Assignment No- 06

(By Shashank Kapadnis 23310)

Aim:

2D Transformation.

Problem Statement:

Implement following 2D transformations on the object with respect to axis: –

i) Scaling ii) Rotation about arbitrary point iii) Reflection

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.

Homogenous Coordinates

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

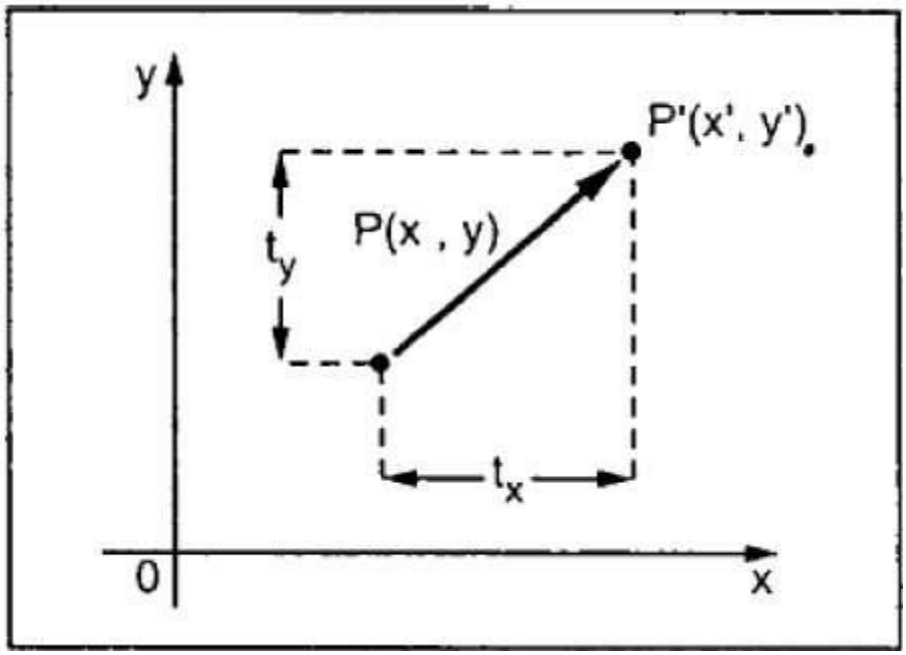
- Translate the coordinates,
- Rotate the translated coordinates, and then
- Scale the rotated coordinates to complete the composite transformation.

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 **Homogenous Coordinate** system. In this system, we can represent all the transformation equations in matrix multiplication. Any Cartesian point PX,YX,Y can be converted to homogenous coordinates by P' (X_h, Y_h, 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 (t_x, t_y) to the original coordinate X,YX,Y to get the new coordinate $X′,Y′X′,Y′$.



From the above figure, you can write that −

$$X' = X + t_x$$

$$Y' = Y + t_y$$

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

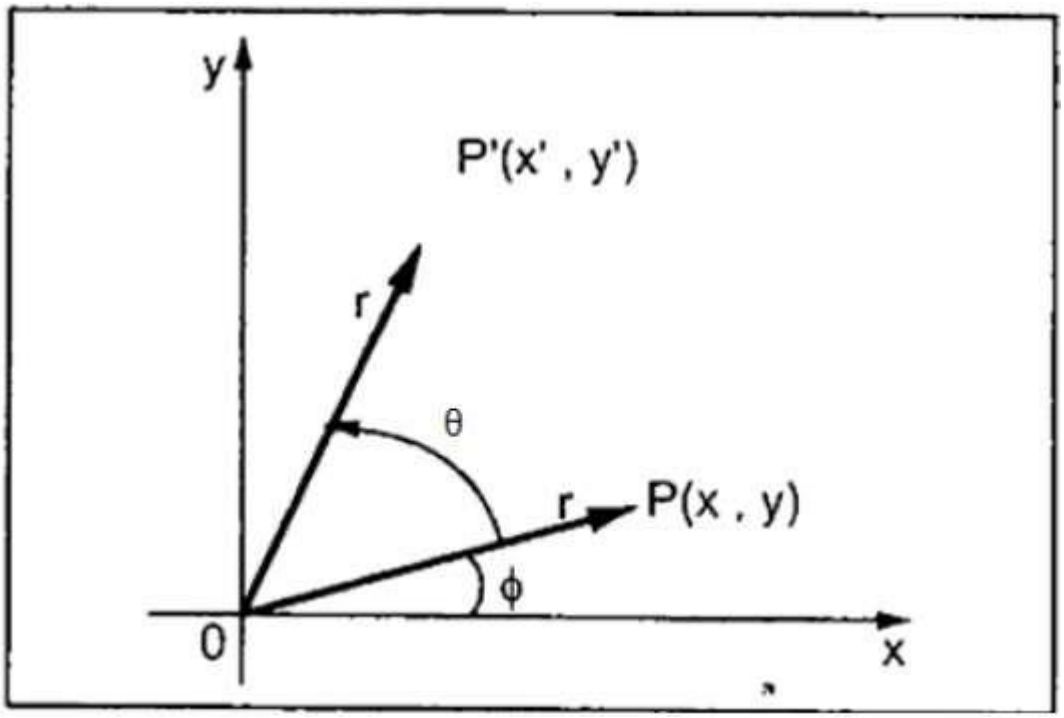
$$P=[X][Y]P=[X][Y]\ p' = [X'][Y'][X'][Y']T = [t_x][t_y][tx][ty]$$

We can write it as −

$$P' = P + T$$

Rotation

In rotation, we rotate the object at particular angle θ thetatheta from its origin. From the following figure, we can see that the point PX,YX,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′X′,Y′.



Using standard trigonometric the original coordinate of point PX,YX,Y can be represented as −

$$X=r\cos\phi.....(1)X=rcos\phi.....(1)$$

$$Y=r\sin\phi.....(2)Y=rsin\phi.....(2)$$

Same way we can represent the point P' $X',Y'X′,Y′$ as −

$$x'=r\cos(\phi+\theta)=r\cos\phi\cos\theta-r\sin\phi\sin\theta.....(3)x′=rcos(ϕ+θ)=rcosϕcosθ−rsinϕsinθ.....(3)$$

$$y'=r\sin(\phi+\theta)=r\cos\phi\sin\theta+r\sin\phi\cos\theta.....(4)y′=rsin(ϕ+θ)=rcosϕsinθ+rsinϕcosθ.....(4)$$

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

$$x'=x\cos\theta-y\sin\theta x′=xcosθ−ysinθ$$

$$y'=x\sin\theta+y\cos\theta y′=xsinθ+ycosθ$$

Representing the above equation in matrix form,

$$[X'Y']=[XY][\cos\theta-\sin\theta\sin\theta\cos\theta]OR[X′Y′]=[XY][cosθsinθ−sinθcosθ]OR$$

P' = P . R

Where R is the rotation matrix

$$R=[\cos\theta-\sin\theta\sin\theta\cos\theta]R=[cosθsinθ−sinθcosθ]$$

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 −

$$R=[\cos(-\theta)-\sin(-\theta)\sin(-\theta)\cos(-\theta)]R=[cos(−θ)sin(−θ)−sin(−θ)cos(−θ)]$$

$$=[\cos\theta\sin\theta-\sin\theta\cos\theta](\because\cos(-\theta)=\cos\theta and \sin(-\theta)=-\sin\theta)=[cosθ−sinθsinθcosθ](∵cos(−θ)=cosθandsin(−θ)=−sinθ)$$

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,YX,Y, the scaling factors are (S_x, S_y), and the produced coordinates are $X′,Y′X′,Y′$. This can be mathematically represented as shown below −

$$\mathbf{X' = X \cdot S_x \text{ and } Y' = Y \cdot S_Y}$$

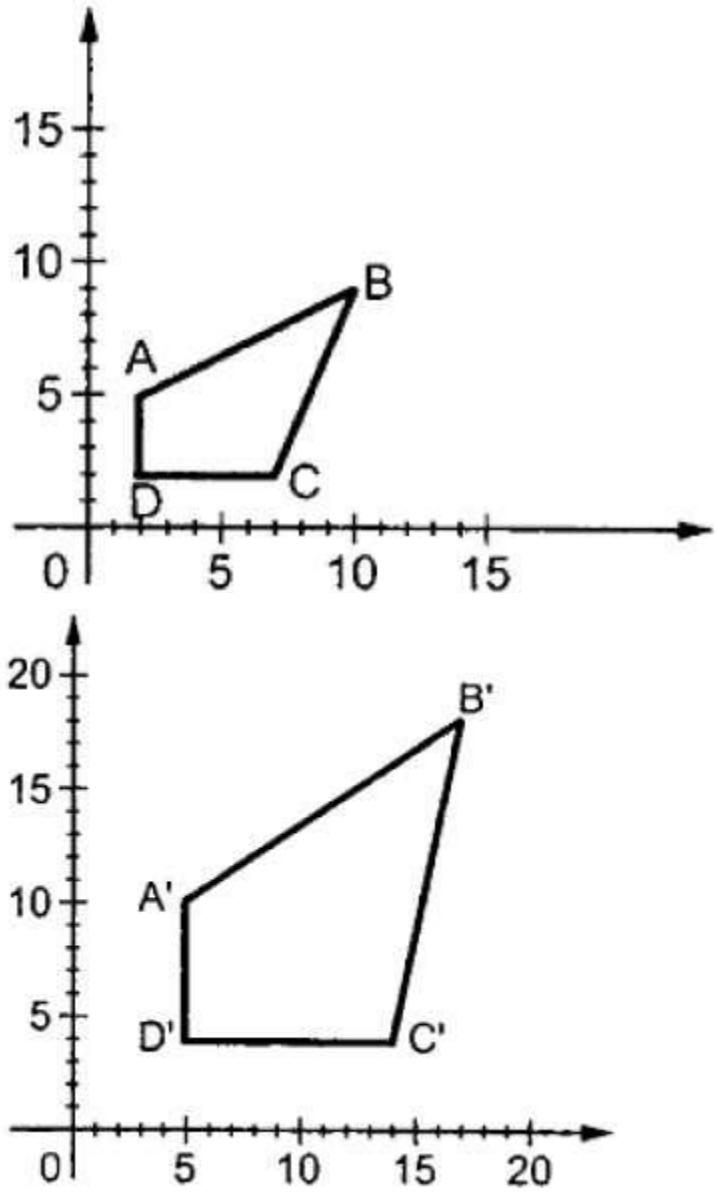
The scaling factor S_x, S_Y scales the object in X and Y direction respectively. The above equations can also be represented in matrix form as below −

$$(X′Y′)=(XY)[S_x00S_y](X′Y′)=(XY)[Sx00Sy]$$

OR

$$\mathbf{P' = P \cdot S}$$

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

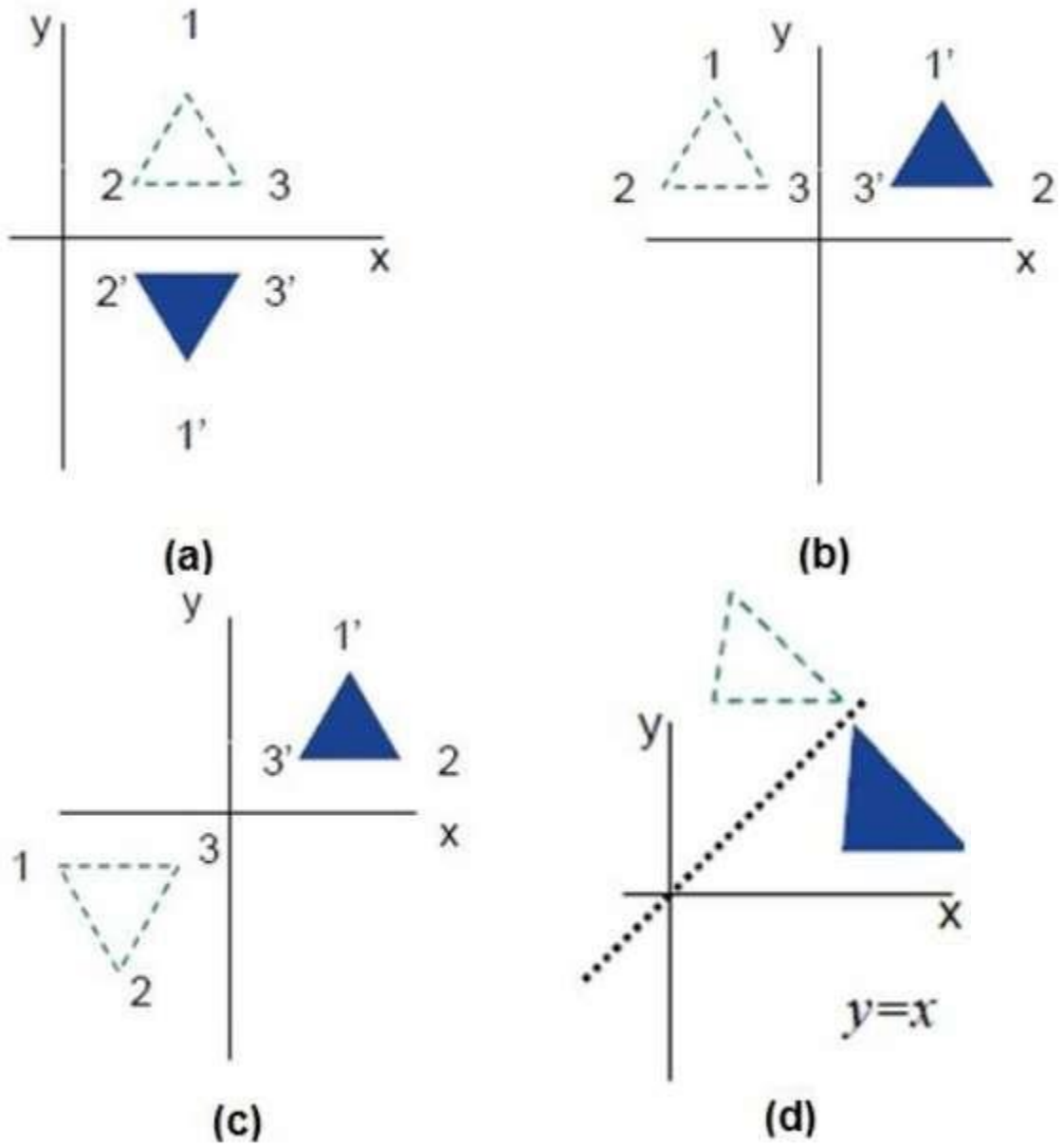


If we provide values less than 1 to the scaling factor S, then we can reduce the size of the object. If we provide values greater than 1, then we can increase the size of the object.

Reflection

Reflection is the mirror image of original object. In other words, we can say that it is a rotation operation with 180°. In reflection transformation, the size of the object does not change.

The following figures show reflections with respect to X and Y axes, and about the origin respectively.

