
void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
```

```
    glColor3f(1.0, 1.0, 1.0);

    glBegin(GL_LINES);
    glVertex2d(0, 420);
    glVertex2d(0, -420);
    glEnd();

    glBegin(GL_LINES);
    glVertex2d(780, 0);
    glVertex2d(-780, 0);
    glEnd();

    glColor3f(0.0, 1.0, 0.0);

    drawLineDDA(0, 0, 400, 200);
    output(420, 200, "(400,200)");

    drawLineDDA(0, 0, 200, 400);
    output(220, 400, "(200,400)");

    drawLineDDA(-400, 200, 0, 0);
    output(-420, 200, "(-400,200)");

    drawLineDDA(-200, 400, 0, 0);
    output(-220, 400, "(-200,400)");

    drawLineDDA(0, 0, 400, -200);
    output(420, -200, "(400,-200)");

    drawLineDDA(0, 0, 200, -400);
    output(220, -400, "(200,-400)");

    drawLineDDA(-400, -200, 0, 0);
    output(-420, -200, "(-400,-200)");

    drawLineDDA(-200, -400, 0, 0);
    output(-220, -400, "(-200,-400)");

    glFlush();
}
int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    // giving window size in X- and Y- direction
    glutInitWindowSize(1366, 768);
    glutInitWindowPosition(0, 0);
    // Giving name to window
    glutCreateWindow("DDA line Drawing");
    myInit();
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```

run.sh:

```
g++ 2.cpp -lGL -lglut -lGLU
./a.out
```

Sample I/O:



Learning Outcomes:

Thus, DDA line drawing algo has been implemented with OpenGL and GLUT frameworks.

Shivcharan T
205001100
CSE - B

SSN COLLEGE OF ENGINEERING, KALAVAKKAM
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
UCS1712 – GRAPHICS AND MULTIMEDIA LAB

Lab Exercise 3 : Bresenham's Line Drawing Algorithm in C++ using OpenGL

To plot points that make up the line with endpoints (x0,y0) and (xn,yn) using DDA line drawing algorithm.
Case 1: +ve slope Left to Right line
Case 2: +ve slope Right to Left line
Case 3: -ve slope Left to Right line
Case 4: -ve slope Right to Left line
Each case has two subdivisions
(i) |m|<= 1 (ii) |m|>1
Note that all four cases of line drawing must be given as test cases.

Aim:
To implement DDA line drawing algorithm

Algorithm:

1. Calculate the differences in the x and y coordinates between the two endpoints, which we'll call Δx and Δy.
2. Determine the direction of the line by checking whether Δx and Δy are positive or negative. This helps decide whether to increment or decrement the x and y coordinates while drawing the line.
3. Initialize an error term to keep track of how far off the line is from the ideal path. This error term is calculated as Δx - Δy.
4. Start at the first point (x1, y1) and draw a pixel at that location.
5. Enter a loop that continues until you reach the second point (x2, y2).
6. In each iteration of the loop, you evaluate the error term. If the error term is greater than or equal to zero, you adjust the y-coordinate (move vertically) and subtract Δy from the error term. If the error term is less than zero, you adjust the x-coordinate (move horizontally) and add Δx to the error term.
7. Continue this loop until you reach the second point, updating the x and y coordinates based on the error term and the direction of the line.

Code:

```
#include <stdio.h>
#include <GL/glut.h>
#include <math.h>
#include <cstring>
#define pi 3.142857

void output(int x, int y, const char *string)
{
    glRasterPos2f(x, y);
```

```
    int len, i;
    len = (int)strlen(string);
    for (i = 0; i < len; i++)
    {
        glutBitmapCharacter(GLUT_BITMAP_HELVETICA_12, string[i]);
    }
}
void draw_pixel(int x, int y)
{
    glBegin(GL_POINTS);
    glVertex2i(x, y);
    glEnd();
}
void myInit(void)
{
    glClearColor(0.0, 0.0, 0.0, 1.0);
    glColor3f(0.0, 1.0, 0.0); // making picture color green (in RGB mode), as middle
argument is 1.0
    glPointSize(1.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(-780, 780, -420, 420);
}
int sign(int x)
{
    return (x > 0) - (x < 0);
}
void bresenham(int x1, int y1, int x2, int y2)
{
    int dx, dy, i, e;
    int incx, incy, inc1, inc2;
    int x, y;
    dx = abs(x2 - x1);
    dy = abs(y2 - y1);
    incx = x2 < x1 ? -1 : 1;
    incy = y2 < y1 ? -1 : 1;
    x = x1;
    y = y1;
    if (dx > dy)
    {
        draw_pixel(x, y);
        e = 2 * dy - dx;
        inc1 = 2 * (dy - dx);
        inc2 = 2 * dy;
        for (i = 0; i < dx; i++)
        {
            if (e >= 0)
            {
                y += incy;
                e += inc1;
            }
            else
                e += inc2;
            x += incx;
            draw_pixel(x, y);
```

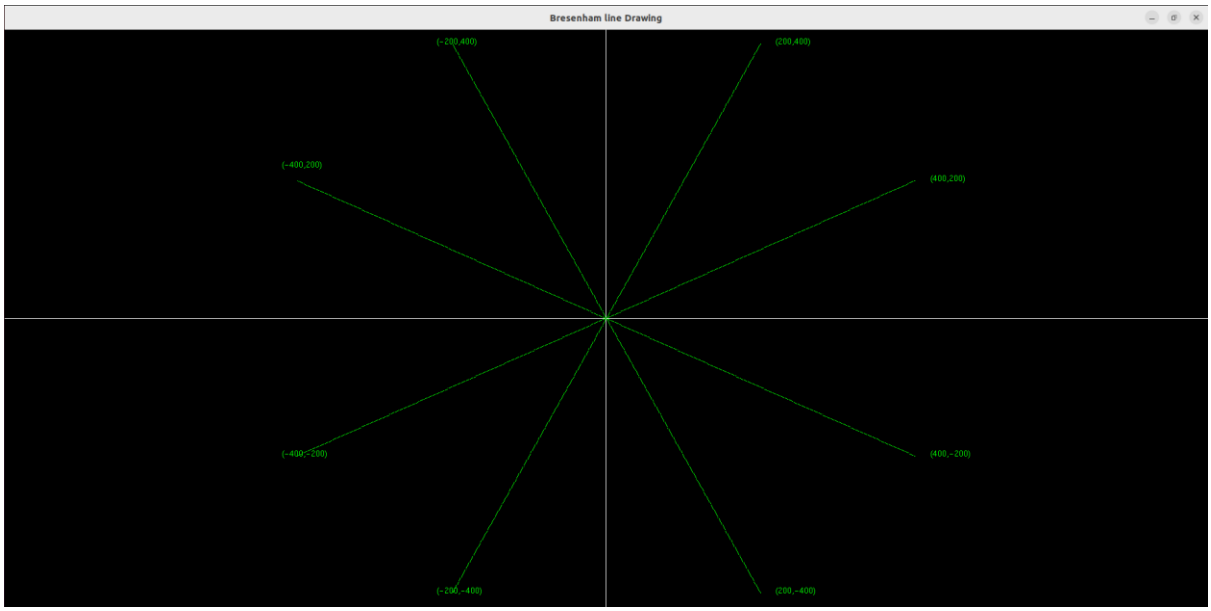
```
        }
    }
    else
    {
        draw_pixel(x, y);
        e = 2 * dx - dy;
        inc1 = 2 * (dx - dy);
        inc2 = 2 * dx;
        for (i = 0; i < dy; i++)
        {
            if (e >= 0)
            {
                x += incx;
                e += inc1;
            }
            else
                e += inc2;
            y += incy;
            draw_pixel(x, y);
        }
    }
}
void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 1.0, 1.0);

    glBegin(GL_LINES);
    glVertex2d(0, 420);
    glVertex2d(0, -420);
    glEnd();

    glBegin(GL_LINES);
    glVertex2d(780, 0);
    glVertex2d(-780, 0);
    glEnd();

    glColor3f(0.0, 1.0, 0.0);

    bresenham(0, 0, 400, 200);
    output(420, 200, "(400,200)");
    bresenham(0, 0, 200, 400);
    output(220, 400, "(200,400)");
    bresenham(-400, 200, 0, 0);
    output(-420, 220, "(-400,200)");
    bresenham(-200, 400, 0, 0);
    output(-220, 400, "(-200,400)");

    bresenham(0, 0, 400, -200);
    output(420, -200, "(400,-200)");
    bresenham(0, 0, 200, -400);
    output(220, -400, "(200,-400)");
    bresenham(-400, -200, 0, 0);
    output(-420, -200, "(-400,-200)");
    bresenham(-200, -400, 0, 0);
```

```
    output(-220, -400, "(-200,-400)");

    glFlush();
}
int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(1366, 768);
    glutInitWindowPosition(0, 0);
    glutCreateWindow("Bresenham line Drawing");
    myInit();
    glutDisplayFunc(display);
    glutMainLoop();
    return 1;
}
```

run.sh:

```
g++ 3.cpp -lGL -lglut -lGLU
./a.out
```

Sample I/O:



Learning Outcomes:

Thus, Bresenham's line drawing algorithm has been implemented with OpenGL and GLUT frameworks.

Shivcharan T
205001100
CSE - B

SSN COLLEGE OF ENGINEERING, KALAVAKKAM
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
UCS1712 – GRAPHICS AND MULTIMEDIA LAB

Lab Exercise 4: Midpoint Circle Drawing Algorithm in C++ using OpenGL

a) To plot points that make up the circle with center (xc,yc) and radius r using Midpoint circle drawing
algorithm. Give atleast 2 test cases.
Case 1: With center (0,0)
Case 2: With center (xc,yc)
b) To draw any object using line and circle drawing algorithms.

Aim:
To implement circle drawing mid point algorithm.

Algorithm:

1. Input radius r and circle center (xc, yc). set the first point (x0 , y0) = (0, r).
2. Calculate the initial value of the decision parameter as p0 = 1 – r.
3. At each xk position, starting at k = 0, perform the following test:
4. If pk <0,
5. plot(xk +1,yk)andpk+1 =pk +2xk+1 +1,
6. Else,
7. where2xk+1 =2xk +2and2yk+1 =2yk –2.
8. plot (xk+1,yk –1)andpk+1 =pk +2xk+1 +1–2yk+1,
9. Determine symmetry points on the other seven octants.
10. Move each calculated pixel position (x, y) onto the circular path centered on (xc, yc) and plot the coordinate values: x = x +xc ,y=y+yc
11. Repeat steps 3 though 5 until x y.
12. For all points, add the center point (xc, yc)

Code:

```
#include <stdio.h>
#include <GL/glut.h>
#include <math.h>
#include <cstring>
#include <iostream>
#define pi 3.142857
using namespace std;

int windowWidth = 1000;
int windowHeight = 1000;

void myInit(void)
{
    glClearColor(0.0, 0.0, 0.0, 1.0);
```

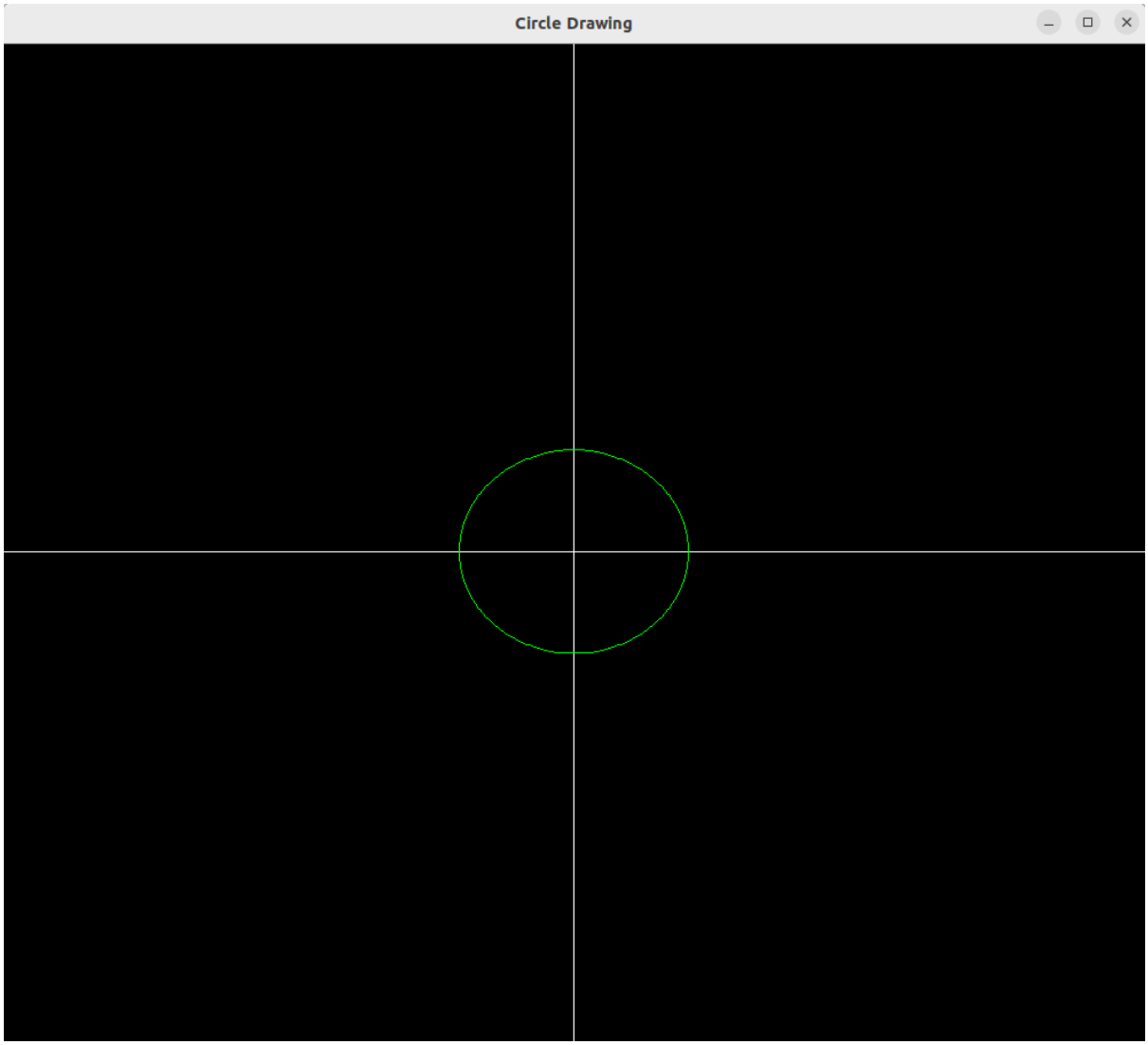
```
    glColor3f(0.0, 1.0, 0.0); // making picture color green (in RGB mode), as middle
argument is 1.0
    glPointSize(1.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(-windowHeight / 2, windowHeight / 2, -windowWidth / 2, windowWidth
/ 2);
}
void draw_pixel(int x, int y)
{
    glBegin(GL_POINTS);
    glVertex2i(x, y);
    glEnd();
    glFlush();
}
void drawLineDDA(float x0, float y0, float xn, float yn)
{
    // glClear(GL_COLOR_BUFFER_BIT);
    float dx = xn - x0;
    float dy = yn - y0;
    float steps = fabs(dx) > fabs(dy) ? fabs(dx) : fabs(dy);
    float xIncrement = dx / steps;
    float yIncrement = dy / steps;
    float x = x0;
    float y = y0;
    for (int i = 0; i <= steps; ++i)
    {
        draw_pixel(static_cast<int>(x + 0.5), static_cast<int>(y + 0.5));
        x += xIncrement;
        y += yIncrement;
    }
}

void draw_axis()
{
    glColor3f(1.0, 1.0, 1.0);
    glBegin(GL_LINES);
    glVertex2d(-2000, 0);
    glVertex2d(2000, 0);
    glEnd();

    glBegin(GL_LINES);
    glVertex2d(0, 2000);
    glVertex2d(0, -2000);
    glEnd();
    glFlush();
}
void draw_in_each_oct(GLint xk, GLint yk, GLint xc, GLint yc)
{
    draw_pixel(xc + xk, yc + yk);
    draw_pixel(xc + yk, yc + xk);
    draw_pixel(xc - yk, yc + xk);
    draw_pixel(xc - xk, yc + yk);
    draw_pixel(xc - xk, yc - yk);
    draw_pixel(xc - yk, yc - xk);
```

