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

EX NO: 5a – 2D Transformations – Translation, Rotation and Scaling

---

Name : Likhitha Verma A
Reg : 185001084
Date : 25/08/2021

**AIM :**

To write a C++ menu-driven program using OPENGL to perform 2D transformations – translation, rotation, scaling for line and polygon.

**ALGORITHM :**

1. Read the no. of edges of the polygon from the user.
2. Read the vertices of the polygon and the line
3. Plot the original polygon and the line
4. Read the transformation from the user given the menu
5. If option is translation :
    a. Read the translation factor (x’, y’) from the user
    b. Add the translation factor to each of the original coordinates of the polygon
    c. Plot the translated polygon
6. If option is scaling :
    a. Read the scaling factor (x’, y’) from the user
    b. Multiply the scaling factor to each of the original coordinates of the polygon
    c. Plot the scaled polygon
    d. You may also translate the scaled polygon so that it doesn’t overlap on the original polygon
7. If option is rotation :
    a. Read the angle of rotation from the user.
    b. Convert angle from degrees to radians.
    c. Apply rotation formula to each of the original coordinates of the polygon
    d. Plot the rotated coordinates of polygon

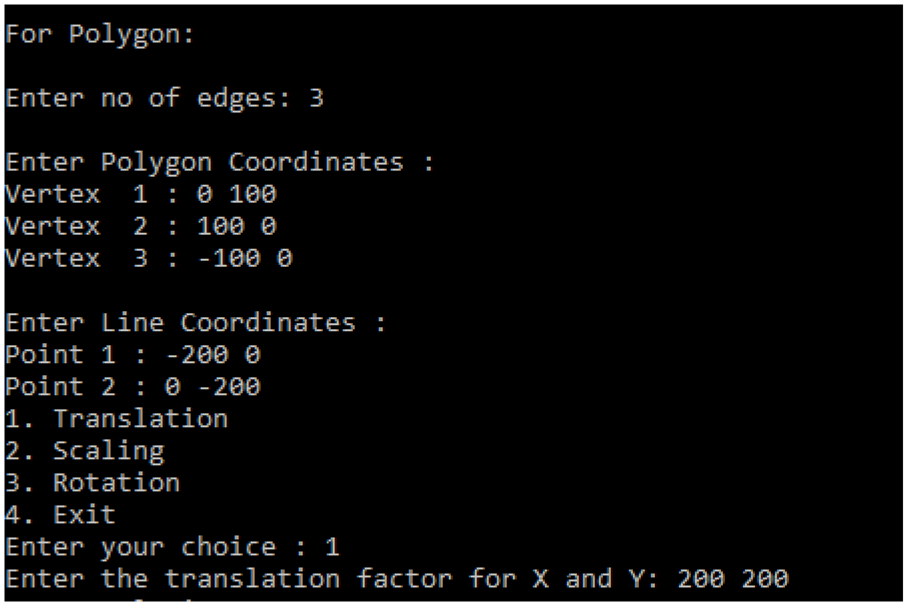
CODE :

```
#include <stdio.h>
#include <math.h>
#include <iostream>
#include <vector>
#include <glut.h>
using namespace std;

int pntX1, pntY1, op = 0, edges;
vector<int> pntX;
vector<int> pntY;
int transX, transY, lineX1, lineX2, lineY1, lineY2;
double scaleX, scaleY;
double angle, angleRad;
char reflectionAxis;

double round(double d)
{
      return floor(d + 0.5);
}

void drawPolygon()
{
      glBegin(GL_POLYGON);
      glColor3f(0.48, 0, 0.7);
      for (int i = 0; i < edges; i++)
      {
            glVertex2i(pntX[i], pntY[i]);
      }
      glEnd();

      glBegin(GL_LINES);
      glVertex2d(lineX1, lineY1);
      glVertex2d(lineX2, lineY2);
      glEnd();
}


void translate(int x, int y)
{
      glBegin(GL_POLYGON);
      glColor3f(0.08, 0.67, 0);
      for (int i = 0; i < edges; i++)
      {
            glVertex2i(pntX[i] + x, pntY[i] + y);
      }
      glEnd();
      glBegin(GL_LINES);
      glVertex2d(lineX1 + x, lineY1 + y);
      glVertex2d(lineX2 + x, lineY2 + y);
      glEnd();
}

void scale(double x, double y)
{
      glBegin(GL_POLYGON);
      glColor3f(0.08, 0.67, 0);
      for (int i = 0; i < edges; i++)
      {
            glVertex2i(round(pntX[i] * x) + 300, round(pntY[i] * y));
      }
      glEnd();
      glBegin(GL_LINES);
      glVertex2d(round(lineX1 * x), round(lineY1 * y));
      glVertex2d(round(lineX2 * x), round(lineY2 * y));
      glEnd();
}

void rotate(double theta)
{
      glBegin(GL_POLYGON);
      glColor3f(0.08,0.67,0);
      for (int i = 0; i < edges; i++)
      {
            glVertex2i(round((pntX[i] * cos(theta)) - (pntY[i] * sin(theta))),
round((pntX[i] * sin(theta)) + (pntY[i] * cos(theta))));
      }
      glEnd();

      glBegin(GL_LINES);
      glVertex2d(round((lineX1 * cos(theta)) - (lineY1 * sin(theta))),
round((lineX1 * sin(theta)) + (lineY1 * cos(theta))));
      glVertex2d(round((lineX2 * cos(theta)) - (lineY2 * sin(theta))),
round((lineX2 * sin(theta)) + (lineY2 * cos(theta))));
```

