<u>Objective :</u>

To understand and implement the 2D Geometric Transformation for Rotation and Scaling of a triangle about a fixed point.

<u>Algorithm / Methodology :</u>

1. Take input all the three end points of the triangle (in the form of a matrix, say **M**) and the coordinates of the point (x,y) about which the triangle has to be rotated and then scaled.
2. Take input the angle (θ), the degree to which the triangle has to be rotated and also the scaling factor in both dimensions about how much the triangle has to be scaled.
3. We will compute a transformation matrix T according to which we will transform the given triangle -
    a. First we will translate the point about which the triangle has to be rotated to origin. The translation matrix (A) will be -

    $$A = \begin{bmatrix} 1 & 0 & -x \\ 0 & 1 & -y \\ 0 & 0 & 1 \end{bmatrix}$$

    b. Then we will use the rotation matrix (B) and rotate the translated triangle about origin to the amount (θ) as given input.

    $$B = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

    c. Then we will apply the scaling matrix (C) and scale the rotated triangle according to the input scaling factors.

    $$C = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

d. Then, finally we will again translate our triangle according to the point given initially i.e., (x,y) using matrix (D) to its original position after transformation -

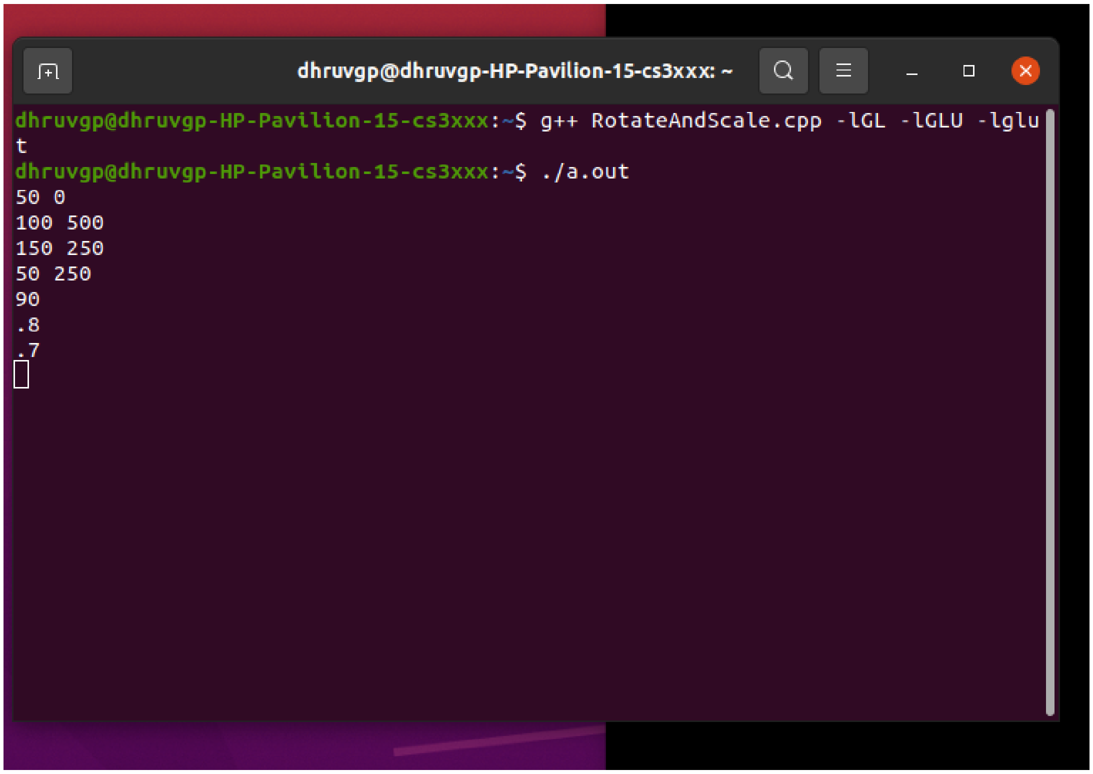
$$D = \begin{bmatrix} 1 & 0 & x \\ 0 & 1 & y \\ 0 & 0 & 1 \end{bmatrix}$$

4. Hence, our final transformation matrix T = D * C * B * A.
5. Our final transformated coordinates of the given input triangle are -

