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

To apply the following 2D transformations on objects and to render the final output along with the original object.

1) Translation

2) Rotation

    a) About origin

    b) With respect to a fixed point (xr,yr)

3) Scaling with respect to

    a) Origin - Uniform Vs Differential Scaling

    b) Fixed point (xf,yf)

4) Reflection with respect to

    a) X-axis

    b) Y-axis

    c) Origin

    d) The line x=y

5) Shearing

    a) x-direction shear

    b) y-direction shear

Note: Use Homogeneous coordinate representations and matrix multiplication to perform transformations. Divide the output window into four quadrants. (Use LINES primitive to draw x and y axis

CODE:

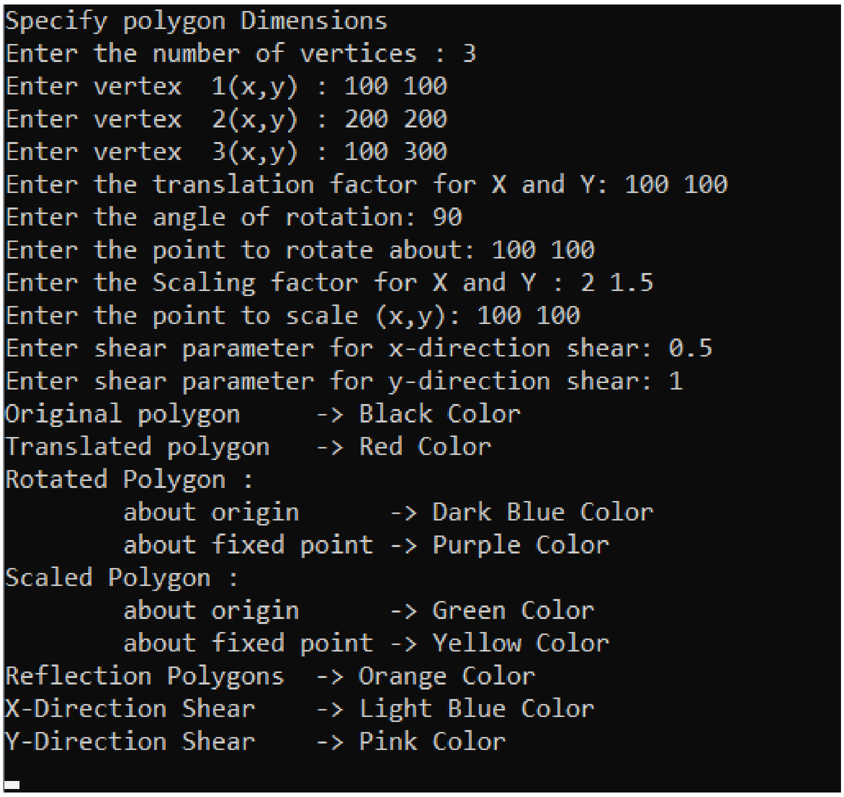
```cpp
#include<bits/stdc++.h>
#include <GL/glut.h>
constexpr auto PI = 3.14;
using namespace std;

vector<pair<int, int>> coords;

int n;
int tx, ty;
int xr, yr;
int xf, yf;
double sx, sy;
double ang, angRad;
double shx, shy;
```

```
void myInit(void)
{
        glClearColor(1.0, 1.0, 1.0, 1.0);
        glColor3f(0.0f, 0.0f, 0.0f);
        glPointSize(4.0);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(-500, 500, -500, 500);
}

void DrawCartesianPlane() {
        glBegin(GL_LINES);
        glColor3f(0.0, 0.0, 0.0);
        glVertex2d(-500, 0);
        glVertex2d(500, 0);
        glVertex2d(0, -500);
        glVertex2d(0, 500);
        glEnd();
}

void drawPolygon()
{
        glBegin(GL_LINE_LOOP);
        glColor3f(0.0, 0.0, 0.0);
        for (int i = 0; i < n; i++)
        {
                glVertex2d(coords[i].first, coords[i].second);
        }
        glEnd();
}


void drawPolygonTrans()
{
        glBegin(GL_LINE_LOOP);
        glColor3f(1.0, 0.0, 0.0);
        for (int i = 0; i < n; i++)
        {
                glVertex2d(coords[i].first+tx, coords[i].second+ty);
        }
        glEnd();
}

void rotatePolygonOrigin() {
        glBegin(GL_LINE_LOOP);
        glColor3f(0.0, 0.0, 1.0);
        for (int i = 0; i < n; i++)
        {
                glVertex2d(round(coords[i].first * cos(angRad)-
coords[i].second*sin(angRad)), round(coords[i].first * sin(angRad)+ coords[i].second *
cos(angRad)));
        }
        glEnd();
}

void rotatePolygonFixed() {
        glBegin(GL_LINE_LOOP);
        glColor3f(1.0, 0.0, 1.0);
        vector<pair<int, int>> newCoords;
        vector<vector<double>> rotatingMatrix(3, vector<double>(3, 0));
        vector<double> curpoint(3, 0), matProduct(3, 0);
        rotatingMatrix[0][0] = cos(angRad);
        rotatingMatrix[1][1] = cos(angRad);
        rotatingMatrix[2][2] = 1;
        rotatingMatrix[1][0] = sin(angRad);
        rotatingMatrix[0][1] = -1*sin(angRad);
        rotatingMatrix[0][2] = xr * (1 - cos(angRad)) + yr * sin(angRad);
        rotatingMatrix[1][2] = yr * (1 - cos(angRad)) - xr * sin(angRad);

        for (int i = 0; i < n; i++) {
                curpoint[0] = coords[i].first;
                curpoint[1] = coords[i].second;
                curpoint[2] = 1;
                matProduct[0] = 0;
                matProduct[1] = 0;
                matProduct[2] = 0;
                for (int j = 0; j < 3; j++) {
                        for (int k = 0; k < 3; k++) {
                                matProduct[j] += rotatingMatrix[j][k] * curpoint[k];
                        }
                }
                newCoords.push_back(make_pair(round(matProduct[0]),
round(matProduct[1])));
        }

        for (int i = 0; i < n; i++)
        {
                glVertex2d(newCoords[i].first, newCoords[i].second);
        }
        glEnd();
}

void scalePolygonOrigin() {
        glBegin(GL_LINE_LOOP);
        glColor3f(0.0, 1.0, 0.0);
        for (int i = 0; i < n; i++)
        {
                glVertex2d(round(coords[i].first*sx), round(coords[i].second*sy));
        }
        glEnd();
}

void scalePolygonFixed() {
        glBegin(GL_LINE_LOOP);
        glColor3f(1.0, 1.0, 0.0);
        vector<pair<int,int>> newCoords;
        vector<vector<double>> scalingMatrix(3, vector<double>(3, 0));
        vector<double> curpoint(3,0), matProduct(3, 0);
        scalingMatrix[0][0] = sx;
        scalingMatrix[1][1] = sy;
        scalingMatrix[2][2] = 1;
        scalingMatrix[0][2] = xf * (1 - sx);
        scalingMatrix[1][2] = yf * (1 - sy);

        for (int i = 0; i < n; i++) {
                curpoint[0] = coords[i].first;
                curpoint[1] = coords[i].second;
                curpoint[2] = 1;
                matProduct[0] = 0;
                matProduct[1] = 0;
                matProduct[2] = 0;
                for (int j = 0; j < 3; j++) {
```

