PUNE INSTITUTE OF COMPUTER TECHNOLOGY

IT Department

CGL Assignment 6

Name: Suyash More **Class**:SE9

Roll No: 23148 **Batch**: G9

Title:

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 −

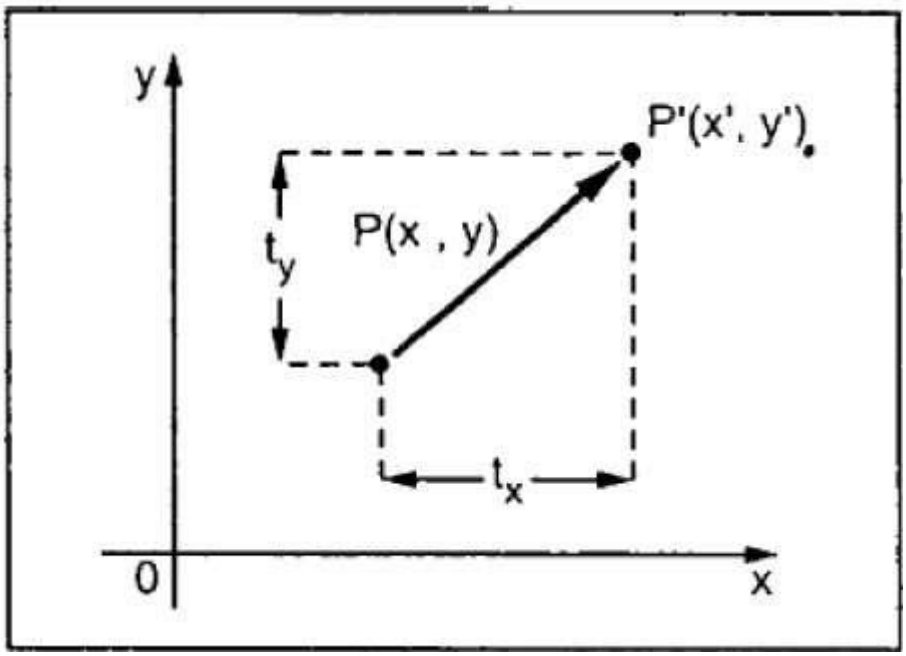
- Translate the coordinates,
- Rotate the translated coordinates, and then
- Scale the rotated coordinates to complete the composite transformation.

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 = [tx][ty][tx][ty]

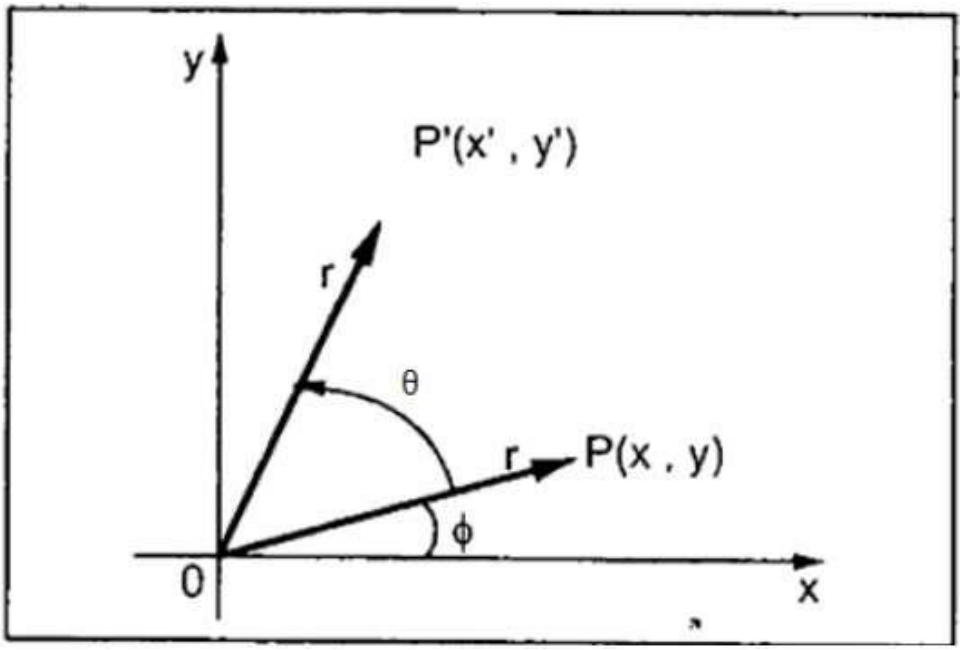
We can write it as −

$$P' = P + T$$

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 −

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

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

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 11 & 22 in 33 & 44 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 −