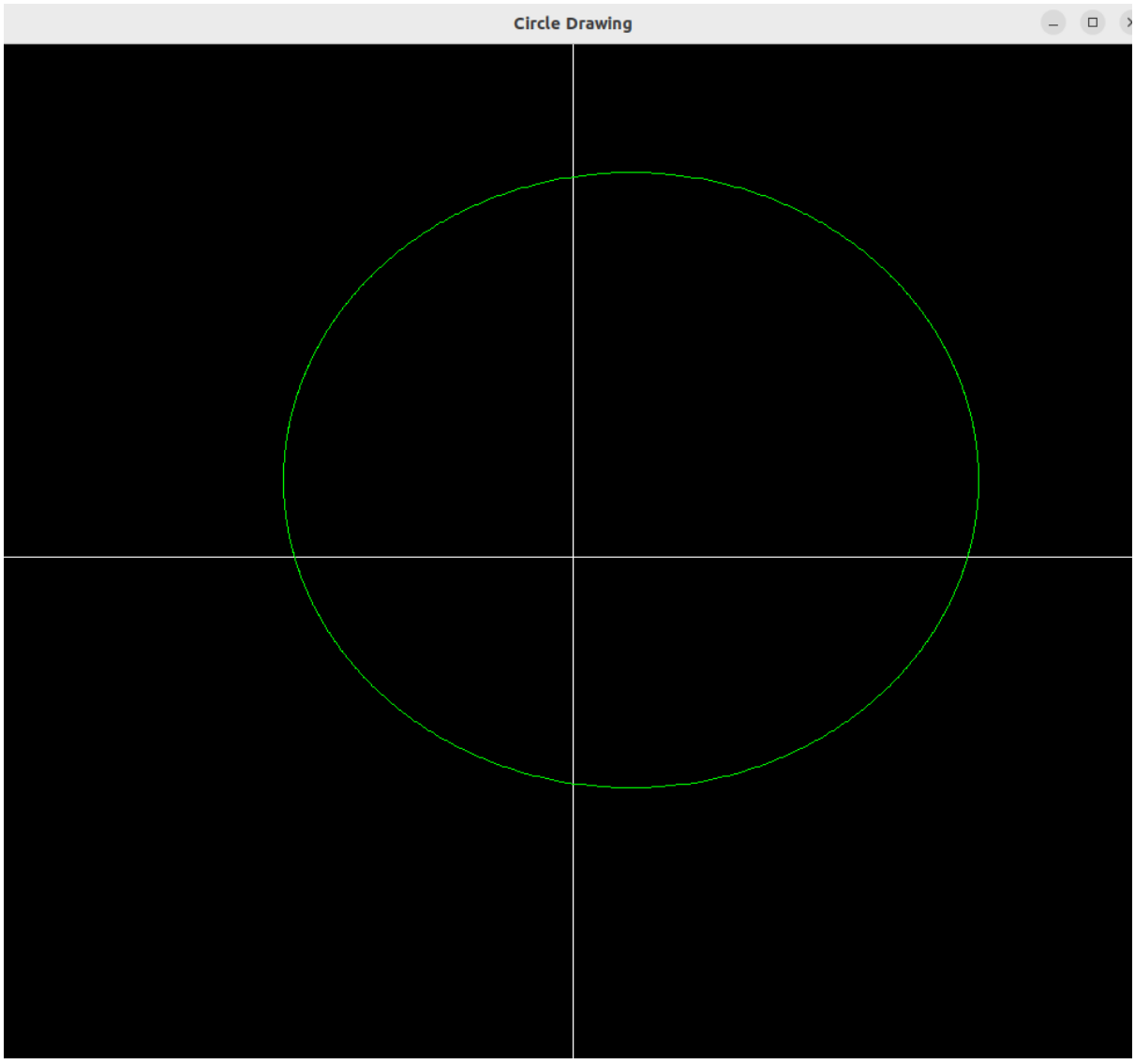
```
    draw_pixel(xc + yk, yc - xk);
    draw_pixel(xc + xk, yc - yk);
}

void midPtCircle(GLint xc, GLint yc, GLint r)
{
    GLint pk, xk, yk;
    pk = 1 - r;
    xk = 0;
    yk = r;
    draw_in_each_oct(xk, yk, xc, yc);
    while (xk <= yk)
    {
        if (pk < 0)
        {
            xk = xk + 1;
            pk = pk + (2 * xk) + 1;
        }
        else
        {
            xk = xk + 1;
            yk = yk - 1;
            pk = pk + (2 * xk) + 1 - (2 * yk);
        }
        draw_in_each_oct(xk, yk, xc, yc);
    }
}
void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    draw_axis();
    glColor3f(0.0, 1.0, 0.0);

    GLint xc, yc, r;
    cout << "Enter xc, yc, radius(resp): ";
    cin >> xc >> yc >> r;
    midPtCircle(xc, yc, r);
    drawLineDDA(0, r, r, 0);
    drawLineDDA(r, 0, 0, -r);
    drawLineDDA(0, -r, -r, 0);
    drawLineDDA(-r, 0, 0, r);
}
int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(windowHeight, windowWidth);
    glutInitWindowPosition(0, 0);
    glutCreateWindow("Circle Drawing");
    myInit();
    glutDisplayFunc(display);
    glutMainLoop();
    return 1;
}
```

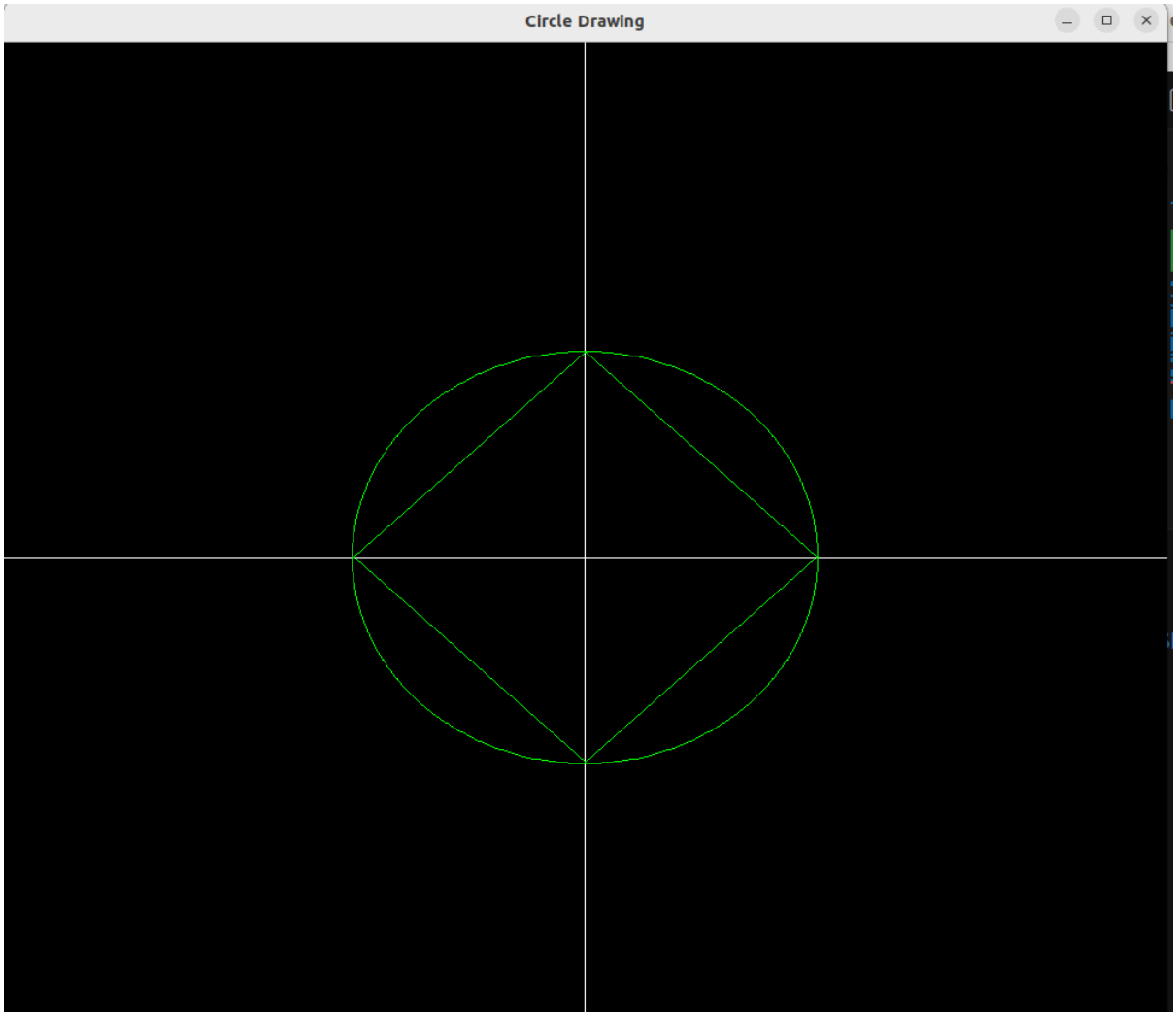
run.sh:

```
g++ 4.cpp -lGL -lglut -lGLU
./a.out
```

Sample I/O:




Circle Drawing



Learning Outcomes:

I learned how to use the midpoint circle drawing algorithm in c++ using the openGL library to draw circles.

Shivcharan T
205001100
CSE - B

SSN COLLEGE OF ENGINEERING, KALAVAKKAM
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
UCS1712 – GRAPHICS AND MULTIMEDIA LAB

Lab Exercise 5: : 2D Transformations in C++ using OpenGL

To apply the following 2D transformations on objects and to render the final output along with the original object.
1) Translation
2) Rotation
 a) about origin
 b) with respect to a fixed point (xr,yr)
3) Scaling with respect to
 a) origin - Uniform Vs Differential Scaling
 b) fixed point (xf,yf)
4) Reflection with respect to
 a) x-axis
 b) y-axis
 c) origin
 d) the line x=y
5) Shearing
 a) x-direction shear
 b) y-direction shear
Note: Use Homogeneous coordinate representations and matrix multiplication to perform transformations. Divide the output window into four quadrants. (Use LINES primitive to draw x and y axis.

Aim:
To implement 2D transformations on objects using C++ using OpenGL

Algorithm:

Application of a sequence of transformations to a point:

P' = M2.M1.P
 = M.P

Composite transformations is formed by calculating the matrix product of the individual transformations and forming products of transformation matrix.

Code:

5a.cpp:

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

```
#include <string.h>
#define pi 3.142857
void mm(double m[3][3], double v[3])
{
    for (int i = 0; i < 3; ++i)
    {
        double temp = 0;
        for (int k = 0; k < 3; ++k)
            temp += m[i][k] * v[k];
        v[i] = temp;
    }
}
int X = 100, Y = -50;
void draw_pixel(int x, int y)
{
    glBegin(GL_POINTS);
    glVertex2i(x, y);
    glEnd();
}
void output(int x, int y, const char *string)
{

    glRasterPos2f(x, y);
    int len, i;
    len = (int)strlen(string);
    for (i = 0; i < len; i++)
    {
        glutBitmapCharacter(GLUT_BITMAP_HELVETICA_12, string[i]);
    }
}
void obj(int a, int b, int c, int d, int w, int x, int y, int z)
{
    glBegin(GL_QUADS);
    glVertex2d(a, b);
    glVertex2d(c, d);
    glVertex2d(w, x);
    glVertex2d(y, z);

    glEnd();
}
void myInit(void)
{
    glClearColor(0.0, 0.0, 0.0, 1.0);
    glColor3f(0.0, 1.0, 0.0);
    glPointSize(1.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(-780, 780, -420, 420);
}
void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 1.0, 1.0);
    glBegin(GL_LINES);
    glVertex2d(0, 420);
```

```
    glVertex2d(0, -420);
    glEnd();
    glBegin(GL_LINES);
    glVertex2d(780, 0);
    glVertex2d(-780, 0);
    glEnd();

    glColor3f(0.0, 1.0, 0.0);

    // TRANSLATION
    double x1[3];
    double x2[3];
    double x3[3];
    double x4[3];
    x1[2] = x2[2] = x3[2] = x4[2] = 1;
    x1[0] = 100;
    x1[1] = 100;
    x2[0] = 200;
    x2[1] = 100;
    x3[0] = 200;
    x3[1] = 200;
    x4[0] = 100;
    x4[1] = 200;
    obj(x1[0], x1[1], x2[0], x2[1], x3[0], x3[1], x4[0], x4[1]);
    output(110, 210, "originalA:");
    double T[3][3] = {{1, 0, 150}, {0, 1, 150}, {0, 0, 1}};
    mm(T, x1);
    mm(T, x2);
    mm(T, x3);
    mm(T, x4);

    obj(x1[0], x1[1], x2[0], x2[1], x3[0], x3[1], x4[0], x4[1]);
    output(260, 360, "translatedA:");

    // ROTATION
    double R[3][3] = {{cos(pi / 4), -sin(pi / 4), 0}, {sin(pi / 4), cos(pi / 4),
0}, {0, 0, 1}};
    x1[0] = 100;
    x1[1] = 100;
    x2[0] = 200;
    x2[1] = 100;
    x3[0] = 200;
    x3[1] = 200;
    x4[0] = 100;
    x4[1] = 200;
    mm(R, x1);
    mm(R, x2);
    mm(R, x3);
    mm(R, x4);
    //  printf("%lf%lf%lf%lf%lf%lf%lf%lf", x1[0], x1[1], x2[0], x2[1], x3[0],
x3[1], x4[0], x4[1]);
    printf("%f%f%f%f%f%f%f%f", x1[0], x1[1], x2[0], x2[1], x3[0], x3[1], x4[0],
x4[1]);
    obj(x1[0], x1[1], x4[0], x4[1], x3[0], x3[1], x2[0], x2[1]);
    output(10, 300, "rotatedA:");
```

```
    printf("%lf", cos(pi / 4));

    // PIVOTROTATION
    double PR[3][3] = {{cos(-pi / 4), -sin(-pi / 4), -X * cos(-pi / 4) + Y * sin(-
pi / 4) + X}, {sin(-pi / 4), cos(-pi / 4), -X * sin(-pi / 4) - Y * cos(-pi / 4) +
Y}, {0, 0, 1}};
    x1[0] = 100;
    x1[1] = 100;
    x2[0] = 200;
    x2[1] = 100;
    x3[0] = 200;
    x3[1] = 200;
    x4[0] = 100;
    x4[1] = 200;
    glColor3f(1.0, 1.0, 1.0);
    output(X - 15, Y - 20, "(rotPivot)");
    glPointSize(5);
    glBegin(GL_POINTS);
    glVertex2d(X, Y);
    glEnd();
    glColor3f(0.0, 1.0, 0.0);
    mm(PR, x1);
    mm(PR, x2);
    mm(PR, x3);
    mm(PR, x4);
    //  printf("%lf%lf%lf%lf%lf%lf%lf%lf", x1[0], x1[1], x2[0], x2[1], x3[0],
x3[1], x4[0], x4[1]);
    printf("%f%f%f%f%f%f%f%f", x1[0], x1[1], x2[0], x2[1], x3[0], x3[1], x4[0],
x4[1]);

    obj(x1[0], x1[1], x4[0], x4[1], x3[0], x3[1], x2[0], x2[1]);
    // obj(x1[0],x1[1],x2[0],x2[1],x3[0],x3[1],x4[0],x4[1]);
    // SCALING
    x1[0] = -50;
    x1[1] = -50;
    x2[0] = -100;
    x2[1] = -50;
    x3[0] = -100;
    x3[1] = -100;
    x4[0] = -50;
    x4[1] = -100;
    glColor3f(1.0, 0.0, 1.0);
    obj(x1[0], x1[1], x2[0], x2[1], x3[0], x3[1], x4[0], x4[1]);
    output(-100, -40, "originalB:");
    double S[3][3] = {{2, 0, 0}, {0, 2, 0}, {0, 0, 1}};
    mm(S, x1);
    mm(S, x2);
    mm(S, x3);
    mm(S, x4);
    obj(x1[0], x1[1], x2[0], x2[1], x3[0], x3[1], x4[0], x4[1]);
    output(-200, -90, "scaledB:");

    // PIVOTSCALING
    int px = -200, py = -200;
    glColor3f(1.0, 1.0, 1.0);
```

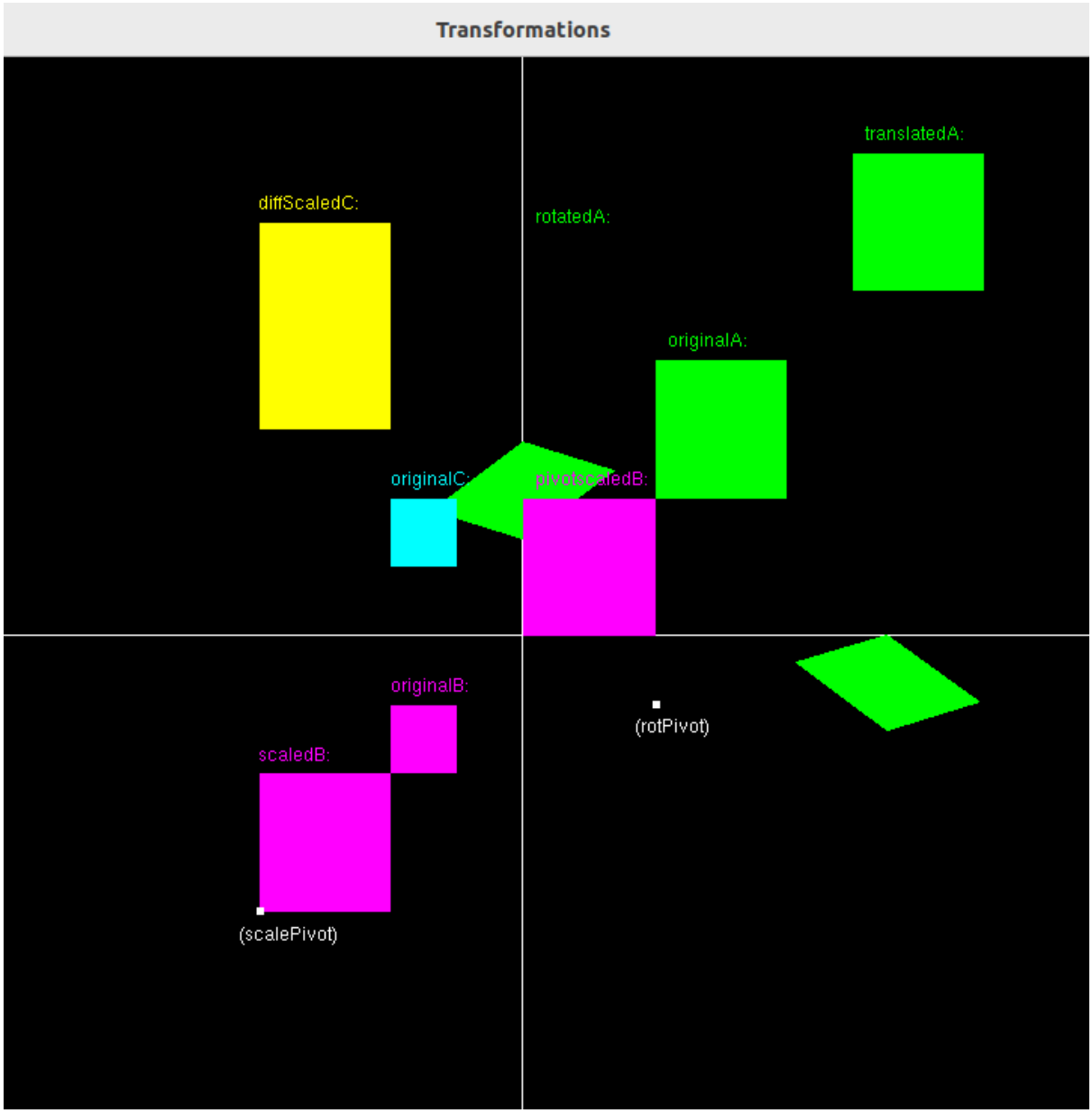
```
    output(px - 15, py - 20, "(scalePivot)");
    glPointSize(5);
    glBegin(GL_POINTS);
    glVertex2d(px, py);
    glEnd();
    glColor3f(1.0, 0.0, 1.0);
    x1[0] = -50;
    x1[1] = -50;
    x2[0] = -100;
    x2[1] = -50;
    x3[0] = -100;
    x3[1] = -100;
    x4[0] = -50;
    x4[1] = -100;
    double PS[3][3] = {{2, 0, 200}, {0, 2, 200}, {0, 0, 1}};
    mm(PS, x1);
    mm(PS, x2);
    mm(PS, x3);
    mm(PS, x4);
    obj(x1[0], x1[1], x2[0], x2[1], x3[0], x3[1], x4[0], x4[1]);
    output(10, 110, "pivotscaledB:");

    // DIISCALING
    glColor3f(0.0, 1.0, 1.0);
    obj(-50, 50, -100, 50, -100, 100, -50, 100);
    output(-100, 110, "originalC:");
    int dsx = 2, dsy = 3.5;
    glColor3f(1.0, 1.0, 0.0);
    obj(-50 * dsx, 50 * dsy, -100 * dsx, 50 * dsy, -100 * dsx, 100 * dsy, -50 *
dsx, 100 * dsy);
    output(-200, 310, "diffScaledC:");

    glFlush();
}
int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(1366, 768);
    glutInitWindowPosition(0, 0);
    glutCreateWindow("Transformations");
    myInit();
    glutDisplayFunc(display);
    glutMainLoop();
}
```

