Shivcharan T
205001100
CSE - B

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

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

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

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

Algorithm:

6a.cpp:

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

6b.cpp:

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

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

c. Similarly, for the y-coordinates.

4. Draw the object on the viewport.

Code:

6a.cpp:

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

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

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

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

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

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

```
}
```

6b.cpp:

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

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

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

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

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

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

// window to viewport transformation
```

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

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

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

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

    *x = x_v;
    *y = y_v;
}

void display1(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

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

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

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

    glFlush();
     glutSwapBuffers();
}

void display2(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

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

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

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

    // Red
```

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

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

    glFlush();
     glutSwapBuffers();
}

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

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

    myInit();
    glutKeyboardFunc(mykey);

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

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

```
    glutMainLoop();
}
```

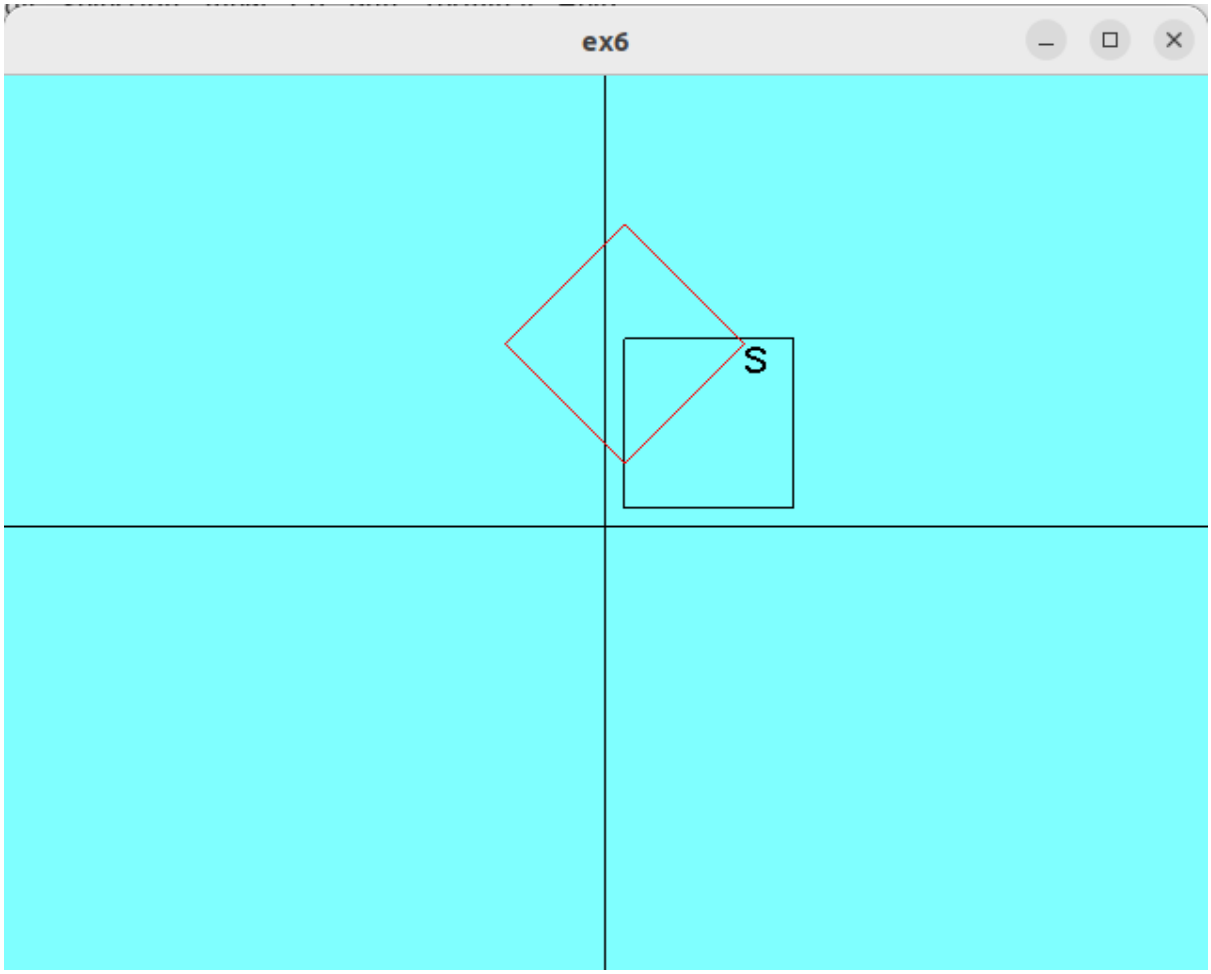
run.sh:

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

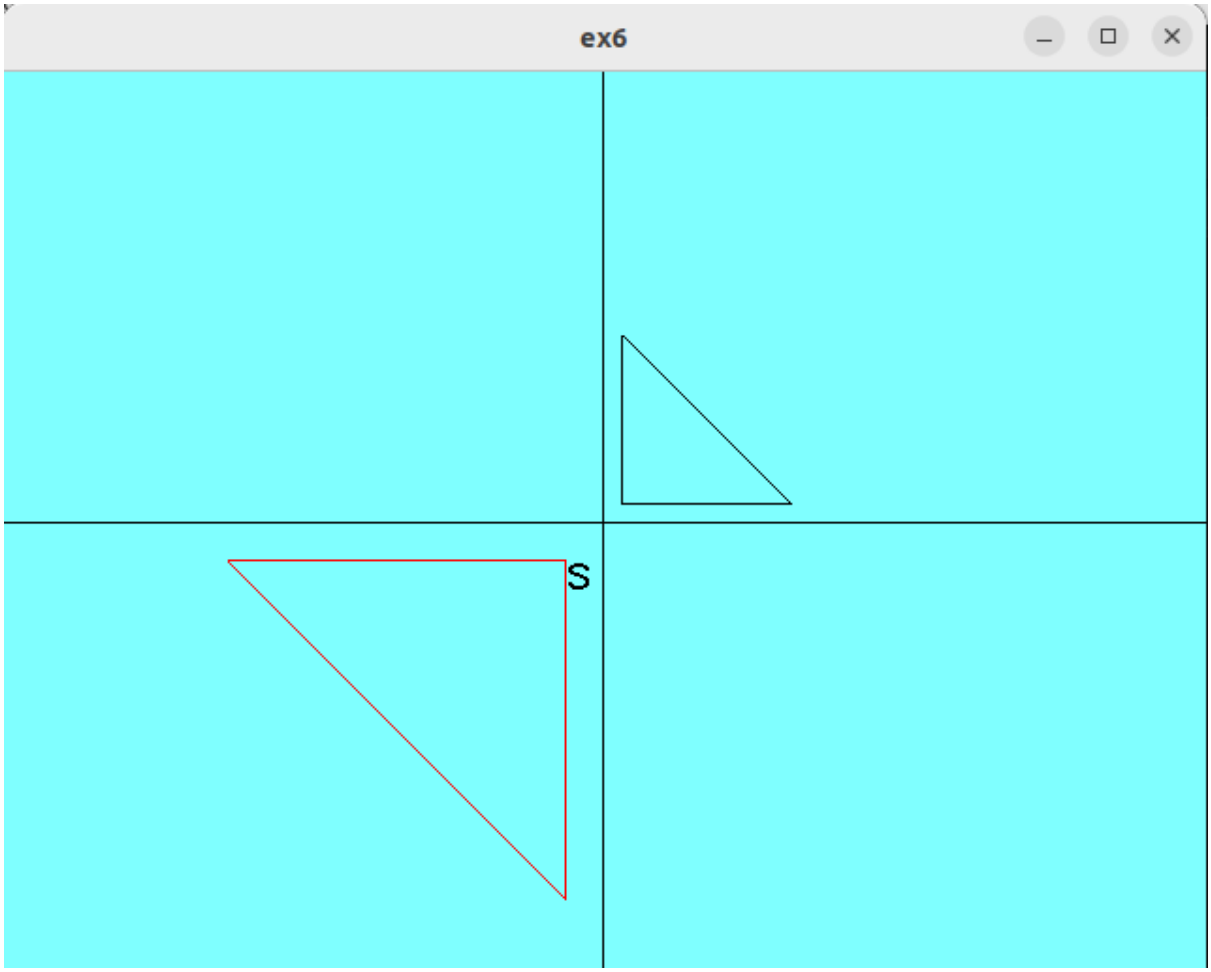
Sample I/O:

6a.cpp:

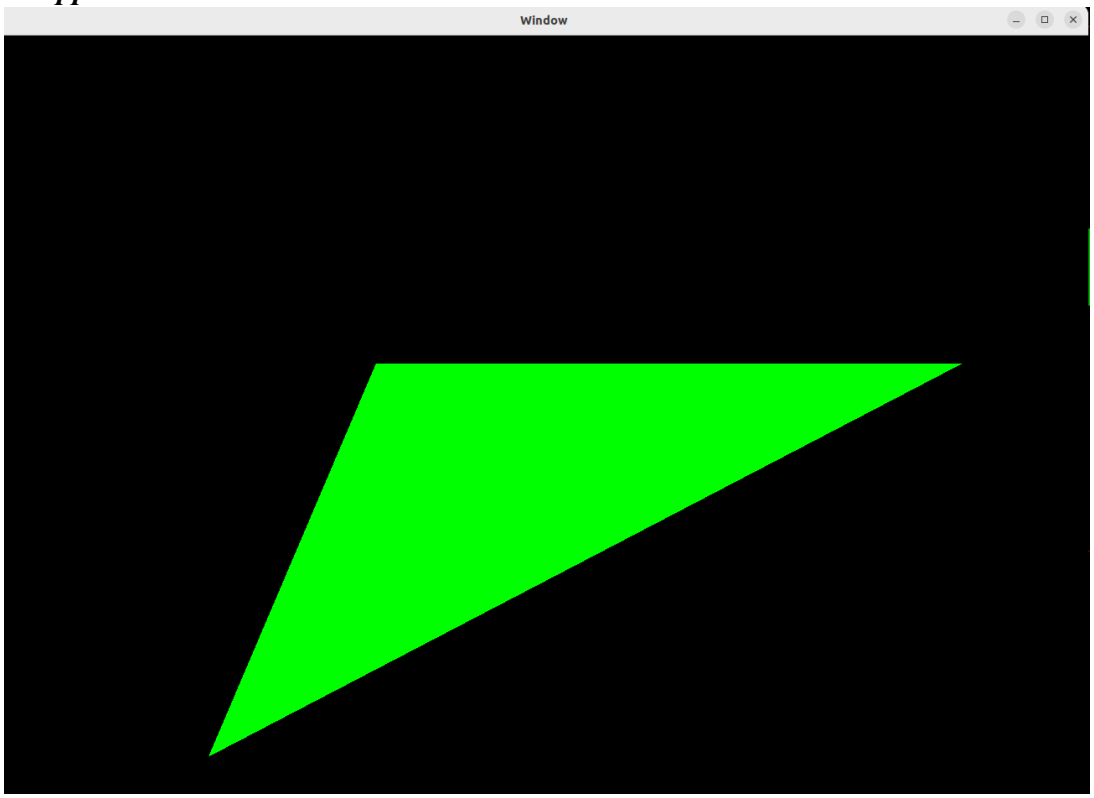
```
Enter any 2 tranformations:-
1.translation
2.rotation
3.scaling
4.reflection
5.shearing(1 / 2 / 3 / 4 / 5)
inc order(op1, op2) : 2 1
Enter deg: 45
0.785398 : deg
Enter tx, ty: 10 20
0.707107 -0.707107 0
0.707107 0.707107 0
0 0 1
1 0 10
0 1 20
0 0 1
Enter no. of points for polygon: 4
Enter x, y coords: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

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

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

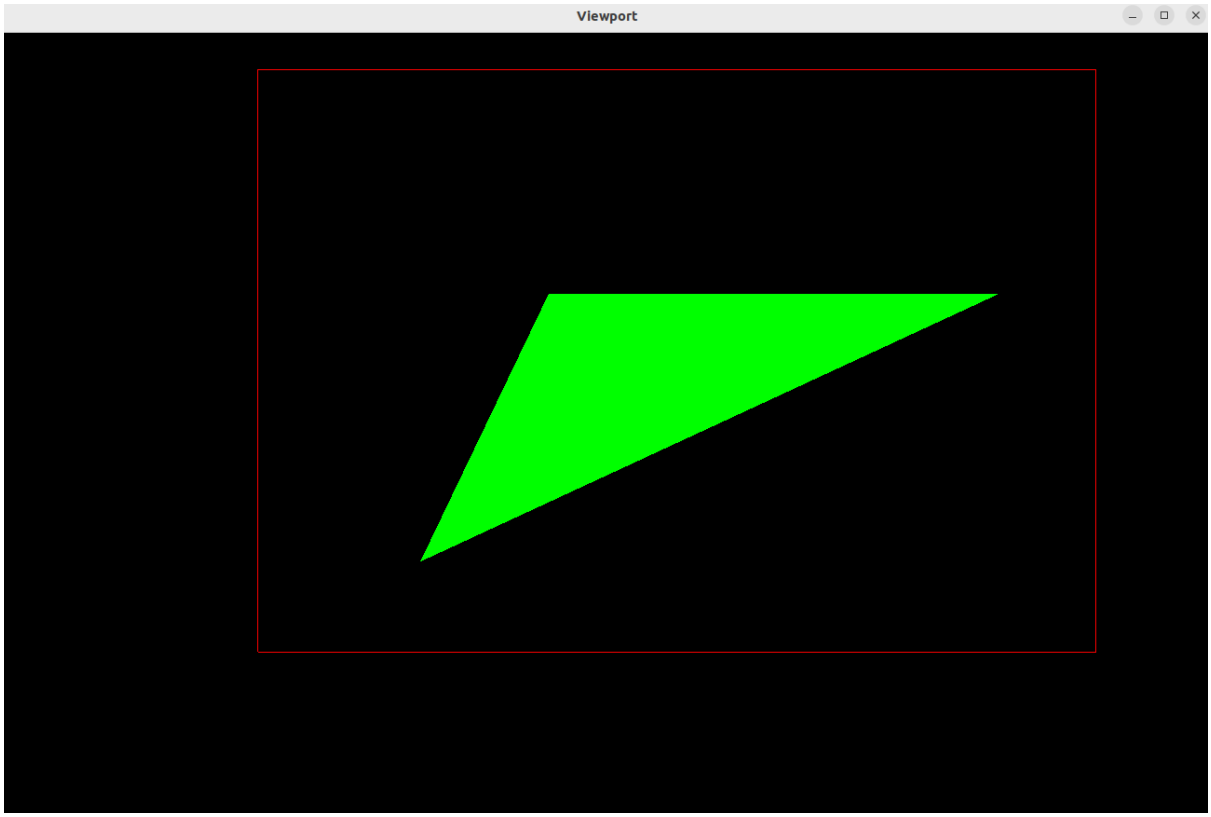

ex6
S

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

-2 0 0
0 -2 0
0 0 1

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



6b.cpp:





Learning Outcomes:

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