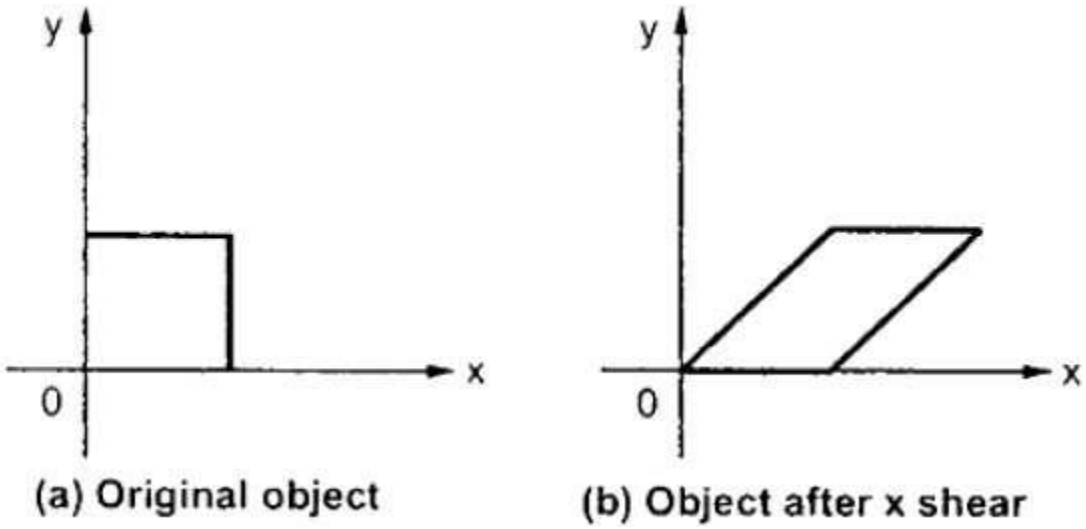


Shear

A transformation that slants the shape of an object is called the shear transformation. There are two shear transformations **X-Shear** and **Y-Shear**. One shifts X coordinates values and other shifts Y coordinate values. However; in both the cases only one coordinate changes its coordinates and other preserves its values. Shearing is also termed as **Skewing**.

X-Shear

The X-Shear preserves the Y coordinate and changes are made to X coordinates, which causes the vertical lines to tilt right or left as shown in below figure.



(a) Original object (b) Object after x shear

The transformation matrix for X-Shear can be represented as −

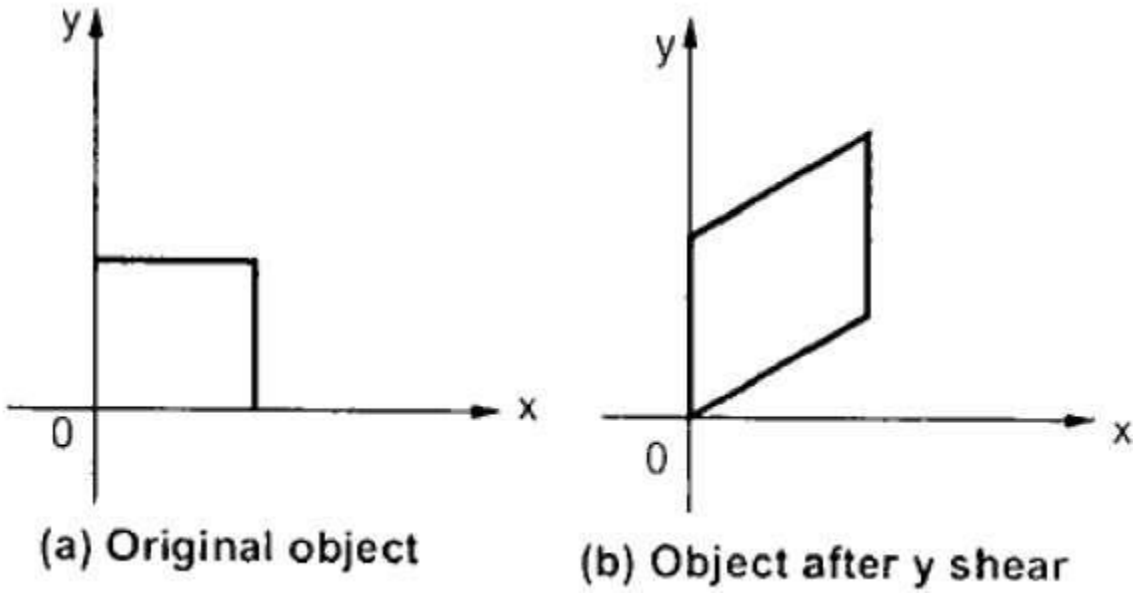
$$X_{sh}=\begin{bmatrix}1 & shx & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix} \quad X_{sh}=[1shx0010001]$$

$$Y' = Y + Sh_y \cdot X$$

$$X' = X$$

Y-Shear

The Y-Shear preserves the X coordinates and changes the Y coordinates which causes the horizontal lines to transform into lines which slopes up or down as shown in the following figure.



(a) Original object (b) Object after y shear

The Y-Shear can be represented in matrix from as −

$$Y_{sh}\begin{bmatrix}1 & 0 & 0\\ shy & 1 & 0\\ 0 & 0 & 1\end{bmatrix} \quad Y_{sh}[100shy10001]$$

$$X' = X + Sh_x \cdot Y$$

Y' = Y

Composite Transformation

If a transformation of the plane T1 is followed by a second plane transformation T2, then the result itself may be represented by a single transformation T which is the composition of T1 and T2 taken in that order. This is written as T = T1·T2.