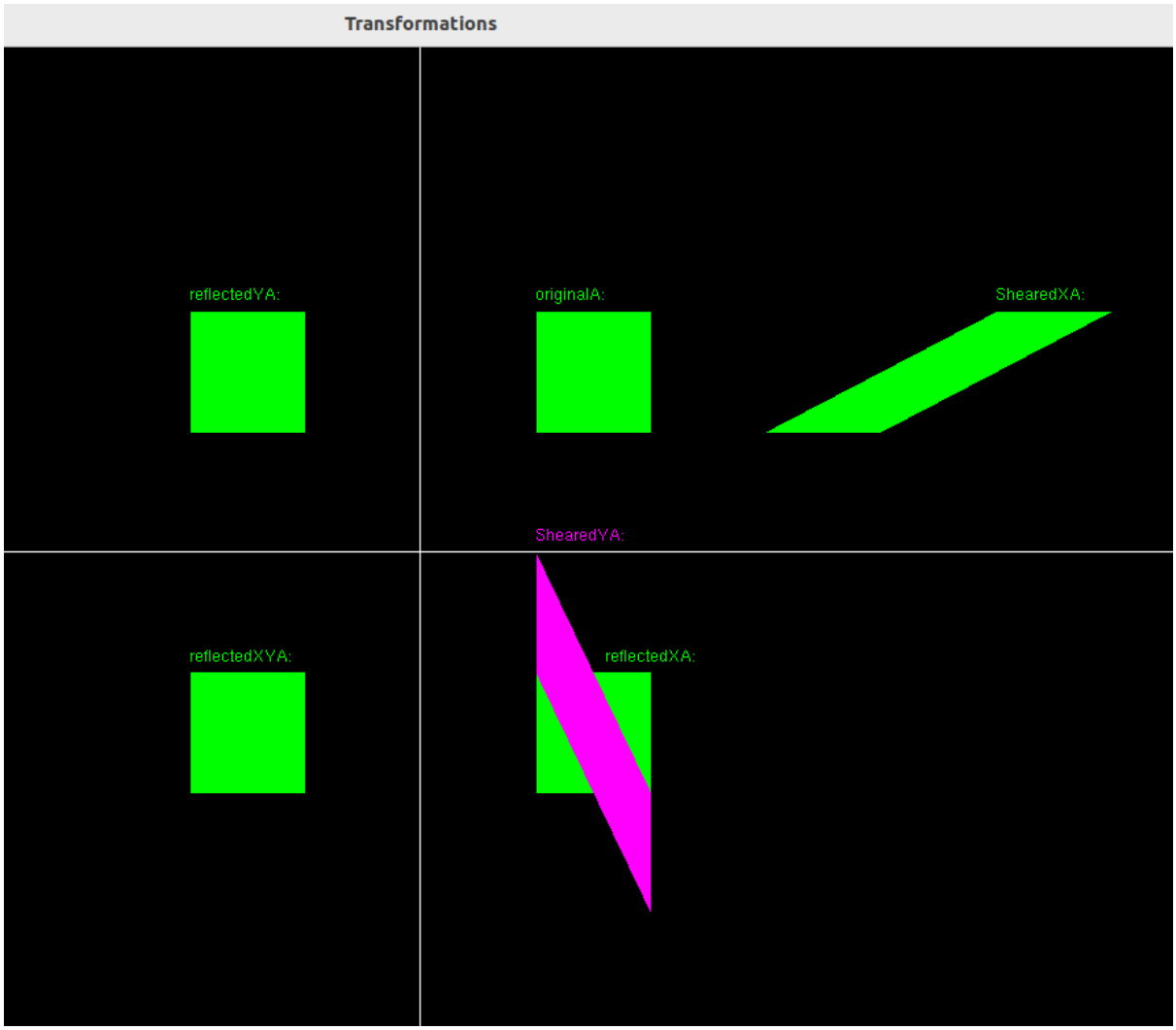
5b.cpp:

run.sh:

```
g++ 5.cpp -lGL -lglut -lGLU
./a.out
```

Sample I/O:





Learning Outcomes:

Learnt to do composite transformations. Learnt to do translation, reflection, shearing, rotation and scaling.

Shivcharan T
205001100
CSE - B

SSN COLLEGE OF ENGINEERING, KALAVAKKAM
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
UCS1712 – GRAPHICS AND MULTIMEDIA LAB

Lab Exercise 6: 2D Composite Transformations and Windowing in C++ using OpenGL

a) To compute the composite transformation matrix for any 2 transformations given as input by
the user and applying it on the object.
The transformation can be any combination of the following.
1) Translation
2) Rotation
3) Scaling
4) Reflection
5) Shearing
Display the original and the transformed object.
Calculate the final transformation matrix by multiplying the two individual transformation matrices and then apply it to the object.
Note: Use Homogeneous coordinate representations and matrix multiplication to perform transformations. Divide the output window into four quadrants. (Use LINES primitive to draw x
and y axis)
b) Create a window with any 2D object and a different sized viewport. Apply window to viewport
transformation on the object. Display both window and viewport.

Aim:
To implement Composite 2D transformations on objects and windowing using C++ using OpenGL

Algorithm:

6a.cpp:

1. Get points of the object as input.
2. Draw the object.
3. Transform each vertex of the object.
4. Draw the object with the transformed vertices.

6b.cpp:

1. Store the window dimensions and the viewport dimensions.
2. Get points of the object as input and draw it on the window.
3. Apply window to viewport transformation on the object as:
    a. Sx = (xvmax - xvmin) / (xwmax - xwmin)

b. xv = xvmin + (xw - xwmin) * Sx

c. Similarly, for the y-coordinates.

4. Draw the object on the viewport.

Code:

6a.cpp:

```
#include <GL/glut.h>
#include <iostream>
#include <vector>
#include <cmath>
#include <cstring>
#include <stdio.h>
#define pi M_PI

using namespace std;
void myInit()
{
    glClearColor(0.5, 1.0, 1.0, 0.0);
    glColor3f(0.0f, 0.0f, 0.0f);
    glPointSize(1);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(-320.0, 320.0, -240.0, 240.0);
}
vector<vector<float>> translation()
{
    float tx, ty;
    cout << "Enter tx, ty: ";
    cin >> tx >> ty;
    vector<vector<float>> translate(3, vector<float>(3, 0.0));
    (translate)[0][0] = 1;
    (translate)[0][2] = tx;
    (translate)[1][1] = 1;
    (translate)[1][2] = ty;
    (translate)[2][2] = 1;
    return translate;
}
vector<vector<float>> rotate()
{
    float deg;
    cout << "Enter deg: ";
    cin >> deg;
    vector<vector<float>> rotate(3, vector<float>(3, 0.0));
    deg *= M_PI / 180;
    cout << deg << " : deg" << endl;
    rotate[0][0] = cos(deg);
    rotate[0][1] = -sin(deg);
    rotate[1][0] = sin(deg);
    rotate[1][1] = cos(deg);
    rotate[2][2] = 1;
    // rotate[0][2] = tx*(1-cos(deg))+ty*sin(deg);
```

```
    // rotate[1][2] = ty*(1-cos(deg))-tx*sin(deg);
    return rotate;
}
vector<vector<float>> scale()
{
    float sx, sy;
    cout << "Enter sx, sy: ";
    cin >> sx >> sy;
    vector<vector<float>> scale(3, vector<float>(3, 0.0));
    scale[0][0] = sx;
    scale[1][1] = sy;
    scale[2][2] = 1;
    // scale[0][2] = tx * (1 - sx);
    // scale[1][2] = ty * (1 - sy);
    return scale;
}
vector<vector<float>> reflect()
{
    float axis;
    cout << "Enter option 1.x-axis 2.y-axis 3.origin 4.x=y (1/2/3/4): ";
    cin >> axis;
    vector<vector<float>> reflect(3, vector<float>(3, 0.0));
    reflect[0][0] = 1;
    reflect[1][1] = 1;
    reflect[2][2] = 1;
    if (axis == 1)
        reflect[1][1] = -1;
    else if (axis == 2)
        reflect[0][0] = -1;
    else if (axis == 3)
    {
        reflect[0][0] = -1;
        reflect[1][1] = -1;
    }
    else if (axis == 4)
    {
        reflect[0][1] = 1;
        reflect[0][0] = 0;
        reflect[1][0] = 1;
        reflect[1][1] = 0;
    }
    return reflect;
}
vector<vector<float>> shear()
{
    float op;
    cout << "Enter option 1.x-shear 2.y-shear (1/2): ";
    cin >> op;
    float sh, ref;
    if (op == 1)
        cout << "Enter shx, yref: ";
    else if (op == 2)
        cout << "Enter shy, xref: ";
    cin >> sh >> ref;
    vector<vector<float>> shear(3, vector<float>(3, 0.0));
```

```
    shear[0][0] = 1;
    shear[1][1] = 1;
    shear[2][2] = 1;
    if (op == 1)
    {
        shear[0][1] = sh;
        shear[0][2] = -sh * ref;
    }
    else if (op == 2)
    {
        shear[1][0] = sh;
        shear[1][2] = -sh * ref;
    }
    return shear;
}
vector<vector<float>> matrixMul(vector<vector<float>> t1,
                                vector<vector<float>>  t2,  vector<vector<float>>
res, int n)
{
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < n; j++)
        {
            res[i][j] = 0;
            for (int k = 0; k < 3; k++)
            {
                res[i][j] += t1[i][k] * t2[k][j];
            }
        }
    }
    return res;
}
void matrixDisp(vector<vector<float>> m)
{
    cout << endl;
    for (auto arrp : m)
    {
        for (auto p : arrp)
        {
            cout << p << " ";
        }
        cout << endl;
    }
}
void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    int op1, op2;
    cout << "Enter any 2 tranformations:- \n1.translation \n2.rotation\n3.scaling
\n4.reflection \n5.shearing(1 / 2 / 3 / 4 / 5) \ninc order(op1, op2) : ";
    cin >> op1 >> op2;
    vector<vector<float>> t1, t2;
    if (op1 == 1)
    {
        t1 = translation();
```

```
}
else if (op1 == 2)
{
    t1 = rotate();
}
else if (op1 == 3)
{
    t1 = scale();
}
else if (op1 == 4)
{
    t1 = reflect();
}
else if (op1 == 5)
{
    t1 = shear();
}
// for op2
if (op2 == 1)
{
    t2 = translation();
}
else if (op2 == 2)
{
    t2 = rotate();
}
else if (op2 == 3)
{
    t2 = scale();
}
else if (op2 == 4)
{
    t2 = reflect();
}
else if (op2 == 5)
{
    t2 = shear();
}
for (auto a : t1)
{
    for (auto x : a)
    {
        cout << x << " ";
    }
    cout << endl;
}
for (auto a : t2)
{
    for (auto x : a)
    {
        cout << x << " ";
    }
    cout << endl;
}
int n;
```

```
    cout << "Enter no. of points for polygon: ";
    cin >> n;
    // points matrix
    vector<vector<float>> points(3, vector<float>(n));
    for (int i = 0; i < n; i++)
    {
        cout << "Enter x, y coords: ";
        cin >> points[0][i] >> points[1][i];
        points[2][i] = 1;
    }
    // order for now is op1 then op2
    // result matrix
    vector<vector<float>> res(3, vector<float>(n));
    // t2 x t1
    res = matrixMul(t2, t1, res, 3);
    matrixDisp(res);
    // t21 x points
    res = matrixMul(res, points, res, n);
    matrixDisp(res);
    // axis
    glBegin(GL_LINES);
    glVertex2d(-320, 0);
    glVertex2d(320, 0);
    glVertex2d(0, -240);
    glVertex2d(0, 240);
    glEnd();
    // original shape
    glBegin(GL_LINE_LOOP);
    for (int i = 0; i < n; i++)
    {
        glVertex2f(points[0][i], points[1][i]);
    }
    glEnd();
    // result shape plot
    glRasterPos2i(res[0][n / 2], res[1][n / 2] - 15);
    glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, int('S'));
    glBegin(GL_LINE_LOOP);
    glColor3f(1.0f, 0.0f, 0.0f);
    for (int i = 0; i < n; i++)
    {
        glVertex2f(res[0][i], res[1][i]);
    }
    glEnd();
    glFlush();
}
int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(640, 480);
    glutCreateWindow("ex6");
    glutDisplayFunc(display);
    myInit();
    glutMainLoop();
    return 0;
```

```
}
```

6b.cpp:

```
#include <cmath>
#include <cstring>
#include <stdio.h>
#include <GL/glut.h>
using namespace std;

// screen dimensions
const int windowWidth = 1300;
const int windowHeight = 1300;

void myInit(void)
{
    glClearColor(0.0, 0.0, 0.0, 1.0);
    glPointSize(1.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(-windowWidth / 2, windowWidth / 2, -windowHeight / 2, windowHeight
/ 2);
    // glViewport(0, 0, windowWidth, windowHeight);
}

void mykey(unsigned char key, int x, int y)
{
    switch (key)
    {
    case 27:
        exit(0);
    }
}

// Just to draw a point
void draw_pixel(int x, int y)
{
    glPointSize(1.0); // Specify point thickness
    glBegin(GL_POINTS);
    glVertex2i(x, y);
    glEnd();
}

void obj(int a, int b, int c, int d, int e, int f)
{
    glBegin(GL_POLYGON);
    glVertex2d(a, b);
    glVertex2d(c, d);
    glVertex2d(e, f);
    glEnd();
}

// window to viewport transformation
```

```
void wov(int *x, int *y, int x_wmax,
         int y_wmax, int x_wmin, int y_wmin,
         int x_vmax, int y_vmax, int x_vmin,
         int y_vmin)
{
    // point on viewport
    int x_v, y_v;

    // scaling factors for x coordinate and y coordinate
    float sx, sy;

    // calculating Sx and Sy
    sx = (float)(x_vmax - x_vmin) / (x_wmax - x_wmin);
    sy = (float)(y_vmax - y_vmin) / (y_wmax - y_wmin);

    // calculating the point on viewport
    x_v = x_vmin + (float)((*x - x_wmin) * sx);
    y_v = y_vmin + (float)((*y - y_wmin) * sy);

    *x = x_v;
    *y = y_v;
}

void display1(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    // Green
    glColor3f(0.0, 1.0, 0.0);

    // Call function
    obj(-200, 150, 500, 150, -400, -450);

    // White
    glColor3f(1.0, 1.0, 1.0);

    glFlush();
     glutSwapBuffers();
}

void display2(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    int xmin = -375, xmax = 525, ymin = -200, ymax = 600;

    // Green
    glColor3f(0.0, 1.0, 0.0);

    // Call function
    int x1[2], x2[2], x3[2];
    x1[0] = -200, x1[1] = 150, x2[0] = 500, x2[1] = 150, x3[0] = -400, x3[1] = -
450;

    // Red
```

```
    glColor3f(1.0, 0.0, 0.0);
    glBegin(GL_LINES);
    glVertex2i(xmin, ymax);
    glVertex2i(xmax, ymax);
    glEnd();
    glBegin(GL_LINES);
    glVertex2i(xmin, ymax);
    glVertex2i(xmin, ymin);
    glEnd();
    glBegin(GL_LINES);
    glVertex2i(xmin, ymin);
    glVertex2i(xmax, ymin);
    glEnd();
    glBegin(GL_LINES);
    glVertex2i(xmax, ymax);
    glVertex2i(xmax, ymin);
    glEnd();

    // Green
    glColor3f(0.0, 1.0, 0.0);
    x1[0] = -200, x1[1] = 150, x2[0] = 500, x2[1] = 150, x3[0] = -400, x3[1] = -
450;
    wov(&x1[0], &x1[1], windowHeight / 2, windowWidth / 2, -windowHeight / 2, -
windowWidth / 2, xmax, ymax, xmin, ymin);
    wov(&x2[0], &x2[1], windowHeight / 2, windowWidth / 2, -windowHeight / 2, -
windowWidth / 2, xmax, ymax, xmin, ymin);
    wov(&x3[0], &x3[1], windowHeight / 2, windowWidth / 2, -windowHeight / 2, -
windowWidth / 2, xmax, ymax, xmin, ymin);
    obj(x1[0], x1[1], x2[0], x2[1], x3[0], x3[1]);

    glFlush();
     glutSwapBuffers();
}

int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize(windowWidth, windowHeight);

    glutInitWindowPosition(0, 0);
    glutCreateWindow("Window");
    // glutReshapeFunc(handleResize);
    glutDisplayFunc(display1);

    myInit();
    glutKeyboardFunc(mykey);

    glutInitWindowPosition(500, 500);
    glutCreateWindow("Viewport");
    // glutReshapeFunc(handleResize);
    glutDisplayFunc(display2);

    myInit();
    glutKeyboardFunc(mykey);
```

```
    glutMainLoop();
}
```