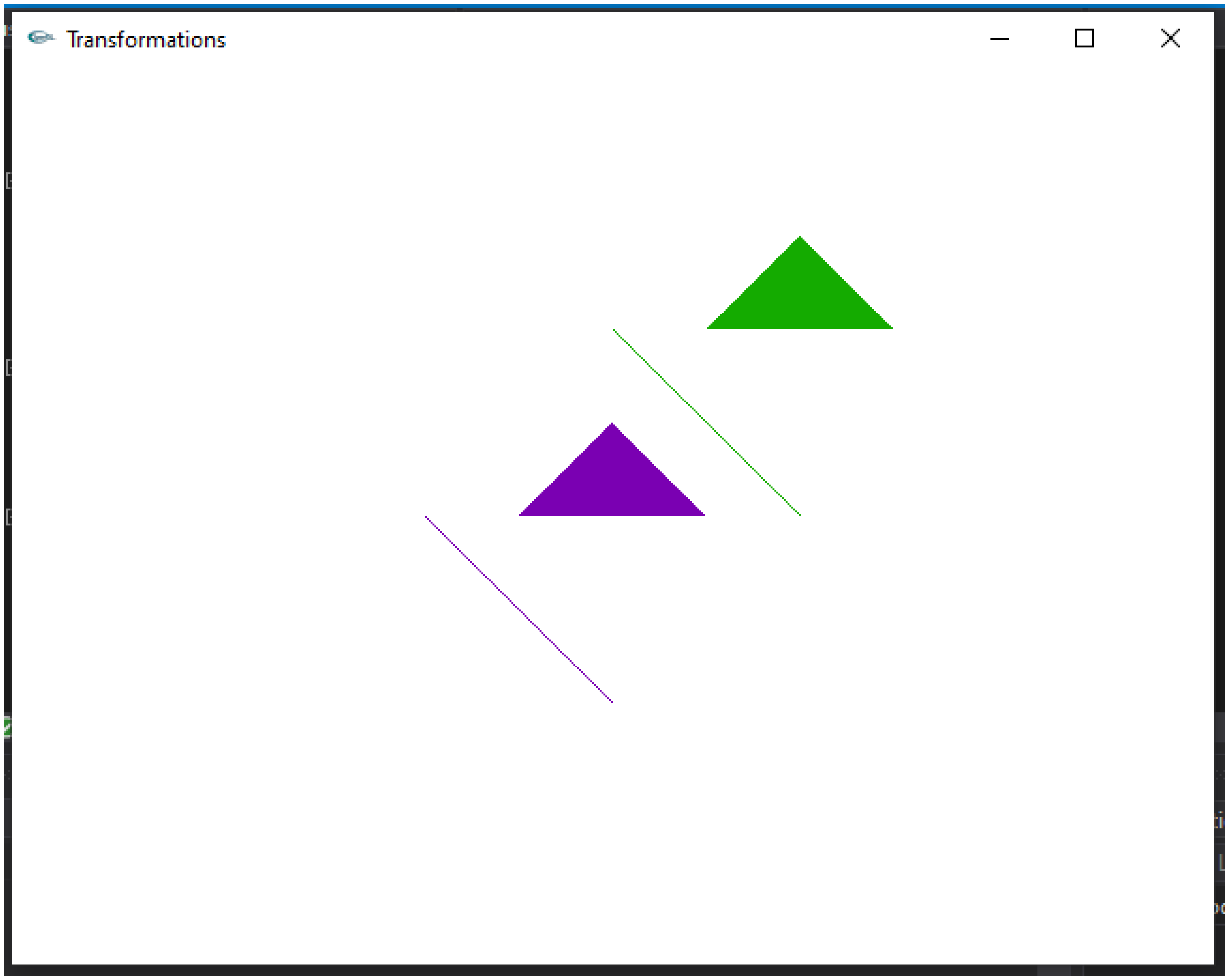
```
      glEnd();
}

void myInit(void)
{
      glClearColor(1.0, 1.0, 1.0, 0.0);
      glColor3f(0.0f, 0.0f, 0.0f);
      glPointSize(4.0);
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      gluOrtho2D(-640.0, 640.0, -480.0, 480.0);
}

void myDisplay(void)
{
      while (true) {
            glClear(GL_COLOR_BUFFER_BIT);
            glColor3f(0.0, 0.0, 0.0);
            drawPolygon();
            cout << "1. Translation\n";
            cout << "2. Scaling\n";
            cout << "3. Rotation\n";
            cout << "4. Exit\n";
            cout << "Enter your choice : ";

            cin >> op;

            if (op == 4) {
                  break;
            }

            if (op == 1)
            {
                  cout << "Enter the translation factor for X and Y: "; cin >>
transX >> transY;
                  translate(transX, transY);
            }
            else if (op == 2)
            {
                  cout << "Enter the scaling factor for X and Y: "; cin >> scaleX
>> scaleY;
                  scale(scaleX, scaleY);
            }
```

```
			else if (op == 3)
			{
				cout << "Enter the angle for rotation: "; cin >> angle;
				angleRad = angle * 3.1416 / 180;
				rotate(angleRad);
			}
			glFlush();
		}

}

void main(int argc, char** argv)
{
		cout << "\nFor Polygon:\n" << endl;
		cout << "Enter no of edges: "; cin >> edges;
		cout << "\nEnter Polygon Coordinates : \n";

		for (int i = 0; i < edges; i++){
			cout << "Vertex  " << i + 1 << " : "; cin >> pntX1 >> pntY1;
			pntX.push_back(pntX1);
			pntY.push_back(pntY1);
		}

		cout << "\nEnter Line Coordinates : \n";
		cout << "Point 1 : "; cin >> lineX1 >> lineY1;
		cout << "Point 2 : "; cin >> lineX2 >> lineY2;

		glutInit(&argc, argv);
		glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
		glutInitWindowSize(640, 480);
		glutInitWindowPosition(100, 150);
		glutCreateWindow("Transformations");
		glutDisplayFunc(myDisplay);
		myInit();
		glutMainLoop();

}
```