Composite transformation can be achieved by concatenation of transformation matrices to obtain a combined transformation matrix.

A combined matrix −

[T][X] = [X] [T1] [T2] [T3] [T4] …. [Tn]

Where [Ti] is any combination of

- Translation
- Scaling
- Shearing
- Rotation
- Reflection

The change in the order of transformation would lead to different results, as in general matrix multiplication is not cumulative, that is [A] . [B] ≠ [B] . [A] and the order of multiplication. The basic purpose of composing transformations is to gain efficiency by applying a single composed transformation to a point, rather than applying a series of transformation, one after another.

For example, to rotate an object about an arbitrary point (X_p, Y_p), we have to carry out three steps −

- Translate point (X_p, Y_p) to the origin.
- Rotate it about the origin.
- Finally, translate the center of rotation back where it belonged.

Algorithm:

Scaling:

1. Make a 2x2 scaling matrix S as:

   S_x 0
   0 S_y
2. For each point of the polygon.
   (i) Make a 2x1 matrix P, where P[0][0] equals to x coordinate of the point and P[1][0] equals to y coordinate of the point.
   (ii) Multiply scaling matrix S with point matrix P to get the new coordinate.
3. Draw the polygon using new coordinates.

Code:

```
#include<GL/glut.h>
#include<stdio.h>

#include<math.h>

static int flag;

int length, xi, yi, choice;
double angle, ET[3][3], ETResult[3][3];
double Rh[4][4], RhResult[4][4];

//------------------DRAW-------------//

void drawET(double ET[3][3])
{
	int i;

	glBegin(GL_LINE_LOOP);
	for(i=0;i<3;i++)
	{
	 glVertex2i(ET[i][0],ET[i][1]);
	}
	glEnd();
}

void drawR(double Rh[4][4])
{
```

```
        int i;

        glBegin(GL_LINE_LOOP);
        for(i=0;i<4;i++)
        {
         glVertex2i(Rh[i][0],Rh[i][1]);
        }
        glEnd();
}
//----------------------------Display---------------------//

void Display()
{
    glClearColor(0,0,0,0);
    glClear(GL_COLOR_BUFFER_BIT);

    glLoadIdentity();
    gluOrtho2D(-320,320,-240,240);//note

    glColor3f(1,1,1);
    glBegin(GL_LINES);
        glVertex2d(-320,0);
        glVertex2d(320,0);
        glVertex2d(0,-240);
        glVertex2d(0,240);
    glEnd();

    glColor3f(1,0,0);
```

```
    if(flag == 0)
     drawET(ET);
    else if(flag == 1)
     drawR(Rh);

    glFlush();
}



//---------MULTIPLY--------------//


void mult3X3(double ET[3][3],double temp[3][3])
{
 double sum;
 int i,j,k;
    for(i=0;i<3;i++)
    {
       for(j=0;j<3;j++)
        {
         sum=0;
            for(k=0;k<3;k++)
            {
               sum=sum+ET[i][k]*temp[k][j];
            }
            ETResult[i][j]=sum;
        }
    }
}
```

```
void mult4X4(double Rh[4][4],double temp[4][4])
{
 double sum;
    int i,j,k;
    for(i=0;i<4;i++)
    {
        for(j=0;j<4;j++)
        {
         sum=0;
            for(k=0;k<4;k++)
            {
                sum=sum+Rh[i][k]*temp[k][j];
   }
            RhResult[i][j]=sum;
        }
 }
}

//----------Translation----------//

void translationET()
{
 double tx,ty,temp[3][3];

    printf("\nTranslating Equilateral triangle");
    printf("\nEnter Tx: ");
    scanf("%lf",&tx);
```

```
    printf("\nEnter Ty: ");
    scanf("%lf",&ty);

 temp[3][3]={0};
    temp[0][0]=1;
    temp[1][1]=1;
    temp[2][2]=1;
    temp[2][0]=tx;
    temp[2][1]=ty;

    mult3X3(ET,temp);
    glColor3f(0.0,1.0,0.0);
    drawET(ETResult);
}

void translationRh()
{
 double tx,ty,temp[4][4];

    printf("\nTranslating Rhombus");
    printf("\nEnter Tx: ");
    scanf("%lf",&tx);
    printf("\nEnter Ty: ");
 scanf("%lf",&ty);

    temp[4][4]={0};
    temp[0][0]=1;
    temp[1][1]=1;
```

```
    temp[2][2]=1;
 temp[3][3]=1;
 temp[3][0]=tx;
    temp[3][1]=ty;

    mult4X4(Rh,temp);
    glColor3f(0.0,1.0,0.0);
    drawR(RhResult);
}

//------------Rotation------------//

void rotationET()
{
 double rx,ry,angle, temp[3][3];

    printf("\n**ROTATION**\n");
    printf("\nArbitrary Point (x,y) : ");
    scanf("%lf %lf",&rx,&ry);
    printf("\nAngle (in degrees) : ");
    scanf("%lf",&angle);