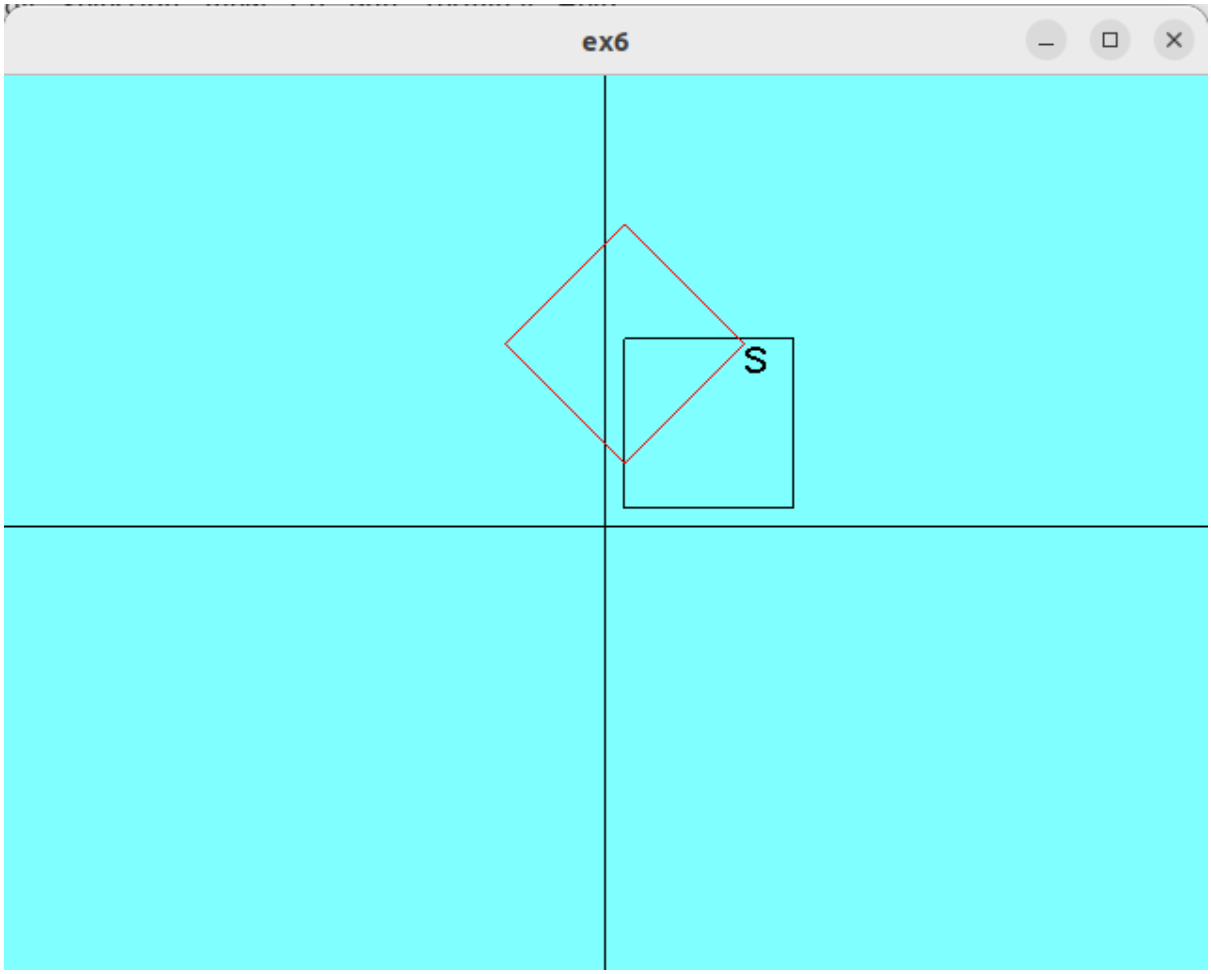
run.sh:

```
g++ 6.cpp -lGL -lglut -lGLU
./a.out
```

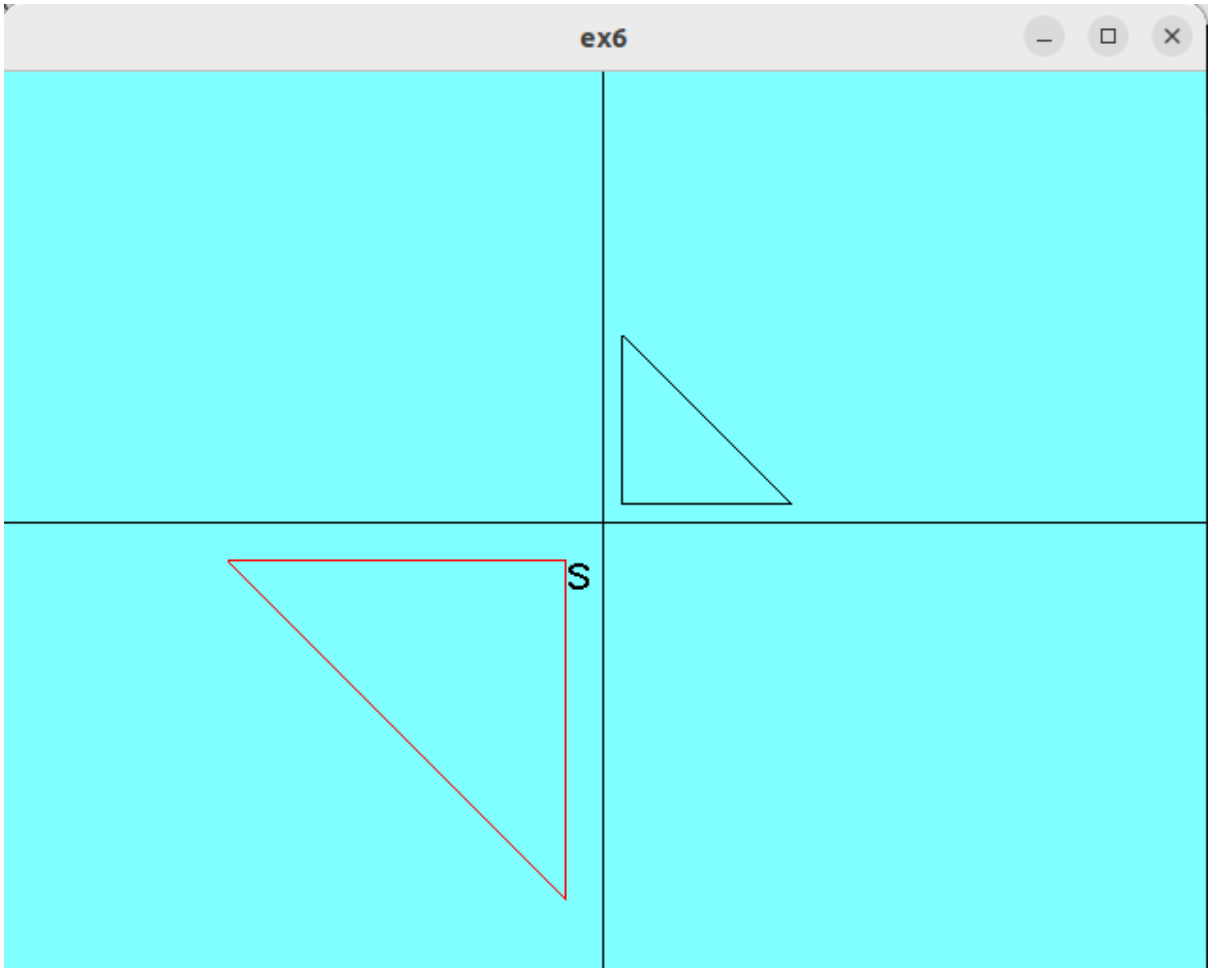
Sample I/O:

6a.cpp:

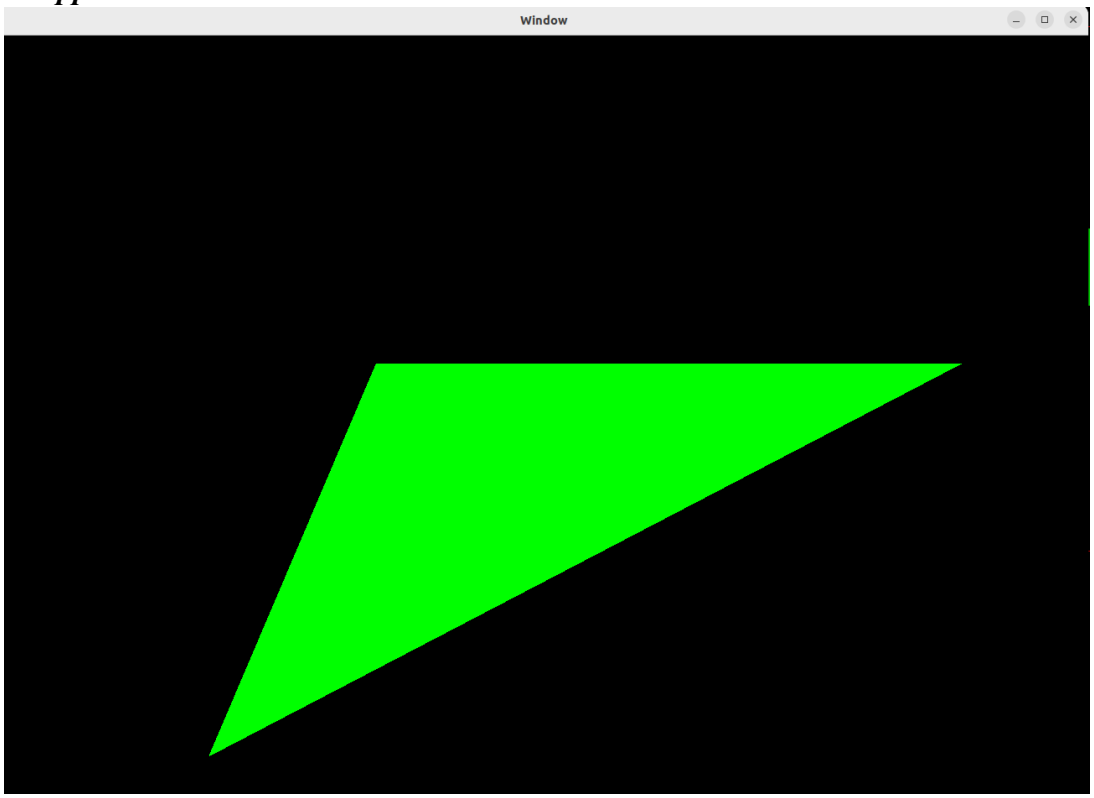
```
Enter any 2 tranformations:-
1.translation
2.rotation
3.scaling
4.reflection
5.shearing(1 / 2 / 3 / 4 / 5)
inc order(op1, op2) : 2 1
Enter deg: 45
0.785398 : deg
Enter tx, ty: 10 20
0.707107 -0.707107 0
0.707107 0.707107 0
0 0 1
1 0 10
0 1 20
0 0 1
Enter no. of points for polygon: 4
Enter x, y coords: 10 100
Enter x, y coords: 100 100
Enter x, y coords: 100 10
Enter x, y coords: 10 10

0.707107 -0.707107 10 0
0.707107 0.707107 20 0
0 0 1 0

-53.6396 10 73.6396 10
97.7817 161.421 97.7817 34.1421
1 1 1 1
```

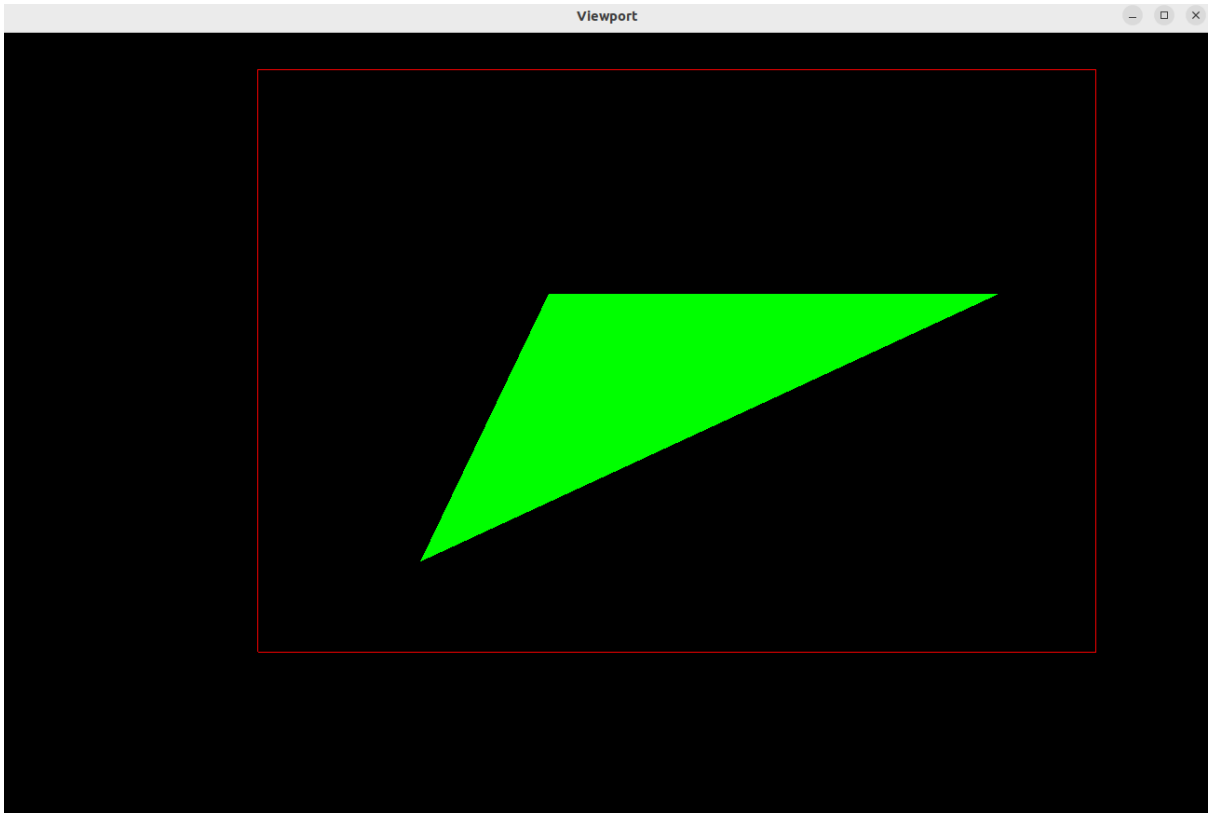

ex6
S

```
cse1100@brokolee:~/SSN/sem7/GML/6$ ./run.sh
Enter any 2 tranformations:-
1.translation
2.rotation
3.scaling
4.reflection
5.shearing(1 / 2 / 3 / 4 / 5)
inc order(op1, op2) : 3 4
Enter sx, sy: 2 2
Enter option 1.x-axis 2.y-axis 3.origin 4.x=y (1/2/3/4): 3
2 0 0
0 2 0
0 0 1
-1 0 0
0 -1 0
0 0 1
Enter no. of points for polygon: 3
Enter x, y coords: 10 100
Enter x, y coords: 10 10
Enter x, y coords: 100 10

-2 0 0
0 -2 0
0 0 1

-20 -20 -200
-200 -20 -20
1 1 1
```



6b.cpp:





Learning Outcomes:

Learnt to do composite transformations. Learnt to do translation, reflection, shearing, rotation and scaling.

Shivcharan T
205001100
CSE - B

SSN COLLEGE OF ENGINEERING, KALAVAKKAM
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
UCS1712 – GRAPHICS AND MULTIMEDIA LAB

Lab Exercise 7: Cohen Sutherland Line clipping in C++ using OpenGL

Apply Cohen Sutherland line clipping on a line (x1,y1) (x2,y2) with respect to a clipping window
(XWmin,YWmin) (XWmax,YWmax).
After clipping with respect to an edge, display the line segment with the calculated intermediate intersection points and the vertex list.
Input: The clipping window co-ordinates and the line endpoints
Note: The output should show the clipping window and the line to be clipped in different colors. You can show the intermediate steps using time delay.

Aim:

To implement Cohen Sutherland Line Clipping Algorithm in C++ using OpenGL

Algorithm:

1. Assign a region code for two endpoints of the given line.
2. If both endpoints have a region code 0000, then the given line is completely inside.
3. Else, perform the logical AND operation for both region codes.
   3.1. If the result is not 0000, then the given line is completely outside.
   3.2. Else, the line is partially inside.
      3.2.1. Choose an endpoint of the line that is outside the given rectangle.
      3.2.2. Find the intersection point of the rectangular boundary (based on the region code).
      3.2.3. Replace the endpoint with the intersection point and update the region code.
      3.2.4. Repeat step 2 until we find a clipped line either trivially accepted or trivially rejected.
4. Repeat step 1 for other lines.

Code:

5a.cpp:

```
#include <cmath>
#include <cstring>
#include <stdio.h>
#include <iostream>
#include <GL/glut.h>
using namespace std;

int i = 0;

// mode
bool hardCode = false;
```

```
// screen dimensions
const int windowWidth = 1000;
const int windowHeight = 1000;

// TBRL
const int INSIDE = 0;
const int LEFT = 1;
const int RIGHT = 2;
const int BOTTOM = 4;
const int TOP = 8;

GLfloat xmin, xmax;
GLfloat ymin, ymax;

typedef struct
{
    GLfloat x, y;
} Point;

Point p1, p2;

void swap_points(Point *p1, Point *p2)
{
    Point t = *p1;
    *p1 = *p2;
    *p2 = t;
}

void swap_codes(GLint *x, GLint *y)
{
    GLint t = *x;
    *x = *y;
    *y = t;
}

GLint inside(GLint code)
{
    return !code;
}

GLint accept(GLint code1, GLint code2)
{
    return !(code1 | code2);
}

GLint reject(GLint code1, GLint code2)
{
    return code1 & code2;
}

int encode(Point p)
{
    GLfloat x = p.x, y = p.y;
    int code = INSIDE; // initialised as being inside of clip window
```

```
    if (y > ymax) // above the clip window
        code |= TOP;
    else if (y < ymin) // below the clip window
        code |= BOTTOM;
    if (x > xmax) // to the right of clip window
        code |= RIGHT;
    else if (x < xmin) // to the left of clip window
        code |= LEFT;
    return code; // return the calculated code
}
void myInit(void)
{
    glClearColor(0.0, 0.0, 0.0, 1.0);
    glColor3f(0.0, 1.0, 0.0);
    glPointSize(1.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(-windowHeight / 2, windowHeight / 2, -windowWidth / 2, windowWidth
/ 2);
}

void draw_axis()
{
    glColor3f(1.0, 1.0, 1.0);
    glBegin(GL_LINE_LOOP);
    glVertex2d(-2000, 0);
    glVertex2d(2000, 0);

    glVertex2d(0, 2000);
    glVertex2d(0, -2000);
    glEnd();
    glFlush();
}

// GLint round(GLfloat a)
// {
//     return (GLint)(a + 0.5f);
// }

void line_clip(Point p1, Point p2)
{
    GLint code1, code2;
    GLint done = 0, plot_line = 0;
    GLfloat m = 0;
    if (p1.x != p2.x)
        m = (p2.y - p1.y) / (p2.x - p1.x);

    while (!done)
    {
        code1 = encode(p1);
        code2 = encode(p2);
        if (accept(code1, code2))
        {
            done = 1;
            plot_line = 1;
```

```
        }
        else if (reject(code1, code2))
        {
            done = 1;
        }
        else
        {
            if (inside(code1))
            {
                swap_points(&p1, &p2);
                swap_codes(&code1, &code2);
            }

            if (code1 & LEFT)
            {
                p1.y += (xmin - p1.x) * m;
                p1.x = xmin;
            }
            else if (code1 & RIGHT)
            {
                p1.y += (xmax - p1.x) * m;
                p1.x = xmax;
            }
            else if (code1 & BOTTOM)
            {
                if (p1.x != p2.x)
                    p1.x += (ymin - p1.y) / m;
                p1.y = ymin;
            }
            else if (code1 & TOP)
            {
                if (p1.x != p2.x)
                    p1.x += (ymax - p1.y) / m;
                p1.y = ymax;
            }
        }
    }

    if (plot_line)
    {
        glColor3f(1, 0, 0);
        glLineWidth(2);
        glBegin(GL_LINES);
        glVertex2i(round(p1.x), round(p1.y));
        glVertex2i(round(p2.x), round(p2.y));
        glEnd();
        glFlush();
    }
}

void draw_line()
{
    glBegin(GL_LINES);
    glVertex2i(round(p1.x), round(p1.y));
    glVertex2i(round(p2.x), round(p2.y));
```