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

$$=[\cos\theta\sin\theta-\sin\theta\cos\theta](\because\cos(-\theta)=\cos\theta\text{and}\sin(-\theta)=-\sin\theta)=[\cos\theta-\sin\theta\sin\theta\cos\theta](\because\cos(-\theta)=\cos\theta\text{and}\sin(-\theta)=-\sin\theta)$$

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 −

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

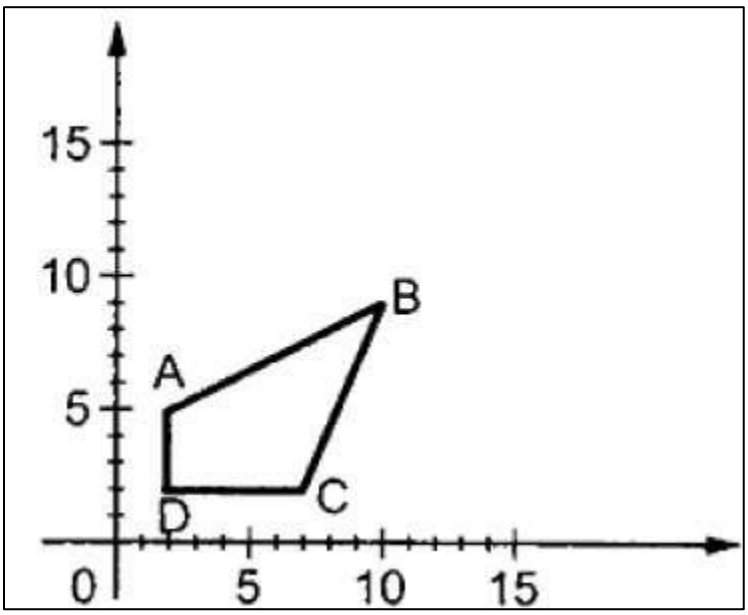
The scaling factor S_x, S_y scales the object in X and Y direction respectively. The above equations can also be represented in matrix form as below −

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

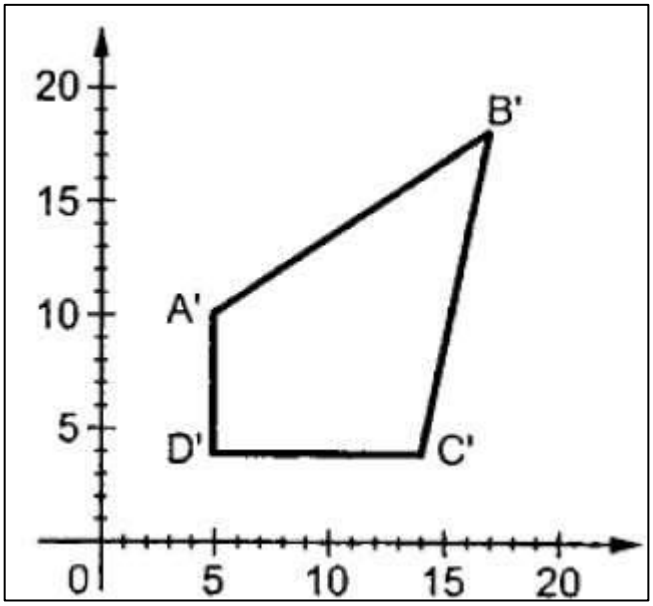
(OR)