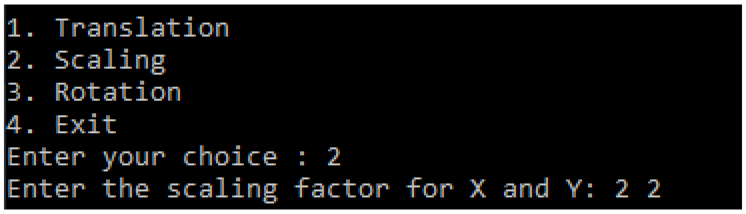
OUTPUT : **1) Translation**



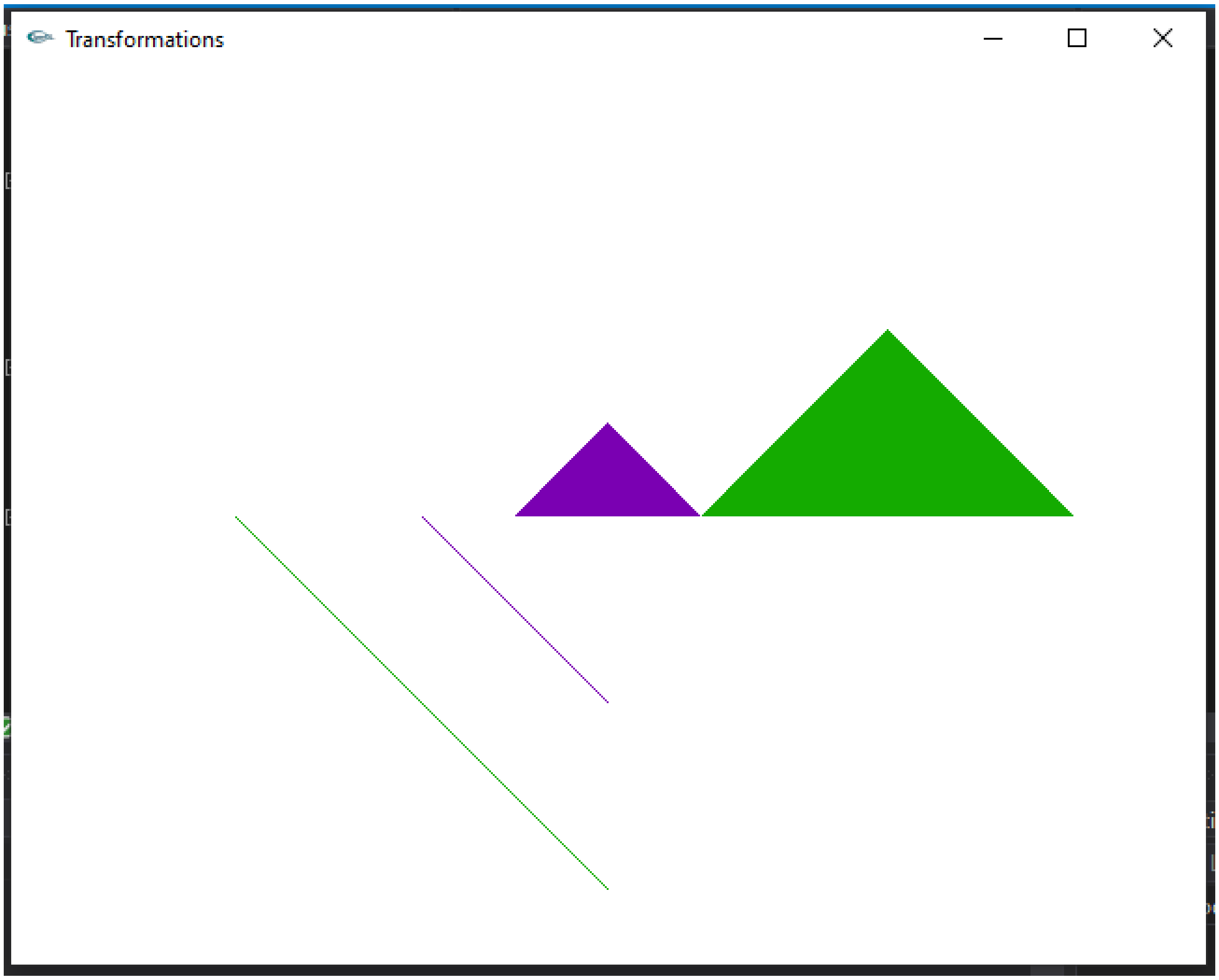
```
For Polygon:

Enter no of edges: 3

Enter Polygon Coordinates :
Vertex  1 : 0 100
Vertex  2 : 100 0
Vertex  3 : -100 0

Enter Line Coordinates :
Point 1 : -200 0
Point 2 : 0 -200
1. Translation
2. Scaling
3. Rotation
4. Exit
Enter your choice : 1
Enter the translation factor for X and Y: 200 200
```