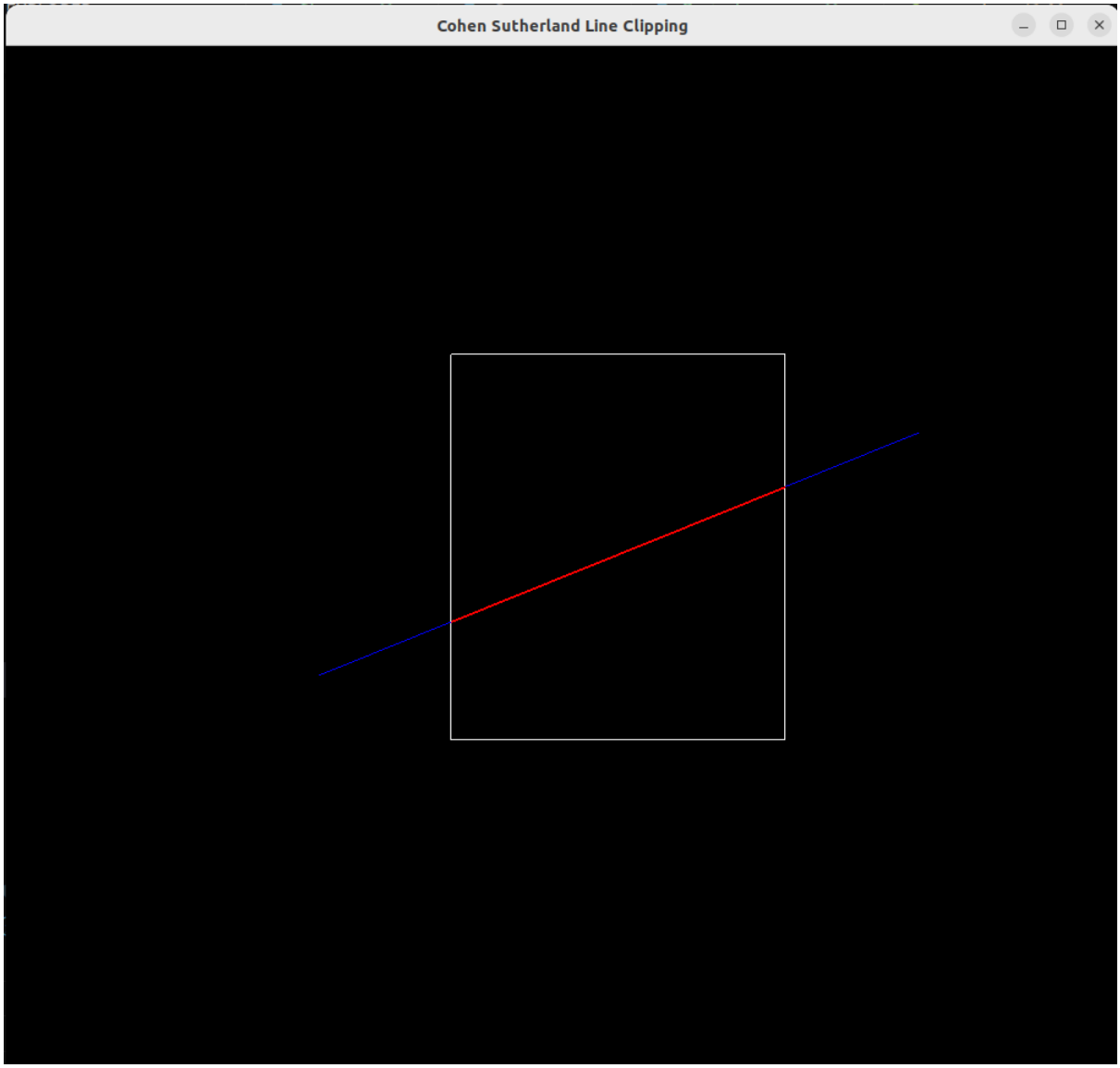
```
    glEnd();
    glFlush();
}

void draw_window()
{
    glColor3f(1, 1, 1);
    glBegin(GL_LINE_LOOP);
    glVertex2i(round(xmin), round(ymin));
    glVertex2i(round(xmin), round(ymax));
    glVertex2i(round(xmax), round(ymax));
    glVertex2i(round(xmax), round(ymin));
    glEnd();
    glFlush();
}

void mykey(unsigned char ch, int x, int y)
{
    if (ch == 'c')
    {
        // clip_flag = 1;
        // glutPostRedisplay();

        line_clip(p1, p2);
        glFlush();
    }
}

void mymouse(int button, int state, int x, int y)
{
    if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN && i < 2)
    {
        if (i == 0)
        {
            p1.x = x - windowWidth / 2;
            p1.y = windowHeight / 2 - y;
        }
        if (i == 1)
        {
            p2.x = x - windowWidth / 2;
            p2.y = windowHeight / 2 - y;
        }
        i++;
    }
    if (i == 2)
    {
        draw_line();
        i++;
    }
}

void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 1.0, 1.0);
```

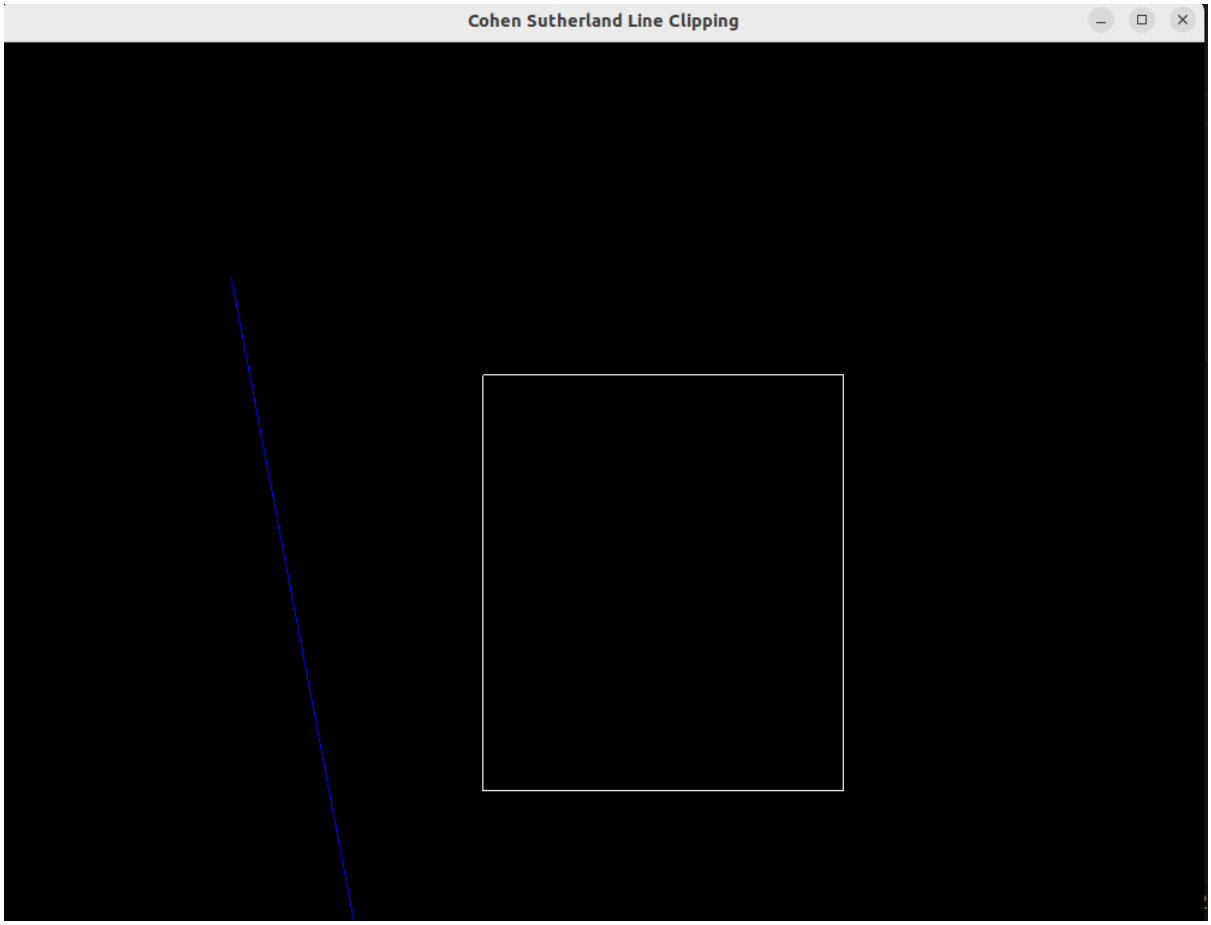
```
    xmin = -100;
    xmax = +200;
    ymin = -123;
    ymax = +223;

    draw_window();

    glColor3f(0, 0, 1);

    if (hardCode)
    {
        p1 = {-450, -500};
        p2 = {250, 310};
        draw_line();
        // line_clip(p1, p2);   //default
    }

    // line_clip(p1, p2);   //default
    glFlush();
}
int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(windowHeight, windowWidth);
    glutInitWindowPosition(0, 0);
    glutCreateWindow("Cohen Sutherland Line Clipping");
    myInit();
    glutDisplayFunc(display);
    glutKeyboardFunc(mykey);
    glutMouseFunc(mymouse);
    glutMainLoop();
    return 1;
}
```

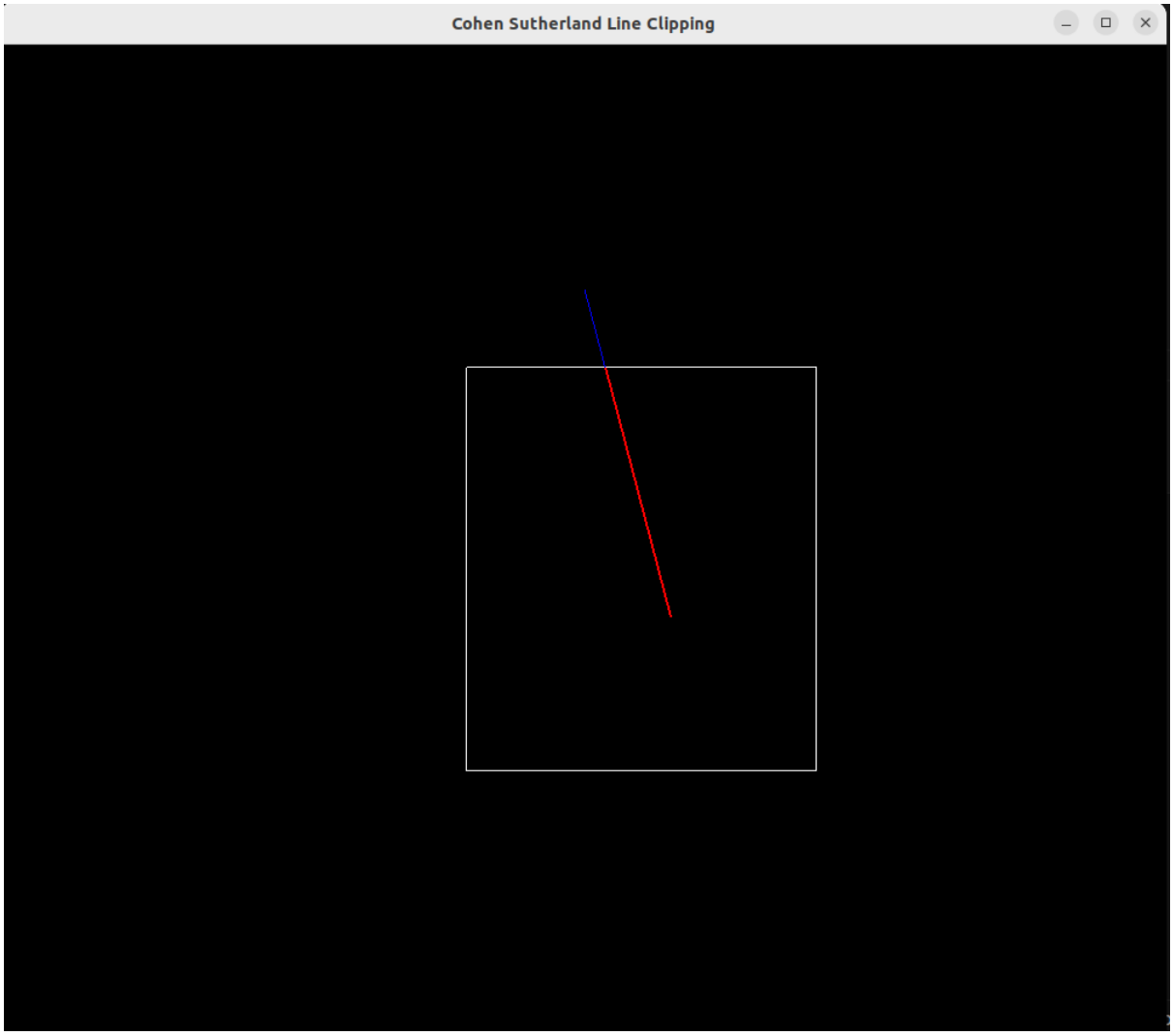
run.sh:

```
g++ 7.cpp -lGL -lglut -lGLU
./a.out
```

Sample I/O:


Cohen Sutherland Line Clipping


Cohen Sutherland Line Clipping



Learning Outcomes:

Learnt to do Line Clipping for a given window using Cohen Sutherland Line Clipping Algorithm.

Shivcharan T
205001100
CSE - B

SSN COLLEGE OF ENGINEERING, KALAVAKKAM
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
UCS1712 – GRAPHICS AND MULTIMEDIA LAB

Lab Exercise 8: : 3-Dimensional Transformations in C++ using OpenGL

Perform the following basic 3D Transformations on any 3D Object.
1) Translation
2) Rotation
3) Scaling
Use only homogeneous coordinate representation and matrix multiplication to perform transformations.
Set the camera to any position on the 3D space. Have (0,0,0) at the center of the screen. Draw X , Y and Z axis.

Aim:
To implement 3D transformations on objects using C++ using OpenGL

Algorithm:
1. Get points of the object as input.
2. Draw the object.
3. Transform each vertex of the object.
4. Draw the object with the transformed vertices.

Code:

8.cpp:

```
#include <stdio.h>
#include <GL/glut.h> //Change to <GLUT/glut.h> in Mac
#include <math.h>
#include <string.h>
#include <iostream>
using namespace std;
#define pi 3.142857

typedef float Matrix4[4][4];
Matrix4 theMatrix;
static GLfloat input[8][3] = {{40, 40, -50}, {90, 40, -50}, {90, 90, -50}, {40, 90,
-50}, {30, 30, 0}, {80, 30, 0}, {80, 80, 0}, {30, 80, 0}};
float output[8][3];
float tx = 100, ty = 100, tz = 100;
float sx = -2, sy = 2, sz = 2;
float angle = 60;
int choice, choiceRot;
void setIdentityM(Matrix4 m)
{
  for (int i = 0; i < 4; i++)
    for (int j = 0; j < 4; j++)
```

```
      m[i][j] = (i == j);
}

// PUT SOME FUNCTION HERE

void translate(int tx, int ty, int tz)
{
  for (int i = 0; i < 8; i++)
  {

    output[i][0] = input[i][0] + tx;
    output[i][1] = input[i][1] + ty;
    output[i][2] = input[i][2] + tz;
  }
}
void scale(int sx, int sy, int sz)
{
  theMatrix[0][0] = sx;
  theMatrix[1][1] = sy;
  theMatrix[2][2] = sz;
}
void RotateX(float angle)
{
  angle = angle * 3.142 / 180;
  theMatrix[1][1] = cos(angle);
  theMatrix[1][2] = -sin(angle);
  theMatrix[2][1] = sin(angle);
  theMatrix[2][2] = cos(angle);
}
void RotateY(float angle)
{
  angle = angle * 3.14 / 180;
  theMatrix[0][0] = cos(angle);
  theMatrix[0][2] = -sin(angle);
  theMatrix[2][0] = sin(angle);
  theMatrix[2][2] = cos(angle);
}
void RotateZ(float angle)
{
  angle = angle * 3.14 / 180;
  theMatrix[0][0] = cos(angle);
  theMatrix[0][1] = sin(angle);
  theMatrix[1][0] = -sin(angle);
  theMatrix[1][1] = cos(angle);
}
void multiplyM()
{
  for (int i = 0; i < 8; i++)
  {
    for (int j = 0; j < 3; j++)
    {
      output[i][j] = 0;
      for (int k = 0; k < 3; k++)
      {
        output[i][j] = output[i][j] + input[i][k] * theMatrix[k][j];
```

```
      }
    }
  }
}

// To draw the solid
void draw(float a[8][3])
{
  glBegin(GL_QUADS);
  glColor3f(0.7, 0.4, 0.5); // behind
  glVertex3fv(a[0]);
  glVertex3fv(a[1]);
  glVertex3fv(a[2]);
  glVertex3fv(a[3]);
  glColor3f(0.8, 0.2, 0.4); // bottom
  glVertex3fv(a[0]);
  glVertex3fv(a[1]);
  glVertex3fv(a[5]);
  glVertex3fv(a[4]);
  glColor3f(0.3, 0.6, 0.7); // left
  glVertex3fv(a[0]);
  glVertex3fv(a[4]);
  glVertex3fv(a[7]);
  glVertex3fv(a[3]);
  glColor3f(0.2, 0.8, 0.2); // right
  glVertex3fv(a[1]);
  glVertex3fv(a[2]);
  glVertex3fv(a[6]);
  glVertex3fv(a[5]);
  glColor3f(0.7, 0.7, 0.2); // up
  glVertex3fv(a[2]);
  glVertex3fv(a[3]);
  glVertex3fv(a[7]);
  glVertex3fv(a[6]);
  glColor3f(1.0, 0.1, 0.1);
  glVertex3fv(a[4]);
  glVertex3fv(a[5]);
  glVertex3fv(a[6]);
  glVertex3fv(a[7]);
  glEnd();
}

/* This is just to call the functions
 also draw X and Y axis here and use output to label stuff) */
void display(void)
{

  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  // black
  glColor3f(0.0, 0.0, 0.0);

  gluLookAt(0, 0, 1, 0, 0, 0, 0, 1, 0); // Camera, Center & Up Vector

  glBegin(GL_LINES); // Plotting X-Axis
  glVertex3d(-1000, 0, 0);
```

```
  glVertex3d(1000, 0, 0);
  glEnd();
  glBegin(GL_LINES); // Plotting Y-Axis
  glVertex3d(0, -1000, 0);
  glVertex3d(0, 1000, 0);
  glEnd();
  glBegin(GL_LINES); // Plotting Z-Axis
  glVertex3d(0, 0, -1000);
  glVertex3d(0, 0, 1000);
  glEnd();

  // Call function
  draw(input);
  setIdentityM(theMatrix);
  switch (choice)
  {
  case 1:
    translate(tx, ty, tz);
    break;
  case 2:
    scale(sx, sy, sz);
    multiplyM();
    break;
  case 3:
    switch (choiceRot)
    {
    case 1:
      RotateX(angle);
      break;
    case 2:
      RotateY(angle);
      break;
    case 3:
      RotateZ(angle);
      break;
    default:
      break;
    }
    multiplyM();
    break;
  }
  // gluLookAt(1, 0, 0, 0, 0, 0, 0, 1, 0); // Camera, Center & Up Vector
  draw(output);
  glFlush();

  glFlush();
}

int main(int argc, char **argv)
{
  /*--------WINDOW INITS-------*/
  glutInit(&argc, argv); // Mandatory. Initializes the GLUT library.
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
  glutInitWindowSize(1380, 700);          // Set the size of output window (kinda
optional)
```