    angle=angle*(M_PI/180);

    temp[3][3]={0};
    temp[0][0]=cos(angle);
    temp[0][1]=sin(angle);
    temp[1][0]=-sin(angle);
```

```
    temp[1][1]=cos(angle);
    temp[2][0]=(-(rx*cos(angle))+(ry*sin(angle))+rx);
    temp[2][1]=(-(rx*sin(angle))-(ry*cos(angle))+ry);
    temp[2][2]=1;

    mult3X3(ET,temp);
    glColor3f(0.0,1.0,0.0);
    drawET(ETResult);
}

void rotationRh()
{
 double rx,ry,angle, temp[4][4];

    printf("\nRotating Rhombus");
    printf("\nArbitrary Point (x,y): ");
    scanf("%lf %lf",&rx,&ry);
    printf("\nAngle (in degree): ");
    scanf("%lf",&angle);

    angle=angle*(M_PI/180);

    temp[4][4]={0};
    temp[0][0]=cos(angle);
    temp[0][1]=sin(angle);
    temp[1][0]=-sin(angle);
    temp[1][1]=cos(angle);
    temp[2][2]=1;
```

```
    temp[3][0]=(-(rx*cos(angle))+(ry*sin(angle))+rx);
    temp[3][1]=(-(rx*sin(angle))-(ry*cos(angle))+ry);
    temp[3][3]=1;

    mult4X4(Rh,temp);
    glColor3f(0.0,1.0,0.0);
    drawR(RhResult);
}

//----------Scaling----------//

void scaleET()
{
 double sx,sy, temp[3][3];

    printf("\nScaling Equilateral triangle");
    printf("\nSx: ");
    scanf("%lf",&sx);
    printf("\nSy: ");
    scanf("%lf",&sy);

    temp[3][3]={0};
 temp[0][0]=sx;
 temp[1][1]=sy;
 temp[2][2]=1;

    mult3X3(ET,temp);
    glColor3f(1.0,1.0,0.0);
```

```
    drawET(ETResult);
}

void scaleRh()
{
    double sx,sy,temp[4][4];

    printf("\nScaling Rhombus");
    printf("\nSx: ");
    scanf("%lf",&sx);
    printf("\nSy: ");
    scanf("%lf",&sy);

    temp[4][4]={0};
 temp[0][0]=sx;
 temp[1][1]=sy;
 temp[2][2]=1;
 temp[3][3]=1;

    mult4X4(Rh,temp);
    glColor3f(1.0,1.0,0.0);
    drawR(RhResult);
}

//-----------Shearing----------//
void shearET()
{
 double xs,ys,temp[3][3];
```

```
  printf("\nShear Equilateral triangle");
  printf("\nPress 1: X - Shear");
  printf("\nPress 2: Y - Shear");
  printf("\nEnter your Choice: ");
  scanf("%d",&choice);

  temp[3][3]={0};

  switch(choice)
  {
   case 1: printf("\nX-shear value: ");
     scanf("%lf",&xs);
              temp[0][0]=1;
              temp[1][0]=xs;
              temp[1][1]=1;
              temp[2][2]=1;
              break;
case 2: printf("\nY-shear value: ");
     scanf("%lf",&ys);
  temp[0][0]=1;
  temp[0][1]=ys;
  temp[1][1]=1;
  temp[2][2]=1;
              break;
  }

  mult3X3(ET,temp);
```

```
    glColor3f(1.0,1.0,0.0);
    drawET(ETResult);
}

void shearRh()
{
 double xs,ys,temp[4][4];

    printf("\nPress 1: X - Shear");
    printf("\nPress 2: Y - Shear");
    printf("\nEnter your Choice: ");
    scanf("%d",&choice);

    temp[4][4]={0};

    switch(choice)
    {
     case 1: printf("\nX-shear value: ");
                scanf("%lf",&xs);
                temp[0][0]=1;
                temp[1][0]=xs;
                temp[1][1]=1;
                temp[2][2]=1;
                temp[3][3]=1;
                break;
  case 2: printf("\nY-shear value: ");
        scanf("%lf",&ys);
                temp[0][0]=1;
```

```
                temp[0][1]=ys;
                temp[1][1]=1;
                temp[2][2]=1;
                temp[3][3]=1;
                break;
 }
    mult4X4(Rh,temp);
    glColor3f(0.0,1.0,0.0);
    drawR(RhResult);
}

//----------MENU--------------//
void Menu(int item)
{
 switch(item)
    {
        case 1: if(choice==1)
                 translationET();
                else
                 translationRh();

                break;

        case 2: if(choice==1)
                 rotationET();
                else
                 rotationRh();
```

```
                break;
        case 3: if(choice==1)
                 scaleET();
                else
                 scaleRh();

                break;
        case 4: if(choice==1)
                {
                 shearET();
                }
                else
                {
                 shearRh();
                }
                break;
        case 5:
          exit(0);
          break;
    }
}

//----------MAIN--------------------//

int main(int argc,char** argv)
{
    printf("\n*MENU");
    printf("\n1. To draw Equilateral Triangle");
```

