Graphics Assignment 02 21118

Problem statement:
write CH program to implement translation, rotation & scaling transformations on equilateral triangle ~~& poly~~ rhombus. Apply the concept of operator overloading.

Objectives
To learn & apply basic transformations on 2D object.

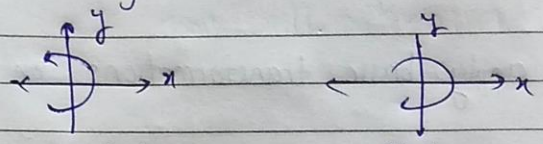
Theory
Transformation means changing some graphic into something else by applying rules.
We can have various types of transformations such as translation, scaling, rotation, shearing, reflection etc when a transformation takes place on 2-d plane it is called 2-d transformation.
Translation, scaling & rotation are basic transformations.

Translation:
It is defined as shifting of object along a straight path.
Translation doesn't alter the shape/size of the object it just moves the entire object from one location to another location in straigh path.
The vector $T = \{t_x, t_y\}$ specifying the amount of translation along x & y direction is called shift vector. ⊛ translation vector.

$$x' = x + t_x$$
$$y' = y + t_y$$

Rotations
- 2D rotation is described by repositioning all the points of an object along a circular path in the 2D plane
- To perform rotation, we need rotation ∠ θ & reference

point (x,y) wrt. which object is to be rotated.
The reference point is also called pivot point.
- If the rotation is performed in an anticlockwise direction, the value of the angle is considered positive otherwise it is negative



Anticlockwise Clockwise

- rotation can be performed wrt. origin (or) some reference point.

5) Scaling:
- Translation & rotation change the position of an object. They do not deform the shape.
- scaling may alter the shape of an object. Scaling can be performed wrt. origin (or) some reference point.
- In the scaling process, you either expand (or) compress the dimensions of the object.
scaling can be achieved by multiplying the original co-ordinates of the object with the scaling factor to get the desired result.
- let the original coordinates are x,y
the scaling factor (Sx & Sy)
produce co-ordinates x',y' as
$x' = x \cdot Sx$ & $y' = y \cdot Sy$

* Software & Hardware Requirements:
Qt creator (latest version)
Window 10, 32-bit, GCC/g++ compiler.

Algorithm

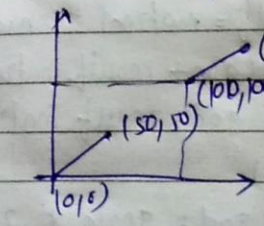
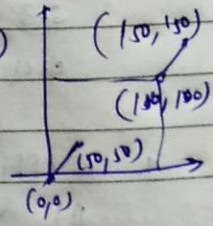
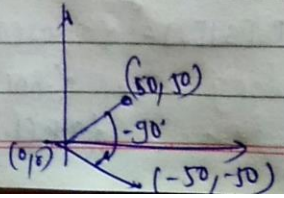
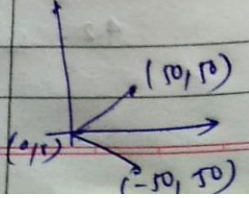
class Multiplication {
initialize matrix[3][3] as double
Multiplication operator * (Multiplication obj) {
// ans is object of multiplication
1. for i ← 0 to 2: i++ do
1.1 for j ← 0 to 3 do
ans.matrix[i][j] = 0
1.1.1 for k ← 1 to 3 do.
ans.matrix[i][j] = ans.matrix[i][j] +
matrix[i][k] * obj.matrix[k][j]
2. return ans
}

Algorithm transformations (flag)
{ // Creating objects of matrix1, matrix2, result of multiplication
1. O[2][3] = {{x, y, 1}, {x2, y2, 1}}
2 for i ← 1 to 2 do
2.1 for j ← 0 to 3 do
2.1.1 matrix1.matrix[i][j] = O[i][j]
3 if (flag == 1)
3.1 S[3][3] = {{Sx, 0, 0}, {0, Sy, 0} {x * (1-Sx), y * (1-Sy), 1}}
3.2 for i ← 1 to 3 do
3.2.1 for j ← 1 to 3 do
3.2.1.1 matrix2.matrix[i][j] = S[i][j]
3.3 result = matrix1 * matrix2.
4.4 DDA(400 + result.matrix[0][0], 400 - result.matrix[0][1],
400 + result.matrix[1,0], 400 - result.matrix[1][1])
4. If (flag == 2)
4.1. Scf rad = angle * 3.14 / 180.
4.2. Scf R[3][3] = {{cos(rad), sin(rad), 0},
{-sin(rad), cos(rad), 0},
{x * (1-cos(rad)) + y * sin(rad), y * (1-cos(rad)
- x * sin(rad)}}