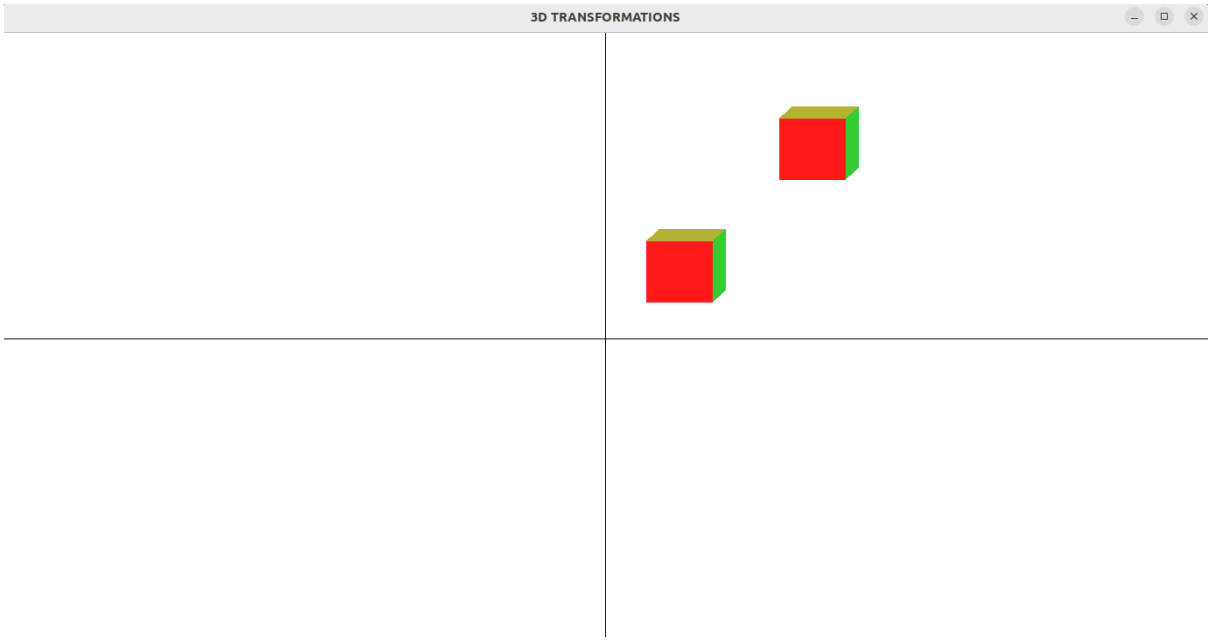
```
  glutInitWindowPosition(200, 200);        // position of output window on screen
(optional)
  glutCreateWindow("3D TRANSFORMATIONS"); // Giving name to window

  /*-------OTHER INITS-------*/
  glClearColor(1.0, 1.0, 1.0, 1.0); // Clear the buffer values for color AND set
these values
  /*can set initial color here also*/
  glMatrixMode(GL_PROJECTION); // Uses something called "projection matrix" to
represent
  glLoadIdentity();           // load the above matrix to fill with identity values
  glOrtho(-454.0, 454.0, -250.0, 250.0, -250.0, 250.0);
  gluPerspective(100, 100, 100, 100);
  glEnable(GL_DEPTH_TEST);
  cout << "Enter your choice number:\n1.Translation\n2.Scaling\n3.Rotation\n=>";
  cin >> choice;
  switch (choice)
  {
  case 1:
    break;
  case 2:
    break;
  case 3:
    cout << "Enter your choice for Rotation about axis:\n1.parallel to X-axis."
         << "(y& z)\n2.parallel to Y-axis.(x& z)\n3.parallel to Z-axis."
         << "(x& y)\n =>";
    cin >> choiceRot;
    break;
  default:
    break;
  }
  glutDisplayFunc(display); // sets the display callback for the current window
  glutMainLoop();           // Enters event processing loop. Compulsory
  return 0;
}
```

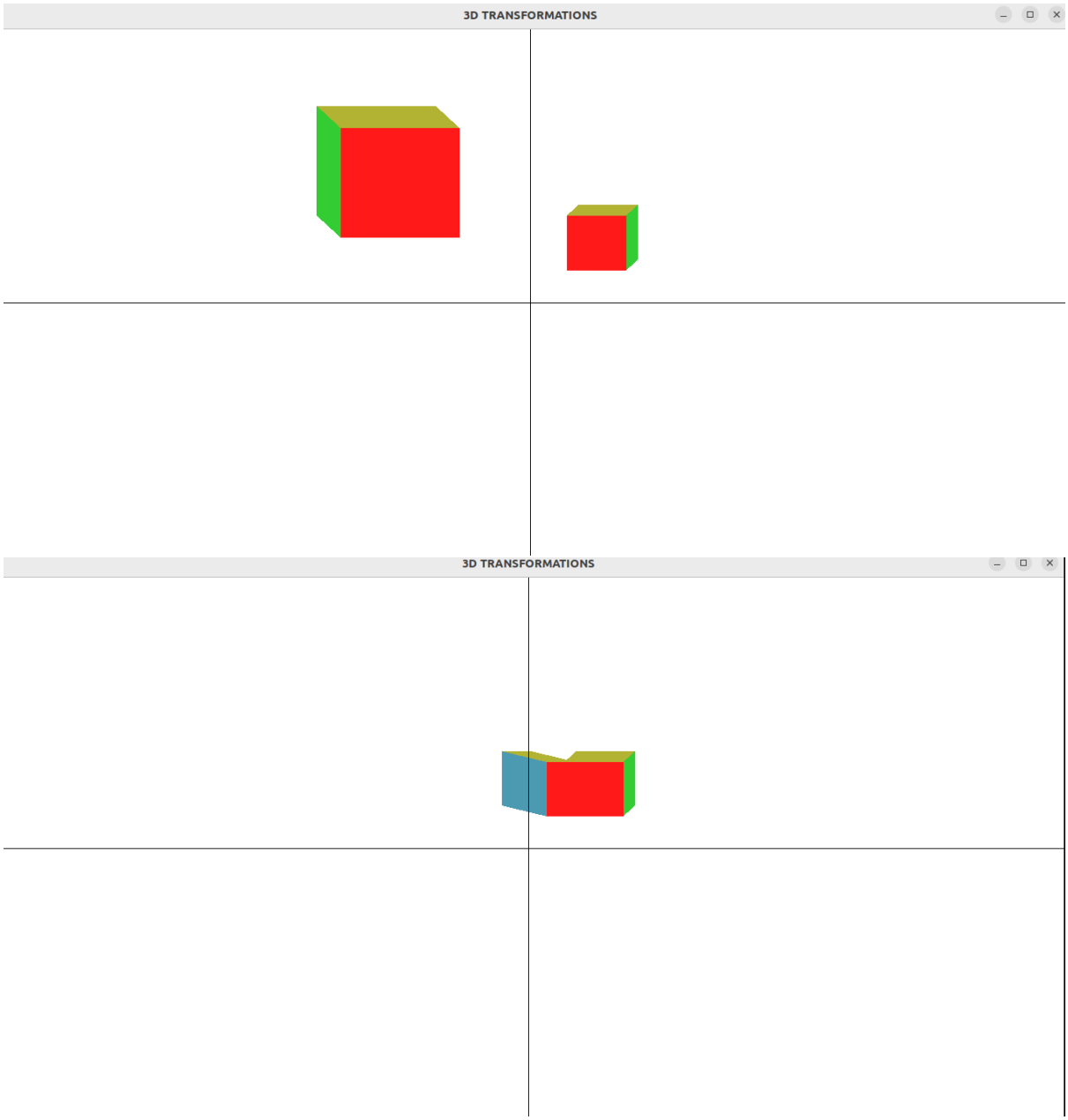
run.sh:

```
g++ 8.cpp -lGL -lglut -lGLU
./a.out
```

Sample I/O:

```
cse1100@brokolee:~/SSN/sem7/GML/8$ ./run.sh
Enter your choice number:
1.Translation
2.Scaling
3.Rotation
=>1
^Ccse1100@brokolee:~/SSN/sem7/GML/8$ ./run.sh
Enter your choice number:
1.Translation
2.Scaling
3.Rotation
=>2
^Ccse1100@brokolee:~/SSN/sem7/GML/8$ ./run.sh
Enter your choice number:
1.Translation
2.Scaling
3.Rotation
=>3
Enter your choice for Rotation about axis:
1.parallel to X-axis.(y& z)
2.parallel to Y-axis.(x& z)
3.parallel to Z-axis.(x& y)
 =>2
```





Learning Outcomes:

Thus, 3D Transformations has been implemented on objects using OpenGL.

Shivcharan T
205001100
CSE - B

SSN COLLEGE OF ENGINEERING, KALAVAKKAM
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
UCS1712 – GRAPHICS AND MULTIMEDIA LAB

Lab Exercise 9: : 3-Dimensional Projections in C++ using OpenGL

Write a menu driven program to perform Orthographic parallel projection and Perspective projection on any 3D object. Set the camera to any position on the 3D space. Have (0,0,0) at the center of the screen. Draw X, Y and Z axis. You can use gluPerspective() to perform perspective projection. Use keyboard functions to rotate and show different views of the object. [Can use built-in functions for 3D transformations].

Aim:
To implement 3D projections on objects using C++ using OpenGL

Algorithm:

1. Include necessary OpenGL and GLUT libraries.

2. Declare global variables for rotation angles and camera position.

3. Specify the vertices of the 3D object.

4. Implement a function to draw X, Y, and Z axes.

5. Create a function to draw the 3D object using glBegin(GL_QUADS) and glVertex3fv.

6. Set up the display function. Use gluLookAt for camera positioning, incorporate rotation transformations using glRotatef, call drawAxes and drawObject within the display function, and use glutSwapBuffers to swap the front and back buffers.

7. Implement a keyboard function to handle user input. Update rotation angles based on key presses (e.g., 'x', 'y', 'z') and allow the user to exit the program (e.g., 'q').

8. Initialize OpenGL and GLUT, set up the window and callback functions for display and keyboard input, configure perspective or orthographic projection, and enter the main event loop.

Code:

9.cpp:

```
#include <iostream>
#include <stdio.h>
#include <cmath>
#include <cstring>
#include <GL/glut.h>
```

```
using namespace std;

// Global constants
const float windowHeight = 1000;
const float windowWidth = 1000;

const float X_MIN = -500;
const float X_MAX = 500;
const float Y_MIN = -500;
const float Y_MAX = 500;
const int FPS = 60;

// Global variables to handle rotation
GLfloat x_rotate = 0;
GLfloat y_rotate = 0;

// Global variable for projection
bool isOrthoProjection = true;
void initializeDisplay();
void keyboardKeys(unsigned char key, int x, int y);
void drawAxes();

void myInit(void)
{
    glClearColor(0.0, 0.0, 0.0, 1.0);
    glColor3f(0.0, 0.0, 1.0);
    glPointSize(1.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(-windowWidth / 2, windowWidth / 2, -windowHeight / 2, windowHeight
/ 2);
}

void drawAxes()
{
    // To draw X and Y axis
    glColor3d(1, 0, 0);
    glBegin(GL_LINES);
    glVertex2f(-2, 0);
    glVertex2f(2, 0);
    glVertex2f(0, -2);
    glVertex2f(0, 2);
    glEnd();
    glFlush();
}

void keyboardKeys(unsigned char key, int x, int y)
{
    // Callback function for keyboard interactivity
    key = tolower(key);
    switch (key)
    {
    case 'w':
    {
```

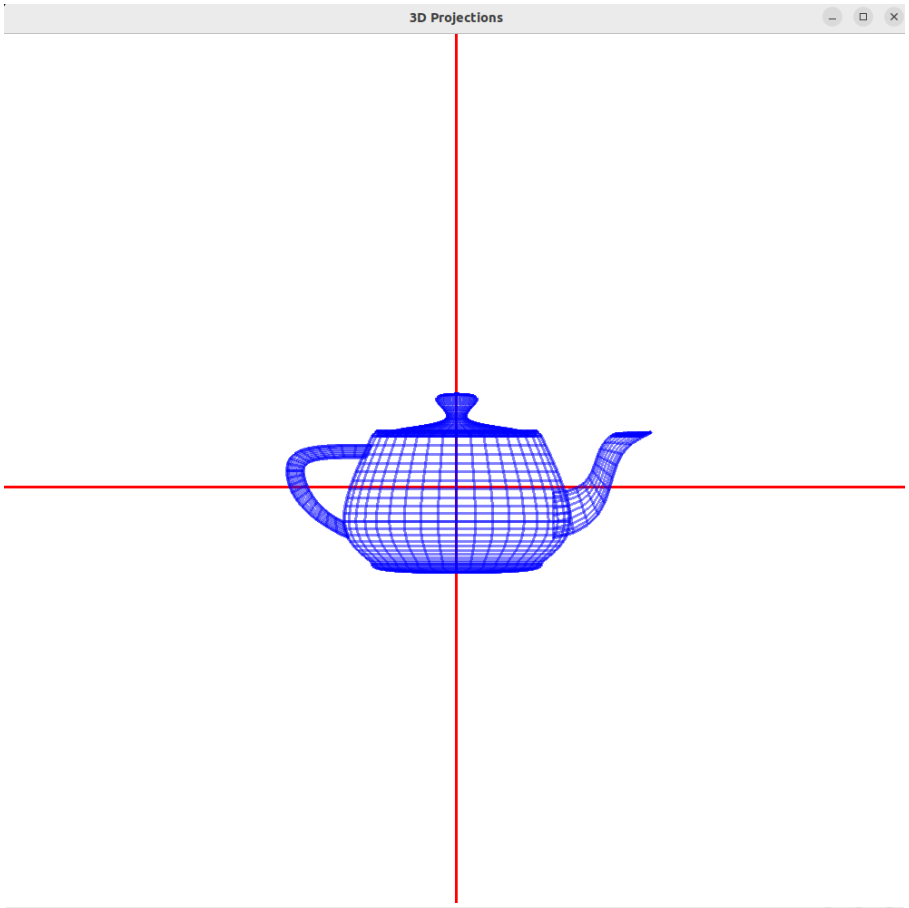
```
        // glLoadIdentity(); // Reset transformations
        x_rotate += 5;
        break;
    }
    case 's':
    {
        x_rotate -= 5;
        break;
    }
    case 'd':
    {
        y_rotate += 5;
        break;
    }
    case 'a':
    {
        y_rotate -= 5;
        break;
    }
    case 27:
        exit(0);
    case 32:
    {
        // Spacebar for changing projections
        isOrthoProjection = !isOrthoProjection;
        x_rotate = 0;
        y_rotate = 0;
        break;
    }
    }
    // Update the display
    glutPostRedisplay();
}

void display()
{
    // Initialize display parameters
    glClearColor(1, 1, 1, 1);
    glClear(GL_COLOR_BUFFER_BIT);
    // Translucency
    glEnable(GL_BLEND);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
    // Line width
    glLineWidth(3);
    // Apply the transformations & drawing on the model view matrix
    glMatrixMode(GL_MODELVIEW);
    // Draw the X and Y axis
    drawAxes();
    // Transform only the drawn object, so use the matrix stack accordingly
    glPushMatrix();
    if (isOrthoProjection)
    {
        // Parallel Projection
        glOrtho(-2, 2, -2, 2, -2, 2);
    }
```

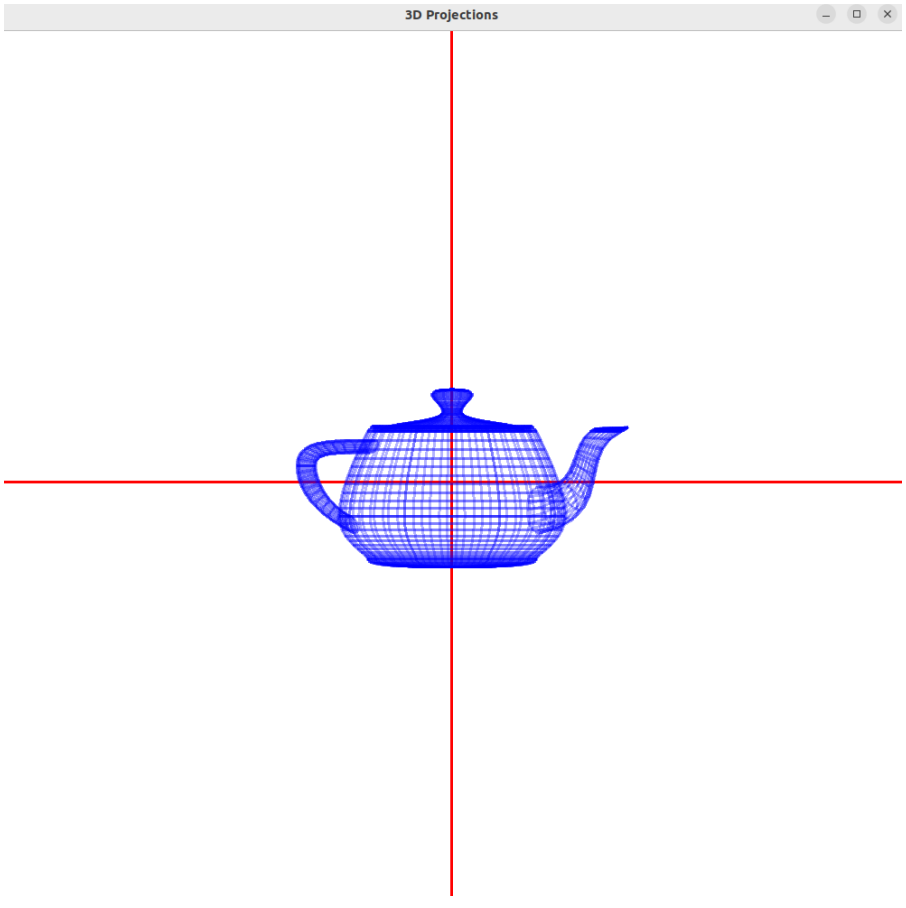
```
    else
    {
        // Perspective Projection
        gluPerspective(120, 1, 0.1, 50); // FoVy = 120, Aspect Ratio = 1
    }
    gluLookAt(0, 0, 1, 0, 0, 0, 0, 1, 0); // Camera, Center & Up Vector
    glPushMatrix(); // Create a separate transformation matrix
    glRotatef(x_rotate, 1, 0, 0);         // Keyboard based rotations
    glRotatef(y_rotate, 0, 1, 0);
    glColor4f(0, 0, 1, 0.3); // Draw the object
    glutWireTeapot(0.5);
    glPopMatrix(); // Pop the transformation matrix
    glPopMatrix(); // Pop the matrix back into the model view stack
    glFlush();
}

int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(windowWidth, windowHeight);
    glutCreateWindow("3D Projections");
    printf("Enter (1) for orthographic and (0) for perspective: ");
    int oop;
    scanf("%d", &oop);
    isOrthoProjection = oop;
    // Register the callback functions
    glutDisplayFunc(display);
    glutKeyboardFunc(keyboardKeys);

    glutMainLoop();
    return 0;
}
```