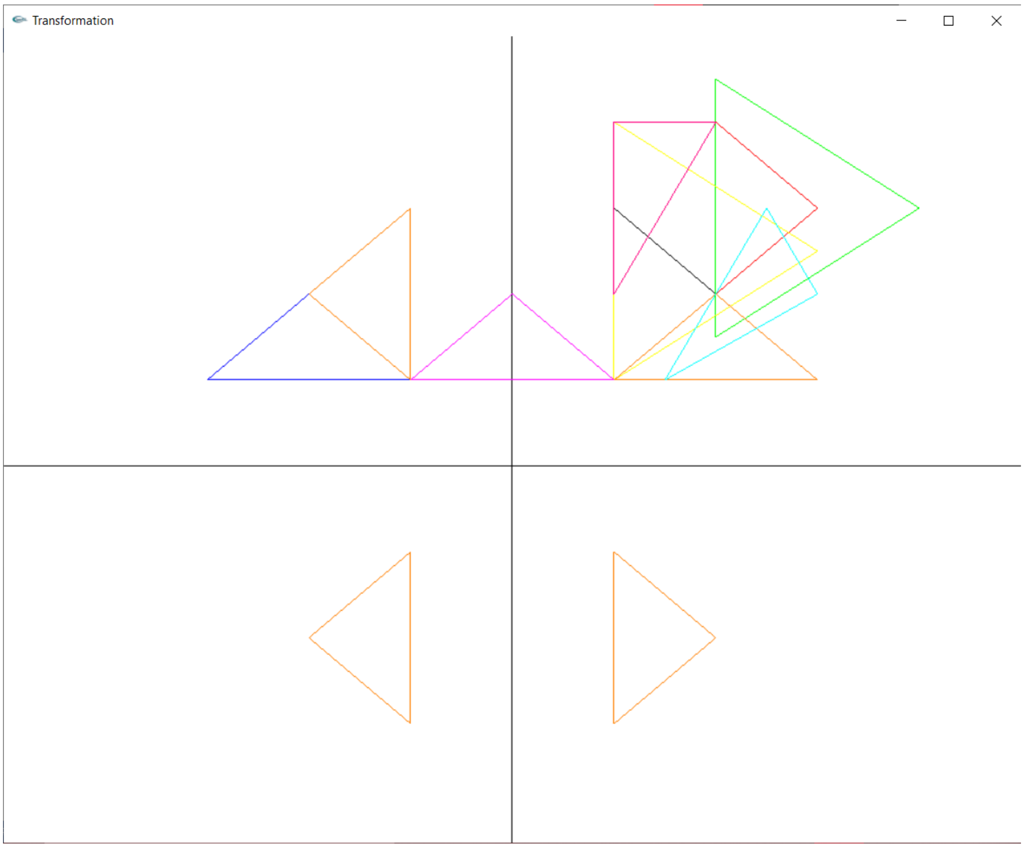
```
				for (int k = 0; k < 3; k++) {
					matProduct[j] += scalingMatrix[j][k] * curpoint[k];
				}
			}
			newCoords.push_back(make_pair(round(matProduct[0]), 
round(matProduct[1])));
		}

		for (int i = 0; i < n; i++)
		{
			glVertex2d(newCoords[i].first, newCoords[i].second);
		}
		glEnd();
}

void reflection_Xaxis() {
		glBegin(GL_LINE_LOOP);
		glColor3f(1.0, 0.5, 0.0);
		vector<pair<int, int>> newCoords;
		vector<vector<double>> reflectMatrix(3, vector<double>(3, 0));
		vector<double> curpoint(3, 0), matProduct(3, 0);
		reflectMatrix[0][0] = 1;
		reflectMatrix[1][1] = -1;
		reflectMatrix[2][2] = 1;

		for (int i = 0; i < n; i++) {
			curpoint[0] = coords[i].first;
			curpoint[1] = coords[i].second;
			curpoint[2] = 1;
			matProduct[0] = 0;
			matProduct[1] = 0;
			matProduct[2] = 0;
			for (int j = 0; j < 3; j++) {
				for (int k = 0; k < 3; k++) {
					matProduct[j] += reflectMatrix[j][k] * curpoint[k];
				}
			}
			newCoords.push_back(make_pair(round(matProduct[0]), 
round(matProduct[1])));
		}

		for (int i = 0; i < n; i++)
		{
			glVertex2d(newCoords[i].first, newCoords[i].second);
		}
		glEnd();
}
void reflection_Yaxis() {
		glBegin(GL_LINE_LOOP);
		glColor3f(1.0, 0.5, 0.0);
		vector<pair<int, int>> newCoords;
		vector<vector<double>> reflectMatrix(3, vector<double>(3, 0));
		vector<double> curpoint(3, 0), matProduct(3, 0);
		reflectMatrix[0][0] = -1;
		reflectMatrix[1][1] = 1;
		reflectMatrix[2][2] = 1;

		for (int i = 0; i < n; i++) {
			curpoint[0] = coords[i].first;
			curpoint[1] = coords[i].second;
			curpoint[2] = 1;
			matProduct[0] = 0;
```

```
        matProduct[1] = 0;
        matProduct[2] = 0;
        for (int j = 0; j < 3; j++) {
            for (int k = 0; k < 3; k++) {
                matProduct[j] += reflectMatrix[j][k] * curpoint[k];
            }
        }
        newCoords.push_back(make_pair(round(matProduct[0]),
round(matProduct[1])));
    }

    for (int i = 0; i < n; i++)
    {
        glVertex2d(newCoords[i].first, newCoords[i].second);
    }
    glEnd();
}
void reflection_origin() {
    glBegin(GL_LINE_LOOP);
    glColor3f(1.0, 0.5, 0.0);
    vector<pair<int, int>> newCoords;
    vector<vector<double>> reflectMatrix(3, vector<double>(3, 0));
    vector<double> curpoint(3, 0), matProduct(3, 0);
    reflectMatrix[0][0] = -1;
    reflectMatrix[1][1] = -1;
    reflectMatrix[2][2] = 1;

    for (int i = 0; i < n; i++) {
        curpoint[0] = coords[i].first;
        curpoint[1] = coords[i].second;
        curpoint[2] = 1;
        matProduct[0] = 0;
        matProduct[1] = 0;
        matProduct[2] = 0;
        for (int j = 0; j < 3; j++) {
            for (int k = 0; k < 3; k++) {
                matProduct[j] += reflectMatrix[j][k] * curpoint[k];
            }
        }
        newCoords.push_back(make_pair(round(matProduct[0]),
round(matProduct[1])));
    }

    for (int i = 0; i < n; i++)
    {
        glVertex2d(newCoords[i].first, newCoords[i].second);
    }
    glEnd();
}

void reflection_XeqYline() {
    glBegin(GL_LINE_LOOP);
    glColor3f(1.0, 0.5, 0.0);
    vector<pair<int, int>> newCoords;
    vector<vector<double>> reflectMatrix(3, vector<double>(3, 0));
    vector<double> curpoint(3, 0), matProduct(3, 0);
    reflectMatrix[0][1] = 1;
    reflectMatrix[1][0] = 1;
    reflectMatrix[2][2] = 1;

    for (int i = 0; i < n; i++) {
        curpoint[0] = coords[i].first;
        curpoint[1] = coords[i].second;
        curpoint[2] = 1;
        matProduct[0] = 0;
        matProduct[1] = 0;
        matProduct[2] = 0;
        for (int j = 0; j < 3; j++) {
            for (int k = 0; k < 3; k++) {
                matProduct[j] += reflectMatrix[j][k] * curpoint[k];
            }
        }
        newCoords.push_back(make_pair(round(matProduct[0]),
round(matProduct[1])));
    }

    for (int i = 0; i < n; i++)
    {
        glVertex2d(newCoords[i].first, newCoords[i].second);
    }
    glEnd();
}

void x_directionShear() {
    glBegin(GL_LINE_LOOP);