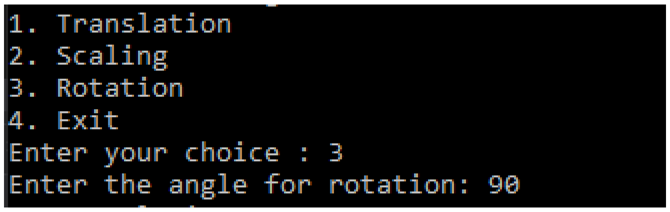
## 2) Scaling



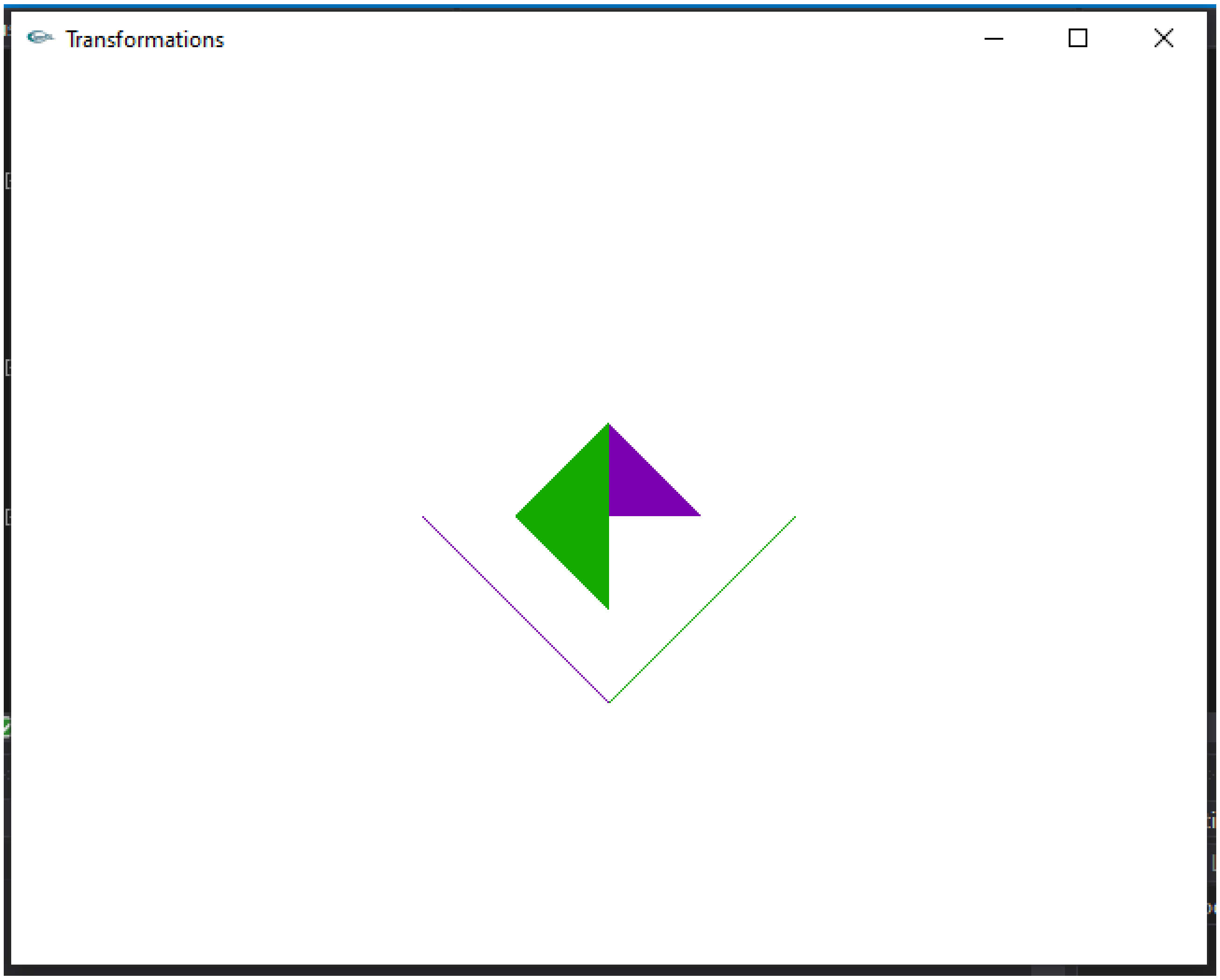
```
1. Translation
2. Scaling
3. Rotation
4. Exit
Enter your choice : 2
Enter the scaling factor for X and Y: 2 2
```



## 3) Rotation



```
1. Translation
2. Scaling
3. Rotation
4. Exit
Enter your choice : 3
Enter the angle for rotation: 90
```



**RESULT :**

Thus compiled and executed a C++ menu-driven program using OPENGL to perform 2D transformations – translation, rotation, scaling for line and polygon successfully.