$$F = T * (M)$$

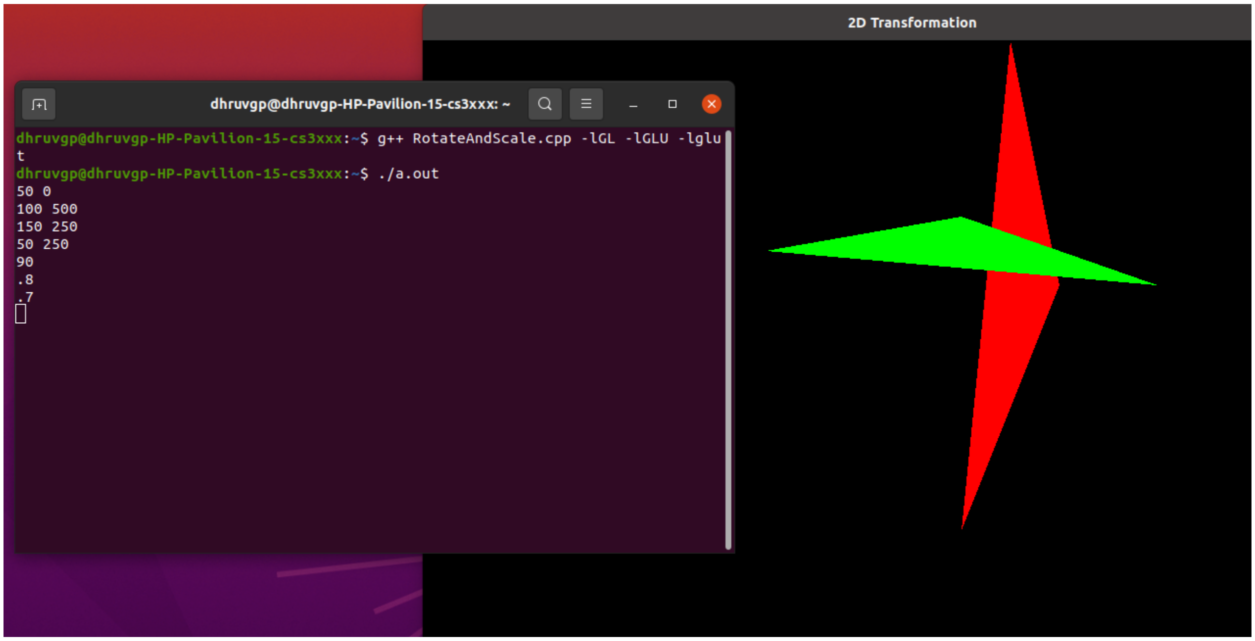
Input :

- **The first three lines denote the end points of the initial triangle taken input.**
- **The next line represents the point (x,y) about which the triangle has to be rotated.**
- **The next line represents the angle in degrees(θ) to which extent the triangle has to be roasted.**
- **The last two lines denote the scaling factors (sx, sy).**



```
dhruvgp@dhruvgp-HP-Pavilion-15-cs3xxx:~$ g++ RotateAndScale.cpp -lGL -lGLU -lglu
t
dhruvgp@dhruvgp-HP-Pavilion-15-cs3xxx:~$ ./a.out
50 0
100 500
150 250
50 250
90
.8
.7
```

Output :

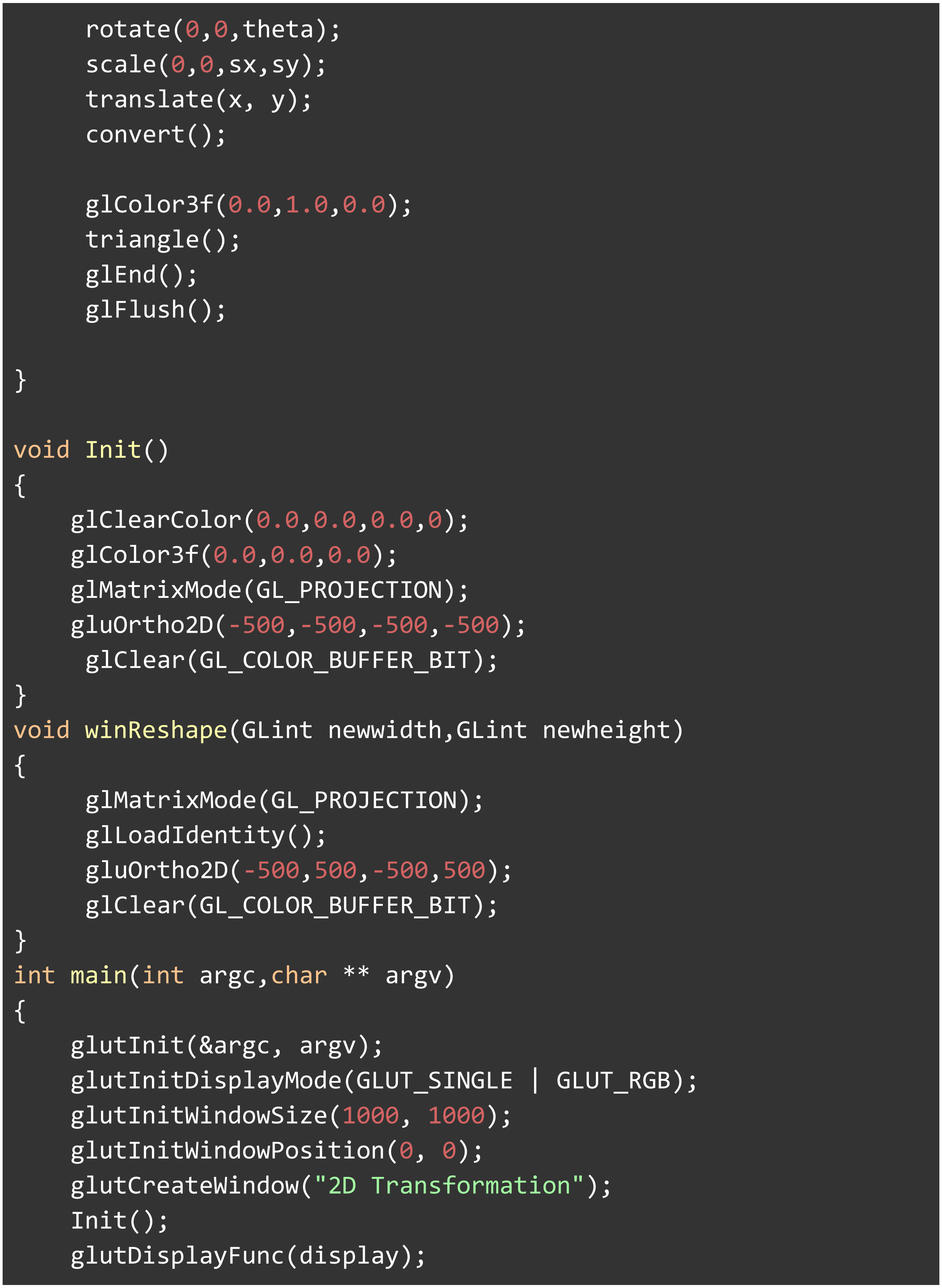


```
dhruvgp@dhruvgp-HP-Pavilion-15-cs3xxx:~$ g++ RotateAndScale.cpp -lGL -lGLU -lglu
t
dhruvgp@dhruvgp-HP-Pavilion-15-cs3xxx:~$ ./a.out
50 0
100 500
150 250
50 250
90
.8
.7
```

