**Practical No 7.A**

**Implementation of 2D transformations.**

**Aim: Write a program to implement a translation, rotation & scaling.**

**Theory:**

Transformation means changing some graphics into something else by applying rules. We can have various types of transformations such as translation, scaling up or down, rotation, shearing, etc. When a transformation takes place on a 2D plane, it is called 2D transformation. Transformations play an important role in computer graphics to reposition the graphics on the screen and change their size or orientation.

**Homogeneous Coordinates**

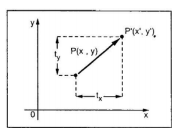
To perform a sequence of transformation such as translation followed by rotation and scaling, we need to follow a sequential process:

* Translate the coordinates
* Rotate the translated coordinates, and then
* Scale the rotated coordinates to complete the composite transformation

To shorten this process, we have to use 3×3 transformation matrix instead of 2×2 transformation matrix. To convert a 2×2 matrix to 3×3 matrix, we have to add an extra dummy coordinate W. In this way, we can represent the point by 3 numbers instead of 2 numbers, which is called Homogeneous Coordinate system. In this system, we can represent all the transformation equations in matrix multiplication. Any Cartesian point P (X, Y) can be converted to homogeneous coordinates by P’ (Xh, Yh, h).

**Translation**

A translation moves an object to a different position on the screen. You can translate a point in 2D by adding translation coordinate (tx, ty) to the original coordinate (X, Y) to get the new coordinate (X’, Y’).

From the figure, we can write that

X’ = X + tx

Y’ = Y + ty

The pair (tx, ty) is called the translation vector or shift vector. The above equations can also be represented using the column vectors.

P=[X][Y] P=[X][Y] p' = [X′] [Y′] [X′] [Y′] T = [tx][ty][tx][ty]

We can write it as: P’ = P + T

**Rotation**

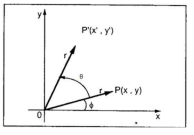
In rotation, we rotate the object at particular angle θ (theta) from its origin. From the following figure, we can see that the point P(X, Y) is located at angle φ from the horizontal X coordinate with distance r from the origin. Let us suppose you want to rotate it at the angle θ. After rotating it to a new location, you will get a new point P’ (X’, Y’).

Using standard trigonometric the original coordinate of point P(X, Y) can be represented as:

X=rcosϕ......(1)

Y=rsinϕ......(2)

Same way we can represent the point P’ (X’, Y’) as:

x′=rcos(ϕ+θ)=rcosϕcosθ−rsinϕsinθ.......(3)

y′=rsin(ϕ+θ)=rcosϕsinθ+rsinϕcosθ.......(4)

Substituting equation (1) & (2) in (3) & (4) respectively, we will get

x′=xcosθ−ysinθ

y′=xsinθ+ycosθ

Representing the above equation in matrix form,

[X′Y′]=[XY][cosθ−sinθsinθcosθ]OR

P’ = P . R

Where R is the rotation matrix

R=[cosθ−sinθsinθcosθ]

The rotation angle can be positive and negative.

For positive rotation angle, we can use the above rotation matrix. However, for negative angle rotation, the matrix will change as shown below:

R=[cos(−θ)−sin(−θ)sin(−θ)cos(−θ)]

=[cosθsinθ−sinθcosθ](∵cos(−θ)=cosθandsin(−θ)=−sinθ)

**Scaling**

To change the size of an object, scaling transformation is used. In the scaling process, you either expand or compress the dimensions of the object. Scaling can be achieved by multiplying the original coordinates of the object with the scaling factor to get the desired result. Let us assume that the original coordinates are (X, Y), the scaling factors are (SX, SY), and the produced coordinates are (X’, Y’). This can be mathematically represented as shown below:

X' = X. SX and Y' = Y. SY

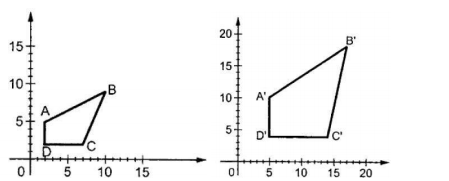
The scaling factor SX, SY scales the object in X and Y direction respectively. The above equations can also be represented in matrix form as below:

(X′Y′) =(XY)[Sx00Sy] (X′Y′) =(XY)[Sx00Sy]

OR

P’ = P. S

Where S is the scaling matrix. The scaling process is shown in the following figure.



If we provide values less than 1 to the scaling factor S, then we can reduce the size of the object. If we provide values greater than 1, then we can increase the size of the object.

**Conclusion: We have implemented translation, rotation & scaling.**

**Code:**

#include<graphics.h>

#include<conio.h>

#include<iostream.h>

#include<math.h>

#include<stdlib.h>

void main() {

clrscr();

int tx,ty,c,t,i,j,k,sx,sy, x1,x2,y1,y2;

float tx1;

x1 = y2 = 150;

x2 = y1 = 200; int x=1;

while(x>0) {

int gd = DETECT, gm;

initgraph(&gd, &gm,"C:\\TURBOC3\\BGI");

rectangle(x1,y1,x2,y2);

cout<<"1: Translation \n2: Rotation \n3: Scaling \n4: Exit";

cout<<"\nEnter your choice :";

cin>>c;

switch(c) {

case 1: cout<<"Enter tx & ty :";

cin>>tx>>ty;

rectangle(x1+tx, y1+ty, x2+tx, y2+ty);

break;

case 2 : int xx1, yy1, xx2, yy2, xx3, yy3, xx4, yy4;

int ax1, ay1, ax2, ay2, ax3, ay3, ax4, ay4;

int refx, refy;

cout<<"Enter angle :";

cin>>tx1;

tx1=tx1\*(3.14/180);

refx = refy = 150;

xx1 = yy1 = yy2 = xx4 = 150;

xx2 = xx3 = yy3 = yy4 = 200;

ax1 = refy +(xx1-refx)\* cos(tx1)-(yy1-refy)\*sin(tx1);

ay1 = refy +(xx1-refx)\* sin(tx1)+(yy1-refy)\*cos(tx1);

ax2 = refy +(xx2-refx)\* cos(tx1)-(yy2-refy)\*sin(tx1);

ay2 = refy +(xx2-refx)\* sin(tx1)+(yy2-refy)\*cos(tx1);

ax3 = refy +(xx3-refx)\* cos(tx1)-(yy3-refy)\*sin(tx1);

ay3 = refy +(xx3-refx)\* sin(tx1)+(yy3-refy)\*cos(tx1);

ax4 = refy +(xx4-refx)\* cos(tx1)-(yy4-refy)\*sin(tx1);

ay4 = refy +(xx4-refx)\* sin(tx1)+(yy4-refy)\*cos(tx1);

line(ax1,ay1, ax2, ay2);

line(ax2,ay2, ax3, ay3);

line(ax3,ay3, ax4, ay4);

line(ax4,ay4, ax1, ay1);

break;

case 3 : cout<<"Enter sx & sy :";

cin>>sx>>sy;

rectangle(x1, y2, x2\*sx, y1\*sy);

break;

case 4 : x=4;

exit(4);

break;

default :

cout<<"Invalid choice";

break;

closegraph(); }

getch();

clrscr(); }

getch(); }

**Output:**