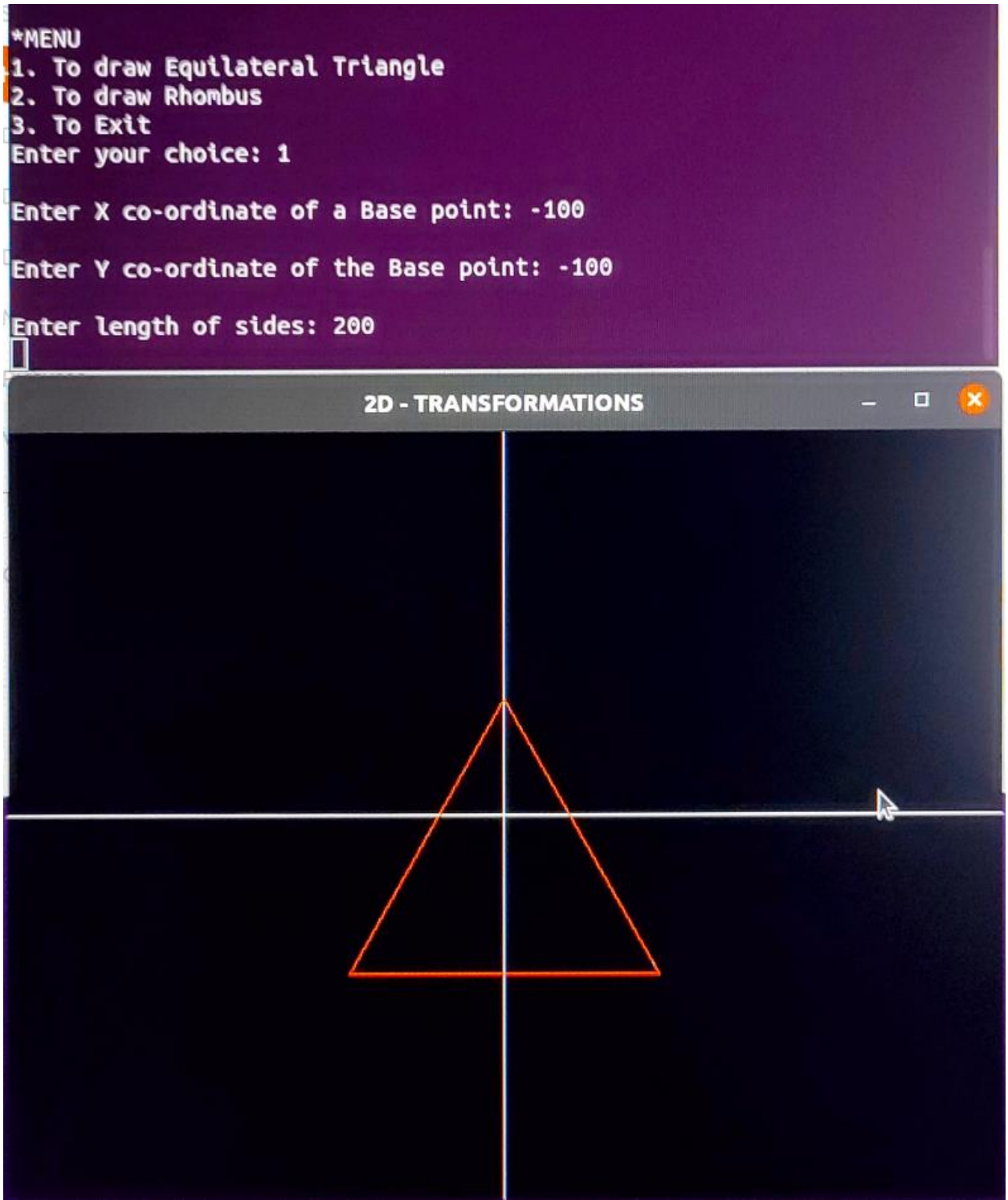
```
printf("\n2. To draw Rhombus");
printf("\n3. To Exit");
printf("\nEnter your choice: ");
scanf("%d",&choice);

switch(choice)
{
 int i, j;

   case 1:
     flag = 0;
     printf("\nEnter X co-ordinate of a Base point: ");
     scanf("%d",&xi);
     printf("\nEnter Y co-ordinate of the Base point: ");
      scanf("%d",&yi);
      printf("\nEnter length of sides: ");
      scanf("%d",&length);

      for(i=0;i<3;i++)
           {
            for(j=0;j<3;j++)
              {
                  ET[i][j]=1;
              }
           }
      ET[0][0]=xi;
      ET[0][1]=yi;
      ET[1][0]=xi+length;
```

```
  ET[1][1]=yi;
  ET[2][0]=length/2+xi;
  ET[2][1]=(sqrt(3)/2*length)+yi;
  break;

case 2:
  flag = 1;
  printf("\nEnter X co-ordinates of a Base point: ");
  scanf("%d",&xi);
  printf("\nEnter Y co-ordinates of the Base point: ");
  scanf("%d",&yi);
  printf("\nEnter length of sides: ");
  scanf("%d",&length);
  printf("\nEnter angle of Rhombus (in degrees): ");
  scanf("%lf",&angle);
  angle = angle * M_PI / 180;

  for(i=0;i<4;i++)
       {
        for(j=0;j<4;j++)
          {
              Rh[i][j]=1;
          }
       }
  Rh[0][0]=xi;
  Rh[0][1]=yi;
  Rh[1][0]=xi+length;
  Rh[1][1]=yi;
```

