UCS1712 – GRAPHICS AND MULTIMEDIA LAB

Ex. No. 5 2D Transformations in C++ using OpenGL

Date: 16/8/21 **Name:** Srinath S

Class: CSE-C **Roll:** 185001205

Question:

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:

```
#include <stdio.h>
#include <math.h>
#include <iostream>
#include <vector>
#include <gl/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;

int shearingX, shearingY;


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 reflection(int option)
{
	if (option == 4) {
		glBegin(GL_LINES); glVertex2i(-640, -640);
		glVertex2i(640, 640); glEnd();
	}
	glBegin(GL_POLYGON); glColor3f(0.02, 0.72, 0.09);

	//X axis reflection

	if (option == 1)
	{
		for (int i = 0; i < edges; i++)
		{
			glVertex2i(round(pntX[i]), round(pntY[i] * -1));

		}
	}//Y axis reflection
	else if (option == 2)
	{
		for (int i = 0; i < edges; i++)
		{
			glVertex2i(round(pntX[i] * -1), round(pntY[i]));
		}
	}//origin reflection
	else if (option == 3) {
		for (int i = 0; i < edges; i++) {
			glVertex2i(round(pntX[i] * -1), round(pntY[i]) * -
1);
		}
	}//Y=X reflection
	else if (option == 4) {
		for (int i = 0; i < edges; i++) {
			glVertex2i(round(pntY[i]), round(pntX[i]));
		}
	}
	glEnd();
}

void shearing(int option)
```

```
{
	glBegin(GL_POLYGON); glColor3f(0.02, 0.72, 0.09);

	//translating the transformed polygon so that it doesn't
overlap on the original polygon
	if (option == 5)
	{
		glVertex2i(pntX[0] + 200, pntY[0]);

		glVertex2i(pntX[1] + shearingX + 200, pntY[1]);
glVertex2i(pntX[2] + shearingX + 200, pntY[2]);

		glVertex2i(pntX[3] + 200, pntY[3]);

	}
	else if (option == 6)
	{
		glVertex2i(pntX[0] + 200, pntY[0]); glVertex2i(pntX[1] +
200, pntY[1]);

		glVertex2i(pntX[2] + 200, pntY[2] + shearingY);
glVertex2i(pntX[3] + 200, pntY[3] + shearingY);
	}
	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)
		{
```

```
                transX >> transY;

            }

            cout << "Enter the translation factor for X and Y: "; cin
>> translate(transX, transY);

            else if (op == 2)
            {


                >> scaleY;

            }

            cout << "Enter the scaling factor for X and Y: "; cin >>
scaleX; 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: