SSN COLLEGE OF ENGINEERING, KALAVAKKAM

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

UCS1712-Graphics and Multimedia Lab

Programming Assignment 1

Study of Basic Output Primitives in C++ using OpenGL

Name: Jayannthan P T *Dept: CSE 'A'* *Roll No.: 205001049*

a)To create an output window using OPENGL and to draw the following basic output primitives:

Source code:

- POINTS

```
void points()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glClearColor(1.0, 1.0, 1.0, 0.0);
    glBegin(GL_POINTS);
    glVertex2f(0.0, 0.0);
    glVertex2f(0.5, 0.0);
    glVertex2f(0.5, 0.5);
    //    glVertex2f(0.0, 0.5);
    glEnd();
    glFlush();
}
```

- LINES

```
void lines()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glClearColor(1.0, 1.0, 1.0, 0.0);
    glBegin(GL_LINES);
    glVertex2f(0.0, 0.0);
    glVertex2f(0.5, 0.0);
    glEnd();
    glFlush();
}
```

- LINE_STRIP

```
void linesstrip()
```

```
{
    glClear(GL_COLOR_BUFFER_BIT);
    glClearColor(1.0, 1.0, 1.0, 0.0);
    glBegin(GL_LINE_STRIP);
    glVertex2f(0.0, 0.0);
    glVertex2f(0.5, 0.0);
    glVertex2f(1.0, 1.0);
    glVertex2f(0.7, 0.7);
    glEnd();
    glFlush();
}
```

- LINE_LOOP

```
void lineloop()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glClearColor(1.0, 1.0, 1.0, 0.0);
    glBegin(GL_LINE_LOOP);
    glVertex2f(0.0, 0.0);
    glVertex2f(0.5, 0.0);
    glVertex2f(1.0, 1.0);
    glVertex2f(0.7, 0.7);
    glEnd();
    glFlush();
}
```

- TRIANGLES

```
void triangle()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glClearColor(1.0, 1.0, 1.0, 0.0);
    glBegin(GL_TRIANGLES);
    glVertex2f(0.0, 0.0);
    glVertex2f(0.5, 0.0);
    glVertex2f(0.5, 0.5);
    //    glVertex2f(0.0, 0.5);
    glEnd();
    glFlush();
}
```

- QUADS

```
void quadrant()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glClearColor(1.0, 1.0, 1.0, 0.0);
    glBegin(GL_QUADS);
    glVertex2f(0.0, 0.0);
    glVertex2f(0.5, 0.0);
```

```
    glVertex2f(0.5, 0.5);
    glVertex2f(0.0, 0.5);
    glEnd();
    glFlush();
}
```

- QUAD_STRIP

```
void quadstrip()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glClearColor(1.0, 1.0, 1.0, 0.0);
    glBegin(GL_QUAD_STRIP);
    glVertex2f(0.0, 0.0);
    glVertex2f(0.5, 0.0);
    glVertex2f(0.5, 0.5);
    glVertex2f(0.0, 0.5);
    glVertex2f(1.0, 1.0);
    glVertex2f(0.7, 0.7);
    glEnd();
    glFlush();
}
```

- POLYGON

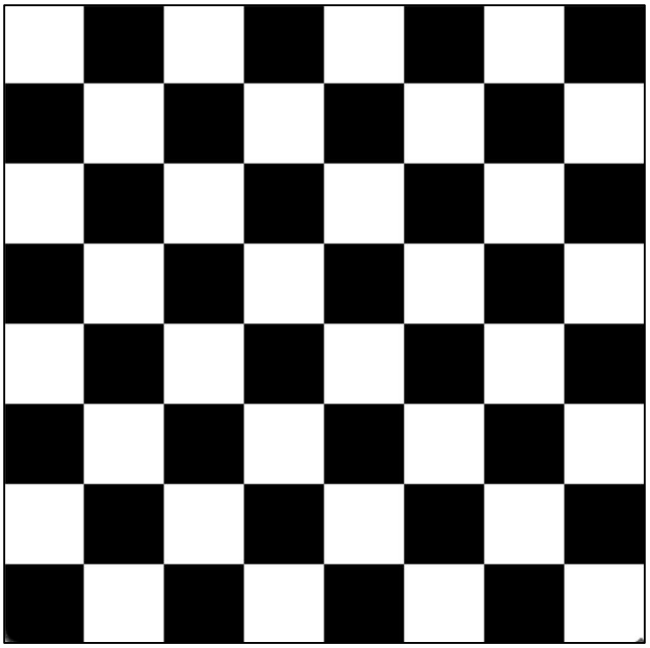
```
void polygon()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glClearColor(1.0, 1.0, 1.0, 0.0);
    glBegin(GL_POLYGON);
    glVertex2f(0.0, 0.0);
    glVertex2f(0.5, 0.0);
    glVertex2f(0.5, 0.5);
    glVertex2f(0.0, 0.5);
    glVertex2f(1.0, 1.0);
    glVertex2f(0.7, 0.7);
    glEnd();
    glFlush();
}
```

b) To create an output window and draw a checkerboard using OpenGL.

Source code:

```
#include <iostream>
#include <GLUT/glut.h>
const int windowWidth = 400;
const int windowHeight = 400;
const int numCheckers = 8;
```

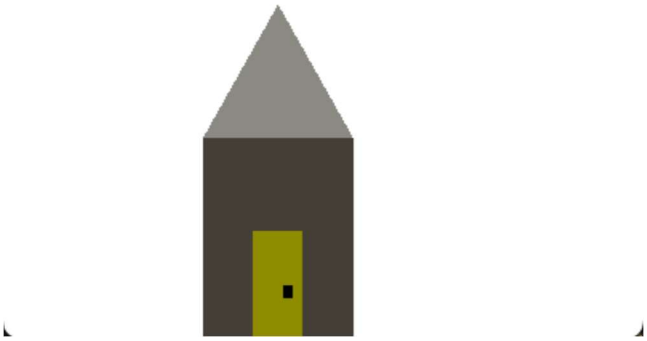
```
const int checkerSize = windowWidth / numCheckers;
void drawCheckerboard()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glClearColor(1.0, 1.0, 1.0, 0.0);
    for (int row = 0; row < numCheckers; row++)
    {
        for (int col = 0; col < numCheckers; col++)
        {
            if ((row + col) % 2 == 0)
                glColor3f(0.0f, 0.0f, 0.0f); // Black
            else
                glColor3f(1.0f, 1.0f, 1.0f); // White
            int x = col * checkerSize;
            int y = row * checkerSize;
            glBegin(GL_QUADS);
            glVertex2f(x, y);
            glVertex2f(x + checkerSize, y);
            glVertex2f(x + checkerSize, y + checkerSize);
            glVertex2f(x, y + checkerSize);
            glEnd();
        }
    }
    glFlush();
}
void myInit()
{
    glClearColor(1.0, 1.0, 1.0, 0.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0, windowWidth, 0, windowHeight);
}
int main(int argc, char *argv[])
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(windowWidth, windowHeight);
    glutCreateWindow("Checkerboard");
    glutDisplayFunc(drawCheckerboard);
    myInit();
    glutMainLoop();
    return 0;
}
```



c) To create an output window and draw a house using POINTS, LINES, TRAINGLES and QUADS/POLYGON.

```
#include <GLUT/glut.h>
void myInit()
{
    glClearColor(1.0, 1.0, 1.0, 0.0);
    glColor3f(0.0f, 0.0f, 0.0f);
    glPointSize(10);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0, 640.0, 0.0, 480.0);
}
void displayPoints()
{
    glBegin(GL_POINTS);
    glVertex2d(150, 100);
    glVertex2d(100, 230);
    glVertex2d(170, 130);
    glVertex2d(300, 350);
    glEnd();
}
void displayRectangle(int x, int y, int w, int h)
{
    glBegin(GL_POLYGON);
    glVertex2d(x, y);
    glVertex2d(x + w, y);
    glVertex2d(x + w, y + h);
    glVertex2d(x, y + h);
    glEnd();
}
void displayTriangle(int x, int y, int w, int h)
{
    glBegin(GL_TRIANGLES);
```

```
    glVertex2d(x, y);
    glVertex2d(x + w, y);
    glVertex2d(x + (w / 2), y + h);
    glEnd();
}
void displayLine(int x1, int y1, int x2, int y2)
{
    glBegin(GL_LINES);
    glVertex2d(x1, y1);
    glVertex2d(x2, y2);
    glEnd();
}
void displayHouse()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor4f(0.268, 0.243, 0.217, 1);
    displayRectangle(200, 0, 150, 150);
    glColor4f(0.54, 0.54, 0.51, 1);
    displayTriangle(200, 150, 150, 100);
    glColor4f(0.56, 0.55, 0, 1);
    displayRectangle(250, 0, 50, 80);
    glColor4f(0, 0, 0, 1);
    displayRectangle(280, 30, 10, 10);
}
```



SSN COLLEGE OF ENGINEERING, KALAVAKKAM

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

UCS1712-Graphics and Multimedia Lab

Programming Assignment 1

DDA Line Drawing Algorithm in C++ using OpenGL

Name: Jayannthan P T *Dept: CSE 'A'* *Roll No.: 205001049*

a) 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

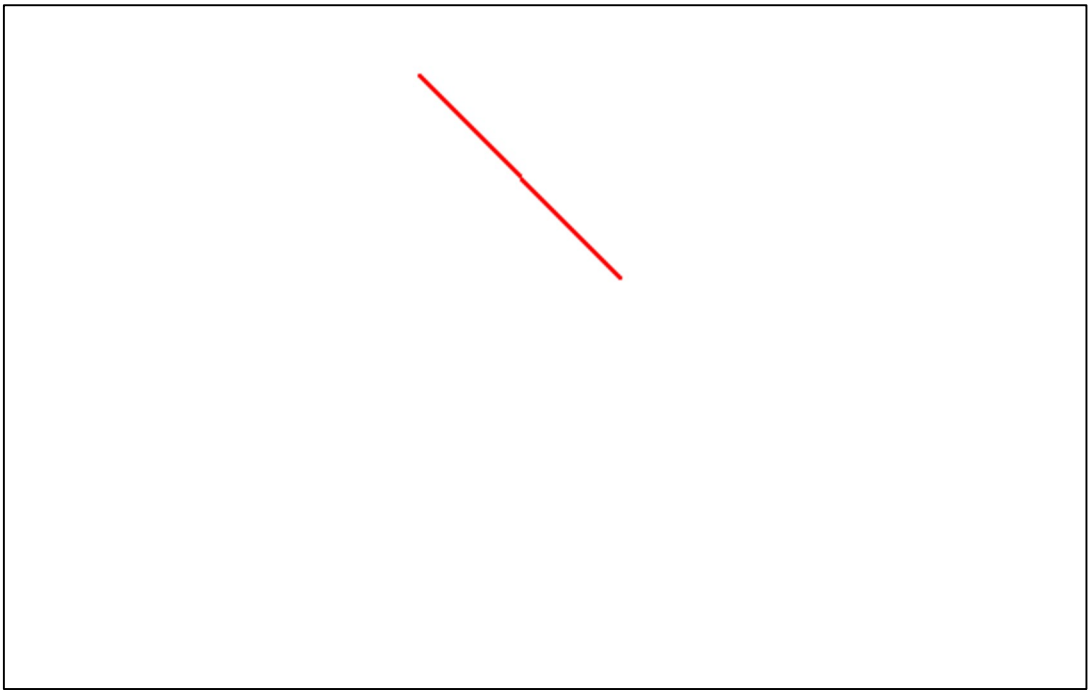
Note that all four cases of line drawing must be given as test cases.

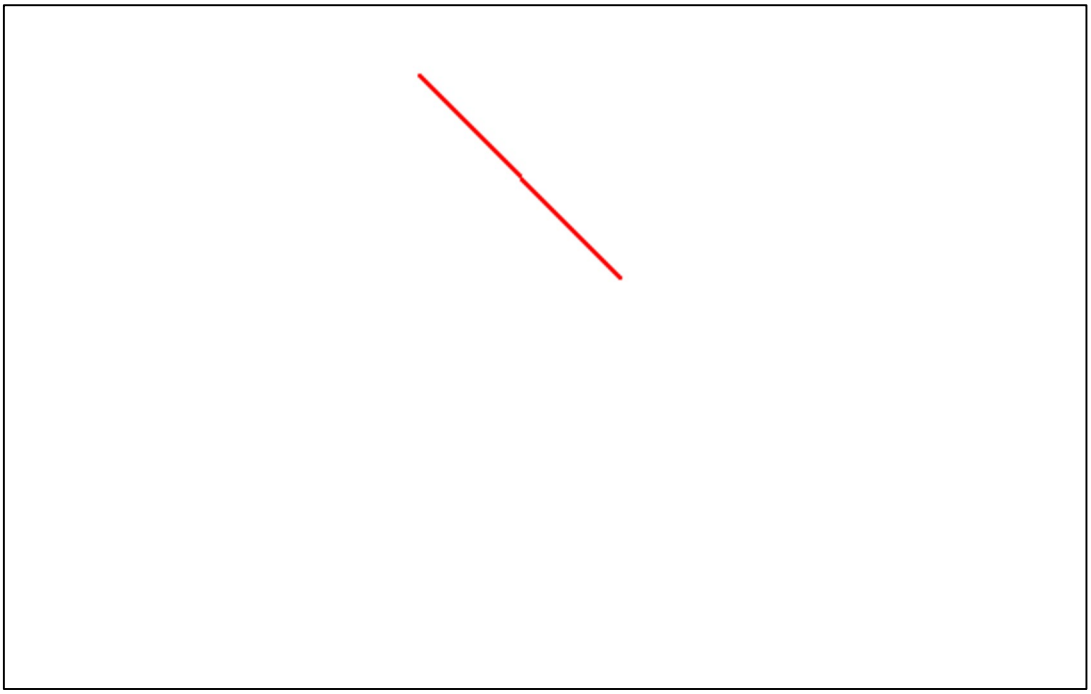
Source code:

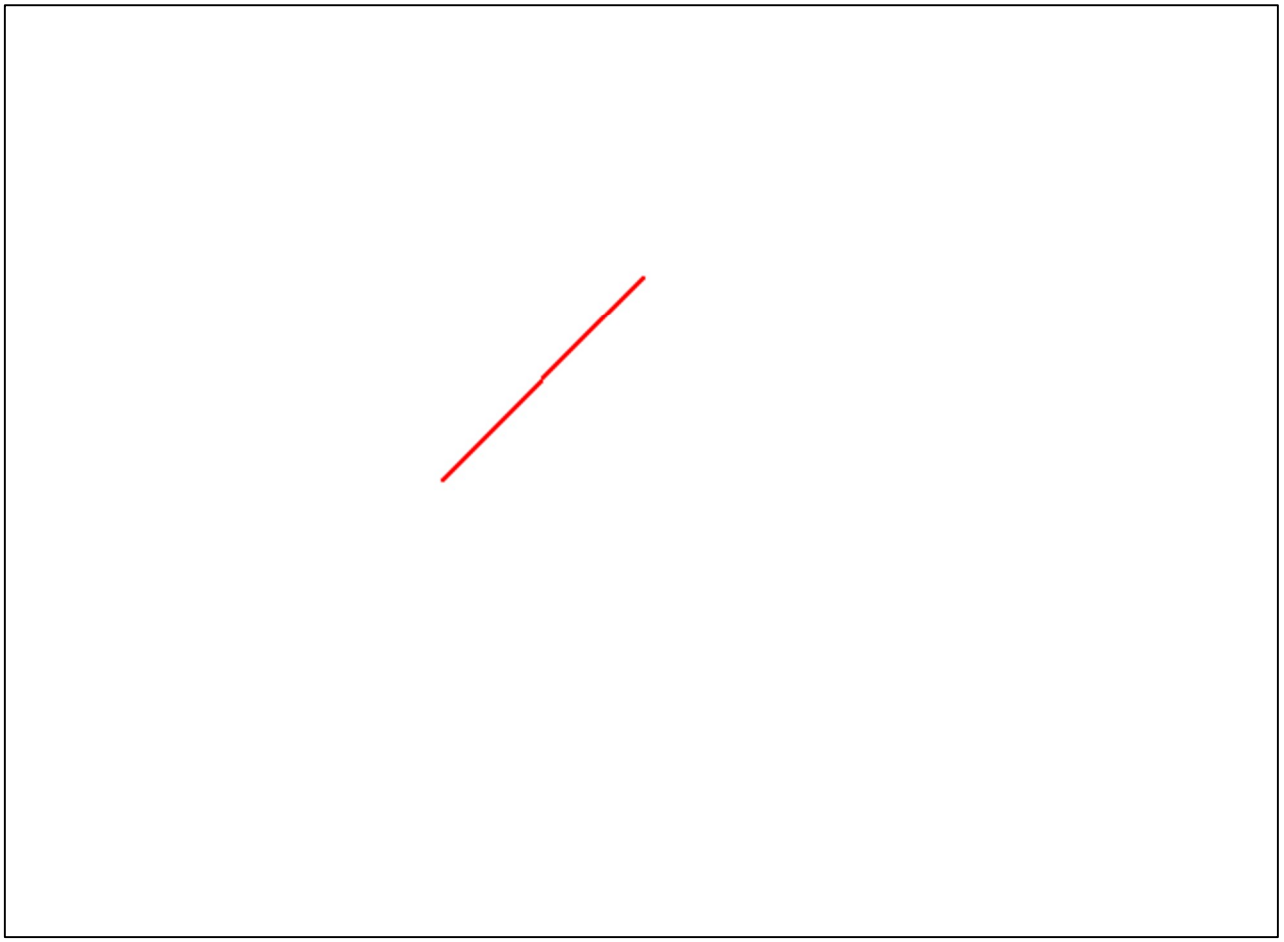
```
include <iostream>

#include <cmath>
#include <GLUT/glut.h>

using namespace std;
int choice = 0, flag = 0;
float x_1 = 0, y_1 = 0, x_2 = 0, y_2 = 0;
string cnt = "YES";

void myInit()
{
    glClearColor(1.0, 1.0, 1.0, 0.0);
    glPointSize(2);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0, 640.0, 0.0, 480.0);
}

void drawLine()
{
```

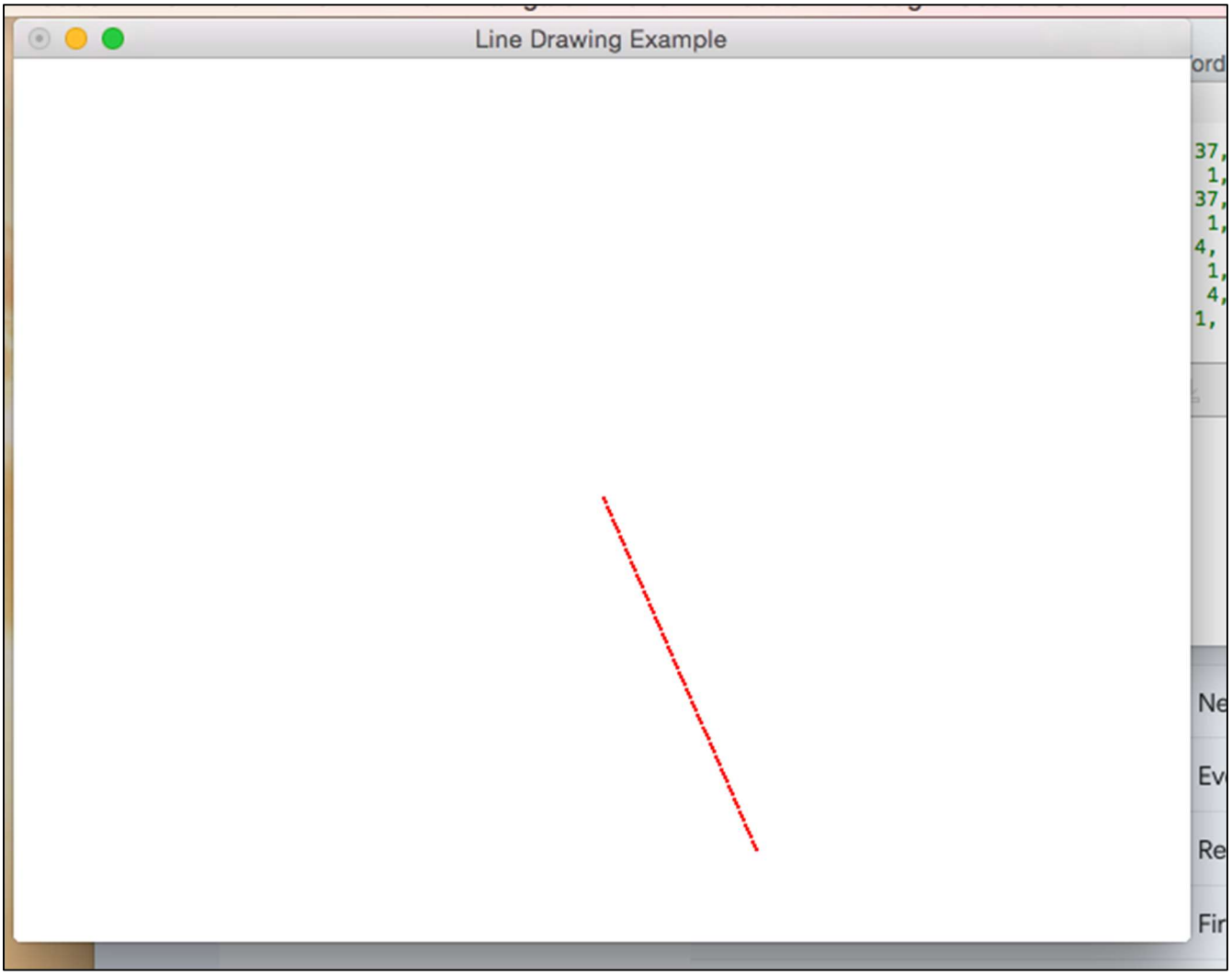
```
    glBegin(GL_POINTS); // Begin drawing points x_1 += 320;
    x_2 += 320;
    y_1 += 240;
    y_2 += 240;
    float dx = x_2 - x_1;
    float dy = y_2 - y_1;
    int steps;
    if (abs(dx) > abs(dy))
        steps = abs(dx);
    else
        steps = abs(dy);
    float x = x_1, y = y_1; // Start from the actual (x_1, y_1) point
    float m = dy / dx;
    for (int i = 0; i <= steps; i++)
    {
        glVertex2f(round(x), round(y)); // Draw the current point
        if (choice == 1)
        {
            x += 1;
            y += m;
        }
        else if (choice == 2)
        {
            x -= 1;
            y -= m;
        }
        else if (choice == 3)
        {
            x += 1;
            y -= m;
        }
        else
        {
            x -= 1;
            y += m;
        }
    }
    glEnd(); // End drawing points
}
void myDisplay()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0f, 0.0f, 0.0f);
    if (flag == 0)
    {
        flag = 1;
        x_1 = 0, y_1 = 0, x_2 = 0, y_2 = 0;
        cout << "Point 1 : ";
        cin >> x_1 >> y_1;
        cout << "Point 2 : ";
        cin >> x_2 >> y_2;
        int m = (y_2 - y_1) / (x_2 - x_1);
        if (x_1 < x_2 && abs(m) <= 1)
        {
```