    glColor3f(0.0, 1.0, 1.0);
    vector<pair<int, int>> newCoords;
    vector<vector<double>> shearMatrix(3, vector<double>(3, 0));
    vector<double> curpoint(3, 0), matProduct(3, 0);
    shearMatrix[0][0] = 1;
    shearMatrix[1][1] = 1;
    shearMatrix[2][2] = 1;
    shearMatrix[0][1] = shx;
    for (int i = 0; i < n; i++) {
        curpoint[0] = coords[i].first;
        curpoint[1] = coords[i].second;
        curpoint[2] = 1;
        matProduct[0] = 0;
        matProduct[1] = 0;
        matProduct[2] = 0;
        for (int j = 0; j < 3; j++) {
            for (int k = 0; k < 3; k++) {
                matProduct[j] += shearMatrix[j][k] * curpoint[k];
            }
        }
        newCoords.push_back(make_pair(round(matProduct[0]),
round(matProduct[1])));
    }

    for (int i = 0; i < n; i++)
    {
        glVertex2d(newCoords[i].first, newCoords[i].second);
    }
    glEnd();
}

void y_directionShear() {
    glBegin(GL_LINE_LOOP);
    glColor3f(1.0, 0.0, 0.5);
    vector<pair<int, int>> newCoords;
    vector<vector<double>> shearMatrix(3, vector<double>(3, 0));
    vector<double> curpoint(3, 0), matProduct(3, 0);
    shearMatrix[0][0] = 1;
    shearMatrix[1][1] = 1;
    shearMatrix[2][2] = 1;
    shearMatrix[1][0] = shy;
    for (int i = 0; i < n; i++) {
        curpoint[0] = coords[i].first;
        curpoint[1] = coords[i].second;
        curpoint[2] = 1;
        matProduct[0] = 0;
        matProduct[1] = 0;
        matProduct[2] = 0;
        for (int j = 0; j < 3; j++) {
            for (int k = 0; k < 3; k++) {
                matProduct[j] += shearMatrix[j][k] * curpoint[k];
            }
        }
        newCoords.push_back(make_pair(round(matProduct[0]),
round(matProduct[1])));
    }

    for (int i = 0; i < n; i++)
    {
        glVertex2d(newCoords[i].first, newCoords[i].second);
    }
    glEnd();
}

void myDisplay()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0.0, 0.0, 0.0);
    DrawCartesianPlane();
    drawPolygon();
    drawPolygonTrans();
    scalePolygonOrigin();
    scalePolygonFixed();
    rotatePolygonOrigin();
    rotatePolygonFixed();
    reflection_Xaxis();
    reflection_Yaxis();
    reflection_origin();
    reflection_XeqYline();
    x_directionShear();
    y_directionShear();
    glFlush();
}

int main(int argc, char** argv)
{
    cout << "Specify polygon Dimensions" << endl;
    cout << "Enter the number of vertices : ";
    cin >> n;
    int x, y;

    for (int i = 0; i < n; i++)
    {
        cout << "Enter vertex  " << i + 1 << "(x,y) : ";
        cin >> x >> y;
        coords.push_back(make_pair(x,y));
    }

    cout << "Enter the translation factor for X and Y: ";
    cin >> tx >> ty;
    cout << "Enter the angle of rotation: ";
    cin >> ang;
    angRad = ang * PI / 180;

    cout << "Enter the point to rotate about: ";
    cin >> xr >> yr;

    cout << "Enter the Scaling factor for X and Y : ";
    cin >> sx >> sy;

    cout << "Enter the point to scale (x,y): ";
    cin >> xf >> yf;

    cout << "Enter shear parameter for x-direction shear: ";
    cin >> shx;

    cout << "Enter shear parameter for y-direction shear: ";
    cin >> shy;


    cout << "Original polygon     -> Black Color" << endl;
    cout << "Translated polygon   -> Red Color" << endl;
    cout << "Rotated Polygon : " << endl;
    cout << "     about origin       -> Dark Blue Color" << endl;
    cout << "     about fixed point -> Purple Color" << endl;
    cout << "Scaled Polygon : " << endl;
    cout << "     about origin       -> Green Color" << endl;
    cout << "     about fixed point -> Yellow Color" << endl;
    cout << "Reflection Polygons  -> Orange Color" << endl;
    cout << "X-Direction Shear    -> Light Blue Color" << endl;
    cout << "Y-Direction Shear    -> Pink Color" << endl;



    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(1000, 1000);
    glutCreateWindow("Transformation");
    glutDisplayFunc(myDisplay);
    myInit();
    glutMainLoop();
    return 0;
}
```

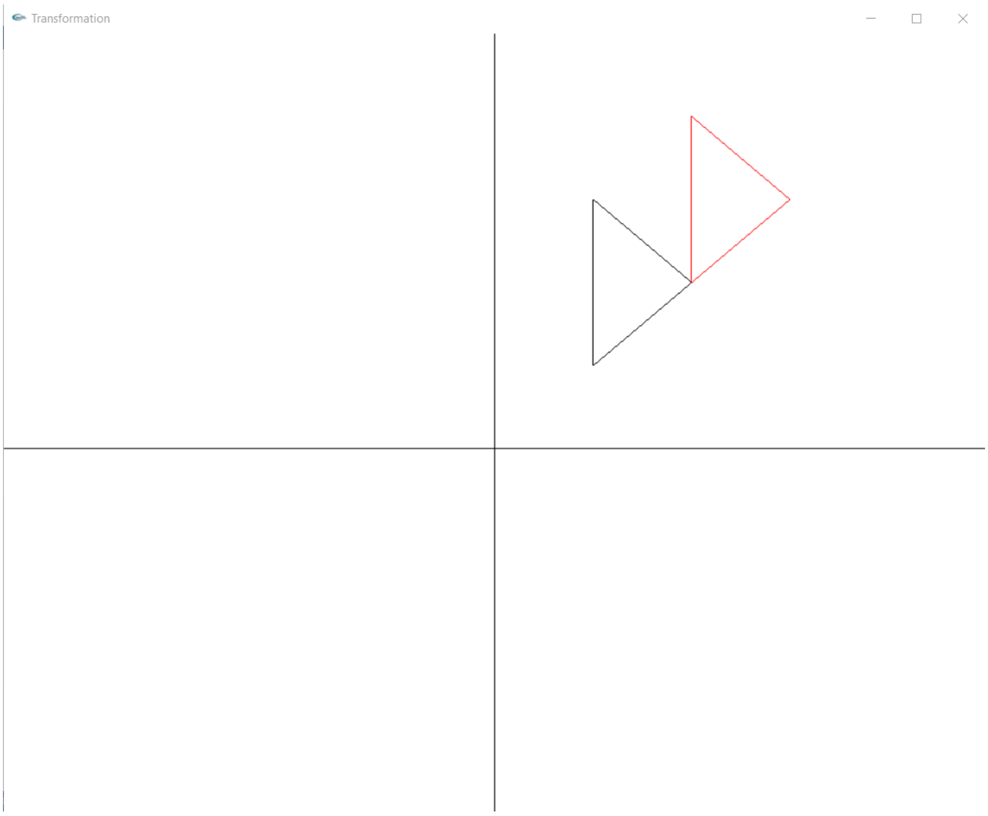
OUTPUT:



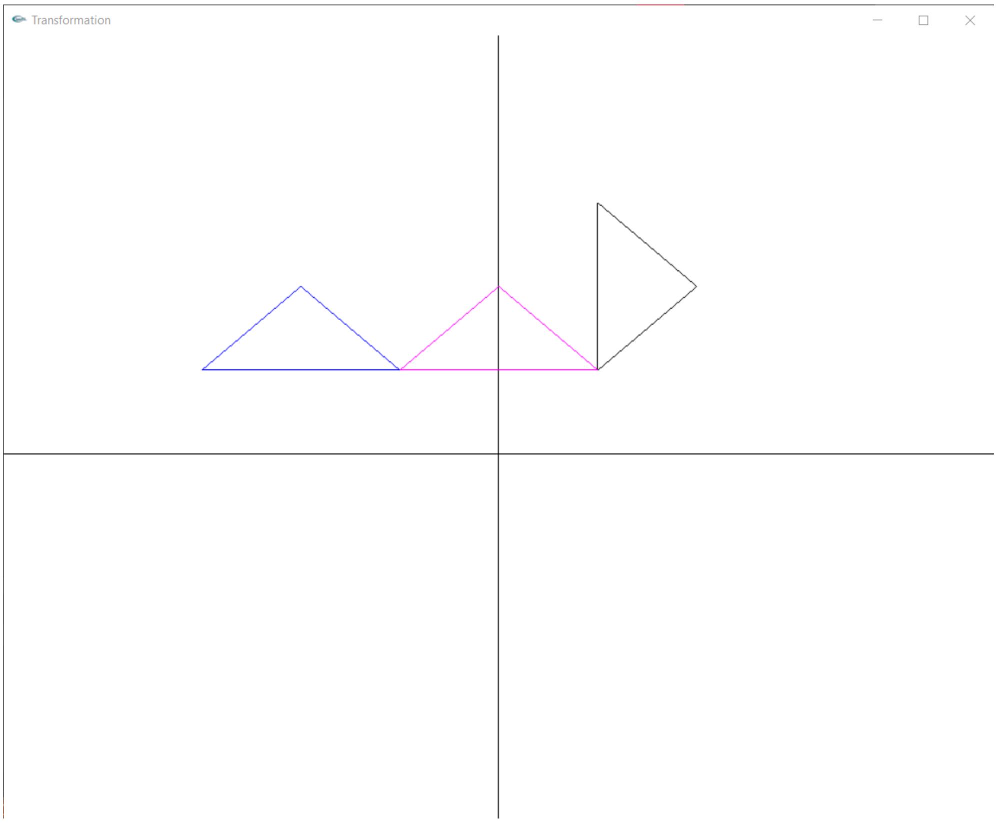
```
Specify polygon Dimensions
Enter the number of vertices : 3
Enter vertex  1(x,y) : 100 100
Enter vertex  2(x,y) : 200 200
Enter vertex  3(x,y) : 100 300
Enter the translation factor for X and Y: 100 100
Enter the angle of rotation: 90
Enter the point to rotate about: 100 100
Enter the Scaling factor for X and Y : 2 1.5
Enter the point to scale (x,y): 100 100
Enter shear parameter for x-direction shear: 0.5
Enter shear parameter for y-direction shear: 1
Original polygon     -> Black Color
Translated polygon   -> Red Color
Rotated Polygon :
        about origin       -> Dark Blue Color
        about fixed point -> Purple Color
Scaled Polygon :
        about origin       -> Green Color
        about fixed point -> Yellow Color
Reflection Polygons  -> Orange Color
X-Direction Shear    -> Light Blue Color
Y-Direction Shear    -> Pink Color
```