4.3 for i ← 1 to 3 do
  4.3.1 for j ← 1 to 3 do
    4.3.1.1 matrix2.matrix[i][j] = R0[i][j]
4.4 result = matrix1 * matrix2.
4.5 DDA(400+result.matrix[0][0], 400-result.matrix[0][1], 400+result.matrix[1][0], 400-result.matrix[1][1])

{

Algorithm DDA(x_1, y_1, x_2, y_2) {
1. if (abs($x_2 - x_1$) > abs($y_2 - y_1$))
   1.1 length = abs($x_2 - x_1$)
2. else length = abs($y_2 - y_1$)
3. $x_1 = (x_2 - x_1)$ / length
4. $y_1 = (y_2 - y_1)$ / length
5. $x_1 = x_1 + 0.5 * x_1$
6. $y_1 = y_1 + 0.5 * y_1$
7. i = 1
8. while(i <= length)
   8.1 &pixel(round(x_1), round(y_1))
   8.2 x_1 += x_1
   8.3 y_1 += y_1
   8.4 i = i++

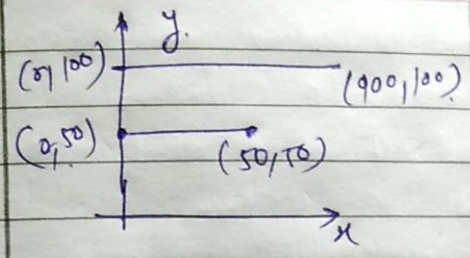
#	Test cases	Actual output	Expected o/p	result
1.	input: $t_x = 100$, $t_y = 100$ $x_1 = 0$, $y_1 = 0$ $x_2 = 50$, $y_2 = 50$.			pass
2.	rotation. $x_1 = 0$, $y_1 = 0$ $x_2 = 50$, $y_2 = 50$ $\theta = -90$.			pass