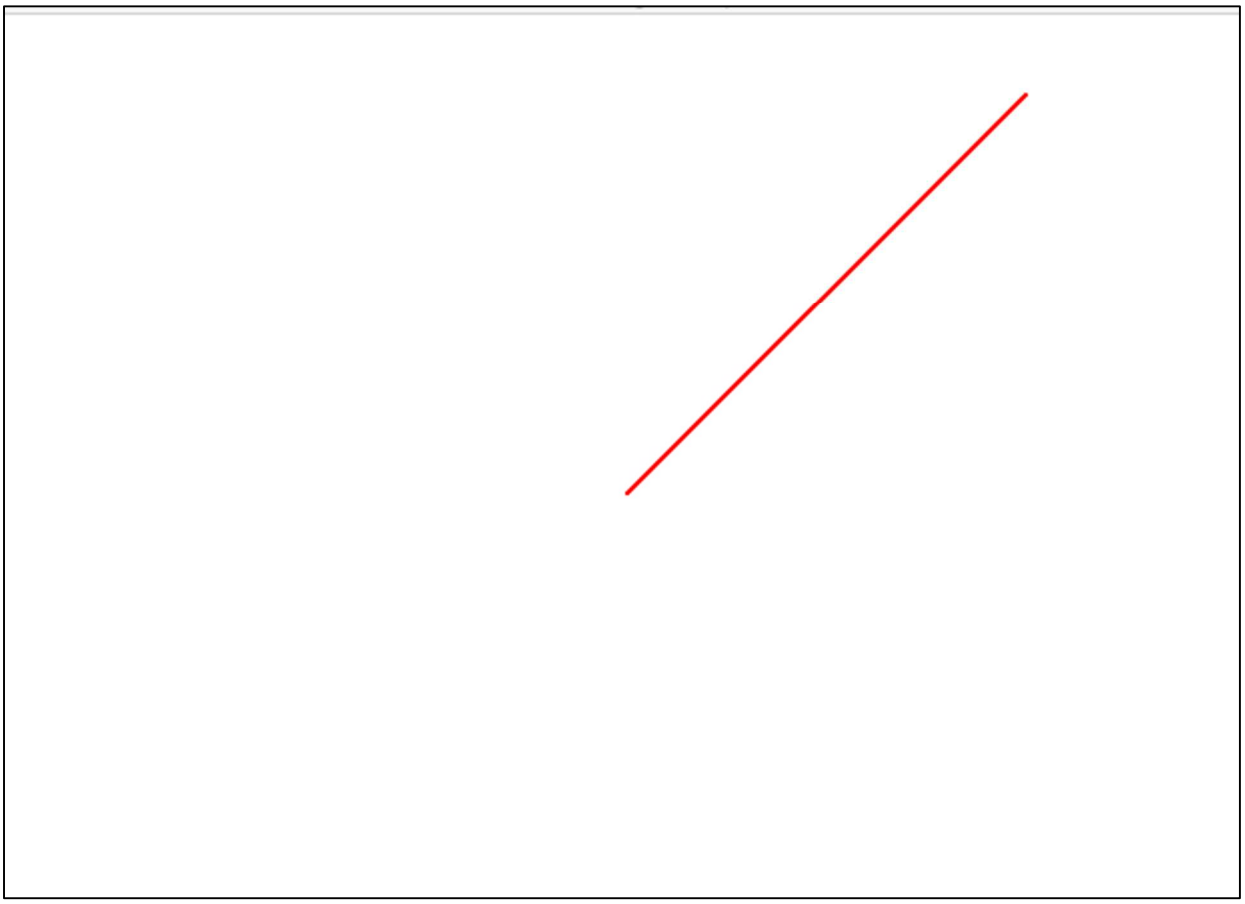
```
            choice = 1;
        }
        else if (x_2 > x_1 && abs(m) <= 1)
        {
            choice = 2;
        }
        else if (x_1 < x_2 && abs(m) > 1)
        {
            choice = 3;
        }
        else
        {
            choice = 4;
        }
    }
    drawLine();
    glFlush();
    cout << "Want to continue (YES/NO) : ";
    cin >> cnt;
    if (cnt == "NO")
    {
        cout << "Exiting...\n";
        exit(0);
    }
    flag = 0;             // Reset flag for the next line
    glutPostRedisplay(); // Continue updating the display
}
int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(640, 480);
    glutCreateWindow("Line Drawing Example");
    myInit();
    glutDisplayFunc(myDisplay);
    glutMainLoop();
    return 0;
}
```








Line Drawing Example



SSN COLLEGE OF ENGINEERING, KALAVAKKAM

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

UCS1712-Graphics and Multimedia Lab

Programming Assignment 3

Bresenham's Line Drawing Algorithm in C++ using OpenGL

Name: Jayannthan P T *Dept: CSE 'A'* *Roll No.: 205001049*

To plot points that make up the line with endpoints (x0,y0) and (xn,yn) using Bresenham's 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

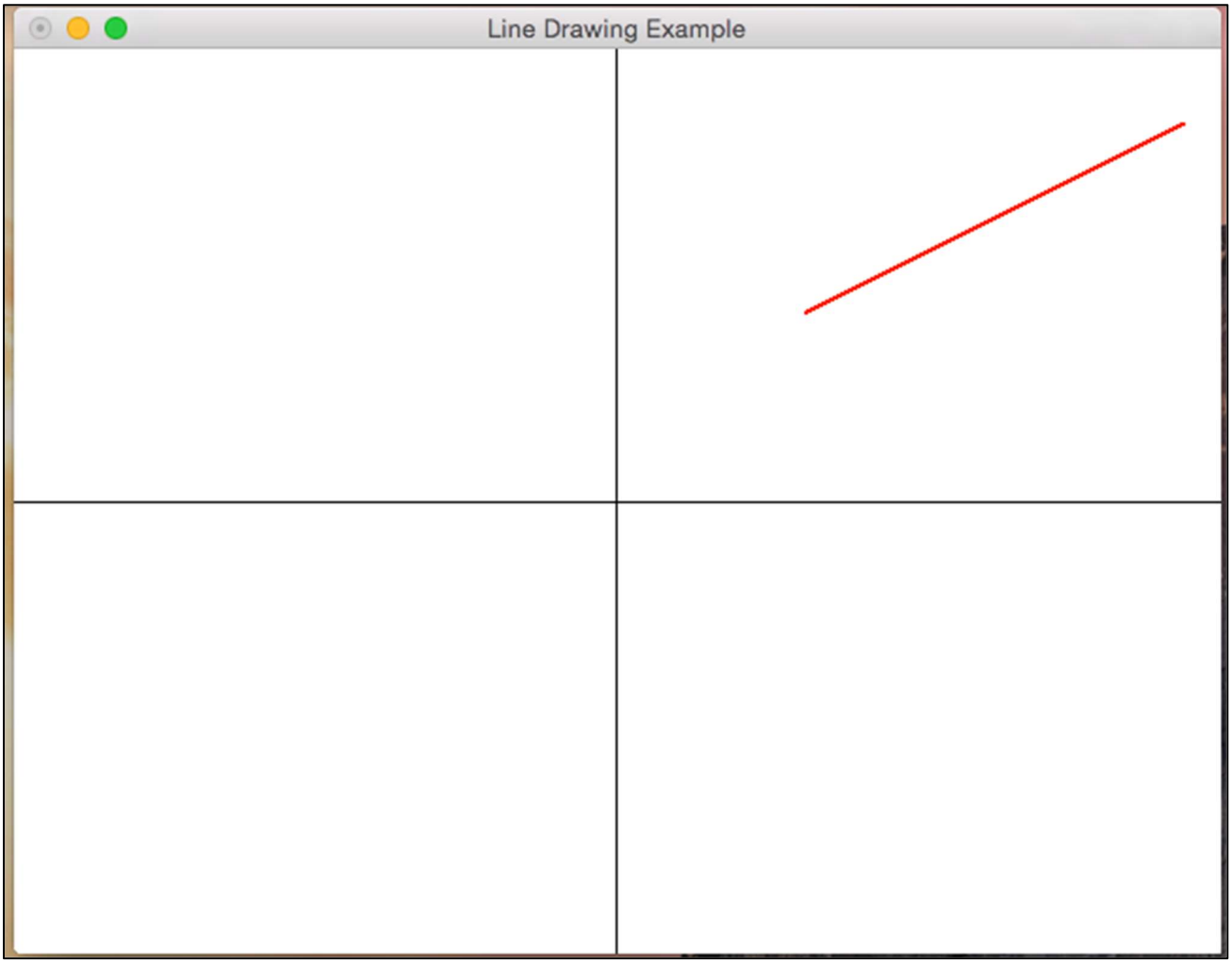
Note that all four cases of line drawing must be given as test cases.

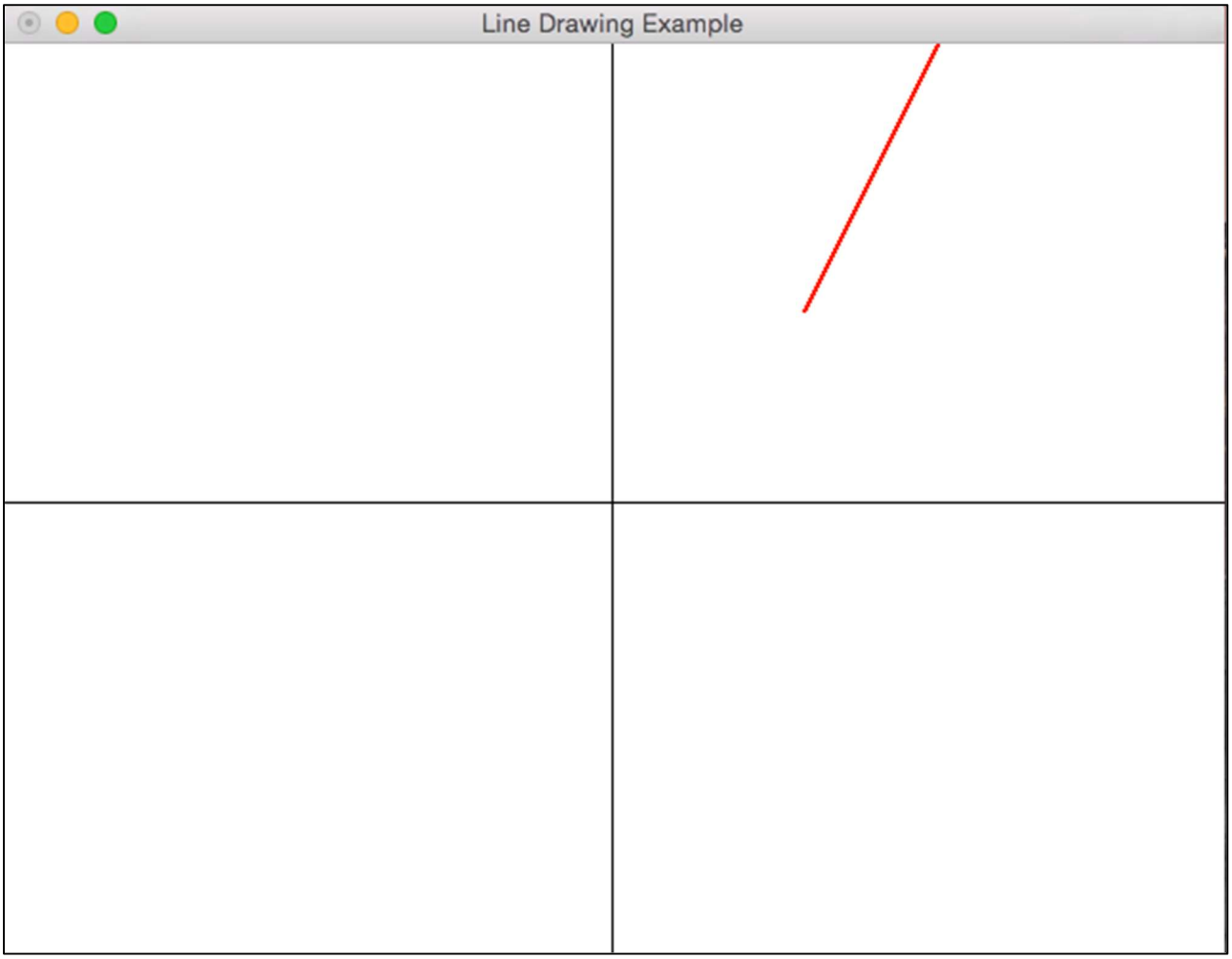
Source code:

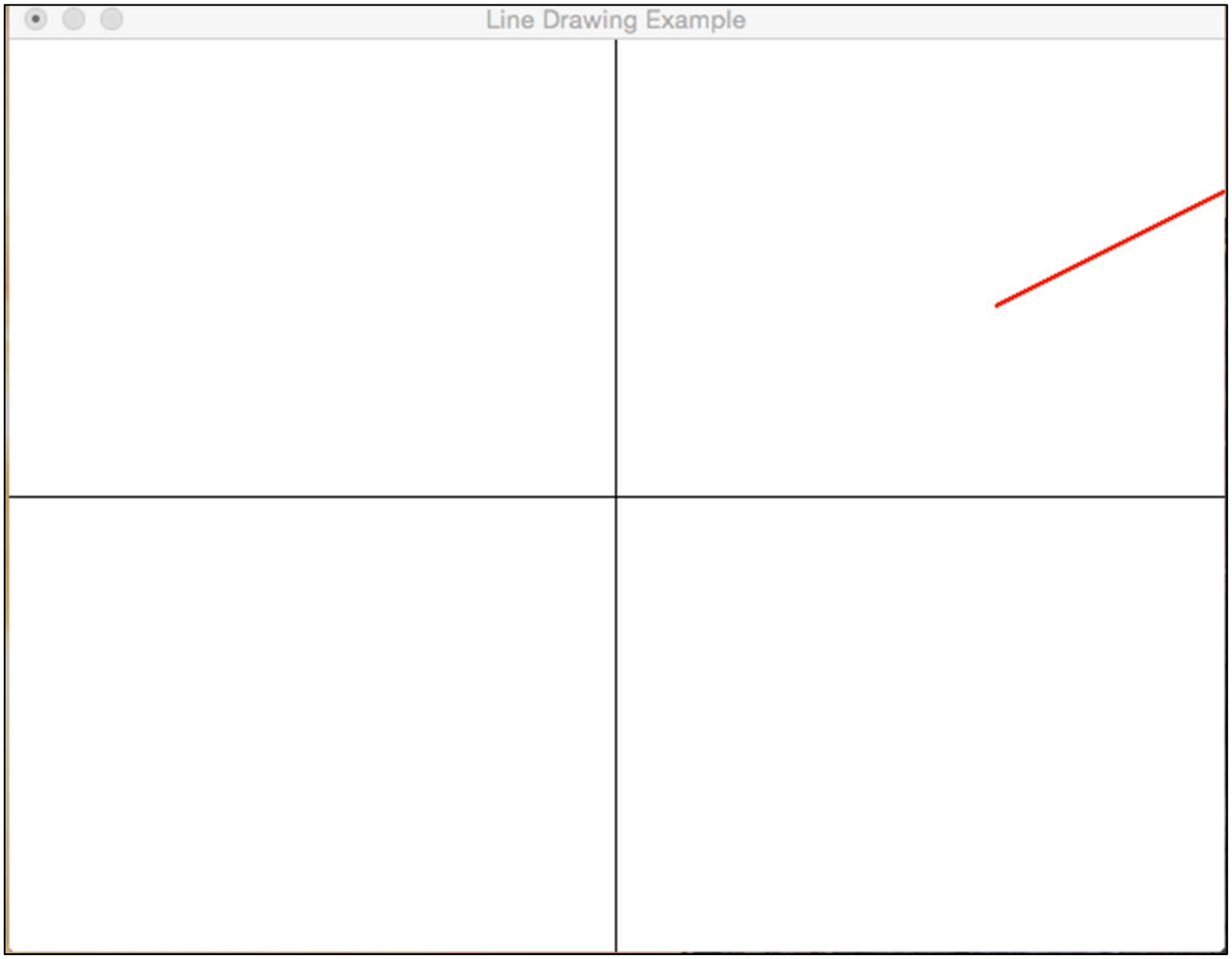
```
#include <iostream>
#include <GLUT/glut.h>
#include <cmath>
using namespace std;

int flag = 0;
int x_1 = 0, y_1 = 0, x_2 = 0, y_2 = 0;
string cnt = "YES";

void myInit()
{
    glClearColor(1.0, 1.0, 1.0, 0.0);
    glPointSize(2);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0, 640.0, 0.0, 480.0);
}

void drawLine()
{
```