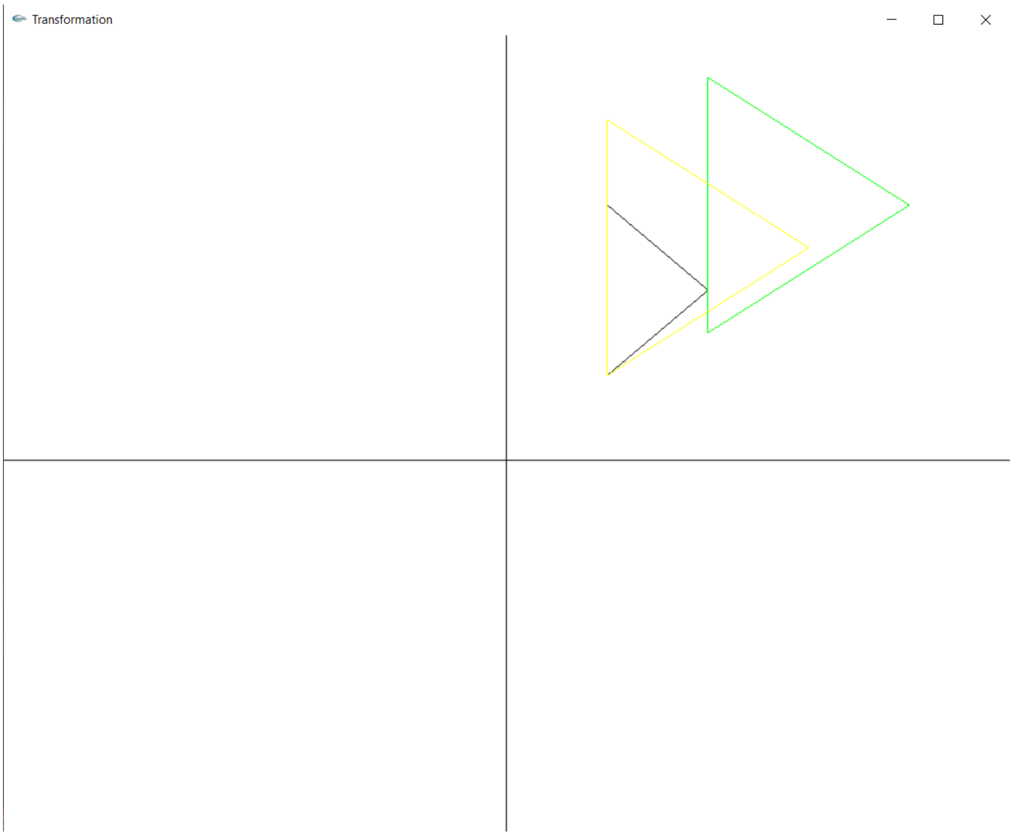
Translation:



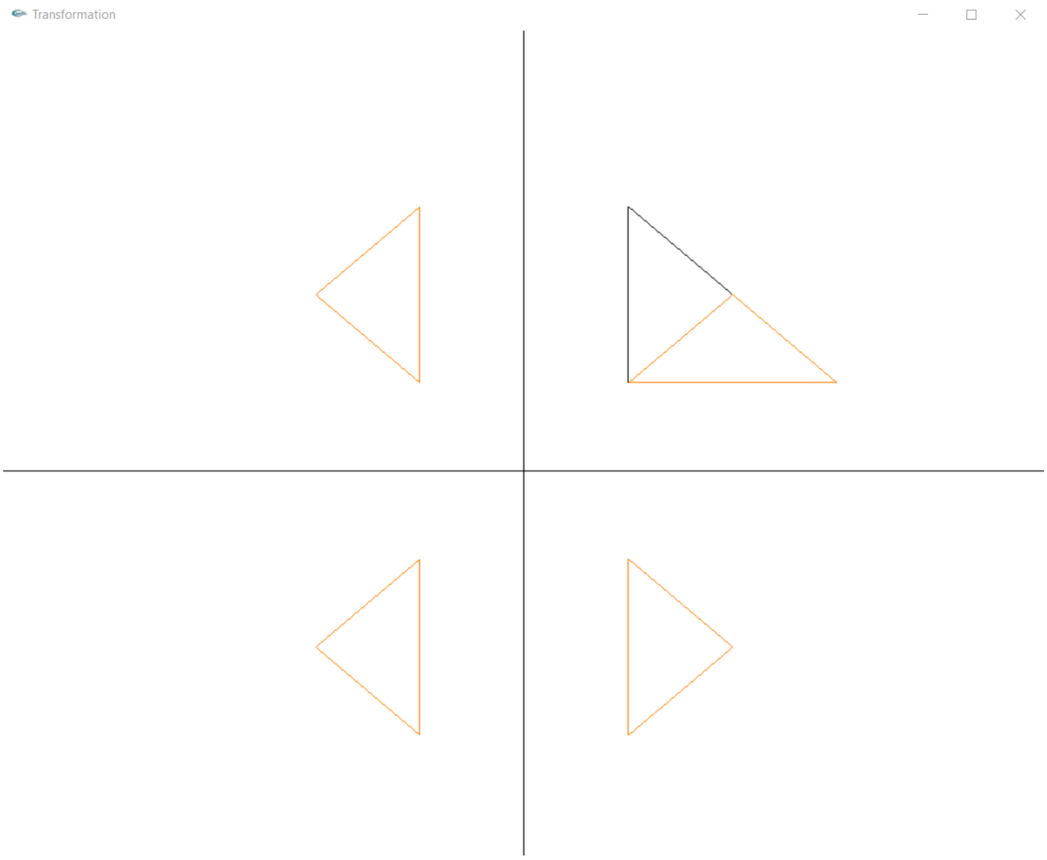
Rotation:



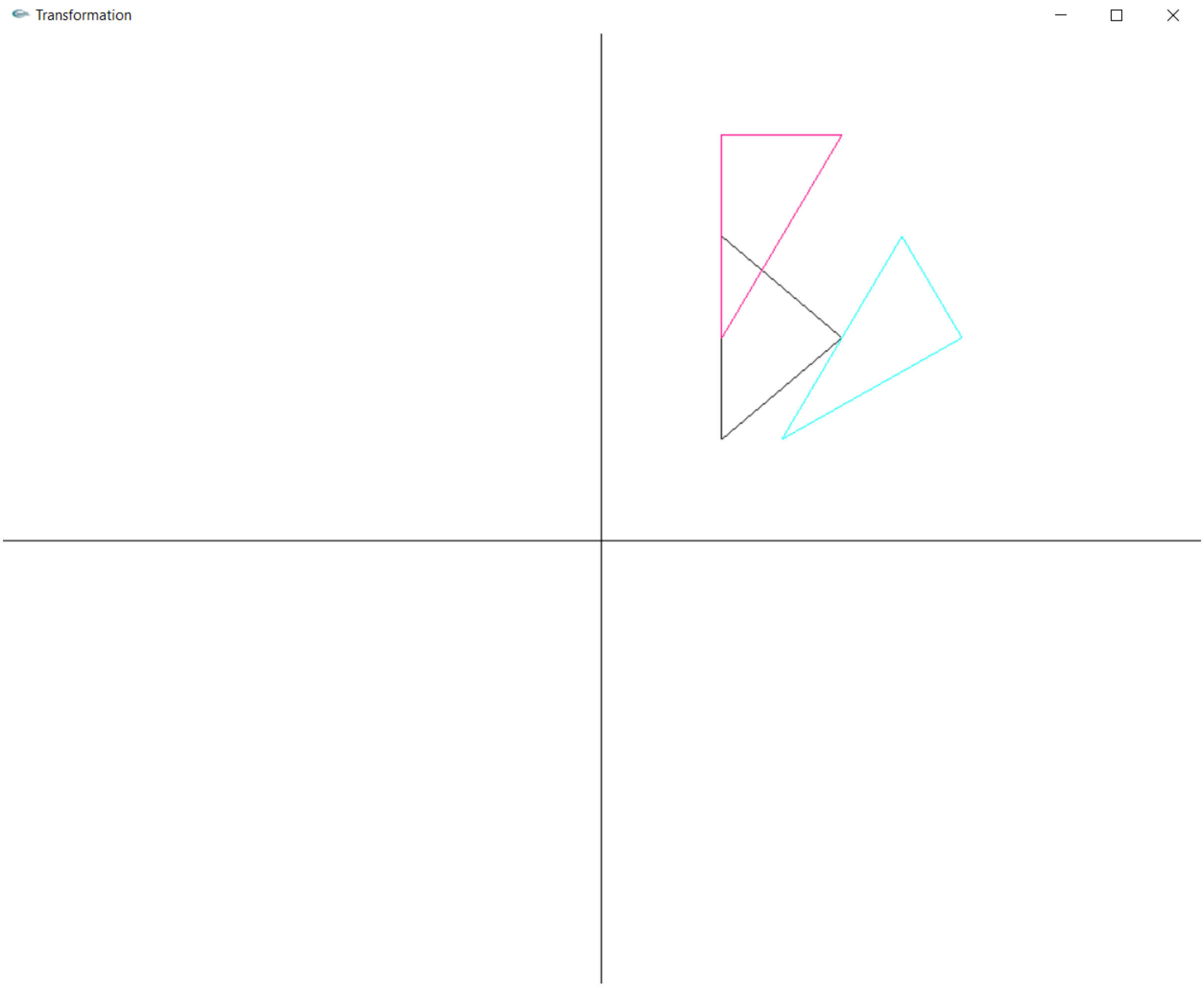
Scaling:



Reflection:



Shearing:



Transformation