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] = {
```

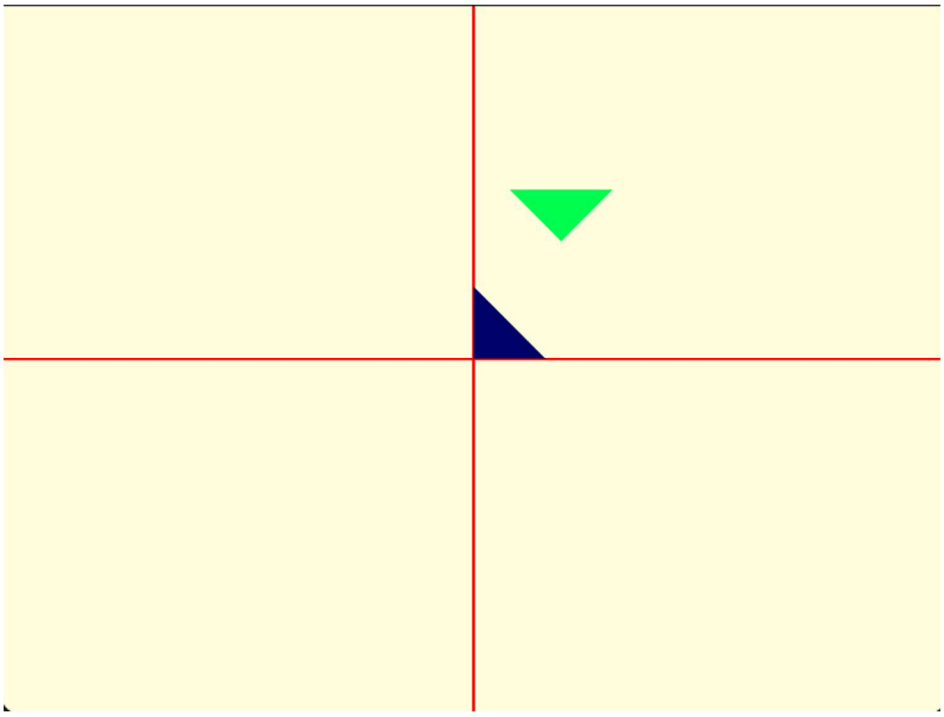
```
                {1, shx,-shx * yref},
                {0, 1, 0},
                {0, 0, 1}
            };
            float* temp = mulTransforms((float*)SH, compositeMatrix);
            delete[] compositeMatrix;
            compositeMatrix = temp;
            break;
        }
        case 11: {
            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
```

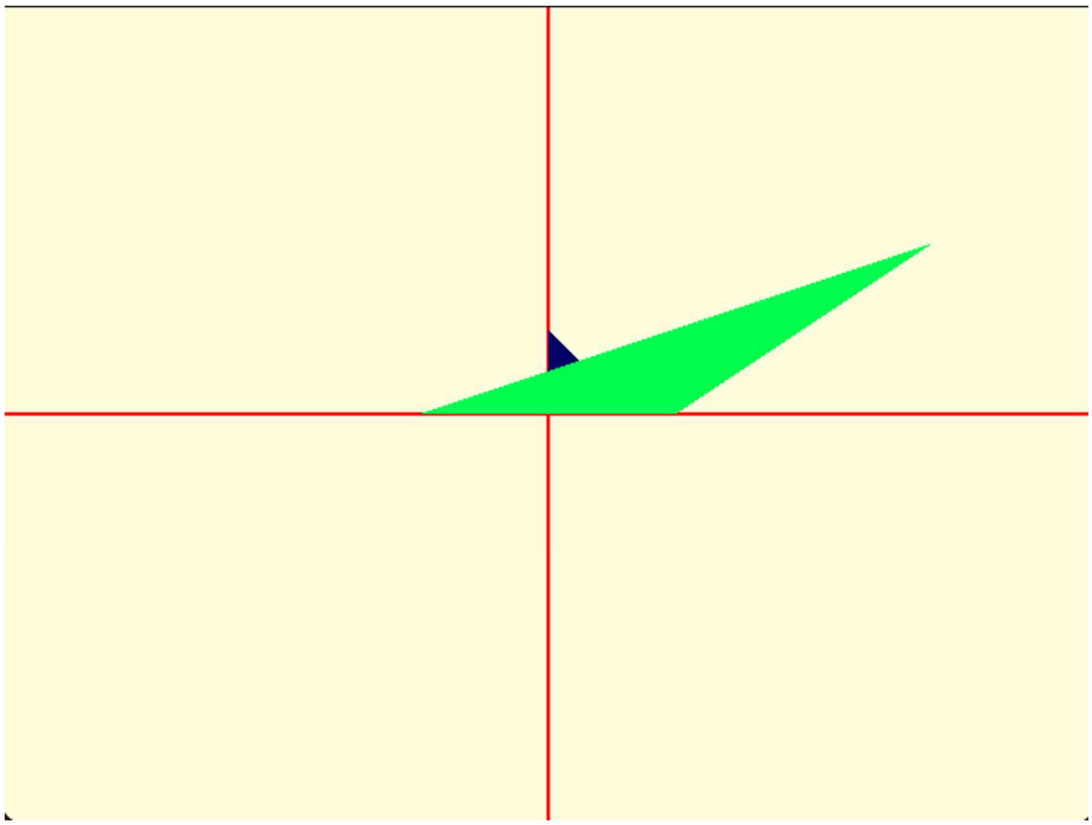
```