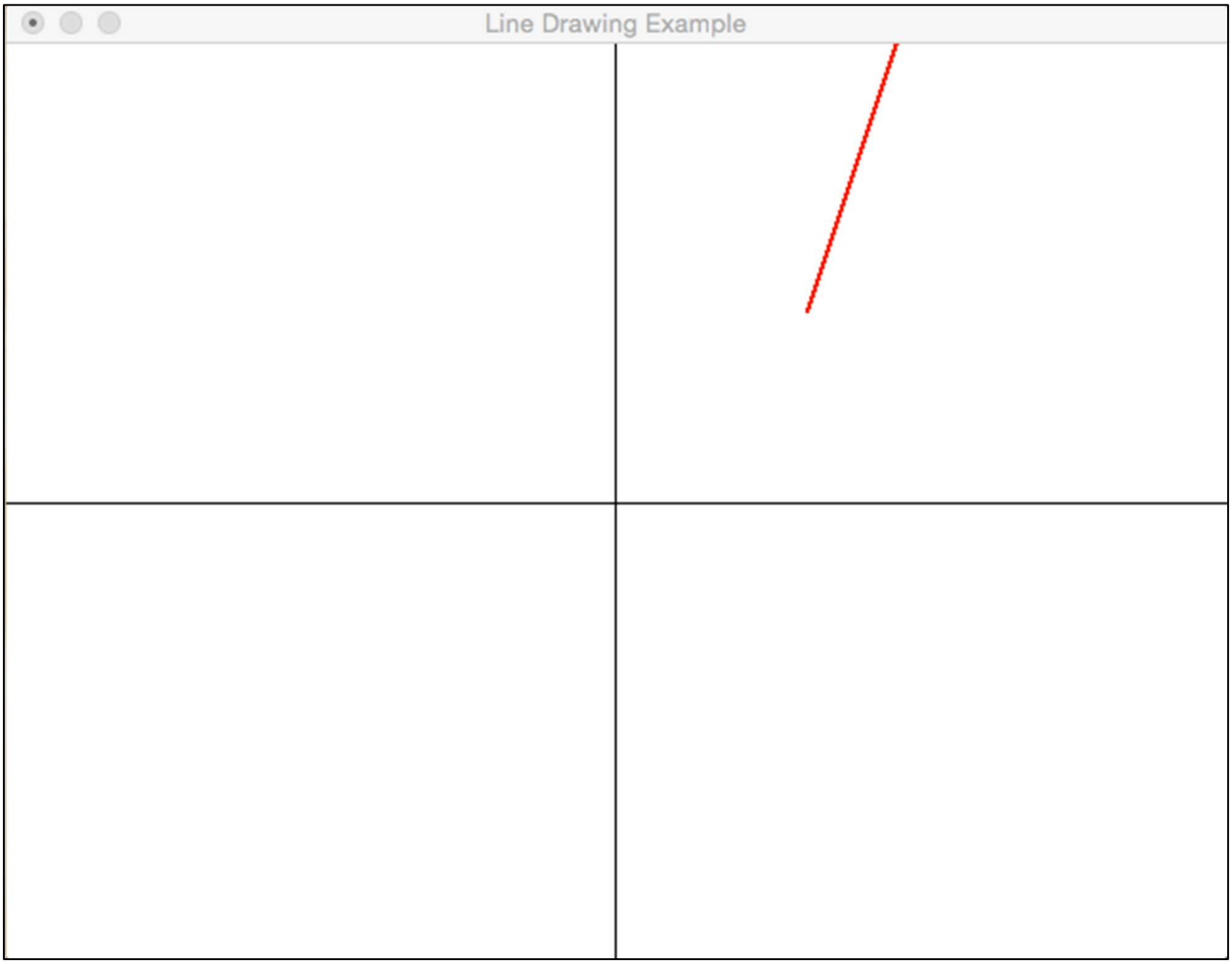
```
    x_1 += 320;
    y_1 += 240;
    x_2 += 320;
    y_2 += 420;
    glColor3f(1.0f, 0.0f, 0.0f);
    float dx = x_2 - x_1;
    float dy = y_2 - y_1;
    int p = 2 * dy - dx;
    glBegin(GL_POINTS); // Begin drawing pointswhile (x_1 < x_2)
    {
        glVertex2f(x_1, y_1);
        x_1++;
        if (x_1 >= 0)
        {
            p -= 2 * dx;
            y_1++;
        }
        p += 2 * dx;
    }
    glEnd();
    // End drawing points
}

void drawAxes()
{
    glBegin(GL_LINES);
    // Draw X-axis
    glColor3f(0.0f, 0.0f, 0.0f); // Set color to black
    glVertex2f(0, 240);          // X-axis starting point
    glVertex2f(640, 240);        // X-axis ending point
    // Draw Y-axis
    glVertex2f(320, 0);   // Y-axis starting point
    glVertex2f(320, 480); // Y-axis ending point
    glEnd();              // End drawing lines
}

void myDisplay()
{
    glClear(GL_COLOR_BUFFER_BIT);
    if (flag == 0)
    {
        flag = 1;
        x_1 = 0, y_1 = 0, x_2 = 0, y_2 = 0;
        cout << "Point 1 : ";
        cin >> x_1 >> y_1;
        cout << "Point 2 : ";
        cin >> x_2 >> y_2;
    }
    drawAxes();
    drawLine();
    glFlush();
    cout << "Want to continue (YES/NO) : ";
    cin >> cnt;
    if (cnt == "NO")
```

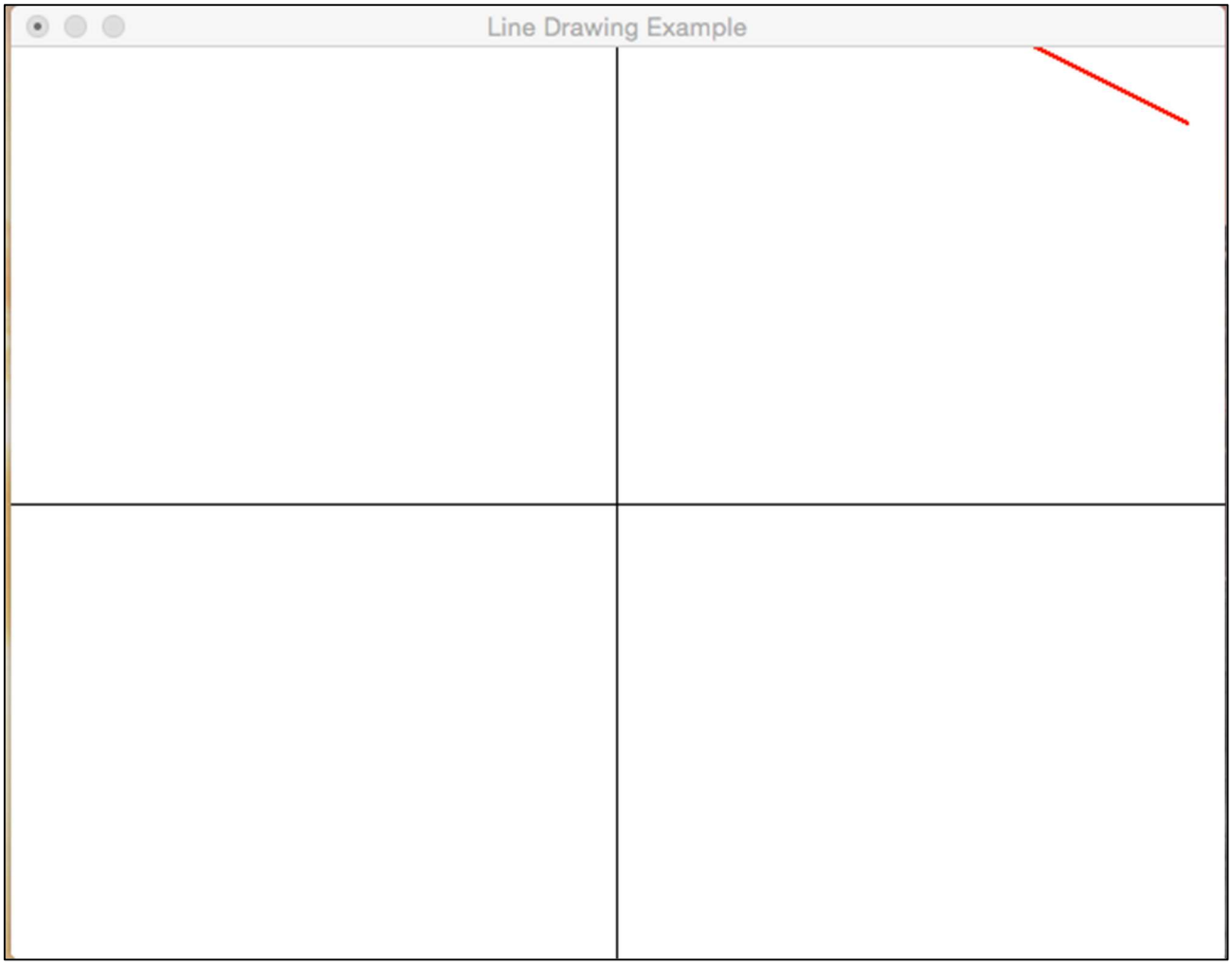
```
    {
        cout << "Exiting...\n";
        exit(0);
    }
    flag = 0;             // Reset flag for the next line
    glutPostRedisplay(); // Continue updating the display
}

int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(640, 480);
    glutCreateWindow("Line Drawing Example");
    myInit();
    glutDisplayFunc(myDisplay);
    glutMainLoop();
    return 0;
}
```

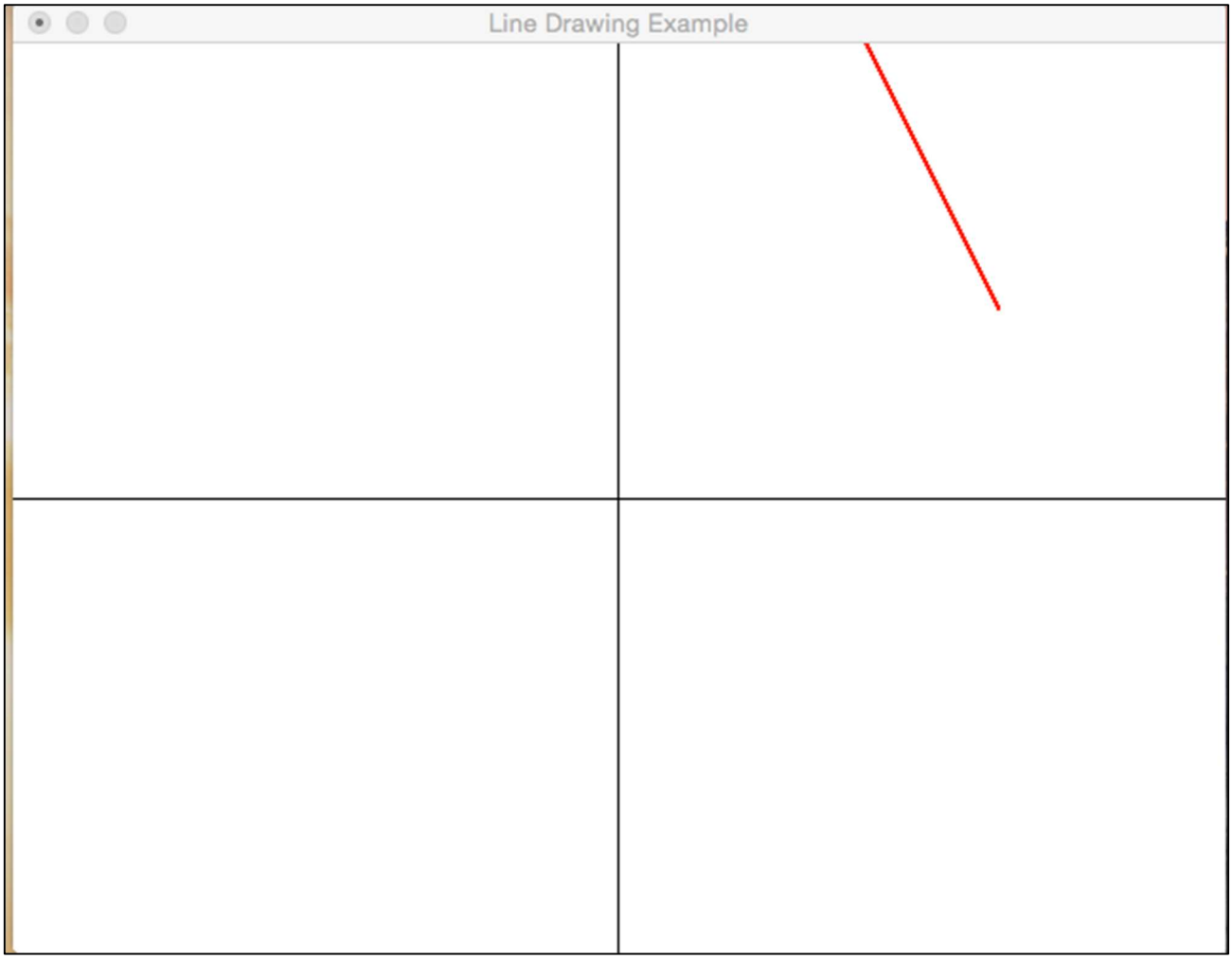

Line Drawing Example

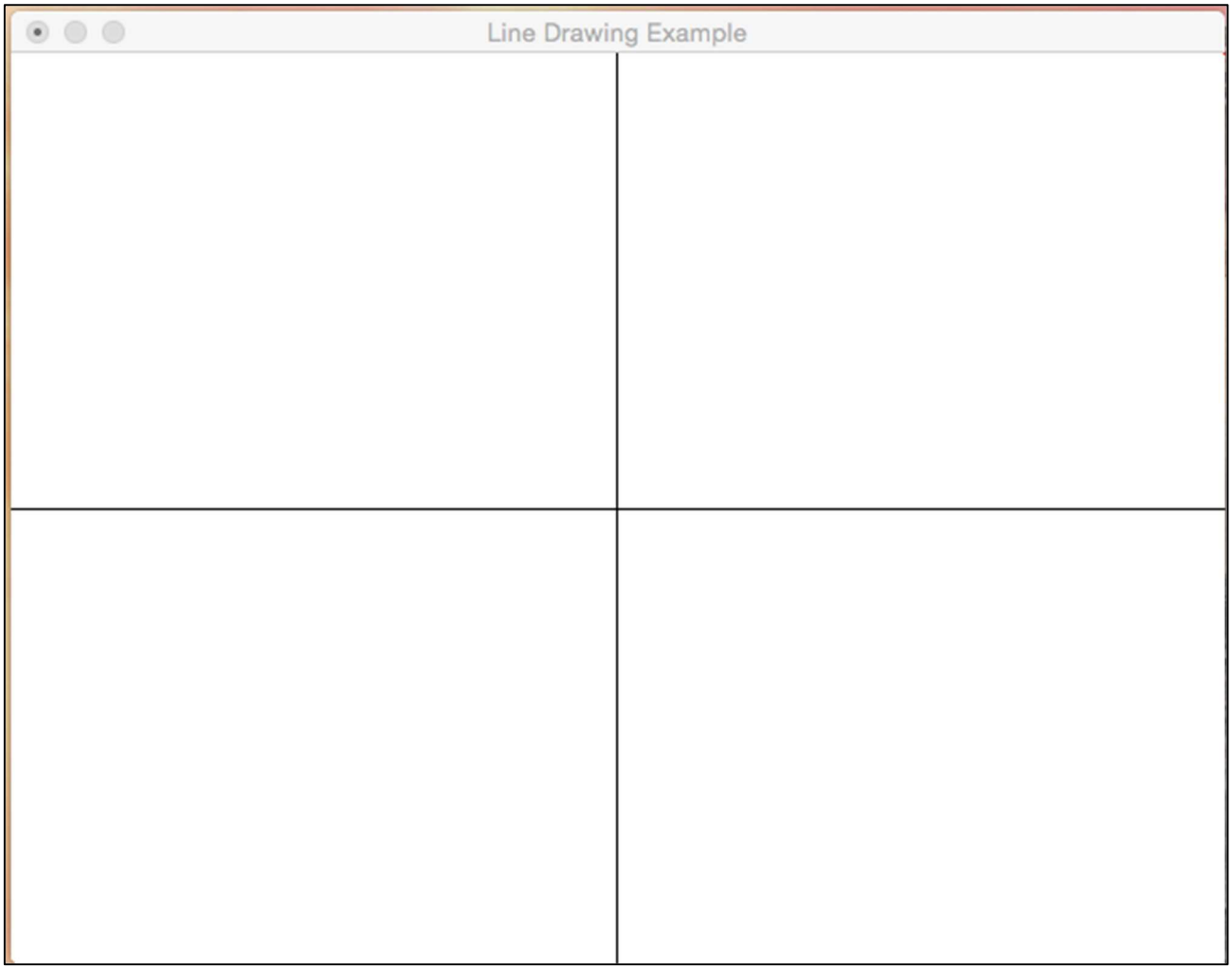

Line Drawing Example

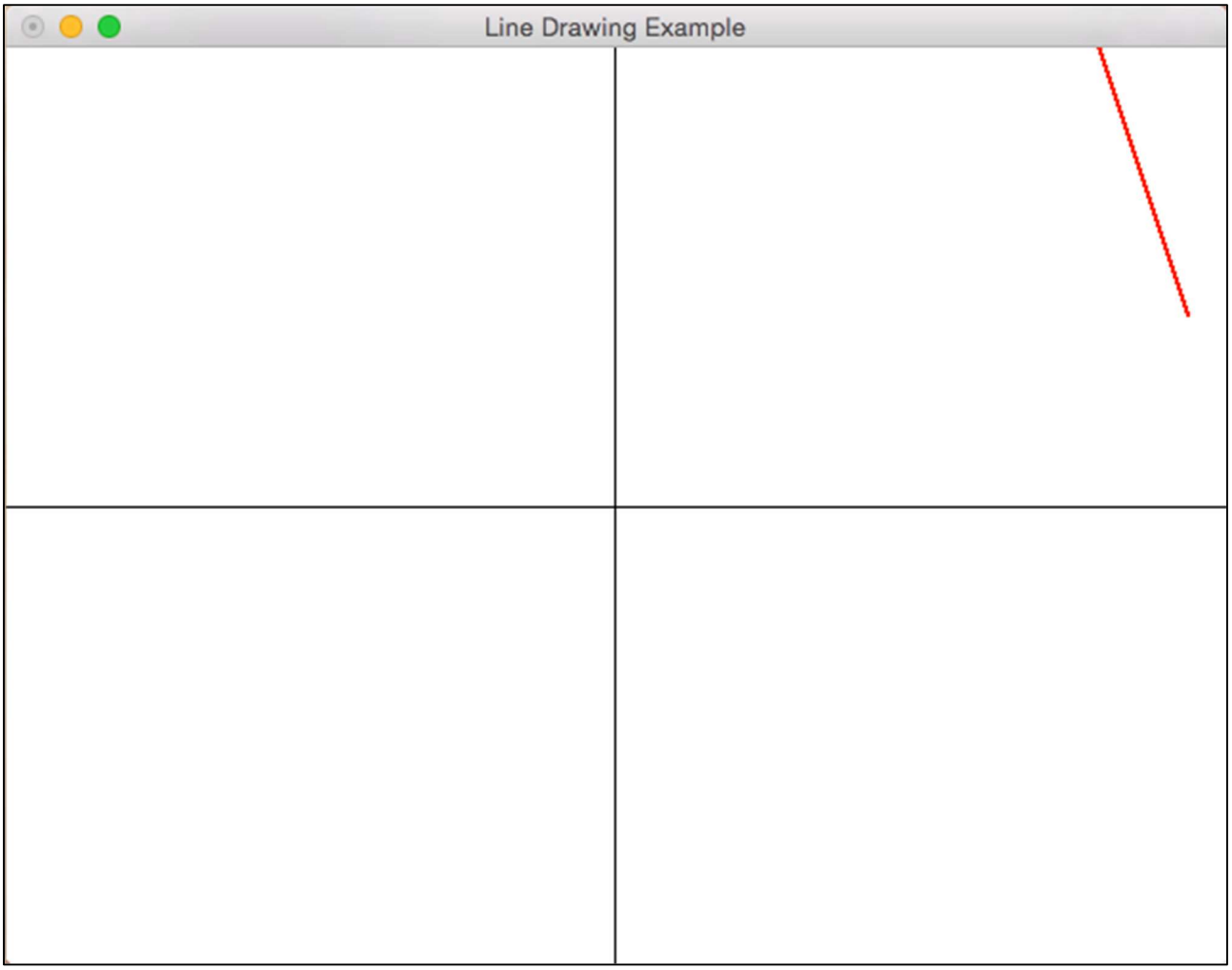

Line Drawing Example


Line Drawing Example


Line Drawing Example


Line Drawing Example


Line Drawing Example




```
Point 1 : 100 100
Point 2 : 300 200
Want to continue (YES/NO) : YES
Sep  4 10:31:18 ssn-macs-Mac-mini.local DDA
update.
Point 1 : 100 100
Point 2 : 200 300
Want to continue (YES/NO) : YES
Sep  4 10:33:14 ssn-macs-Mac-mini.local DDA
update.
Point 1 : 400 200
Point 2 : 200 100
Want to continue (YES/NO) : YES
Point 1 : 200 400
Point 2 : 100 100
Want to continue (YES/NO) : YES
Sep  4 10:36:37 ssn-macs-Mac-mini.local DDA
update.
Point 1 : 100 300
Point 2 : 300 200
Want to continue (YES/NO) : YES
Sep  4 10:37:10 ssn-macs-Mac-mini.local DDA
update.
Point 1 : 100 300
Point 2 : 200 100
Want to continue (YES/NO) : YES
Sep  4 10:38:16 ssn-macs-Mac-mini.local DDA
update.
Point 1 : 400 200 200 300
Point 2 : Want to continue (YES/NO) : YES
```

```
Point 1 : 400 200 200 300
Point 2 : Want to continue (YES/NO) : YES
Sep  4 10:38:46 ssn-macs-Mac-mini.local DDA[8
update.
Point 1 : 200 400
Point 2 : 300 100
Want to continue (YES/NO) : NO
```

SSN COLLEGE OF ENGINEERING, KALAVAKKAM

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

UCS1712-Graphics and Multimedia Lab

Programming Assignment 4

Midpoint Circle Drawing Algorithm in C++ using OpenGL

Name: Jayannthan P T *Dept: CSE 'A'* *Roll No.: 205001049*

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)