Appendix - Code :

```
#include <GL/glut.h>
#include <bits/stdc++.h>
using namespace std;

const double pi = acos(-1);

float x[3], y[3];

float T[3][3];

void triangle() {
     for(int i=0;i<3;++i) {
          glVertex2f(x[i], y[i]);
     }
}

void identity_matrix(float a[][3]) {
     for(int i=0;i<3;++i) {
          for(int j=0;j<3;++j) {
               a[i][j] = (i == j);
          }
     }
}

// multiplying mat to T (transformation matrix)
void mult(float mat[][3])
{
     float tmp[3][3];
     for(int i=0;i<3;++i) {
          for(int j=0;j<3;++j) {
               tmp[i][j] = 0;
               for(int k=0;k<3;++k) {
                    tmp[i][j] += mat[i][k] * T[k][j];
```

```
            }
        }
    }
    for(int i=0;i<3;++i) {
        for(int j=0;j<3;++j) {
            T[i][j] = tmp[i][j];
        }
    }
}

void translate(float tx, float ty) {
    float transition[3][3];
    identity_matrix(transition);
    transition[0][2] = tx;
    transition[1][2] = ty;
    mult(transition);
}

void rotate(float x, float y,float angle) {
    angle = angle / 180 * pi;

    // Translate - Rotate - Translate

    translate(-x, -y);

    float transition[3][3];
    identity_matrix(transition);
    transition[0][0] = transition[1][1] = cos(angle);
    transition[0][1] = -sin(angle);
    transition[1][0] = - transition[0][1];
    mult(transition);

    translate(x, y);
}

void scale(float x, float y, float sx, float sy) {
    translate(-x, -y);

    float transition[3][3];
    identity_matrix(transition);
    transition[0][0] = sx;
    transition[1][1] = sy;
    mult(transition);

    translate(x,y);
}

void convert() {
    for(int i=0;i<3;++i) {
        float X = T[0][0] * x[i] + T[0][1] * y[i] + T[0][2];
        float Y = T[1][0] * x[i] + T[1][1] * y[i] + T[1][2];
        x[i] = X, y[i] = Y;
    }
}


void display() {
    for(int i=0;i<3;++i) {
        cin >> x[i] >> y[i];
    }
    identity_matrix(T);

    glBegin(GL_TRIANGLES);
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0,0.0,0.0);

    triangle();

    float x, y, theta;
    float sx, sy;
    cin >> x >> y >> theta >> sx >> sy;
    translate(-x,-y);
```



```
     rotate(0,0,theta);
     scale(0,0,sx,sy);
     translate(x, y);
     convert();

     glColor3f(0.0,1.0,0.0);
     triangle();
     glEnd();
     glFlush();

}

void Init()
{
    glClearColor(0.0,0.0,0.0,0);
    glColor3f(0.0,0.0,0.0);
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(-500,-500,-500,-500);
     glClear(GL_COLOR_BUFFER_BIT);
}
void winReshape(GLint newwidth,GLint newheight)
{
     glMatrixMode(GL_PROJECTION);
     glLoadIdentity();
     gluOrtho2D(-500,500,-500,500);
     glClear(GL_COLOR_BUFFER_BIT);
}
int main(int argc,char ** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(1000, 1000);
    glutInitWindowPosition(0, 0);
    glutCreateWindow("2D Transformation");
    Init();
    glutDisplayFunc(display);
    glutReshapeFunc(winReshape);
    glutMainLoop();
    return 0;
}
```