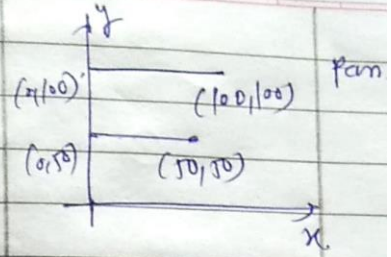
1) scaling

$x_1=0, y_1=50$

$x_2=50, y_2=50$

$S_x=2, S_y=2$





Pan

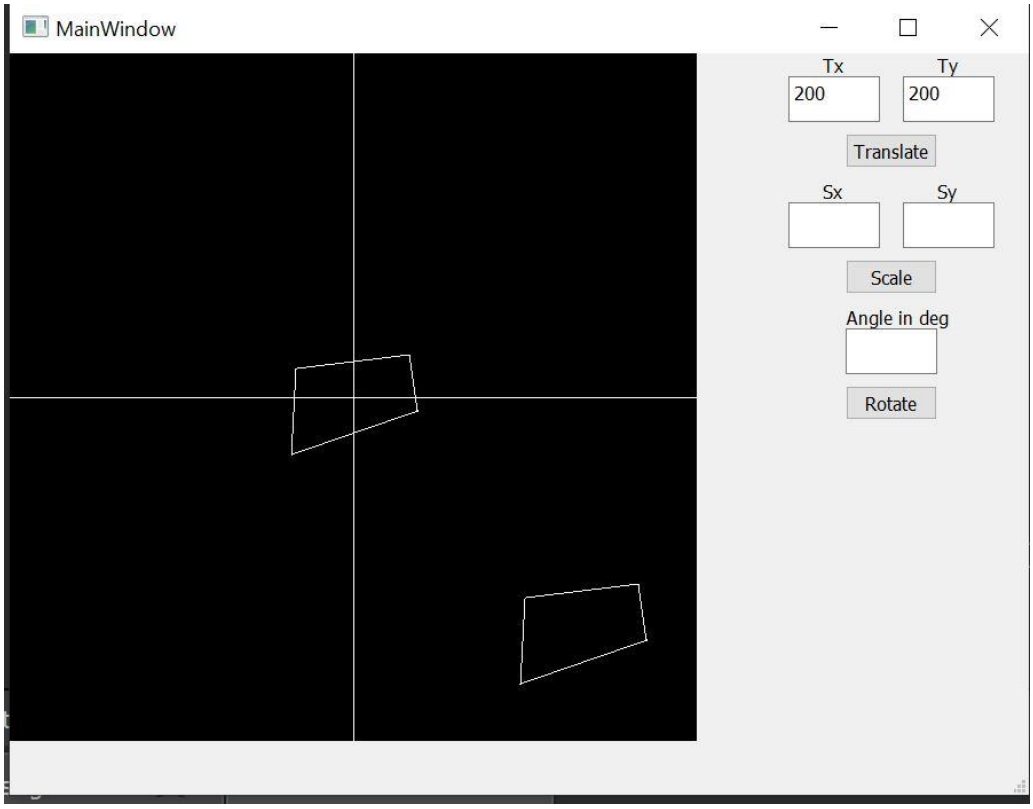
Time complexity of Algorithm: $O(4^3)$

Conclusion

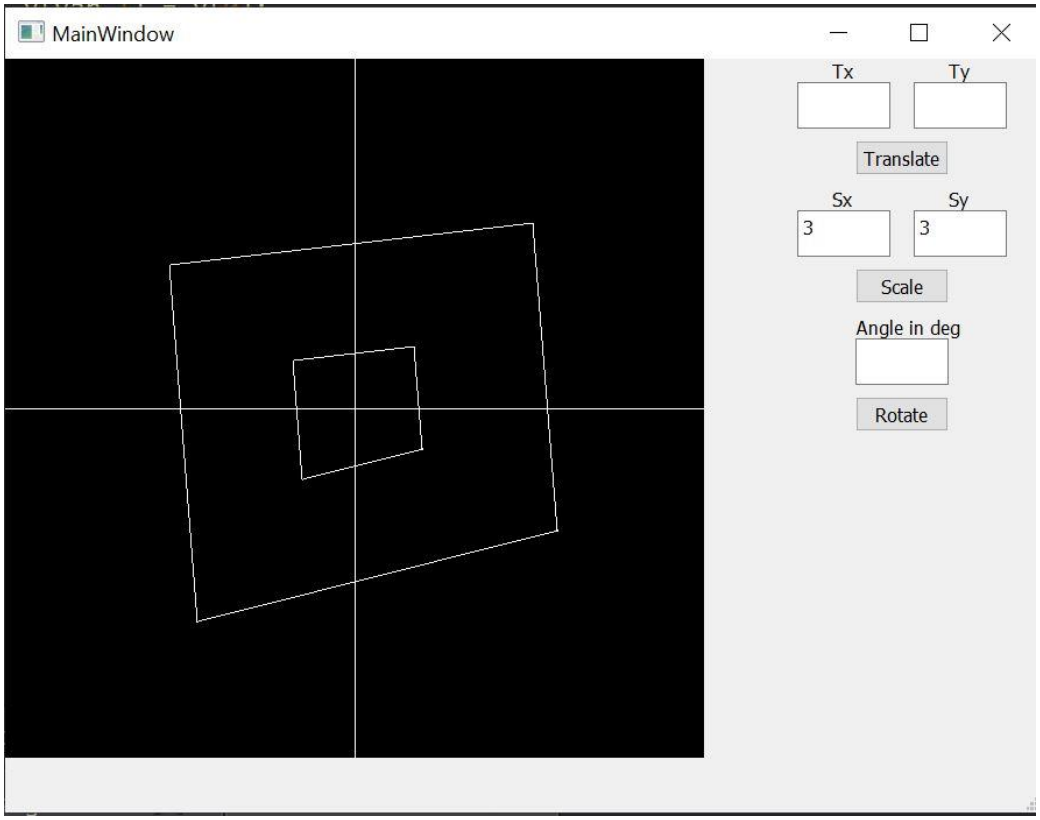
We learnt about the 2D transformations & also learnt how to apply operator overloading to the rotation, scaling translation using matrix multiplication.

Output and Explanations:

Translation: The polygon (4 – sides) which in initially drawn at almost centre of console is translated by 200 units in x-direction and by 200 units in y-direction. Since in screen coordinate system +y is in the downword direction and +x is in the right direction, translated object appears at bottom right side of console window.



Scaling: The polygon (4 - sided) is scaled by 4 units in both x and y direction due to which expands and eventually gets bigger than original.



Rotation: The four sided polygon is rotated by an angle 180 int x-y plane. The rotated polygon is exactly opposite to the original one (due to 180 degree turn).