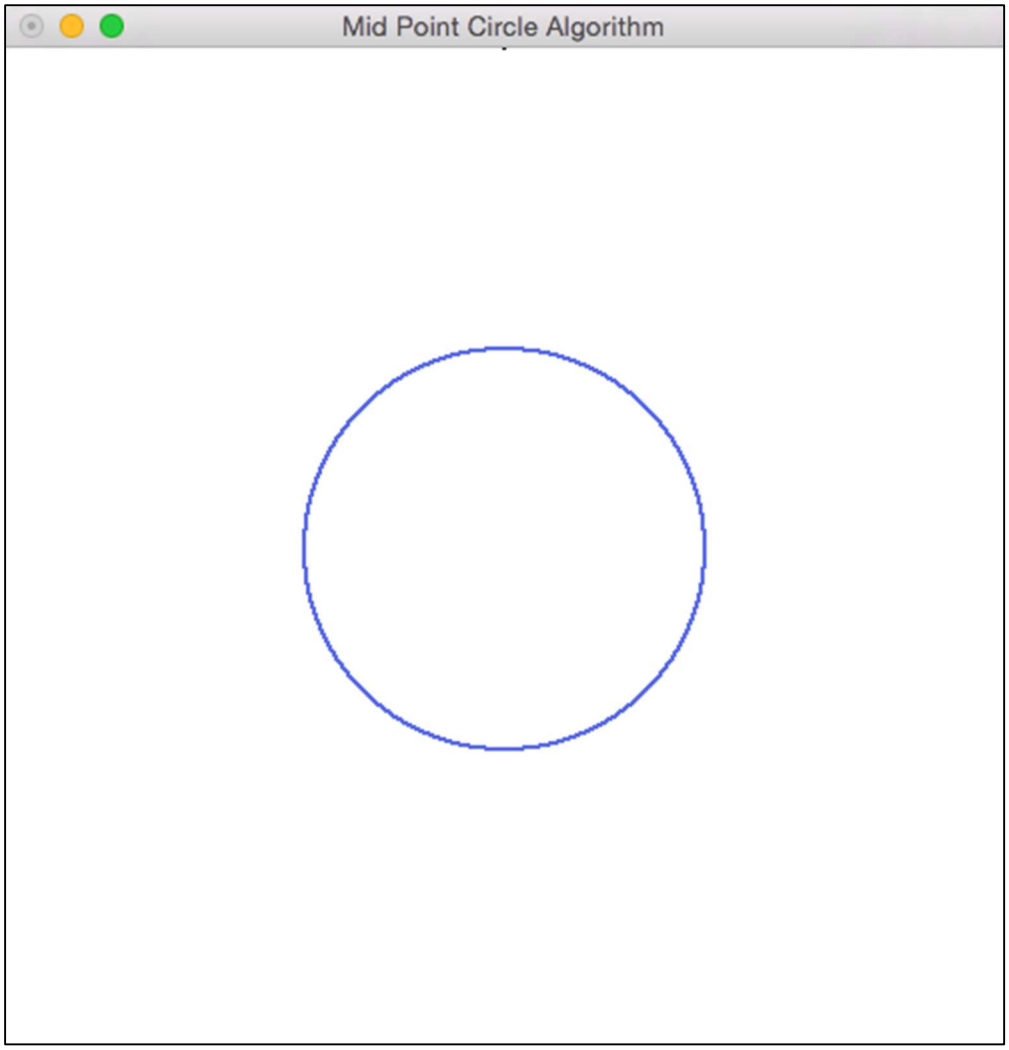
Case 2: With center (xc,yc)

b) To draw any object using line and circle drawing algorithms.

Source code:

```
#include <stdlib.h>
#include <GLUT/glut.h>
#include <iostream>
using namespace std;

int xc, yc, r;

void myInit()

{
    glClearColor(1.0, 1.0, 1.0, 0.0);
    glColor3f(0.4, 0.4, 0.9);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glPointSize(2);
    gluOrtho2D(-250.0, 250.0, -250.0, 250.0);
}

void plotAll(int x, int y, int xc, int yc)

{
    glVertex2d(x + xc, y + yc);
    glVertex2d(x + xc, -y + yc);
    glVertex2d(-x + xc, y + yc);
    glVertex2d(-x + xc, -y + yc);
    glVertex2d(y + xc, x + yc);
    glVertex2d(y + xc, -x + yc);
```

```
    glVertex2d(-y + xc, x + yc);
    glVertex2d(-y + xc, -x + yc);
}

void circle()

{
    glClear(GL_COLOR_BUFFER_BIT);
    int x = r, y = 0, pk = 1 - r;
    glBegin(GL_POINTS);       // Draw the x-axis and y-axis
    glColor3f(0.0, 0.0, 0.0); // Set color to black for axes
    // X-axis
    glVertex2d(-250, 0);
    glVertex2d(250, 0);
    // Y-axis
    glVertex2d(0, -250);
    glVertex2d(0, 250);
    // Draw the circle using Mid-Point Circle Algorithm
    lColor3f(0.4, 0.4, 0.9); // Set color back to blue for the circle
    plotAll(x, y, xc, yc);
    while (x > y)
    {
        y++;
        if (pk < 0)
        {
            pk += (2 * y) + 1;
        }
        else
        {
            x--;
            pk += (2 * y) - (2 * x) + 1;
        }
        plotAll(x, y, xc, yc);
    }
    glEnd();
    glFlush();
}

int main(int argc, char *argv[])

{
    cout << "Enter circle center coordinates:";
    cin >> xc >> yc;
    cout << "Enter radius:";
    cin >> r;
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutCreateWindow("Mid Point Circle Algorithm");
    glutDisplayFunc(circle);
    myInit();
    glutMainLoop();
    return 1;
}
```


Mid Point Circle Algorithm

SSN COLLEGE OF ENGINEERING, KALAVAKKAM

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

UCS1712-Graphics and Multimedia Lab

Programming Assignment 5

2D Transformations in C++ using OpenGL

Name: Jayannthan P T *Dept: CSE 'A'* *Roll No.: 205001049*

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 the x and y axis).

Source code:

```
#define GL_SILENCE_DEPRECATION
#include <GLUT/glut.h>
#include <stdio.h>
#include <iostream>
#include <math.h>
using namespace std;
float toRad(float xDeg)
{
    return xDeg * 3.14159 / 180;
}
void myInit()
{
    glClearColor(1, 1, 1, 1);    // violet
    glColor3f(0.0f, 0.0f, 0.5f); // dark blue
    // glPointSize(10);
    glMatrixMode(GL_PROJECTION);
    glLineWidth(2);
    glLoadIdentity();
    gluOrtho2D(0.0, 640.0, 0.0, 480.0);
}
void displayPoint(float x, float y)
{
    glBegin(GL_POINTS);
    glVertex2d(x + 320, y + 240);
    glEnd();
}
void displayHomogeneousPoint(float *h)
{
    float x = *(h + 0);
    float y = *(h + 1);
    glColor4f(0, 1, 0.4, 1); // green
    displayPoint(x, y);
}
void displayLine(int x1, int y1, int x2, int y2)
{
    glBegin(GL_LINES);
    glVertex2d(x1 + 320, y1 + 240);
    glVertex2d(x2 + 320, y2 + 240);
    glEnd();
}
void displayTriangle(int x1, int y1, int x2, int y2, int x3, int y3)
{
    glBegin(GL_TRIANGLES);
    glVertex2d(x1 + 320, y1 + 240);
    glVertex2d(x2 + 320, y2 + 240);
    glVertex2d(x3 + 320, y3 + 240);
    glEnd();
}
void displayTransformedTriangle(float *p1, float *p2, float *p3)
{
    float x1 = *(p1 + 0);
```

```
    float y1 = *(p1 + 1);
    float x2 = *(p2 + 0);
    float y2 = *(p2 + 1);
    float x3 = *(p3 + 0);
    float y3 = *(p3 + 1);
    glColor4f(0, 1, 0.4, 1); // green
    displayTriangle(x1, y1, x2, y2, x3, y3);
}
void drawPlane()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor4f(0, 0, 0, 1);        // yellow
    displayLine(-320, 0, 320, 0); // x-axis
    displayLine(0, -240, 0, 240); // y-axis
    glFlush();
}
void printMenu()
{
    cout << "1 - Translation" << endl;
    cout << "2 - Rotation about origin" << endl;
    cout << "3 - Rotation wrt fixed point" << endl;
    cout << "4 - Scaling wrt origin" << endl;
    cout << "5 - Scaling wrt fixed point" << endl;
    cout << "6 - Reflection wrt x-axis" << endl;
    cout << "7 - Reflection wrt y-axis" << endl;
    cout << "8 - Reflection wrt origin" << endl;
    cout << "9 - Reflection wrt line x=y" << endl;
    cout << "10 - Shearing along x-dir" << endl;
    cout << "11 - Shearing along y-dir" << endl;
    cout << "0 - All done" << endl;
}
void printMatrix(float *arr, int m, int n)
{
    int i, j;
    for (i = 0; i < m; i++)
    {
        for (j = 0; j < n; j++)
            cout << *((arr + i * n) + j) << " ";
        cout << endl;
    }
}
float *mulMatrix(float *a, int m1, int n1, float *b, int m2, int n2)
{
    if (n1 != m2)
    {
        cout << "Multiplication Input Error" << endl;
        return NULL;
    }
    float *res = new float[m1 * n2];
    for (int i = 0; i < m1; i++)
    {
        for (int j = 0; j < n2; j++)
        {
            *((res + i * n2) + j) = 0;
```

```
            for (int k = 0; k < n1; k++)
            {
                *((res + i * n2) + j) += *((a + i * n1) + k) * *((b + k * n2) + j);
            }
        }
    }
    return res;
}
void printPoint(float *P)
{
    printMatrix(P, 3, 1);
}
void printMatrix3(float *M)
{
    printMatrix(M, 3, 3);
}
float *transformPoint(float *m, float *p)
{
    return mulMatrix(m, 3, 3, p, 3, 1);
}
float *mulTransforms(float *m1, float *m2)
{
    return mulMatrix(m1, 3, 3, m2, 3, 3);
}
float *getTransformationMatrix()
{
    cout << "COMPOSITE TRANSFORMATION" << endl;
    float *compositeMatrix = new float[3 * 3];
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            compositeMatrix[i * 3 + j] = (i == j) ? 1 : 0;
        }
    }
    printMenu();
    int ch;
    do
    {
        cout << "\nChoose required transformation: ";
        cin >> ch;
        switch (ch)
        {
        case 1:
        {
            cout << "TRANSLATION" << endl;
            float tx, ty;
            cout << "Enter translation values: ";
            cin >> tx >> ty;
            float T[3][3] = {
                {1, 0, tx},
                {0, 1, ty},
                {0, 0, 1}};
            float *temp = mulTransforms((float *)T, compositeMatrix);
```

```
            delete[] compositeMatrix;
            compositeMatrix = temp;
            break;
        }
        case 2:
        {
            cout << "ROTATION ABOUT ORIGIN" << endl;
            float angle;
            cout << "Enter rotation angle: ";
            cin >> angle;
            float theta = toRad(angle);
            float c = cos(theta);
            float s = sin(theta);
            float R[3][3] = {
                {c, -s, 0},
                {s, c, 0},
                {0, 0, 1}};
            float *temp = mulTransforms((float *)R, compositeMatrix);
            delete[] compositeMatrix;
            compositeMatrix = temp;
            break;
        }
        case 3:
        {
            cout << "ROTATION WRT FIXED POINT" << endl;
            float angle;
            cout << "Enter rotation angle: ";
            cin >> angle;
            float theta = toRad(angle);
            float c = cos(theta);
            float s = sin(theta);
            float xr, yr;
            cout << "Enter fixed point coords: ";
            cin >> xr >> yr;
            float R[3][3] = {
                {c, -s, (xr * (1 - c)) + (yr * s)},
                {s, c, (yr * (1 - c)) - (xr * s)},
                {0, 0, 1}};
            float *temp = mulTransforms((float *)R, compositeMatrix);
            delete[] compositeMatrix;
            compositeMatrix = temp;
            break;
        }
        case 4:
        {
            cout << "SCALING WRT ORIGIN" << endl;
            float sx, sy;
            cout << "Enter scaling factor values: ";
            cin >> sx >> sy;
            float S[3][3] = {
                {sx, 0, 0},
                {0, sy, 0},
                {0, 0, 1}};
            float *temp = mulTransforms((float *)S, compositeMatrix);
```

```
            delete[] compositeMatrix;
            compositeMatrix = temp;
            break;
        }
        case 5:
        {
            cout << "SCALING WRT FIXED POINT" << endl;
            float sx, sy;
            cout << "Enter scaling factor values: ";
            cin >> sx >> sy;
            float xf, yf;
            cout << "Enter fixed point coords: ";
            cin >> xf >> yf;
            float S[3][3] = {
                {sx, 0, xf * (1 - sx)},
                {0, sy, yf * (1 - sy)},
                {0, 0, 1}};
            float *temp = mulTransforms((float *)S, compositeMatrix);
            delete[] compositeMatrix;
            compositeMatrix = temp;
            break;
        }
        case 6:
        {
            cout << "REFLECTION WRT X-AXIS" << endl;
            float RF[3][3] = {
                {1, 0, 0},
                {0, -1, 0},
                {0, 0, 1}};
            float *temp = mulTransforms((float *)RF, compositeMatrix);
            delete[] compositeMatrix;
            compositeMatrix = temp;
            break;
        }
        case 7:
        {
            cout << "REFLECTION WRT Y-AXIS" << endl;
            float RF[3][3] = {
                {-1, 0, 0},
                {0, 1, 0},
                {0, 0, 1}};
            float *temp = mulTransforms((float *)RF, compositeMatrix);
            delete[] compositeMatrix;
            compositeMatrix = temp;
            break;
        }
        case 8:
        {
            cout << "REFLECTION WRT ORIGIN" << endl;
            float RF[3][3] = {
                {-1, 0, 0},
                {0, -1, 0},
                {0, 0, 1}};
            float *temp = mulTransforms((float *)RF, compositeMatrix);
```

```
            delete[] compositeMatrix;
            compositeMatrix = temp;
            break;
        }
        case 9:
        {
            cout << "REFLECTION WRT LINE X=Y" << endl;
            float RF[3][3] = {
                {0, 1, 0},
                {1, 0, 0},
                {0, 0, 1}};
            float *temp = mulTransforms((float *)RF, compositeMatrix);
            delete[] compositeMatrix;
            compositeMatrix = temp;
            break;
        }
        case 10:
        {
            cout << "SHEARING ALONG X-DIR" << endl;
            float shx, yref = 0;
            cout << "Enter shear value: ";
            cin >> shx;
            cout << "Enter yref value: ";
            cin >> yref;
            float SH[3][3] = {
                {1, shx, -shx * yref},
                {0, 1, 0},
                {0, 0, 1}};
            float *temp = mulTransforms((float *)SH, compositeMatrix);
            delete[] compositeMatrix;
            compositeMatrix = temp;
            break;
        }
        case 11:
        {
            cout << "SHEARING ALONG Y-DIR" << endl;
            float shy, xref = 0;
            cout << "Enter shear value: ";
            cin >> shy;
            cout << "Enter yref value: ";
            cin >> xref;
            float SH[3][3] = {
                {1, 0, 0},
                {shy, 1, -shy * xref},
                {0, 0, 1}};
            float *temp = mulTransforms((float *)SH, compositeMatrix);
            delete[] compositeMatrix;
            compositeMatrix = temp;
            break;
        }
        case 0:
        {
            cout << "ALL DONE" << endl;
        }
```

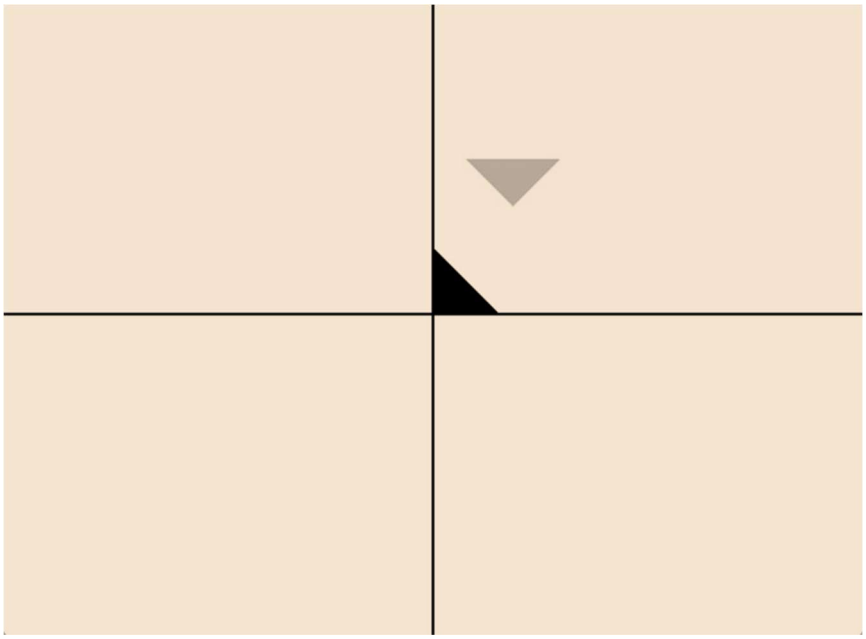
```
        default:
            break;
        }
    } while (ch != 0);
    return compositeMatrix;
}
void plotTransform()
{
    cout << "TRANSFORMATION OF A TRIANGLE" << endl;
    // Point P1
    float x1, y1;
    cout << "Enter point P1 coords: ";
    cin >> x1 >> y1;
    float *P1 = new float[3]{{x1}, {y1}, {1}};
    cout << "Homogeneous representation of P1: " << endl;
    printPoint(P1);
    cout << endl;
    // Point P2
    float x2, y2;
    cout << "Enter point P2 coords: ";
    cin >> x2 >> y2;
    float *P2 = new float[3]{{x2}, {y2}, {1}};
    cout << "Homogeneous representation of P2: " << endl;
    printPoint(P2);
    cout << endl;
    // Point P3
    float x3, y3;
    cout << "Enter point P3 coords: ";
    cin >> x3 >> y3;
    float *P3 = new float[3]{{x3}, {y3}, {1}};
    cout << "Homogeneous representation of P3: " << endl;
    printPoint(P3);
    cout << endl;
    // plot triangle
    displayTriangle(x1, y1, x2, y2, x3, y3);
    float *M = getTransformationMatrix();
    if (M != NULL)
    {
        cout << "\nTransformation Matrix: " << endl;
        printMatrix3(M);
        cout << "\nP1': " << endl;
        float *Q1 = transformPoint(M, P1);
        printPoint(Q1);
        cout << "\nP2': " << endl;
        float *Q2 = transformPoint(M, P2);
        printPoint(Q2);
        cout << "\nP3': " << endl;
        float *Q3 = transformPoint(M, P3);
        printPoint(Q3);
        displayTransformedTriangle(Q1, Q2, Q3);
        delete[] Q1;
        delete[] Q2;
        delete[] Q3;
    }
```

```
    delete[] M;
    delete[] P1;
    delete[] P2;
    delete[] P3;
}
void plotChart()
{
    glClear(GL_COLOR_BUFFER_BIT);
    drawPlane();
    plotTransform();
    glFlush();
}
int main(int argc, char *argv[])
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA);
    glutInitWindowSize(640, 480);
    glutCreateWindow("Transformations");
    glutDisplayFunc(plotChart);
    myInit();
    glutMainLoop();
    return 1;
}
```

Output

```
TRANSFORMATION OF A TRIANGLE
Enter point P1 coords: 0 0
Homogeneous representation of P1:
0
0
1

Enter point P2 coords: 50 0
Homogeneous representation of P2:
50
0
1

Enter point P3 coords: 0 50
Homogeneous representation of P3:
0
50
1

COMPOSITE TRANSFORMATION
1 - Translation
2 - Rotation about origin
3 - Rotation wrt fixed point
4 - Scaling wrt origin
5 - Scaling wrt fixed point
6 - Reflection wrt x-axis
7 - Reflection wrt y-axis
8 - Reflection wrt origin
9 - Reflection wrt line x=y
10 - Shearing along x-dir
11 - Shearing along y-dir
0 - All done
```

```
Choose required transformation: 1
TRANSLATION
Enter translation values: 60 80

Choose required transformation: 3
ROTATION WRT FIXED POINT
Enter rotation angle: 45
Enter fixed point coords: 60 80

Choose required transformation: 0
ALL DONE

Transformation Matrix:
0.707107 -0.707106 60
0.707106 0.707107 80
0 0 1

P1':
60
80
1

P2':
95.3554
115.355
1

P3':
24.6447
115.355
1
```



SSN COLLEGE OF ENGINEERING, KALAVAKKAM

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

UCS1712-Graphics and Multimedia Lab

Programming Assignment 6

2D Composite Transformations and Windowing in C++ using OpenGL

Name: Jayannthan P T *Dept: CSE 'A'* *Roll No.: 205001049*

a) To compute the composite transformation matrix for any 2 transformations input by the user and apply it on the object.

1) Translation
2) Rotation
3) Scaling
4) Reflection
5) Shearing

Display the original and the transformed 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.