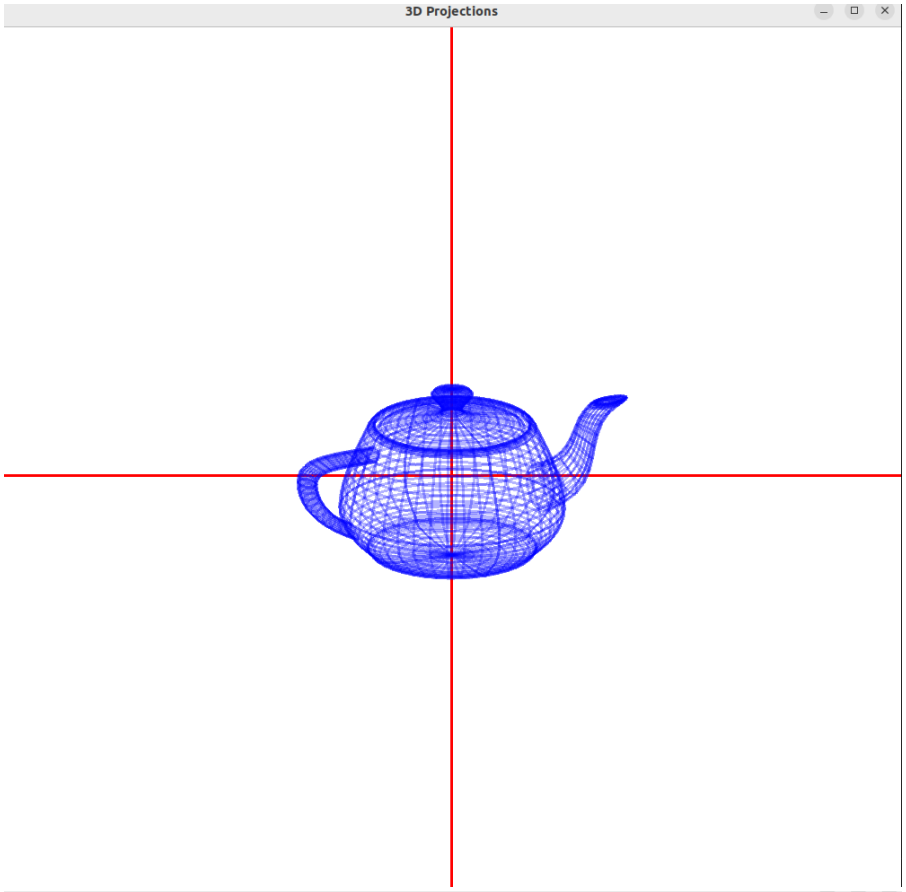
run.sh:

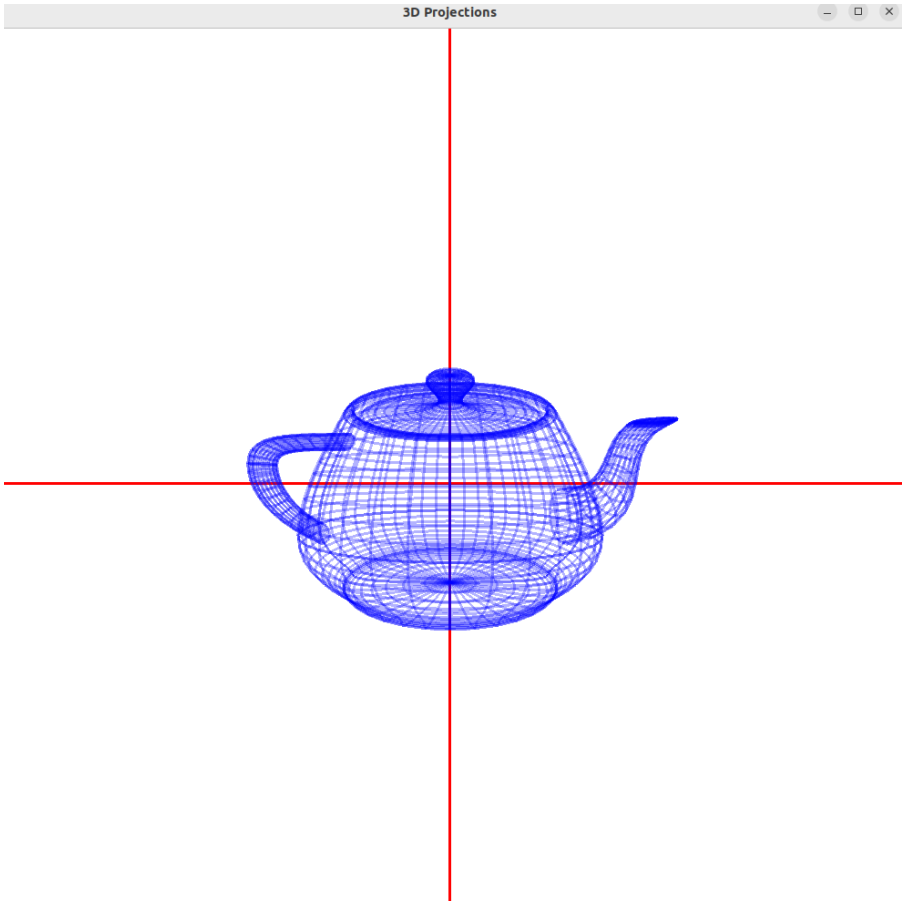
```
g++ 9.cpp -lGL -lglut -lGLU
./a.out
```

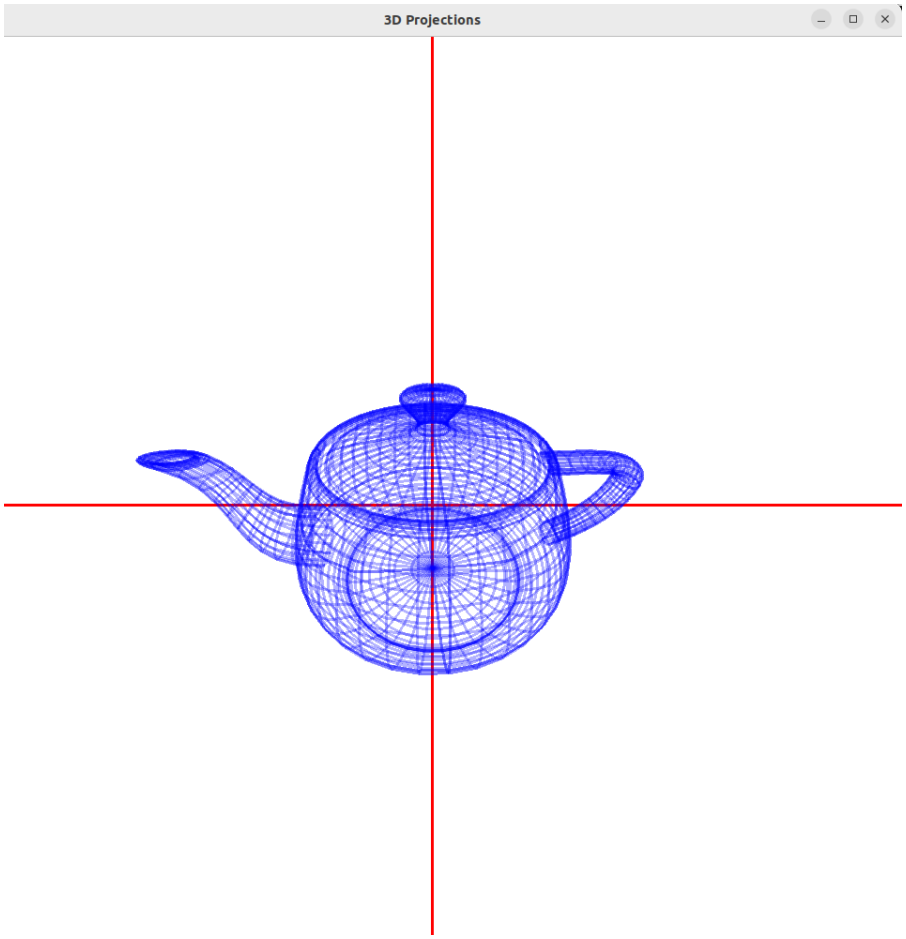
Sample I/O:


3D Projections


3D Projections


3D Projections


3D Projections



Learning Outcomes:

Thus, 3D projections has been implemented on objects using OpenGL.

Shivcharan T
205001100
CSE - B

SSN COLLEGE OF ENGINEERING, KALAVAKKAM
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
UCS1712 – GRAPHICS AND MULTIMEDIA LAB

Lab Exercise 10: : Creating a 3D Scene in C++ using OpenGL

Write a C++ program using Opengl to draw atleast four 3D objects. Apply lighting and texture and render the scene. Apply transformations to create a simple 3D animation. [Use built-in transformation functions.

Aim:

To create a 3D Scene in C++ using OpenGL

Algorithm:

1. Initialize OpenGL, set window dimensions, and create a window using GLUT.
2. Set clear color and enable depth testing for accurate rendering.
3. Load a texture and set its parameters for later use in the scene using SOIL.
4. Enable lighting and set light parameters (position, ambient, diffuse, specular).
5. Enable texture mapping and set the shading model to GL_FLAT.
6. Set up the perspective projection using gluPerspective.
7. Define the display function to clear buffers, set the projection and modelview matrices, and draw 3D objects.
8. Inside the display function, draw a teapot, a scaled sphere, a scaled and translated cone, and a torus using built-in functions.
9. Apply transformations to each object (translation, rotation, scaling) to create the desired arrangement using built-in functions..
10. Implement an update function to control the rotation angle for animation.
11. Set up the main function, specify the display and update functions, and initialize the scene.
12. Enter the GLUT main loop to handle events and continuously render the scene.

10.cpp:

```
#include <GL/glut.h>
#include <SOIL/SOIL.h>

const int windowWidth = 800;
const int windowHeight = 600;

GLfloat angle = 0.0f; // Initial rotation angle

// Texture variables
GLuint textureID;

// Rotation angles
float angleX = 0.0f;
float angleY = 0.0f;
```

```
// Texture coordinates
float texCoordX = 0.0f;
float texCoordY = 0.0f;

void init()
{
    glClearColor(0.0, 0.0, 0.0, 1.0);
    glEnable(GL_DEPTH_TEST);

    // Load texture
    glGenTextures(1, &textureID);
    glBindTexture(GL_TEXTURE_2D, textureID);
    int width, height;
    unsigned char *image = SOIL_load_image("texture1.jpg", &width, &height, 0,
SOIL_LOAD_RGB);
    glTexImage2D(GL_TEXTURE_2D,   0,   GL_RGB,   width,   height,   0,   GL_RGB,
GL_UNSIGNED_BYTE, image);
    SOIL_free_image_data(image);

    // Set texture parameters
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

    // Enable lighting and set light parameters
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);

    GLfloat light_position[] = {1.0, 1.0, 1.0, 0.0};
    GLfloat ambient[] = {0.2, 0.2, 0.2, 1.0};
    GLfloat diffuse[] = {1.0, 1.0, 1.0, 1.0};
    GLfloat specular[] = {1.0, 1.0, 1.0, 1.0};

    glLightfv(GL_LIGHT0, GL_POSITION, light_position);
    glLightfv(GL_LIGHT0, GL_AMBIENT, ambient);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuse);
    glLightfv(GL_LIGHT0, GL_SPECULAR, specular);

    // Enable texture and set shading model
    glEnable(GL_TEXTURE_2D);
    glShadeModel(GL_FLAT);

    // Set up perspective projection
    glMatrixMode(GL_PROJECTION);
    gluPerspective(45.0f, 1.0f, 1.0f, 100.0f);
    glMatrixMode(GL_MODELVIEW);
}

void drawTeapot()
{
    glEnable(GL_TEXTURE_2D);
    glBindTexture(GL_TEXTURE_2D, textureID); // Bind the texture

    glutSolidTeapot(1.0); // Draw a teapot
```

```
    glDisable(GL_TEXTURE_2D);
}

// glusolid

void drawSphere()
{
    glEnable(GL_TEXTURE_2D);
    glBindTexture(GL_TEXTURE_2D, textureID); // Bind the texture

    GLUquadricObj *sphere = gluNewQuadric();
    gluQuadricTexture(sphere, GL_TRUE);

    gluSphere(sphere, 0.5, 20, 20);

    gluDeleteQuadric(sphere);
    glDisable(GL_TEXTURE_2D);
}

void drawConeInit()
{
    glEnable(GL_TEXTURE_2D);
    glBindTexture(GL_TEXTURE_2D, textureID); // Bind the texture

    GLUquadricObj *cone = gluNewQuadric();
    gluQuadricTexture(cone, GL_TRUE);

    gluCylinder(cone, 0.0, 0.5, 1.0, 20, 20); // Draw a cone

    gluDeleteQuadric(cone);
    glDisable(GL_TEXTURE_2D);
}

void drawCone()
{
    glEnable(GL_TEXTURE_2D);
    glBindTexture(GL_TEXTURE_2D, textureID); // Bind the texture

    GLUquadricObj *cone = gluNewQuadric();
    gluQuadricTexture(cone, GL_TRUE);

    gluCylinder(cone, 0.0, 0.5, 1.0, 20, 20); // Draw the cone

    glPushMatrix();
    glTranslatef(0.0, 0.0, 1.0);     // Move to the base of the cone
    gluDisk(cone, 0.0, 0.5, 20, 20); // Draw the base circle
    glPopMatrix();

    gluDeleteQuadric(cone);
    glDisable(GL_TEXTURE_2D);
}

void drawCylinder()
{
    glEnable(GL_TEXTURE_2D);
```

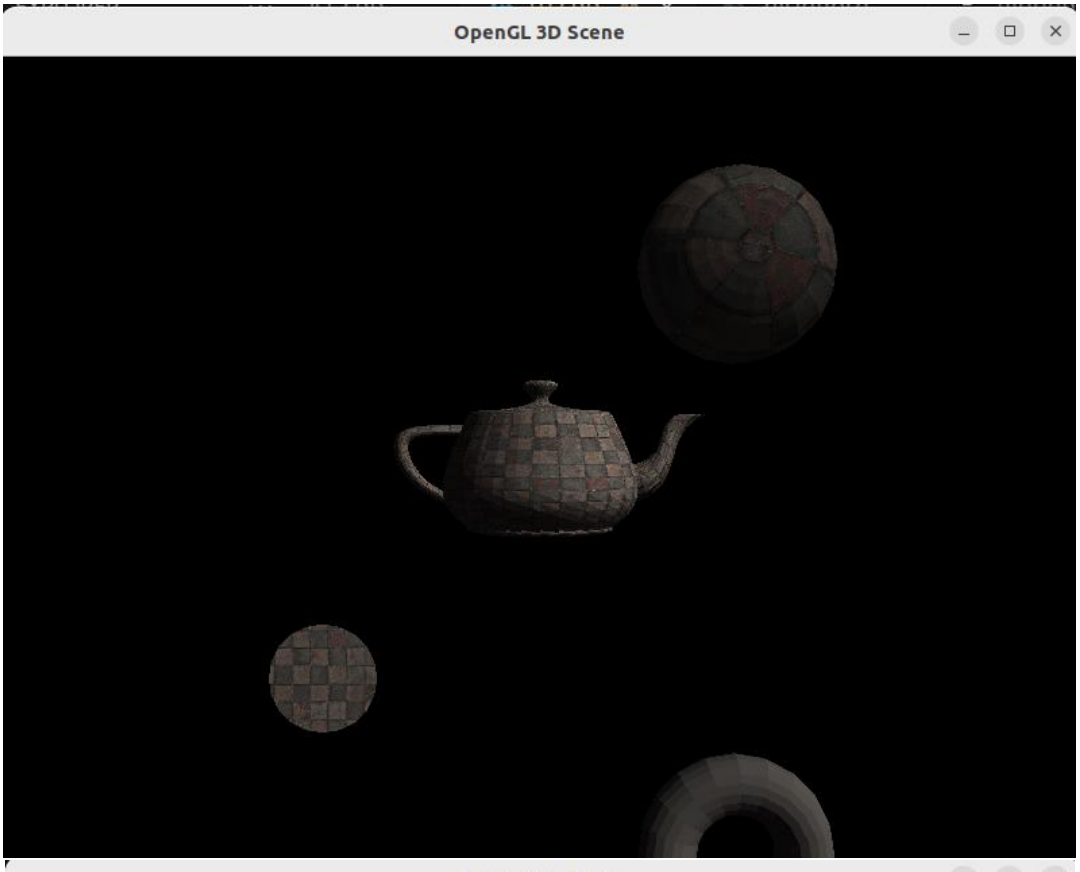
```
    glBindTexture(GL_TEXTURE_2D, textureID); // Bind the texture

    GLUquadricObj *cylinder = gluNewQuadric();
    gluQuadricTexture(cylinder, GL_TRUE);

    gluCylinder(cylinder, 0.5, 0.5, 1.0, 20, 20);

    gluDeleteQuadric(cylinder);
    glDisable(GL_TEXTURE_2D);
}

void drawTorus()
{
    glEnable(GL_TEXTURE_2D);
    glBindTexture(GL_TEXTURE_2D, textureID); // Bind the texture

    glutSolidTorus(0.3, 0.7, 20, 20);

    glDisable(GL_TEXTURE_2D);
}

void display()
{
    glViewport(0, 0, windowWidth, windowHeight); // Set the viewport size

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(45.0,  static_cast<double>(windowWidth)  /  windowHeight,  0.1,
100.0); // Adjusted near and far planes

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(0.0, 0.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0); // Adjusted camera
position

    glTranslatef(0.0f, 0.0f, -5.0f);    // Move the scene back along the z-axis
    glRotatef(angle, 1.0f, 1.0f, 0.0f); // Rotate around the x and y-axis

    // Apply material properties (color, etc.)
    GLfloat material_diffuse[] = {0.7f, 0.7f, 0.7f, 1.0f};
    glMaterialfv(GL_FRONT, GL_DIFFUSE, material_diffuse);

    // Draw 3D objects
    drawTeapot(); // Draw a teapot
    glTranslatef(2.0f, 2.0f, 0.0f);
    glScalef(2.0f, 2.0f, 2.0f);  // Scale
    drawSphere(); // Draw a sphere
    glScalef(0.5f, 0.5f, 0.5f);  // Scale
    glTranslatef(-4.0f, -4.0f, 0.0f);
    drawCone(); // Draw a cylinder
    glTranslatef(4.0f, -2.0f, 0.0f);
    drawTorus(); // Draw a torus
```