Choose required transformation: 1
TRANSLATION
Enter translation values: 60 80

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);
```

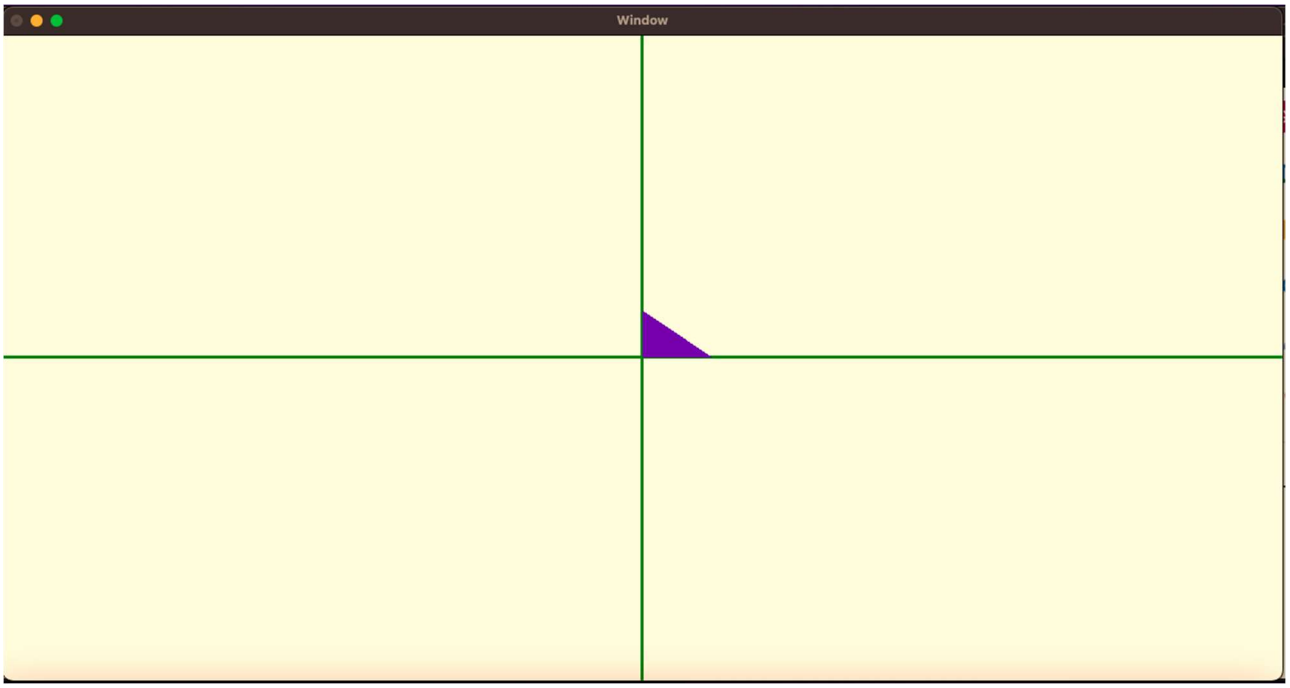
```
    glFlush();
    glutSwapBuffers();
}
void plotWindow_viewport() {
    myInit_viewport();
    glClear(GL_COLOR_BUFFER_BIT);
    displayaxes_viewport();
    drawObject(0);
    glFlush();
    glutSwapBuffers();
}
int main(int argc, char* argv[]) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE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