Source code:

```
#define GL_SILENCE_DEPRECATION
#include<GLUT/glut.h>
#include<stdio.h>
#include<iostream>
#include<math.h>
using namespace std;
float toRad(float xDeg) {
    return xDeg * 3.14159 / 180;
}
void myInit() {
    glClearColor(1, 1, 1, 1); // violet
    glColor3f(0.0f, 0.0f, 0.5f); //dark blue
    //glPointSize(10);
    glMatrixMode(GL_PROJECTION);
    glLineWidth(2);
    glLoadIdentity();
    gluOrtho2D(0.0, 640.0, 0.0, 480.0);
}
void displayPoint(float x, float y) {
    glBegin(GL_POINTS);
    glVertex2d(x + 320, y + 240);
    glEnd();
}
void displayHomogeneousPoint(float* h) {
    float x = *(h + 0);
    float y = *(h + 1);
    glColor4f(0, 1, 0.4, 1); //green
    displayPoint(x, y);
}
void displayLine(int x1, int y1, int x2, int y2) {
    glBegin(GL_LINES);
    glVertex2d(x1 + 320, y1 + 240);
    glVertex2d(x2 + 320, y2 + 240);
    glEnd();
}
void displayTriangle(int x1, int y1, int x2, int y2, int x3, int y3) {
    glBegin(GL_TRIANGLES);
    glVertex2d(x1 + 320, y1 + 240);
    glVertex2d(x2 + 320, y2 + 240);
    glVertex2d(x3 + 320, y3 + 240);
    glEnd();
}
void displayTransformedTriangle(float* p1, float* p2, float* p3) {
    float x1 = *(p1 + 0);
    float y1 = *(p1 + 1);
    float x2 = *(p2 + 0);
    float y2 = *(p2 + 1);
    float x3 = *(p3 + 0);
    float y3 = *(p3 + 1);
    glColor4f(0, 1, 0.4, 1); //green
    displayTriangle(x1, y1, x2, y2, x3, y3);
```

```
}
void drawPlane() {
    glClear(GL_COLOR_BUFFER_BIT);
    glColor4f(0, 0, 0, 1); //yellow
    displayLine(-320, 0, 320, 0); //x-axis
    displayLine(0, -240, 0, 240); //y-axis
    glFlush();
}
void printMenu() {
    cout << "1 - Translation" << endl;
    cout << "2 - Rotation about origin" << endl;
    cout << "3 - Rotation wrt fixed point" << endl;
    cout << "4 - Scaling wrt origin" << endl;
    cout << "5 - Scaling wrt fixed point" << endl;
    cout << "6 - Reflection wrt x-axis" << endl;
    cout << "7 - Reflection wrt y-axis" << endl;
    cout << "8 - Reflection wrt origin" << endl;
    cout << "9 - Reflection wrt line x=y" << endl;
    cout << "10 - Shearing along x-dir" << endl;
    cout << "11 - Shearing along y-dir" << endl;
    cout << "0 - All done" << endl;
}
void printMatrix(float* arr, int m, int n)
{
    int i, j;
    for (i = 0; i < m; i++) {
        for (j = 0; j < n; j++)
            cout << *((arr + i * n) + j) << " ";
        cout << endl;
    }
}
float* mulMatrix(float* a, int m1, int n1, float* b, int m2, int n2) {
    if (n1 != m2) {
        cout << "Multiplication Input Error" << endl;
        return NULL;
    }
    float* res = new float[m1 * n2];
    for (int i = 0; i < m1; i++) {
        for (int j = 0; j < n2; j++) {
            *((res + i * n2) + j) = 0;
            for (int k = 0; k < n1; k++) {
                *((res + i * n2) + j) += *((a + i * n1) + k) * *((b + k * n2) + j);
            }
        }
    }
    return res;
}
void printPoint(float* P) {
    printMatrix(P, 3, 1);
}
void printMatrix3(float* M) {
    printMatrix(M, 3, 3);
}
float* transformPoint(float* m, float* p) {
```

```
    return mulMatrix(m, 3, 3, p, 3, 1);
}
float* mulTransforms(float* m1, float* m2) {
    return mulMatrix(m1, 3, 3, m2, 3, 3);
}
float* getTransformationMatrix() {
    cout << "COMPOSITE TRANSFORMATION" << endl;
    float* compositeMatrix = new float[3 * 3];
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            compositeMatrix[i * 3 + j] = (i == j) ? 1 : 0;
        }
    }
    printMenu();
    int ch;
    do {
        cout << "\nChoose required transformation: ";
        cin >> ch;
        switch (ch) {
        case 1: {
            cout << "TRANSLATION" << endl;
            float tx, ty;
            cout << "Enter translation values: ";
            cin >> tx >> ty;
            float T[3][3] = {
                {1, 0, tx},
                {0, 1, ty},
                {0, 0, 1}
            };
            float* temp = mulTransforms((float*)T, compositeMatrix);
            delete[] compositeMatrix;
            compositeMatrix = temp;
            break;
        }
        case 2: {
            cout << "ROTATION ABOUT ORIGIN" << endl;
            float angle;
            cout << "Enter rotation angle: ";
            cin >> angle;
            float theta = toRad(angle);
            float c = cos(theta);
            float s = sin(theta);
            float R[3][3] = {
                {c, -s, 0},
                {s, c, 0},
                {0, 0, 1}
            };
            float* temp = mulTransforms((float*)R, compositeMatrix);
            delete[] compositeMatrix;
            compositeMatrix = temp;
            break;
        }
        case 3: {
            cout << "ROTATION WRT FIXED POINT" << endl;
```

```
            float angle;
            cout << "Enter rotation angle: ";
            cin >> angle;
            float theta = toRad(angle);
            float c = cos(theta);
            float s = sin(theta);
            float xr, yr;
            cout << "Enter fixed point coords: ";
            cin >> xr >> yr;
            float R[3][3] = {
                {c, -s, (xr * (1 - c)) + (yr * s)},
                {s, c, (yr * (1 - c)) - (xr * s)},
                {0, 0, 1}
            };
            float* temp = mulTransforms((float*)R, compositeMatrix);
            delete[] compositeMatrix;
            compositeMatrix = temp;
            break;
        }
        case 4: {
            cout << "SCALING WRT ORIGIN" << endl;
            float sx, sy;
            cout << "Enter scaling factor values: ";
            cin >> sx >> sy;
            float S[3][3] = {
                {sx, 0, 0},
                {0, sy, 0},
                {0, 0, 1}
            };
            float* temp = mulTransforms((float*)S, compositeMatrix);
            delete[] compositeMatrix;
            compositeMatrix = temp;
            break;
        }
        case 5: {
            cout << "SCALING WRT FIXED POINT" << endl;
            float sx, sy;
            cout << "Enter scaling factor values: ";
            cin >> sx >> sy;
            float xf, yf;
            cout << "Enter fixed point coords: ";
            cin >> xf >> yf;
            float S[3][3] = {
                {sx, 0, xf * (1 - sx)},
                {0, sy, yf * (1 - sy)},
                {0, 0, 1}
            };
            float* temp = mulTransforms((float*)S, compositeMatrix);
            delete[] compositeMatrix;
            compositeMatrix = temp;
            break;
        }
        case 6: {
            cout << "REFLECTION WRT X-AXIS" << endl;
```

```
            float RF[3][3] = {
                {1, 0, 0},
                {0, -1, 0},
                {0, 0, 1}
            };
            float* temp = mulTransforms((float*)RF, compositeMatrix);
            delete[] compositeMatrix;
            compositeMatrix = temp;
            break;
        }
        case 7: {
            cout << "REFLECTION WRT Y-AXIS" << endl;
            float RF[3][3] = {
                {-1, 0, 0},
                {0, 1, 0},
                {0, 0, 1}
            };
            float* temp = mulTransforms((float*)RF, compositeMatrix);
            delete[] compositeMatrix;
            compositeMatrix = temp;
            break;
        }
        case 8: {
            cout << "REFLECTION WRT ORIGIN" << endl;
            float RF[3][3] = {
                {-1, 0, 0},
                {0, -1, 0},
                {0, 0, 1}
            };
            float* temp = mulTransforms((float*)RF, compositeMatrix);
            delete[] compositeMatrix;
            compositeMatrix = temp;
            break;
        }
        case 9: {
            cout << "REFLECTION WRT LINE X=Y" << endl;
            float RF[3][3] = {
                {0, 1, 0},
                {1, 0, 0},
                {0, 0, 1}
            };
            float* temp = mulTransforms((float*)RF, compositeMatrix);
            delete[] compositeMatrix;
            compositeMatrix = temp;
            break;
        }
        case 10: {
            cout << "SHEARING ALONG X-DIR" << endl;
            float shx, yref = 0;
            cout << "Enter shear value: ";
            cin >> shx;
            cout << "Enter yref value: ";
            cin >> yref;
            float SH[3][3] = {
```

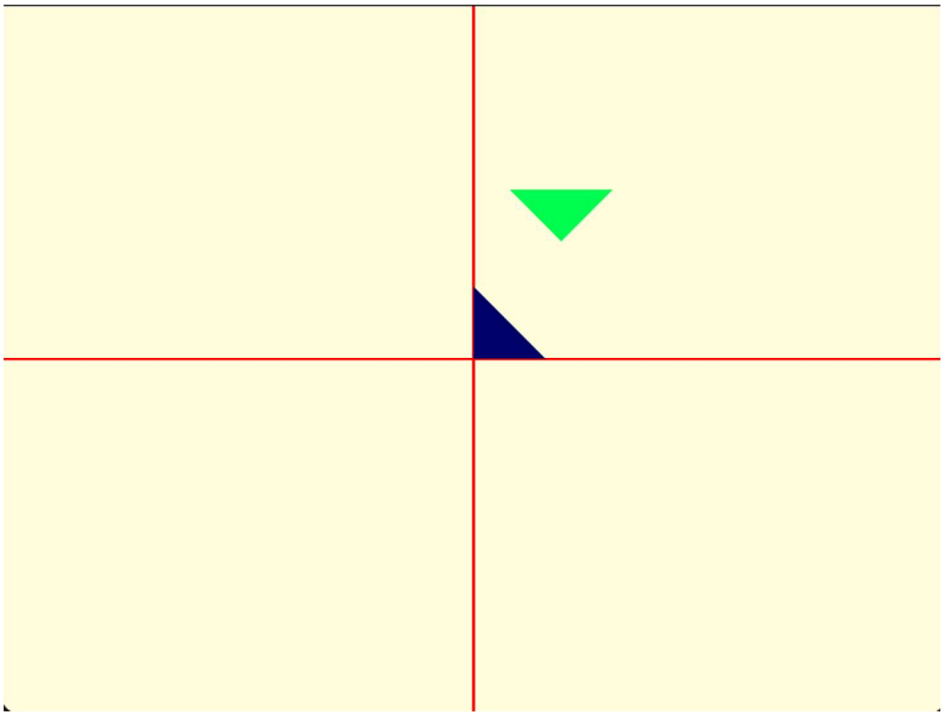
```
                {1, shx,-shx * yref},
                {0, 1, 0},
                {0, 0, 1}
            };
            float* temp = mulTransforms((float*)SH, compositeMatrix);
            delete[] compositeMatrix;
            compositeMatrix = temp;
            break;
        }
        case 11: {
            cout << "SHEARING ALONG Y-DIR" << endl;
            float shy, xref = 0;
            cout << "Enter shear value: ";
            cin >> shy;
            cout << "Enter yref value: ";
            cin >> xref;
            float SH[3][3] = {
                {1, 0, 0},
                {shy, 1, -shy * xref},
                {0, 0, 1}
            };
            float* temp = mulTransforms((float*)SH, compositeMatrix);
            delete[] compositeMatrix;
            compositeMatrix = temp;
            break;
        }
        case 0: {
            cout << "ALL DONE" << endl;
        }
        default: break;
        }
    } while (ch != 0);
    return compositeMatrix;
}
void plotTransform()
{
    cout << "TRANSFORMATION OF A TRIANGLE" << endl;
    //Point P1
    float x1, y1;
    cout << "Enter point P1 coords: ";
    cin >> x1 >> y1;
    float* P1 = new float[3] { {x1}, { y1 }, { 1 } };
    cout << "Homogeneous representation of P1: " << endl;
    printPoint(P1);
    cout << endl;
    //Point P2
    float x2, y2;
    cout << "Enter point P2 coords: ";
    cin >> x2 >> y2;
    float* P2 = new float[3] { {x2}, { y2 }, { 1 } };
    cout << "Homogeneous representation of P2: " << endl;
    printPoint(P2);
    cout << endl;
    //Point P3
```

```
    float x3, y3;
    cout << "Enter point P3 coords: ";
    cin >> x3 >> y3;
    float* P3 = new float[3] { {x3}, { y3 }, { 1 } };
    cout << "Homogeneous representation of P3: " << endl;
    printPoint(P3);
    cout << endl;
    //plot triangle
    displayTriangle(x1, y1, x2, y2, x3, y3);
    float* M = getTransformationMatrix();
    if (M != NULL) {
        cout << "\nTransformation Matrix: " << endl;
        printMatrix3(M);
        cout << "\nP1': " << endl;
        float* Q1 = transformPoint(M, P1);
        printPoint(Q1);
        cout << "\nP2': " << endl;
        float* Q2 = transformPoint(M, P2);
        printPoint(Q2);
        cout << "\nP3': " << endl;
        float* Q3 = transformPoint(M, P3);
        printPoint(Q3);
        displayTransformedTriangle(Q1, Q2, Q3);
        delete[] Q1;
        delete[] Q2;
        delete[] Q3;
    }
    delete[] M;
    delete[] P1;
    delete[] P2;
    delete[] P3;
}
void plotChart() {
    glClear(GL_COLOR_BUFFER_BIT);
    drawPlane();
    plotTransform();
    glFlush();
}
int main(int argc, char* argv[]) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA);
    glutInitWindowSize(640, 480);
    glutCreateWindow("Transformations");
    glutDisplayFunc(plotChart);
    myInit();
    glutMainLoop();
    return 1;
}
```

Output

```
TRANSFORMATION OF A TRIANGLE
Enter point P1 coords: 0 0
Homogeneous representation of P1:
0
0
1

Enter point P2 coords: 50 0
Homogeneous representation of P2:
50
0
1

Enter point P3 coords: 0 50
Homogeneous representation of P3:
0
50
1

COMPOSITE TRANSFORMATION
1 - Translation
2 - Rotation about origin
3 - Rotation wrt fixed point
4 - Scaling wrt origin
5 - Scaling wrt fixed point
6 - Reflection wrt x-axis
7 - Reflection wrt y-axis
8 - Reflection wrt origin
9 - Reflection wrt line x=y
10 - Shearing along x-dir
11 - Shearing along y-dir
0 - All done
```

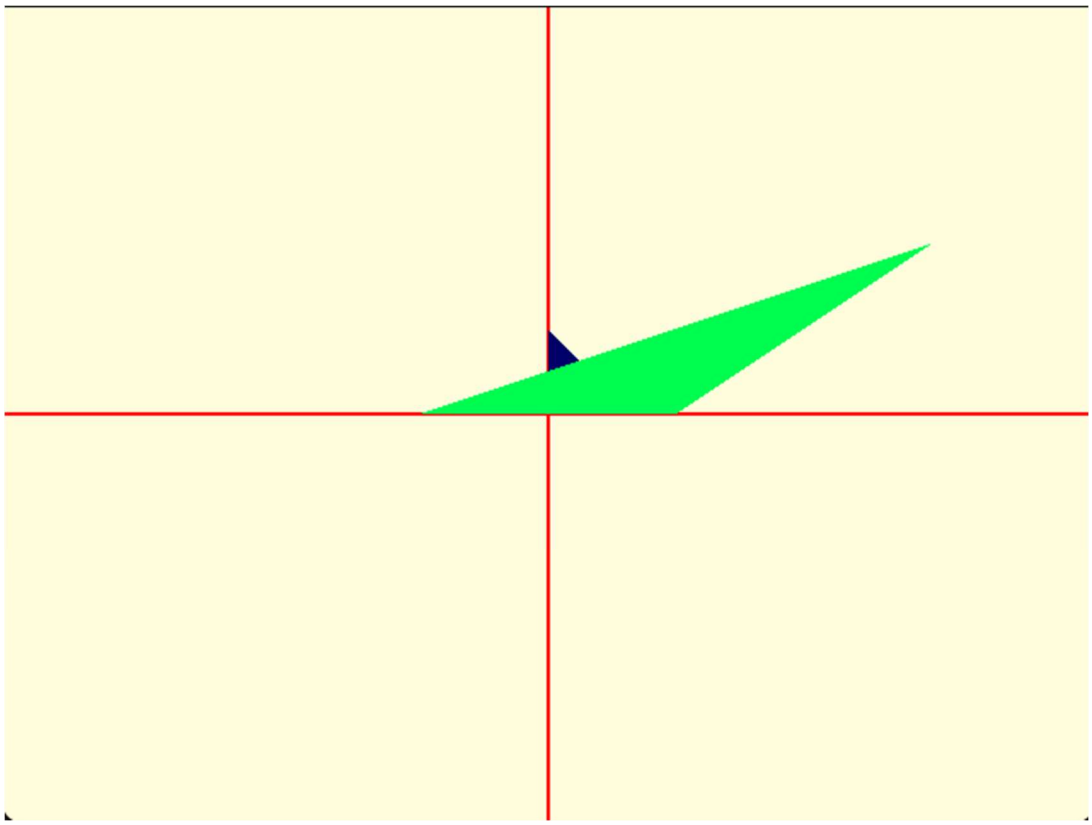
```
Choose required transformation: 1
TRANSLATION
Enter translation values: 60 80

Choose required transformation: 3
ROTATION WRT FIXED POINT
Enter rotation angle: 45
Enter fixed point coords: 60 80

Choose required transformation: 0
ALL DONE

Transformation Matrix:
0.707107 -0.707106 60
0.707106 0.707107 80
0 0 1

P1':
60
80
1

P2':
95.3554
115.355
1

P3':
24.6447
115.355
1
```



```
TRANSFORMATION OF A TRIANGLE
Enter point P1 coords: 0 0
Homogeneous representation of P1:
0
0
1

Enter point P2 coords: 0 50
Homogeneous representation of P2:
0
50
1

Enter point P3 coords: 50 0
Homogeneous representation of P3:
50
0
1

COMPOSITE TRANSFORMATION
1 - Translation
2 - Rotation about origin
3 - Rotation wrt fixed point
4 - Scaling wrt origin
5 - Scaling wrt fixed point
6 - Reflection wrt x-axis
7 - Reflection wrt y-axis
8 - Reflection wrt origin
9 - Reflection wrt line x=y
10 - Shearing along x-dir
11 - Shearing along y-dir
0 - All done
```

```
Choose required transformation: 4
SCALING WRT ORIGIN
Enter scaling factor values: 3 2

Choose required transformation: 8
REFLECTION WRT ORIGIN

Choose required transformation: 6
REFLECTION WRT X-AXIS

Choose required transformation: 10
SHEARING ALONG X-DIR
Enter shear value: 1.5
Enter yref value: -50

Choose required transformation: 0
ALL DONE

Transformation Matrix:
-3 3 75
0 2 0
0 0 1

P1':
75
0
1

P2':
225
100
1
```

```
Choose required transformation: 0
ALL DONE

Transformation Matrix:
-3 3 75
0 2 0
0 0 1

P1':
75
0
1

P2':
225
100
1

P3':
-75
0
1
```



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.

Source code:

```
#define GL_SILENCE_DEPRECATION
#include<GLUT/glut.h>
#include<stdio.h>
#include<iostream>
#include<math.h>
using namespace std;
float xvmax = 640, yvmax = 480, xwmax = 1280, ywmax = 960;
void myInit_window() {
    glClearColor(1, 1, 1, 1.0);
    glColor3f(0.0f, 0.0f, 0.0f);
    glPointSize(3);
    glLineWidth(3);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0, 1280.0, 0.0, 960.0);
}
void myInit_viewport() {
    glClearColor(1, 1, 1, 1.0);
    glColor3f(0.0f, 0.0f, 0.0f);
    glPointSize(3);
    glLineWidth(3);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0, 640.0, 0.0, 480.0);
}
void displayaxes_window() {
    glBegin(GL_LINES);
    glColor4f(0, 0.5, 0, 1);
    //y - axis
    glVertex2d(640, 0);
    glVertex2d(640, 960);
    //x - axis
    glVertex2d(0, 480);
    glVertex2d(1280, 480);
    glEnd();
}
void displayaxes_viewport() {
    glBegin(GL_LINES);
    glColor4f(0, 0.5, 0, 1);
    //y - axis
    glVertex2d(320, 0);
    glVertex2d(320, 480);
    //x - axis
    glVertex2d(0, 240);
    glVertex2d(640, 240);
    glEnd();
}
void drawObject(int window) {
```

```
    float x1, y1;
    cout << "Enter point 1 coordinates: ";
    cin >> x1 >> y1;

    float x2, y2;
    cout << "Enter point 2 coordinates: ";
    cin >> x2 >> y2;

    float x3, y3;
    cout << "Enter point 3 coordinates: ";
    cin >> x3 >> y3;

    if (window) {
        cout << "window\n";
        glBegin(GL_TRIANGLES);
        glColor4f(0.4, 0, 0.8, 1);
        glVertex2d(x1 + (xwmax / 2), y1 + (ywmax / 2));
        glVertex2d(x2 + (xwmax / 2), y2 + (ywmax / 2));
        glVertex2d(x3 + (xwmax / 2), y3 + (ywmax / 2));
        glEnd();
        glFlush();
    }
    else {
        cout << "viewport\n";
        float sx = xvmax / xwmax, sy = yvmax / ywmax;
        float S[3][3] = { {sx, 0, 0}, {0, sy, 0}, {0, 0, 1} };
        float T[3][3] = { {x1, y1, 1}, {x2, y2, 1}, {x3, y3, 1} };
        float R[3][3] = { {0, 0, 0}, {0, 0, 0}, {0, 0, 0} };

        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                for (int k = 0; k < 3; k++) {
                    R[i][j] += S[i][k] * T[k][j];
                }
            }
        }
        glBegin(GL_TRIANGLES);
        glColor4f(0, 0, 0.8, 1);

        glVertex2d(R[0][0] + (xvmax / 2), R[0][1] + (yvmax / 2));
        glVertex2d(R[1][0] + (xvmax / 2), R[1][1] + (yvmax / 2));
        glVertex2d(R[2][0] + (xvmax / 2), R[2][1] + (yvmax / 2));
        glEnd();
        glFlush();
    }

}
void plotWindow_window() {
    myInit_window();
    glClear(GL_COLOR_BUFFER_BIT);
    displayaxes_window();

    drawObject(1);
```

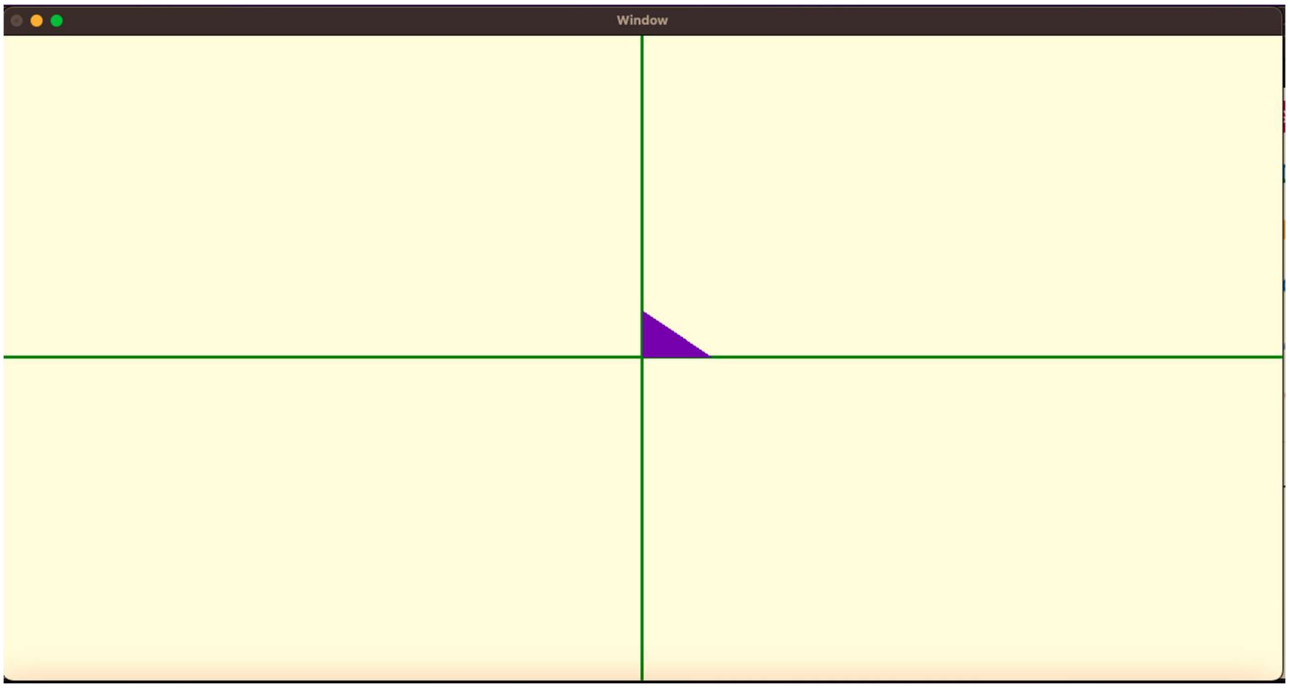
```
    glFlush();
    glutSwapBuffers();
}
void plotWindow_viewport() {
    myInit_viewport();
    glClear(GL_COLOR_BUFFER_BIT);
    displayaxes_viewport();
    drawObject(0);
    glFlush();
    glutSwapBuffers();
}
int main(int argc, char* argv[]) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA);

    glutInitWindowSize(xwmax, ywmax);
    int window = glutCreateWindow("Window");

    glutInitWindowSize(xvmax, yvmax);
    int viewport = glutCreateWindow("Viewport");

    glutSetWindow(window);
    glutDisplayFunc(plotWindow_window);

    glutSetWindow(viewport);
    glutDisplayFunc(plotWindow_viewport);

    glutMainLoop();
    return 1;
}
```

Output

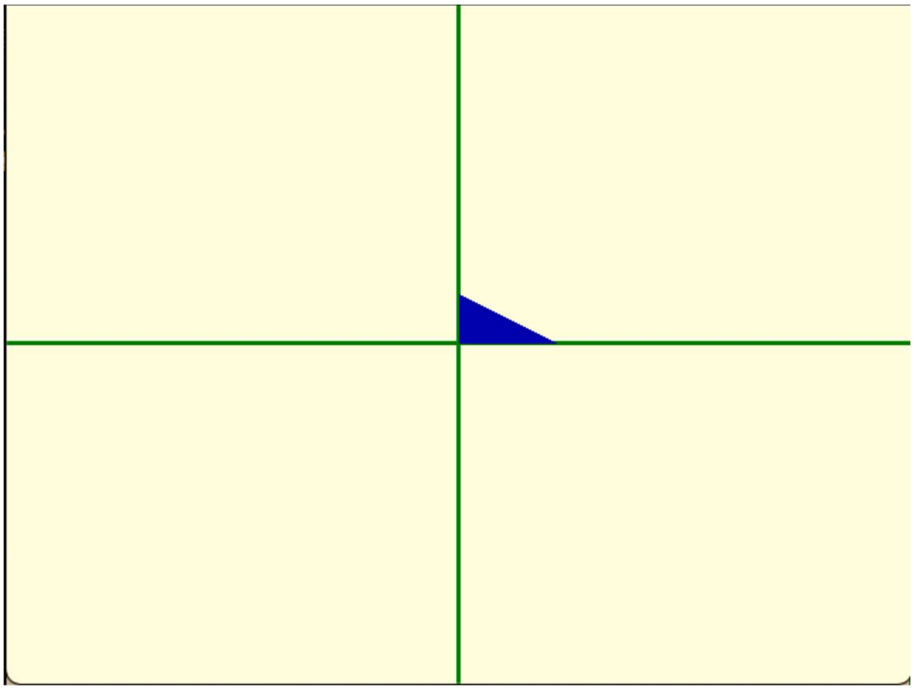
Input

```
Enter point 1 coordinates: 0 0
Enter point 2 coordinates: 0 70
Enter point 3 coordinates: 70 0
window
Enter point 1 coordinates: 0 0
Enter point 2 coordinates: 0 70
Enter point 3 coordinates: 70 0
viewport
```

Window


Window

Viewport



SSN COLLEGE OF ENGINEERING, KALAVAKKAM

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

UCS1712-Graphics and Multimedia Lab

Programming Assignment 7

Cohen Sutherland Line clipping in C++ using OpenGL

Name: Jayannthan P T *Dept: CSE 'A'* *Roll No.: 205001049*

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 an edge, display the line segment with the calculated intermediate intersection points.

Source code:

```
#define GL_SILENCE_DEPRECATION

#include<GL/glut.h>
#include<iostream>
#include<math.h>
using namespace std;

float xmin, xmax, ymin, ymax;
char letter = 'A';

void myInit() {
    glClearColor(0.9, 0.9, 0.9, 0);
    glColor3f(0.0f, 0.0f, 0.0f);
    glPointSize(5);
    glMatrixMode(GL_PROJECTION);
    glLineWidth(1.5);
    glLoadIdentity();
    gluOrtho2D(0.0, 640.0, 0.0, 480.0);
}


void labelPoint(float x, float y, char label) {

    glRasterPos2f(x + 320, y + 240);
    glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, label);
}

void displayPoint(float x, float y) {
    glBegin(GL_POINTS);
    glVertex2d(x + 320, y + 240);
```

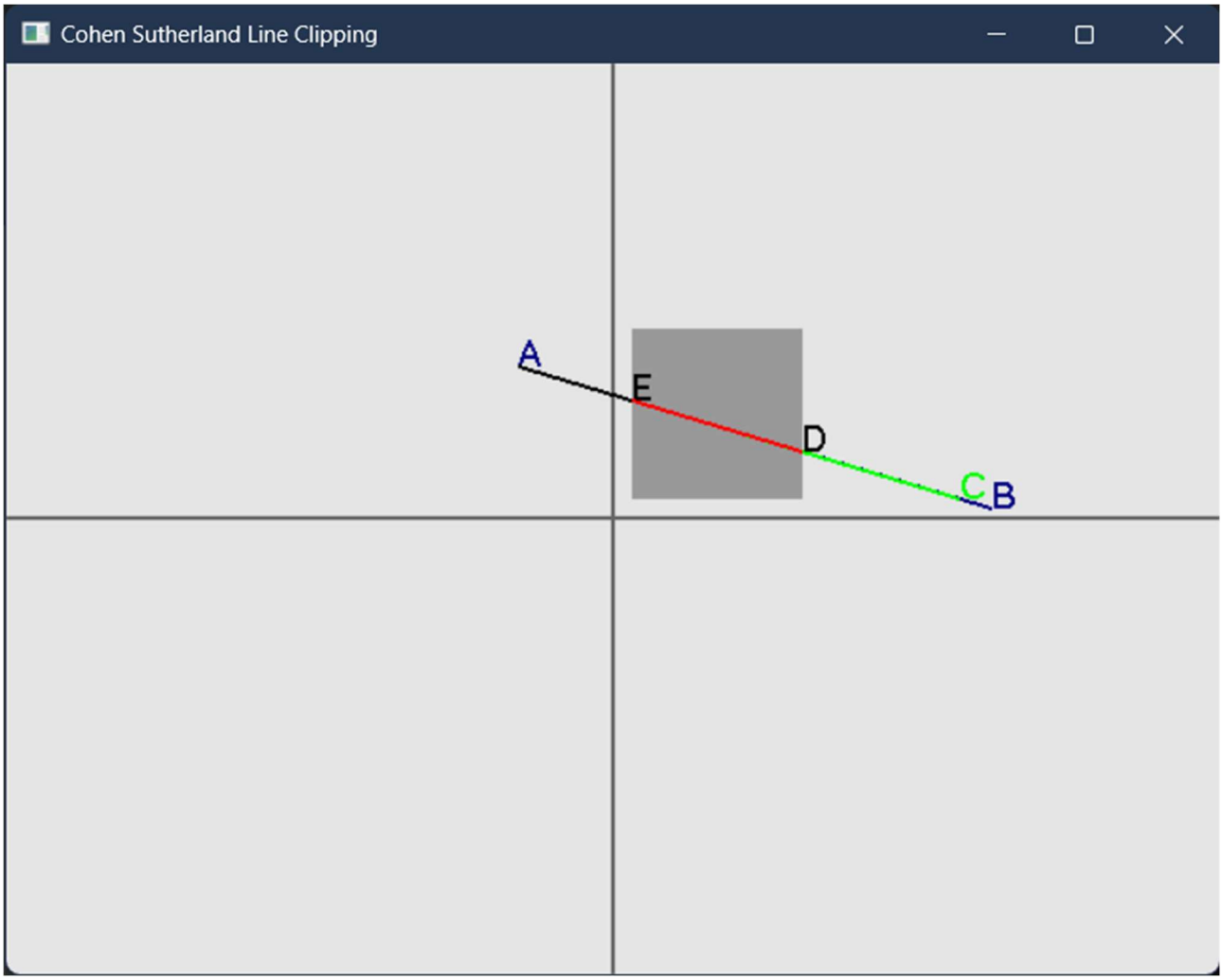
```
    glEnd();
}

void displayLine(int x1, int y1, int x2, int y2) {
    glBegin(GL_LINES);
    glVertex2d(x1 + 320, y1 + 240);
    glVertex2d(x2 + 320, y2 + 240);
    glEnd();
}

void displayQuads(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4) {
    glBegin(GL_QUADS);
    glVertex2d(x1 + 320, y1 + 240);
    glVertex2d(x2 + 320, y2 + 240);
    glVertex2d(x3 + 320, y3 + 240);
    glVertex2d(x4 + 320, y4 + 240);
    glEnd();
}
void drawPlane() {
    glClear(GL_COLOR_BUFFER_BIT);
    glColor4f(0.4, 0.4, 0.4, 1);
    displayLine(-320, 0, 320, 0);
    displayLine(0, -240, 0, 240);
    glFlush();
}
void printMatrix(float* arr, int m, int n)
{
    int i, j;
    for (i = 0; i < m; i++) {
        for (j = 0; j < n; j++)
            cout << *((arr + i * n) + j) << " ";
        cout << endl;
    }
}

void printRegionCode(float* RC) {
    printMatrix(RC, 4, 1);
}

float* getRegionCode(float x, float y) {
    float* TBRL = new float[4];

    TBRL[0] = (y > ymax) ? 1 : 0;
    TBRL[1] = (y < ymin) ? 1 : 0;
    TBRL[2] = (x > xmax) ? 1 : 0;
    TBRL[3] = (x < xmin) ? 1 : 0;

    return TBRL;
}

bool isTrivialAccept(float* OR) {
    for (int i = 0; i < 4; i++) {
        if (OR[i] == 1) {
            cout << "not accept" << endl;
```

```
            return false;
        }
    }
    cout << "accept" << endl;
    return true;
}

bool isTrivialReject(float* AND) {
    for (int i = 0; i < 4; i++) {
        if (AND[i] == 1) {
            cout << "reject" << endl;
            return true;
        }
    }
    cout << "not reject" << endl;
    return false;
}

int firstRegion(float* OR) {
    int i;
    for (i = 0; OR[i] == 0; i++);
    return i;
}
void plotClippingWindow() {
    glColor4f(0.6, 0.6, 0.6, 1);
    displayQuads(xmin, ymin, xmin, ymax, xmax, ymax, xmax, ymin);
}

void plotInputLine(float x1, float y1, float x2, float y2) {
    glColor4f(0.0, 0.0, 0.5, 1);
    displayLine(x1, y1, x2, y2);
}

void applyCSLineClipping(float x1, float y1, float x2, float y2) {
    float* RC1 = getRegionCode(x1, y1);
    cout << "\nRegion code of P1: " << endl;
    printRegionCode(RC1);
    cout << endl;

    float* RC2 = getRegionCode(x2, y2);
    cout << "\nRegion code of P2: " << endl;
    printRegionCode(RC2);
    cout << endl;

    bool algoEnd = false;
    int green = 1;
    do {
        float* OR = new float[4];
        float* AND = new float[4];
        for (int i = 0; i < 4; i++) {
            OR[i] = max(RC1[i], RC2[i]);
            AND[i] = min(RC1[i], RC2[i]);
        }
        cout << "\nOR: " << endl;
```

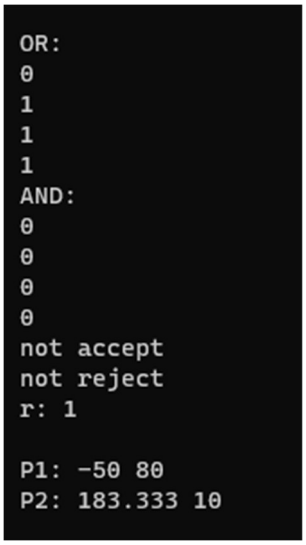
```
        printRegionCode(OR);
        cout << "AND: " << endl;
        printRegionCode(AND);

        if (isTrivialAccept(OR) || isTrivialReject(AND)) {
            cout << "\nend" << endl;
            algoEnd = true;
            glColor4f(1, 0, 0, 1);
            displayLine(x1, y1, x2, y2);
        }
        else {
            int r = firstRegion(OR);
            cout << "r: " << r << endl;
            float x = x1, y = y1;
            float m = (y2 - y1) / (x2 - x1);
            glColor4f(0, green, 0, 1); green -= 0.2;
            if (r == 0) {
                y = ymax;
                x = x1 + (1 / m) * (y - y1);
                RC1 = getRegionCode(x, y);
                x1 = x; y1 = y;
                labelPoint(x1, y1, letter++);
            }
            else if (r == 1) {
                y = ymin;
                x = x1 + (1 / m) * (y - y1);
                RC2 = getRegionCode(x, y);
                x2 = x; y2 = y;
                labelPoint(x2, y2, letter++);
            }
            else if (r == 2) {
                x = xmax;
                y = y1 + m * (x - x1);
                RC2 = getRegionCode(x, y);
                x2 = x; y2 = y;
                labelPoint(x2, y2, letter++);
            }
            else if (r == 3) {
                x = xmin;
                y = y1 + m * (x - x1);
                RC1 = getRegionCode(x, y);
                x1 = x; y1 = y;
                labelPoint(x1, y1, letter++);
            }
            cout << "\nP1: " << x1 << " " << y1 << endl;
            cout << "P2: " << x2 << " " << y2 << endl;
            displayLine(x1, y1, x2, y2);
        }
    } while (!algoEnd);
}

void plotChart() {
    glClear(GL_COLOR_BUFFER_BIT);
```

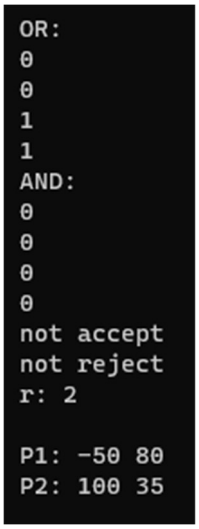
```
    drawPlane();

    cout << "COHEN SUTHERLAND LINE CLIPPING" << endl;


    cout << "\nEnter clipping window edges: " << endl;
    cout << "x_min: "; cin >> xmin;
    cout << "x_max: "; cin >> xmax;
    cout << "y_min: "; cin >> ymin;
    cout << "y_max: "; cin >> ymax;
    plotClippingWindow();


    float x1, y1, x2, y2;
    cout << "\nEnter line endpoints: " << endl;
    cout << "P1: "; cin >> x1 >> y1;
    cout << "P2: "; cin >> x2 >> y2;
    if (x2 < x1) {
        cout << "Swapping points so that P1 lies to the left of P2..." << endl;
        float tx = x1, ty = y1;
        x1 = x2; y1 = y2;
        x2 = tx; y2 = ty;
    }
    plotInputLine(x1, y1, x2, y2);

    labelPoint(x1, y1, letter++);
    labelPoint(x2, y2, letter++);


    applyCSLineClipping(x1, y1, x2, y2);

    glFlush();
}

int main(int argc, char* argv[]) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGBA);
    glutInitWindowSize(640, 480);
    glutCreateWindow("Cohen Sutherland Line Clipping");
    glutDisplayFunc(plotChart);
    myInit();
    glutMainLoop();
    return 1;
}
```

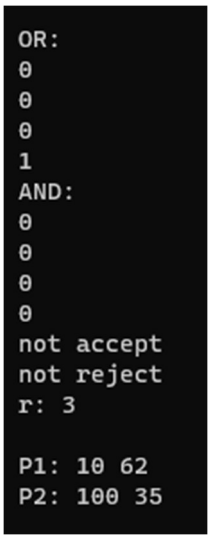
Output

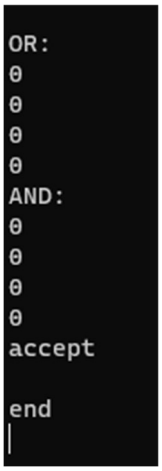



```
COHEN SUTHERLAND LINE CLIPPING

Enter clipping window edges:
x_min: 10
x_max: 100
y_min: 10
y_max: 100

Enter line endpoints:
P1: -50 80
P2: 200 5

Region code of P1:
0
0
0
1


Region code of P2:
0
1
1
0
```


OR:
0
1
1
1
AND:
0
0
0
0
not accept
not reject
r: 1
P1: -50 80
P2: 183.333 10


OR:
0
0
1
1
AND:
0
0
0
0
not accept
not reject
r: 2
P1: -50 80
P2: 100 35


OR:
0
0
0
1
AND:
0
0
0
0
not accept
not reject
r: 3
P1: 10 62
P2: 100 35


OR:
0
0
0
0
AND:
0
0
0
0
accept
end

SSN COLLEGE OF ENGINEERING, KALAVAKKAM

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

UCS1712-Graphics and Multimedia Lab

Programming Assignment 6

3-Dimensional Transformations in C++ using OpenGL

Name: Jayannthan P T *Dept: CSE 'A'* *Roll No.: 205001049*

a) 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.