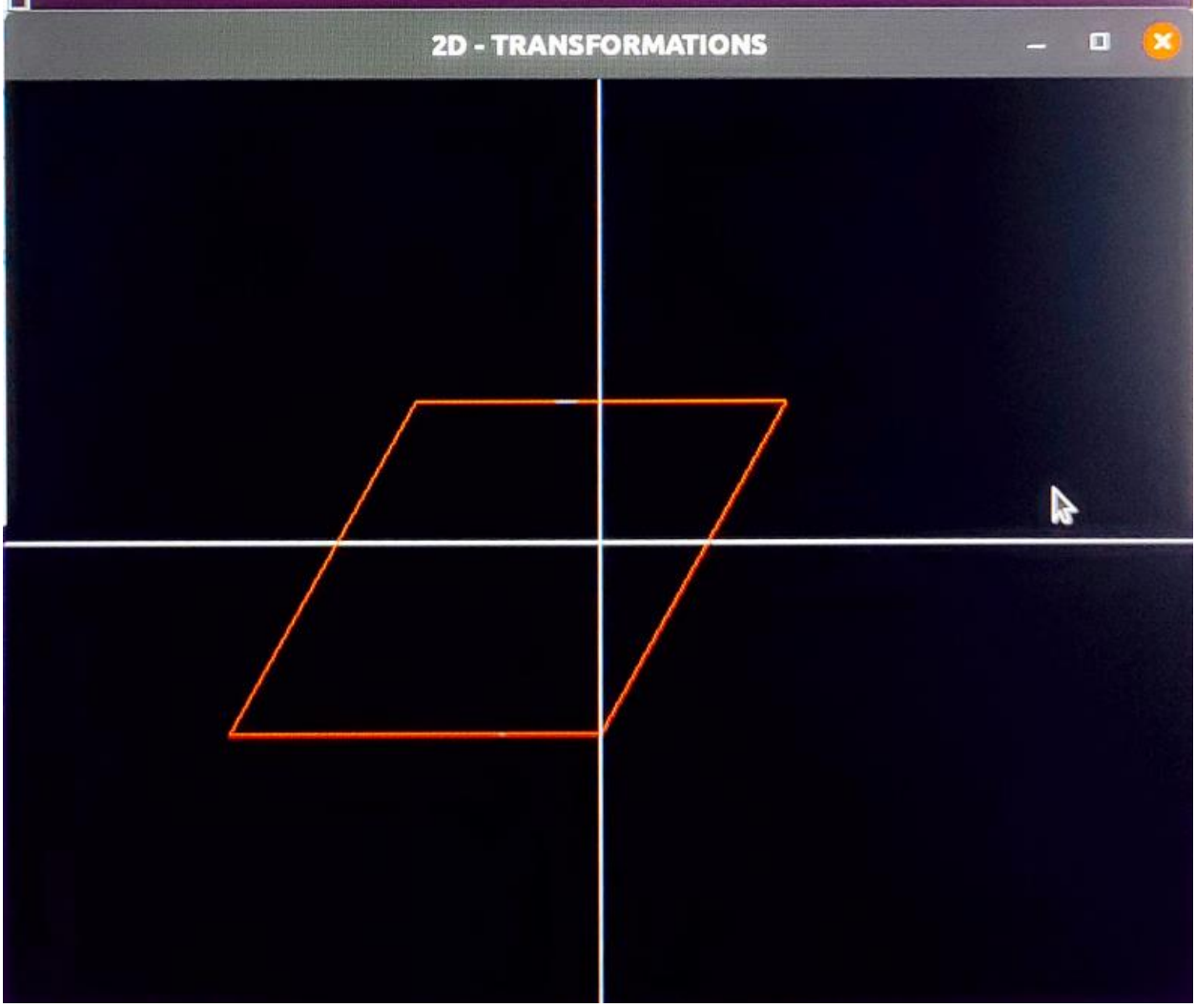
```
        Rh[2][0]=length+xi+length*cos(angle);
        Rh[2][1]=yi+length*sin(angle);
        Rh[3][0]=xi+length*cos(angle);
        Rh[3][1]=yi+length*sin(angle);
        break;

      case 3:
        exit(0);
        break;

      default:printf("\nInvalid Input!");
        break;
}

  glutInit(&argc,argv);
  glutInitDisplayMode(GLUT_SINGLE);
  glutInitWindowSize(640,480);
  glutInitWindowPosition(0,0);
  glutCreateWindow("2D - TRANSFORMATIONS");

  glutDisplayFunc(Display);

  glutCreateMenu(Menu);
   glutAddMenuEntry("1.Translation",1);
   glutAddMenuEntry("2.Rotation",2);
   glutAddMenuEntry("3.Scaling",3);
   glutAddMenuEntry("4.Shear",4);
   glutAddMenuEntry("5.EXIT",5);
```

```
    glutAttachMenu(GLUT_RIGHT_BUTTON);


    glutMainLoop();

    return 0;
}
```

Output:



```
*MENU
1. To draw Equilateral Triangle
2. To draw Rhombus
3. To Exit
Enter your choice: 2

Enter X co-ordinates of a Base point: -200

Enter Y co-ordinates of the Base point: -100

Enter length of sides: 200

Enter angle of Rhombus (in degrees): 60
```



Conclusion:

Successfully implemented 2D transformation .