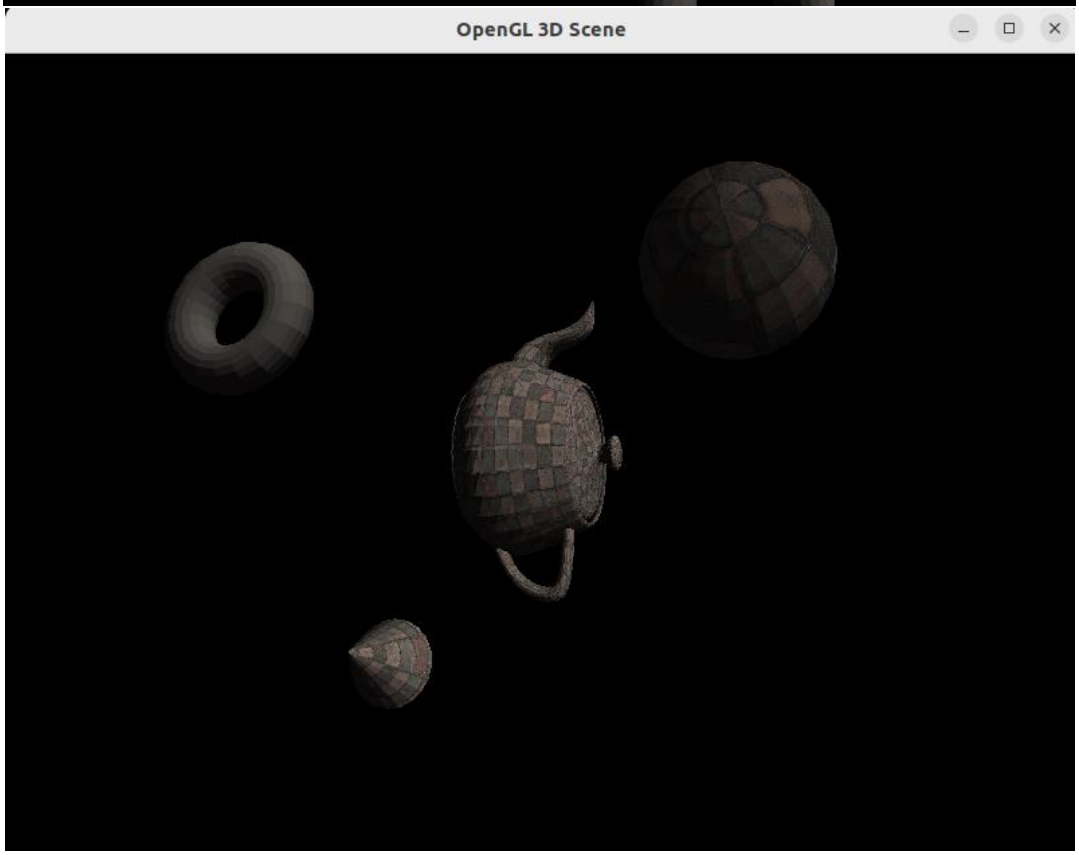
```
    glTranslatef(0.0f, 0.0f, -5.0f);
    glutSwapBuffers();
}

void update(int value)
{
    angle += 2.0f; // Update rotation angle
    if (angle > 360)
    {
        angle -= 360; // Keep the angle within 0 to 360 degrees
    }

    glutPostRedisplay();          // Trigger a redraw
    glutTimerFunc(16, update, 0); // Call update function every 16 milliseconds
}

int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(windowWidth, windowHeight); // Set window size
    glutCreateWindow("OpenGL 3D Scene");

    // Add these lines for proper initialization
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glEnable(GL_DEPTH_TEST);

    glutDisplayFunc(display);
    glutTimerFunc(25, update, 0);

    init();

    glutMainLoop();
    return 0;
}
```

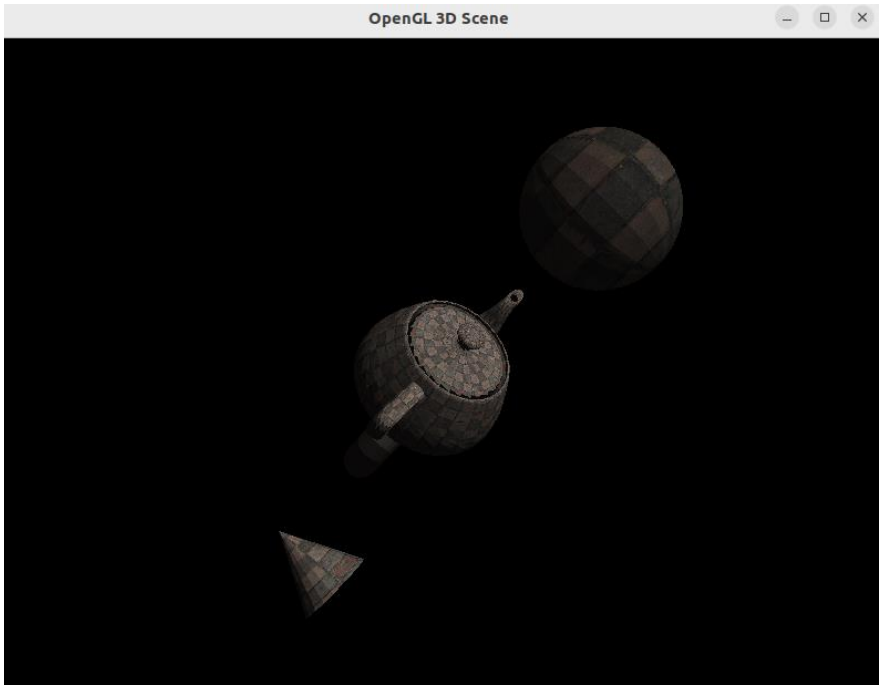
run.sh:

```
g++ 10.cpp -lGL -lglut -lGLU -lSOIL
./a.out
```

Sample I/O:


OpenGL 3D Scene


OpenGL 3D Scene



Learning Outcomes:

Thus, 3D objects were drawn and lighting and textures were applied and the scene was rendered in C++ using OpenGL.

Shivcharan T
205001100
CSE - B

SSN COLLEGE OF ENGINEERING, KALAVAKKAM
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
UCS1712 – GRAPHICS AND MULTIMEDIA LAB

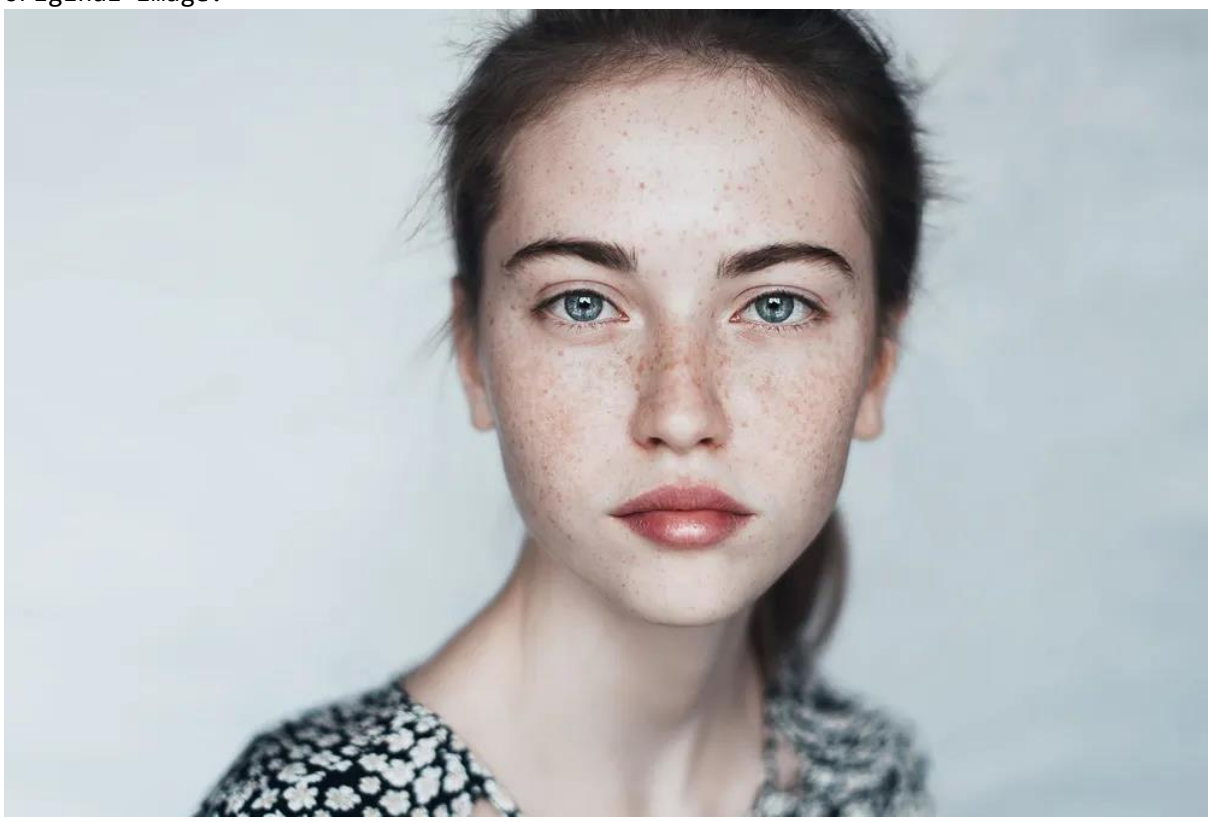
Lab Exercise 11: Image Editing and Manipulation

a) Using GIMP, include an image and apply filters, noise and masks.
b) Using GIMP, create a GIF animated image.

Aim:
To do image editing and manipulation on a image file using GIMP software.

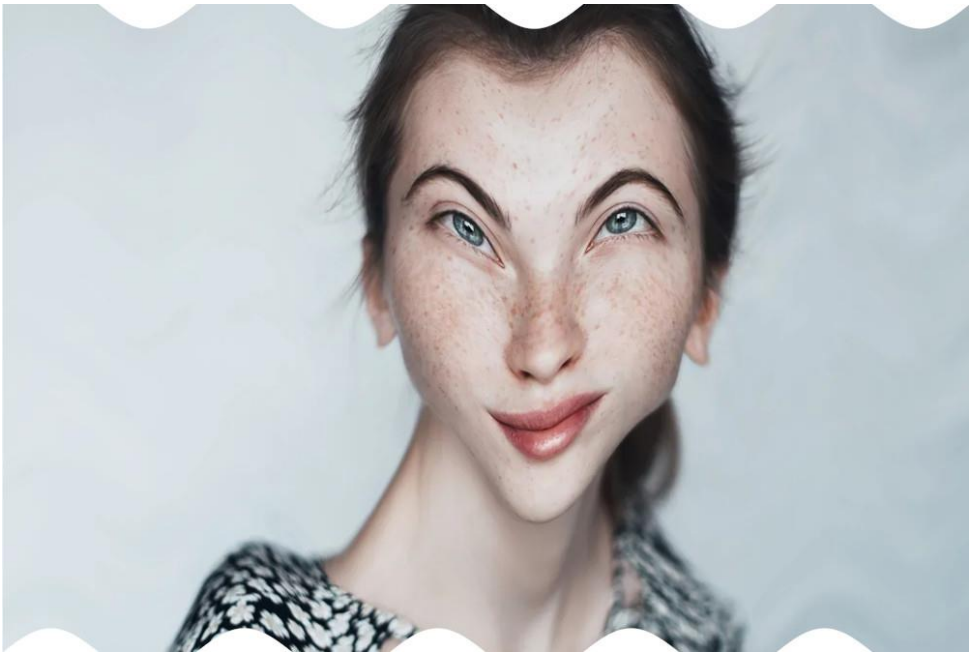
a:

`original image:`

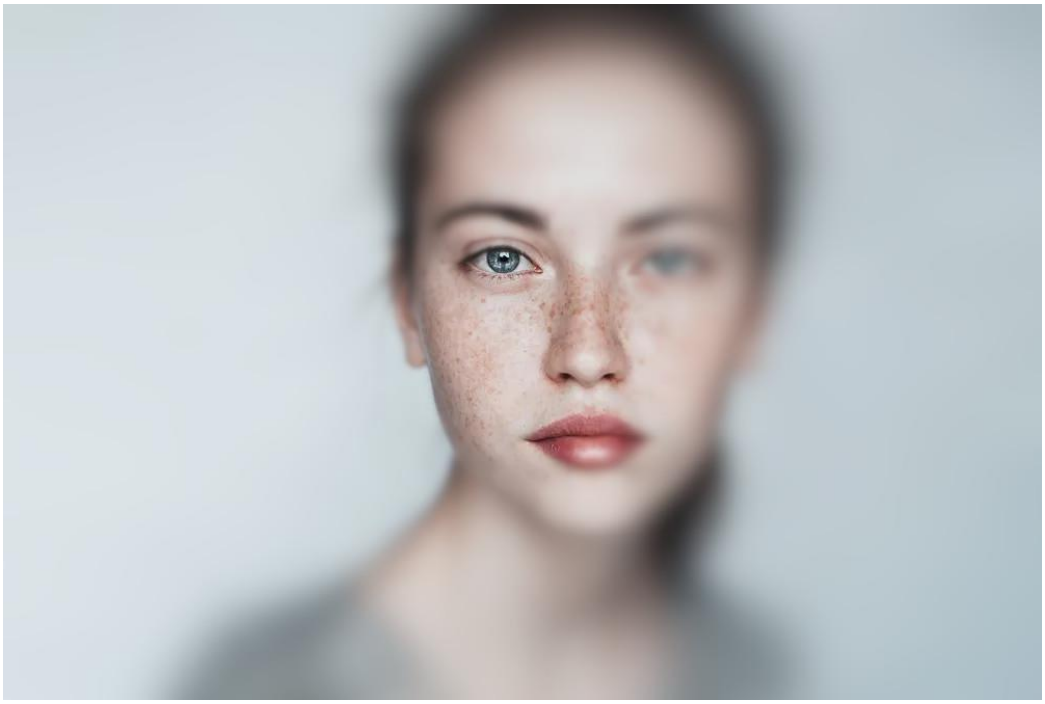


Sample Output:

1. ***Filters:***
    a. ***Disort – Ripple:***

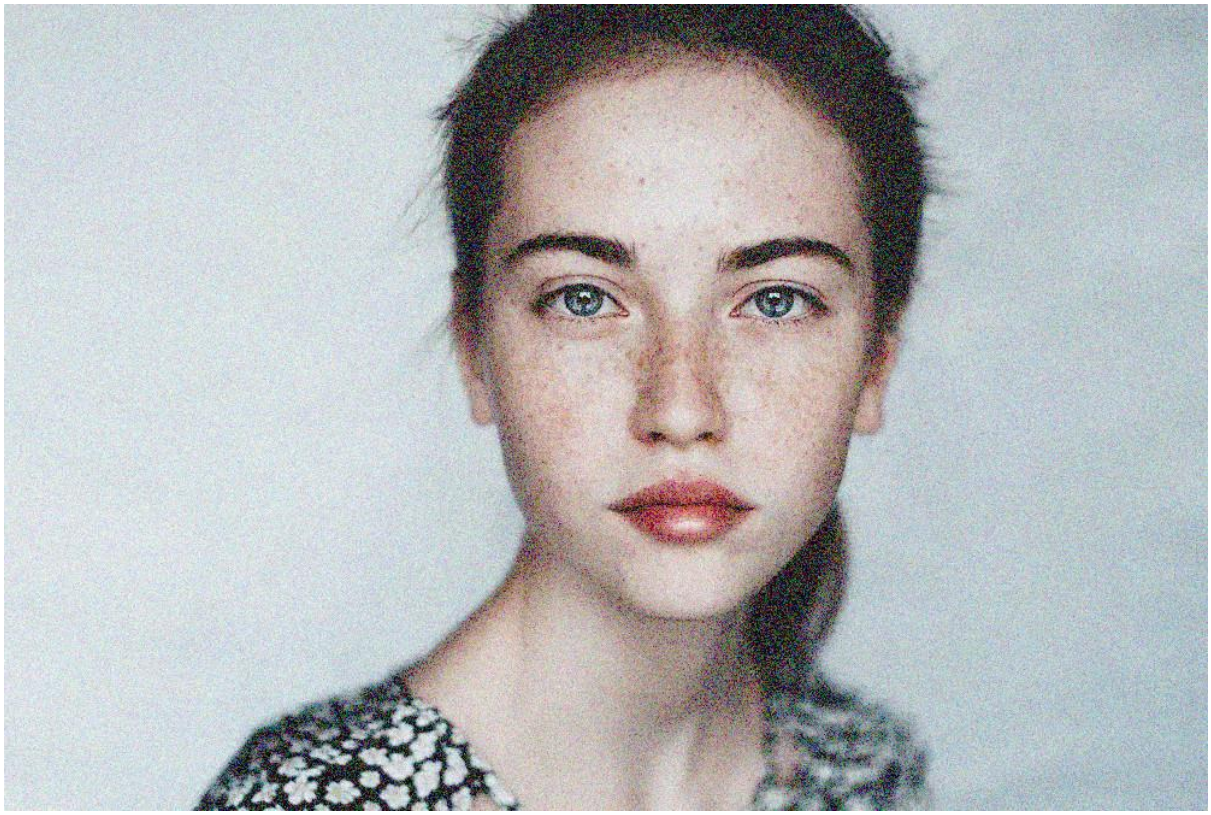


b. Focus – Blur:



2. Noise:

a. Noise - RBG Noise:



3. ***Mask:***
    a. ***Enchance – Unsharp Mask:***

b:

Learning Outcomes:

Learnt to do image editing and manipulation on a image file using GIMP software.

Shivcharan T
205001100
CSE - B

SSN COLLEGE OF ENGINEERING, KALAVAKKAM
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
UCS1712 – GRAPHICS AND MULTIMEDIA LAB

Lab Exercise 11: Creating 2D animation

Using GIMP, include layers and create a simple animation of your choice.

Aim:
To do a simple animation using layers in GIMP software.

Sample Output:

Learning Outcomes:

Learnt to do simple animation using layers in GIMP software.