Source code:

```
#include <stdio.h>
#include <GL/glut.h>
#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);
}

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)
{
    180;
    theMatrix[1][1] = cos(angle);
    theMatrix[1][2] = -sin(angle);
    theMatrix[2][1] = sin(angle);
```

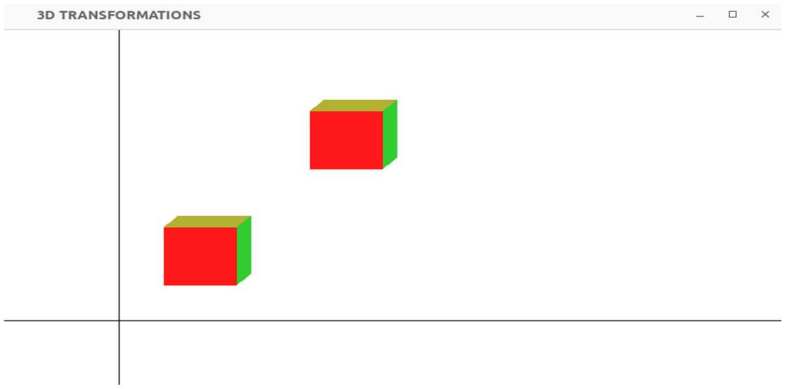
```
    theMatrix[2][2] = cos(angle);
}
void RotateY(float angle)
{
    180;
    theMatrix[0][0] = cos(angle);
    theMatrix[0][2] = -sin(angle);
    theMatrix[2][0] = sin(angle);
    theMatrix[2][2] = cos(angle);
}
void RotateZ(float angle)
{
    180;
    theMatrix[0][0] = cos(angle);
    theMatrix[0][1] = 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];
            }
        }
    }
}

void draw(float a[8][3])
{
    glBegin(GL_QUADS);
    glColor3f(0.7, 0.4, 0.5);
    glVertex3fv(a[0]);
    glVertex3fv(a[1]);
    glVertex3fv(a[2]);
    glVertex3fv(a[3]);
    glColor3f(0.8, 0.2, 0.4);
    glVertex3fv(a[0]);
    glVertex3fv(a[1]);
    glVertex3fv(a[5]);
    glVertex3fv(a[4]);
    glColor3f(0.3, 0.6, 0.7);
    glVertex3fv(a[0]);
    glVertex3fv(a[4]);
    glVertex3fv(a[7]);
    glVertex3fv(a[3]);
    glColor3f(0.2, 0.8, 0.2);
    glVertex3fv(a[1]);
    glVertex3fv(a[2]);
```

```
    glVertex3fv(a[6]);
    glVertex3fv(a[5]);
    glColor3f(0.7, 0.7, 0.2);
    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();
}

void display(void)
{

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    glColor3f(0.0, 0.0, 0.0);

    glBegin(GL_LINES);
    glVertex3d(-1000, 0, 0);
    glVertex3d(1000, 0, 0);
    glEnd();
    glBegin(GL_LINES);
    glVertex3d(0, -1000, 0);
    glVertex3d(0, 1000, 0);
    glEnd();
    glBegin(GL_LINES);
    glVertex3d(0, 0, -1000);
    glVertex3d(0, 0, 1000);
    glEnd();

    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);
```

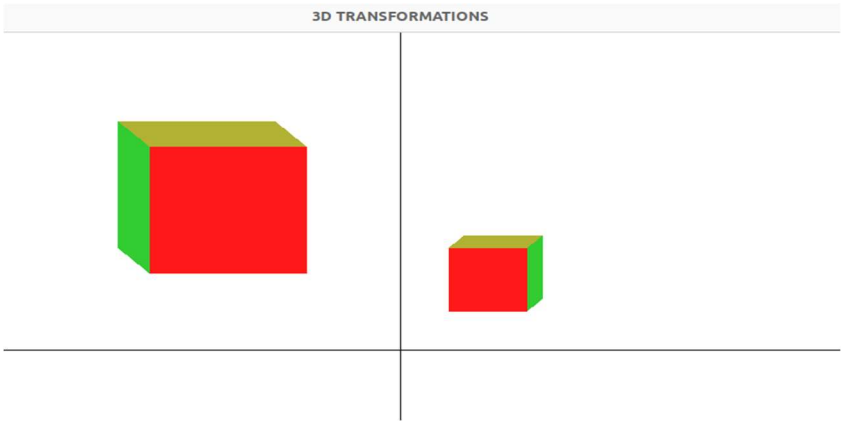
```
            break;
        case 3:
            RotateZ(angle);
            break;
        default:
            break;
        }
        multiplyM();
        break;
    }
    draw(output);
    glFlush();

    glFlush();
}

int main(int argc, char **argv)
{

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(1380, 700);
    glutInitWindowPosition(200, 200);
    glutCreateWindow("3D TRANSFORMATIONS");

    glClearColor(1.0, 1.0, 1.0, 1.0);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    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);
    glutMainLoop();
    return 0;
}
```

Output

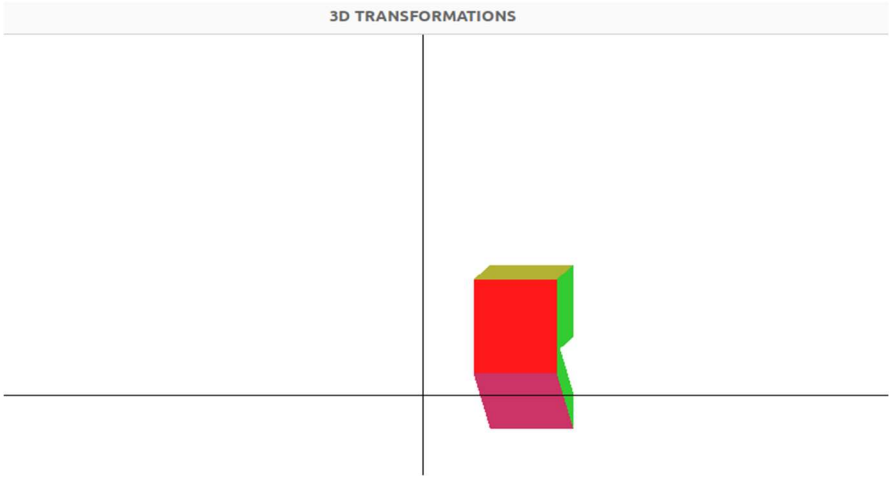
Translate along X and Y and Z



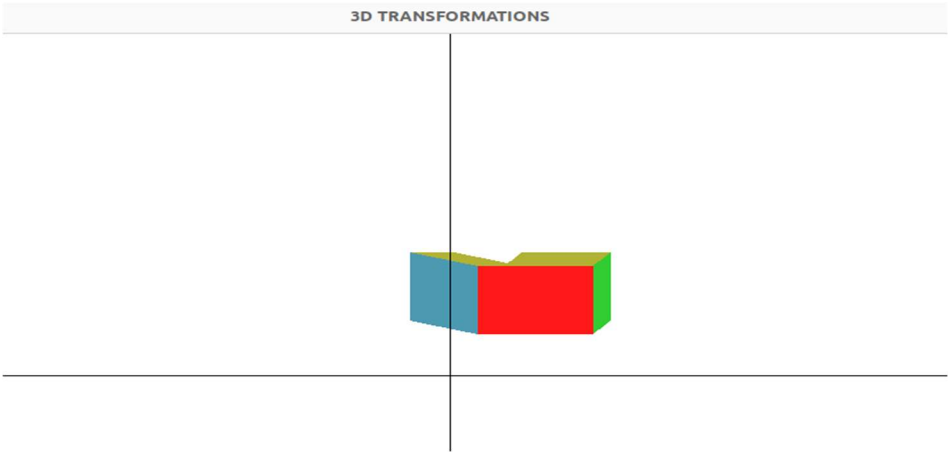
Scaling at X Y and Z



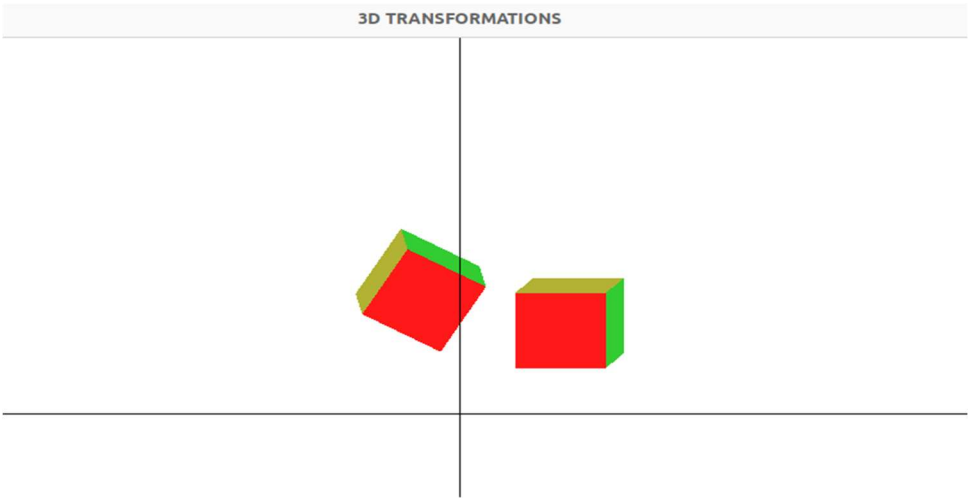
Rotation wrt X



Rotate wrt Y



Rotate wrt Z



SSN COLLEGE OF ENGINEERING, KALAVAKKAM

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

UCS1712-Graphics and Multimedia Lab

Programming Assignment 6

3-Dimensional Projections in C++ using OpenGL

Name: Jayannthan P T *Dept: CSE 'A'* *Roll No.: 205001049*

Write a menu driven program to perform Orthographic 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.

Use gluPerspective() to perform perspective projection. Also, can use inbuilt functions for 3D trasformations.

Source code:

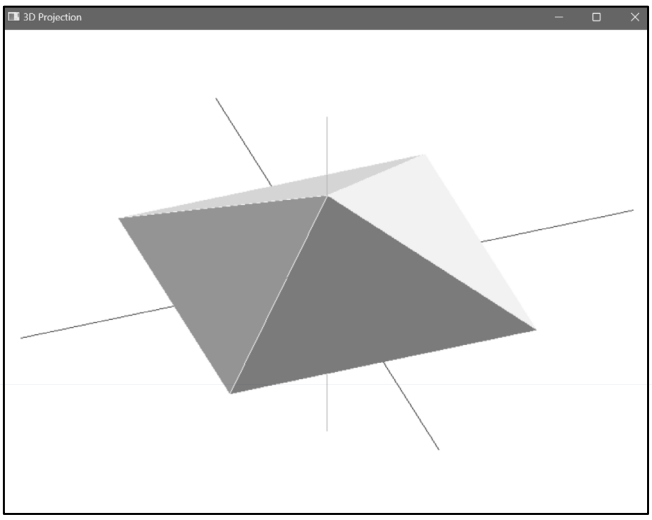
```
#include <GL/glut.h>
int projectionMode = 0;
float rotateX = 0.0f;
float rotateY = 0.0f;
float cameraX = 0.0f;
float cameraY = 0.0f;
float cameraZ = 5.0f;
void draw3DObject()
{
    glBegin(GL_TRIANGLES);
    glColor3f(0.682, 0.871, 0.988);
    glVertex3f(0, 1, 0);
    glVertex3f(-1, 0, -1);
    glVertex3f(1, 0, -1);
    glColor3f(1, 0.984, 0.451);
    glVertex3f(0, 1, 0);
    glVertex3f(1, 0, -1);
    glVertex3f(1, 0, 1);
    glColor3f(0.694, 0.369, 1);
    glVertex3f(0, 1, 0);
    glVertex3f(1, 0, 1);
    glVertex3f(-1, 0, 1);
    glColor3f(0.972, 0.459, 0.667);
    glVertex3f(0, 1, 0);
    glVertex3f(-1, 0, 1);
    glVertex3f(-1, 0, -1);
    glEnd();
```

```
    glColor3f(0.914, 0.722, 0.141);
    glBegin(GL_QUADS);
    glVertex3f(-1, 0, -1);
    glVertex3f(1, 0, -1);
    glVertex3f(-1, 0, 1);
    glVertex3f(-1, 0, -1);
    glEnd();
    glColor3f(1.0, 1.0, 1.0);
    glBegin(GL_LINES);
    glVertex3f(0, 1, 0);
    glVertex3f(-1, 0, -1);
    glVertex3f(0, 1, 0);
    glVertex3f(1, 0, -1);
    glVertex3f(0, 1, 0);
    glVertex3f(1, 0, 1);
    glVertex3f(0, 1, 0);
    glVertex3f(-1, 0, 1);
    glVertex3f(-1, 0, -1);
    glVertex3f(1, 0, -1);
    glVertex3f(1, 0, -1);
    glVertex3f(1, 0, 1);
    glVertex3f(1, 0, 1);
    glVertex3f(-1, 0, 1);
    glVertex3f(-1, 0, -1);
    glVertex3f(-1, 0, 1);
    glEnd();
}
void setPerspectiveProjection()
{
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(45.0, 1.0, 1.0, 100.0);
    glMatrixMode(GL_MODELVIEW);
}
void setOrthographicProjection()
{
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-2.0, 2.0, -2.0, 2.0, 1.0, 100.0);
    glMatrixMode(GL_MODELVIEW);
}
void display()
{
    glClearColor(1.0, 1.0, 1.0, 0.0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    gluLookAt(cameraX, cameraY, cameraZ, 0, 0, 0, 0, 1, 0);
    glRotatef(rotateX, 1.0f, 0.0f, 0.0f);
    glRotatef(rotateY, 0.0f, 1.0f, 0.0f);
    glBegin(GL_LINES);
    glColor3f(1.0f, 0.0f, 0.0f);
    glVertex3f(-2, 0, 0);
    glVertex3f(2, 0, 0);
    glColor3f(0.0f, 1.0f, 0.0f);
```

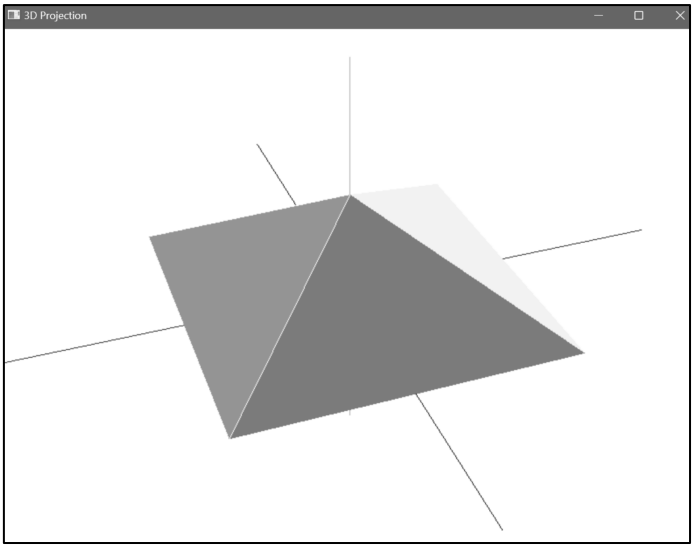
```
    glVertex3f(0, -2, 0);
    glVertex3f(0, 2, 0);
    glColor3f(0.0f, 0.0f, 1.0f);
    glVertex3f(0, 0, -2);
    glVertex3f(0, 0, 2);
    glEnd();
    glColor3f(1.0f, 1.0f, 1.0f);
    draw3DObject();
    glutSwapBuffers();
}
void keyboard(unsigned char key, int x, int y)
{
    switch (key)
    {
    case 'o':
        projectionMode = 0;
        setOrthographicProjection();
        break;
    case 'p':
        projectionMode = 1;
        setPerspectiveProjection();
        break;
    case 'r':
        rotateX += 10.0f;
        break;
    case 'l':
        rotateY += 10.0f;
        break;
    case 'f':
        cameraZ -= 0.25f;
        break;
    case 'b':
        cameraZ += 0.25f;
        break;
    case 27:
        exit(0);
    }
    glutPostRedisplay();
}
int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(800, 600);
    glutCreateWindow("3D Projection");
    glEnable(GL_DEPTH_TEST);
    glutDisplayFunc(display);
    glutKeyboardFunc(keyboard);
    setOrthographicProjection();
    glutMainLoop();
    return 0;
}
```

Output

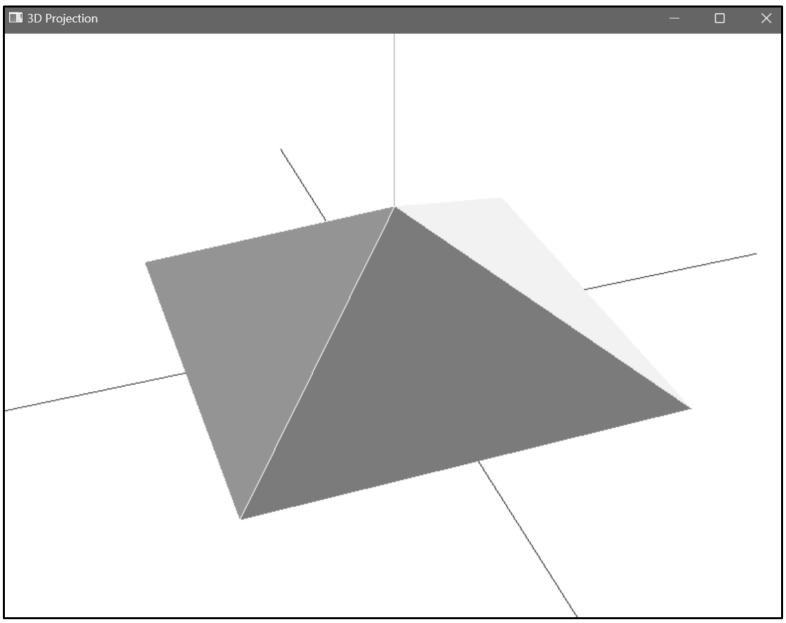
1 Orthographic Projection



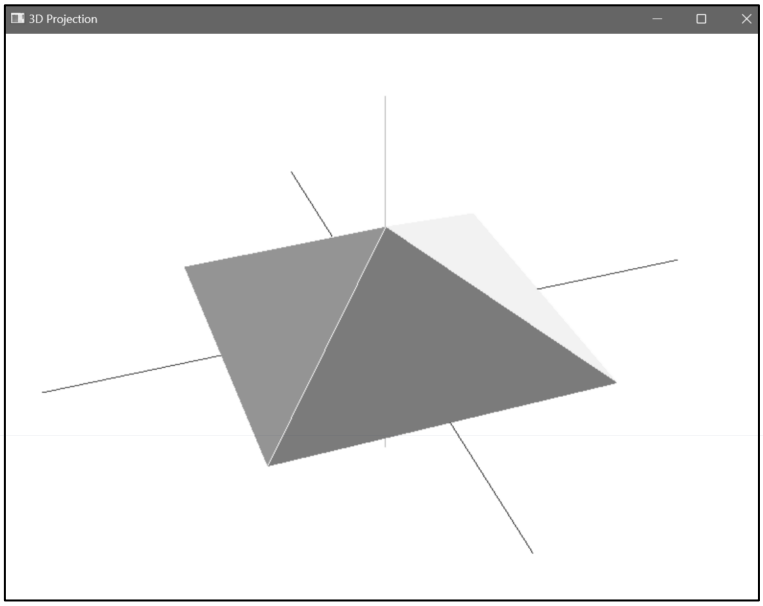
2 Perspective Projection



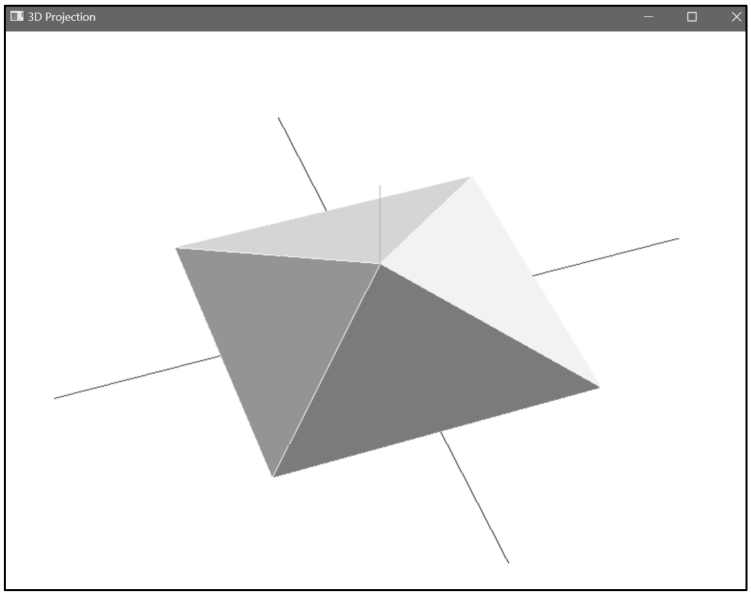
3 Moving Camera Forward



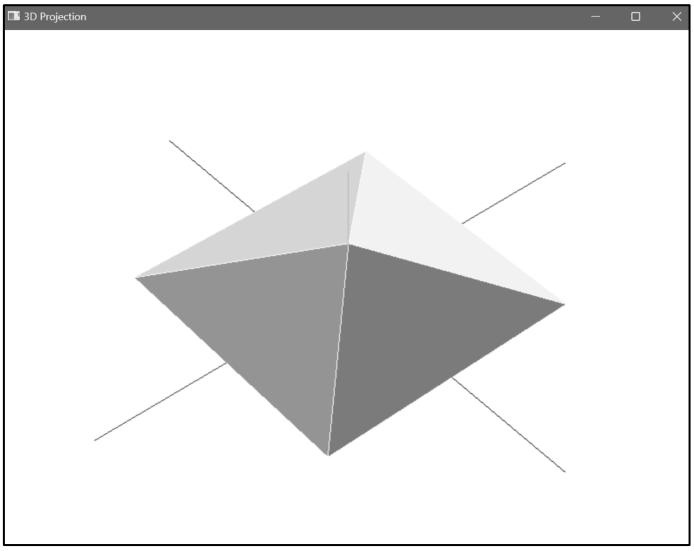
4 Moving Camera Backward



5 Rotate Right



6 Rotate Left



SSN COLLEGE OF ENGINEERING, KALAVAKKAM

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

UCS1712-Graphics and Multimedia Lab

Programming Assignment 10

Creating a 3D Scene in C++ using OpenGL

Name: Jayannthan P T *Dept: CSE 'A'* *Roll No.: 205001049*

Write a C++ program using Opengl to draw atleast 2 3D objects. Apply lighting and texture and render the scene.

OpenGL Functions to use:

glShadeModel()

glMaterialfv()

glLightfv()

glEnable()

glGenTextures()

glTexEnvf()

glBindTexture()

glTexParameteri()

glTexCoord2f()

Source code:

```
#include <GL/glut.h>
#include <Windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
GLfloat black[] = {0.0, 0.0, 0.0, 1.0};
GLfloat white[] = {1.0, 1.0, 1.0, 1.0};
GLfloat direction[] = {1.0, 1.0, 1.0, 0.0};
float teapot_rotate = 0.2, teapot_rotate_direction = 1, teapot_posx = -0.5, teapot_posy =
1.0, teapot_xplace = 0, teapot_yplace = 0;
float teaspoon_posx = 0.75, teaspoon_posy = 2.5, teaspoon_yplace = 0;
float sugar1_posx = 0.65, sugar1_posy = 2.5, sugar2_posx = 0.8, sugar2_posy = 2.75,
sugar1_yplace = 0, sugar2_yplace = 0;
float teacup_rotate = 0;
void display()
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glMatrixMode(GL_MODELVIEW);
    glPushMatrix();
    glEnable(GL_TEXTURE_2D);
    glDisable(GL_TEXTURE_2D);
    glPushMatrix();
    GLfloat teacup_color[] = {0.482, 1, 0.161, 0.0};
    GLfloat teacup_mat_shininess[] = {100};
    glMaterialfv(GL_FRONT, GL_DIFFUSE, teacup_color);
    glMaterialfv(GL_FRONT, GL_SHININESS, teacup_mat_shininess);
    glTranslatef(0.75, -0.25, 0.0);
    glRotatef(teacup_rotate, 0, 1, 0);
    glutSolidTeacup(1.0);
    glPopMatrix();
    glPushMatrix();
    GLfloat teapot_color[] = {0.486, 0.212, 0.871, 0.0};
    GLfloat teapot_mat_shininess[] = {100};
    glMaterialfv(GL_FRONT, GL_DIFFUSE, teapot_color);
    glMaterialfv(GL_FRONT, GL_SHININESS, teapot_mat_shininess);
    glTranslatef(teapot_posx, teapot_posy, 0.0);
    glRotatef(teapot_rotate, 0, 0, 1);
    glutSolidTeapot(0.75);
    glPopMatrix();
    GLfloat sugar_color[] = {1, 1, 1, 0.0};
    GLfloat sugar_mat_shininess[] = {50};
    glPushMatrix();
    glMaterialfv(GL_FRONT, GL_DIFFUSE, sugar_color);
    glMaterialfv(GL_FRONT, GL_SHININESS, sugar_mat_shininess);
    glTranslatef(sugar1_posx, sugar1_posy, 0.0);
    glRotatef(-45.0, 0, 0, 1);
    glutSolidCube(0.1);
    glPopMatrix();
    glPushMatrix();
    glMaterialfv(GL_FRONT, GL_DIFFUSE, sugar_color);
    glMaterialfv(GL_FRONT, GL_SHININESS, sugar_mat_shininess);
    glTranslatef(sugar2_posx, sugar2_posy, 0.0);
```

```
    glRotatef(45.0, 0, 0, 1);
    glutSolidCube(0.1);
    glPopMatrix();
    glPushMatrix();
    GLfloat teaspoon_color[] = {0.2, 0.2, 0.2, 0.0};
    GLfloat teaspoon_mat_shininess[] = {100};
    glMaterialfv(GL_FRONT, GL_DIFFUSE, teaspoon_color);
    glMaterialfv(GL_FRONT, GL_SHININESS, teaspoon_mat_shininess);
    glTranslatef(teaspoon_posx, teaspoon_posy, 0.0);
    glRotatef(135, 0, 1, 0);
    glRotatef(-60, 1, 0, 0);
    glutSolidTeaspoon(1.25);
    glPopMatrix();
    if (teapot_rotate_direction == 1 && teapot_rotate > -45.0)
        teapot_rotate -= 0.5;
    if (teapot_rotate_direction == 1 && teapot_rotate <= -45.0)
        teapot_rotate_direction = -1;
    if (teapot_rotate_direction == -1 && teapot_rotate < 0)
        teapot_rotate += 0.5;
    if (teapot_rotate_direction == -1 && teapot_rotate >= 0)
        teapot_rotate_direction = 0;
    teacup_rotate -= 0.2;
    if (teapot_rotate_direction == 0)
    {
        if (teapot_posx > -1.25 && teapot_xplace == 0)
            teapot_posx -= 0.05;
        if (teapot_posx <= -1.25)
            teapot_xplace = 1;
        if (teapot_posy > 0 && teapot_yplace == 0)
            teapot_posy -= 0.05;
        if (teapot_posy <= -1)
            teapot_yplace = 1;
    }
    if (teapot_rotate_direction == 0)
    {
        if (sugar1_posy > -0.5 && sugar1_yplace == 0)
            sugar1_posy -= 0.05;
        if (sugar1_posy <= -0.5)
            sugar1_yplace = 1;
        if (sugar2_posy > -0.5 && sugar2_yplace == 0)
            sugar2_posy -= 0.05;
        if (sugar2_posy <= -0.5)
            sugar2_yplace = 1;
    }
    if (sugar1_yplace == 1 && sugar2_yplace == 1)
    {
        if (teaspoon_posy > -0.25 && teaspoon_yplace == 0)
            teaspoon_posy -= 0.05;
        if (teaspoon_posy <= -0.5)
            teaspoon_yplace = 1;
    }
    glutSwapBuffers();
}
void reshape(GLint w, GLint h)
```

```
{
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    GLfloat aspect = GLfloat(w) / GLfloat(h);
    glLoadIdentity();
    glOrtho(-2.5, 2.5, -2.5 / aspect, 2.5 / aspect, -10.0, 10.0);
}
void init()
{
    glClearColor(1, 1, 1, 1);
    glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, white);
    glMaterialfv(GL_FRONT, GL_SPECULAR, white);
    glMaterialf(GL_FRONT, GL_SHININESS, 30);
    glLightfv(GL_LIGHT0, GL_AMBIENT, black);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, white);
    glLightfv(GL_LIGHT0, GL_SPECULAR, white);
    glLightfv(GL_LIGHT0, GL_POSITION, direction);
    glEnable(GL_LIGHTING);
    light glEnable(GL_LIGHT0);
    glEnable(GL_DEPTH_TEST);
    depth glShadeModel(GL_FLAT);
    glEnable(GL_TEXTURE_2D);
    glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
    glTexImage2D(GL_TEXTURE_2D, 0, 3, 2, 2, 0, GL_RGB, GL_UNSIGNED_BYTE, texture);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
    glRotatef(20.0, 1.0, 0.0, 0.0);
}
void sceneDemo(int v)
{
    glutPostRedisplay();
    glutTimerFunc(1000 / 24, sceneDemo, 0);
}
int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowPosition(80, 80);
    glutInitWindowSize(800, 600);
    glutCreateWindow("Exercise 10");
    glutReshapeFunc(reshape);
    glutDisplayFunc(display);
    glutTimerFunc(1000, sceneDemo, 0);
    init();
    glutMainLoop();
}
```

Output:

Frames:

SSN COLLEGE OF ENGINEERING, KALAVAKKAM

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

UCS1712-Graphics and Multimedia Lab

Programming Assignment 11

Image Editing and Manipulation

Name: Jayannthan P T *Dept: CSE 'A'* *Roll No.: 205001049*

AIM:

a) Using GIMP, include an image and apply filters, noise and masks.

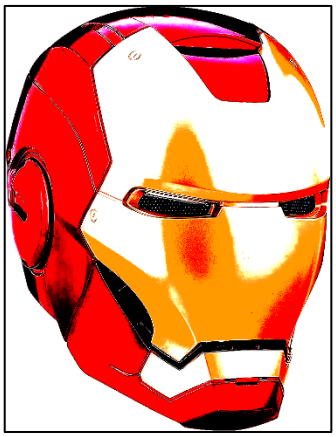
b) Using GIMP, create a GIF animated image.

Output

INPUT IMAGE:

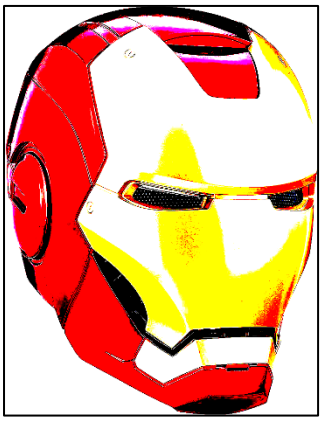
GAUSSIAN BLUR:

Lighting

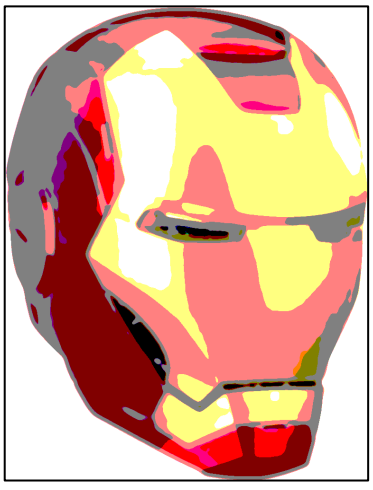


NEON EDGE DETECTION:

ENHANCED CONTRAST:



Cartoon



b) **ANIMATION - FRAMES**

SSN COLLEGE OF ENGINEERING, KALAVAKKAM

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

UCS1712-Graphics and Multimedia Lab

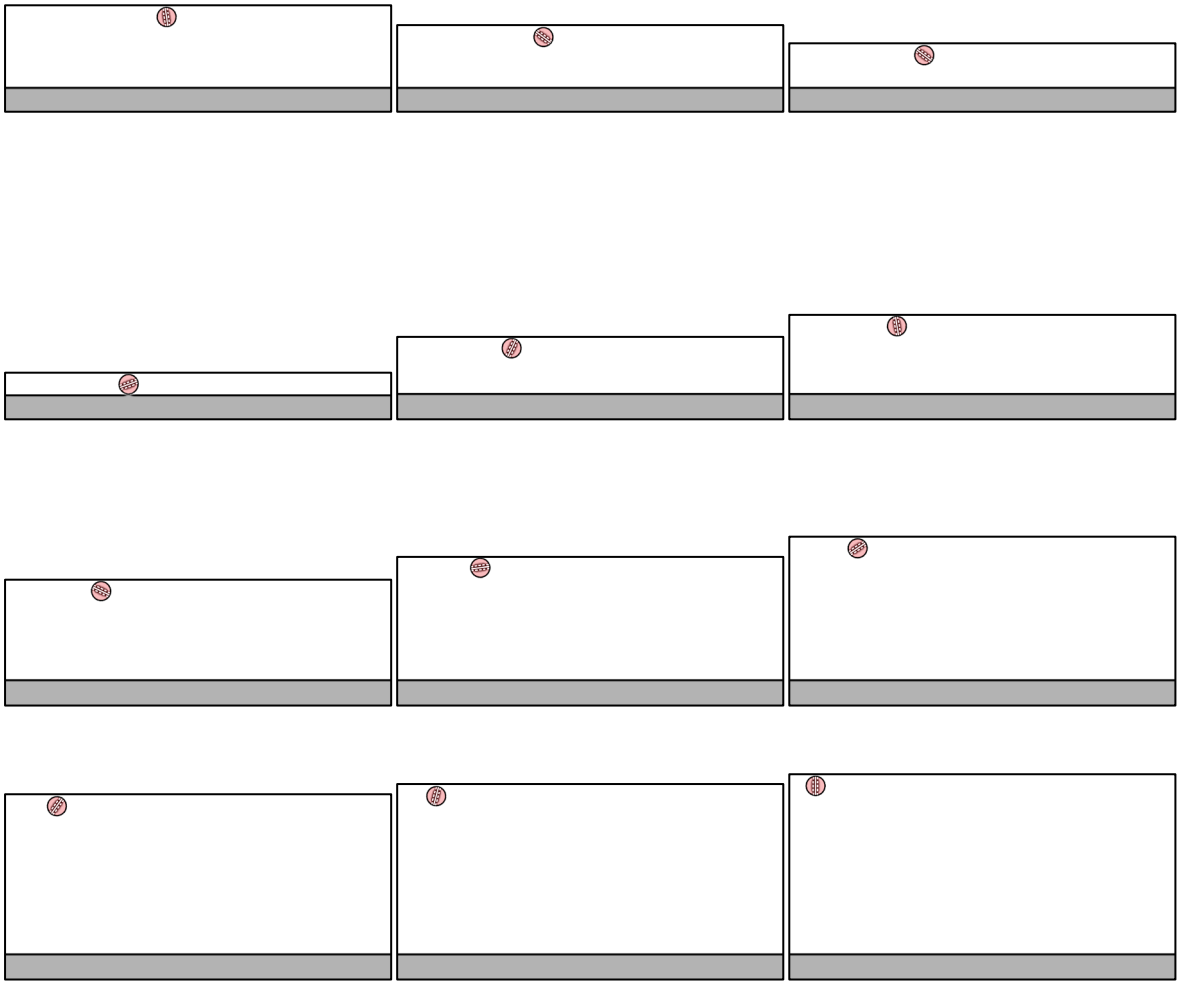
Programming Assignment 12

Creating 2D animation

Name: Jayannthan P T *Dept: CSE 'A'* *Roll No.: 205001049*

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

Ball Bouncing



UCS1712
Graphics and Multimedia Lab

Mini Project

Group Members

P T Jayannthan – 205001049

Koushik Viswanath S - 205001055

Graphics and Multimedia LAB

Mini Project Report

The Batman

Aim:

To render a 2D model and animation of a Batman Sequence to explore various graphics and multimedia tools of Blender.

Tools:

Blender - version 3.6

Implementation:

1. **Creation of 2D Batman Model:**
   A detailed 2D representation of Batman was crafted within Blender by tracing the character's outline, ensuring a faithful portrayal.

2. **Filling the Silhouette:**
   After establishing the outline, the interior of the Batman silhouette was completely filled with a black color.

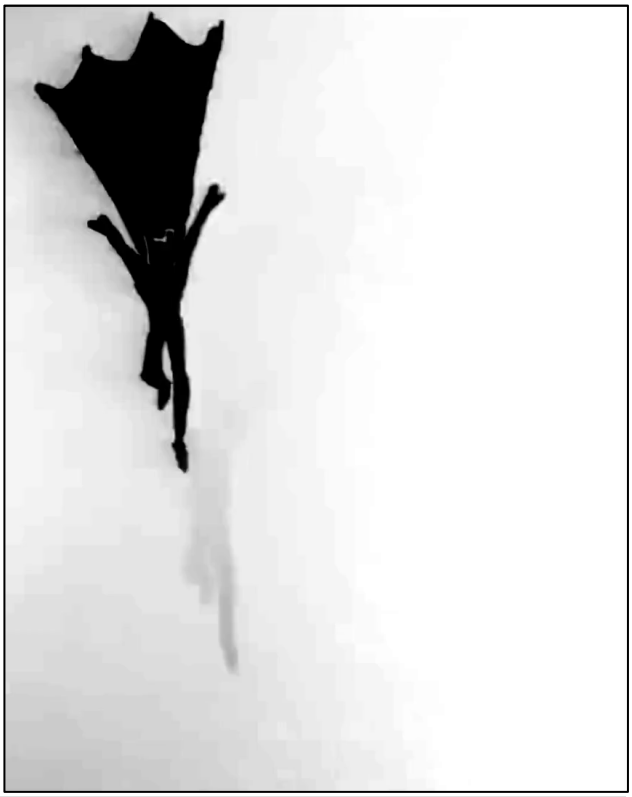
3. **Isolating and Emphasizing the Eyes:**
   A distinctive touch was introduced by isolating the eyes and preserving them from the black fill. This particular emphasis on the eyes was carefully maintained at specific keyframes, enhancing the character's expressiveness and preserving Batman's recognizable look.

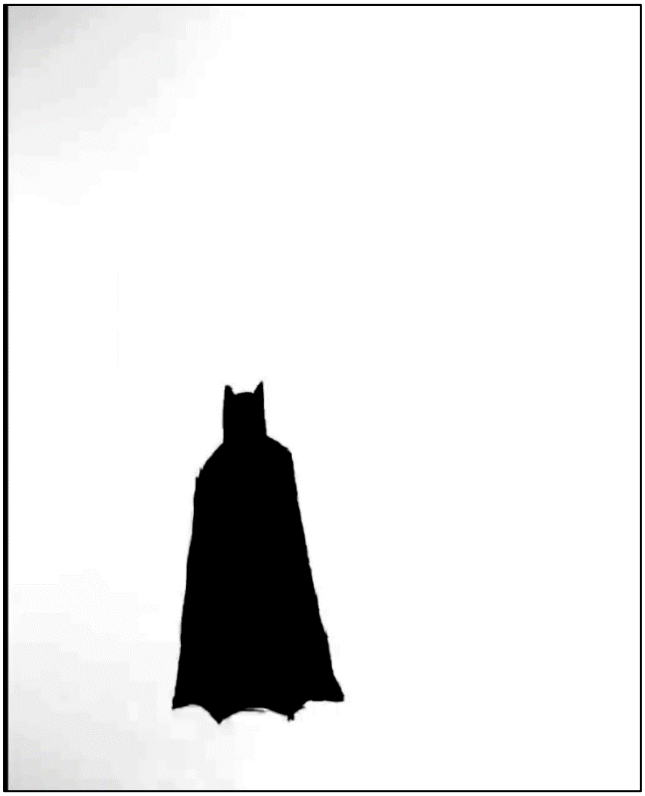
4. **Keyframe Animation:**
   Dynamic movement was infused into the animation through the strategic placement of keyframes. Altering the shape of the Batman model at different keyframes resulted in a fluid animation sequence. This technique facilitated seamless transitions between various poses, breathing life into the character.

5. **Transformations in Keyframes:**
   Each keyframe represented a pivotal moment in the animation timeline. Transformations applied to the Batman model in these keyframes encompassed changes in position, scale, and shape.

6. **Integration of Audio:**
   An additional layer of richness was introduced to the animation sequence by incorporating audio elements. The synchronization of sound with the visual elements enhanced the overall immersive experience, contributing to a more captivating portrayal of the Batman sequence.

Screenshots of the Demo:





Result:

The 2D animation depicting the Batman sequence has been successfully created and rendered.