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

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



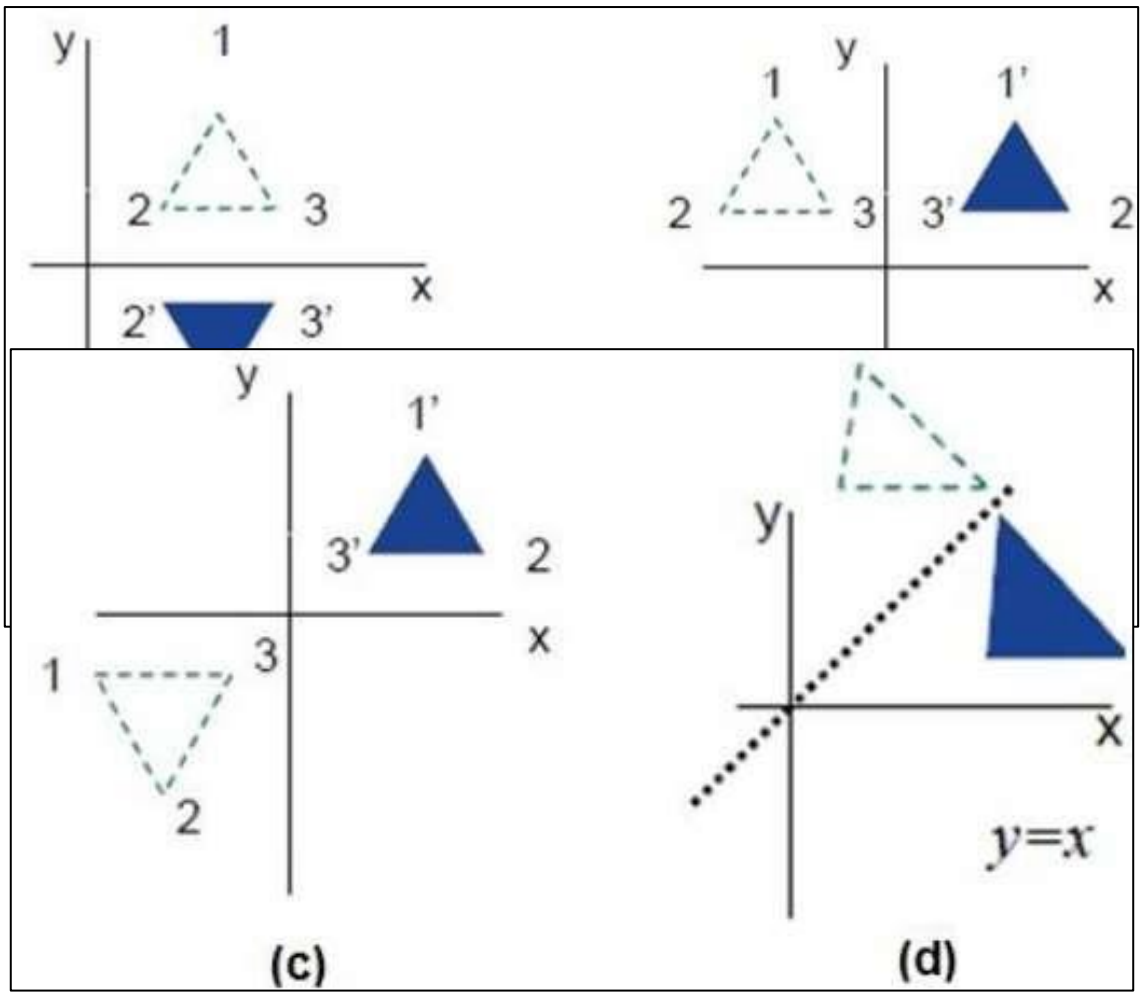
If we provide values less than 1 to the scaling factor S, then we can reduce the size of the object. If we provide values greater than 1, then we can increase the size of the object.



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.

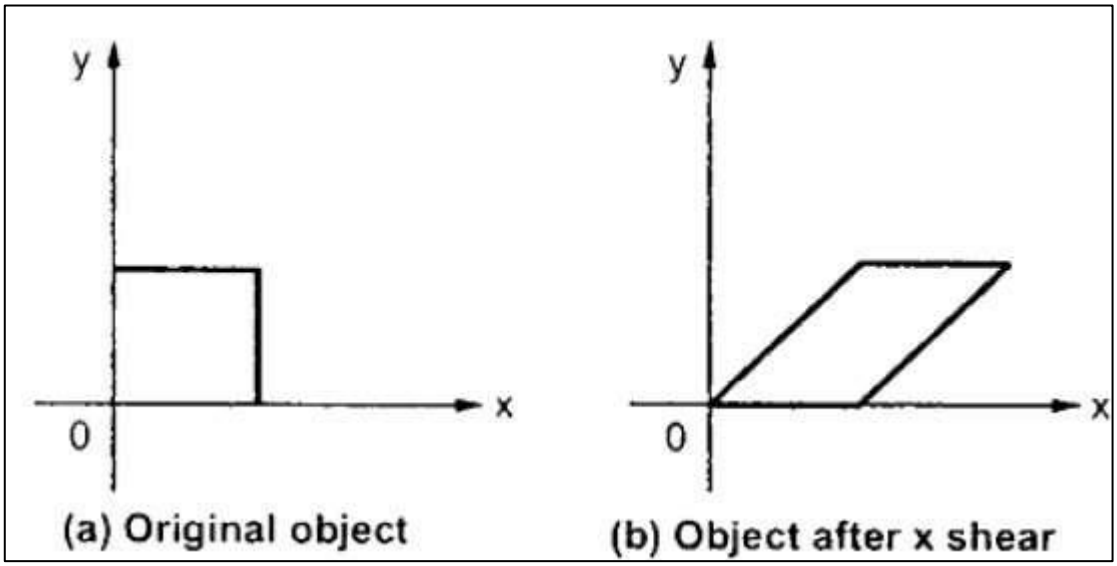


Shear :-

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**.

i) 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.



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

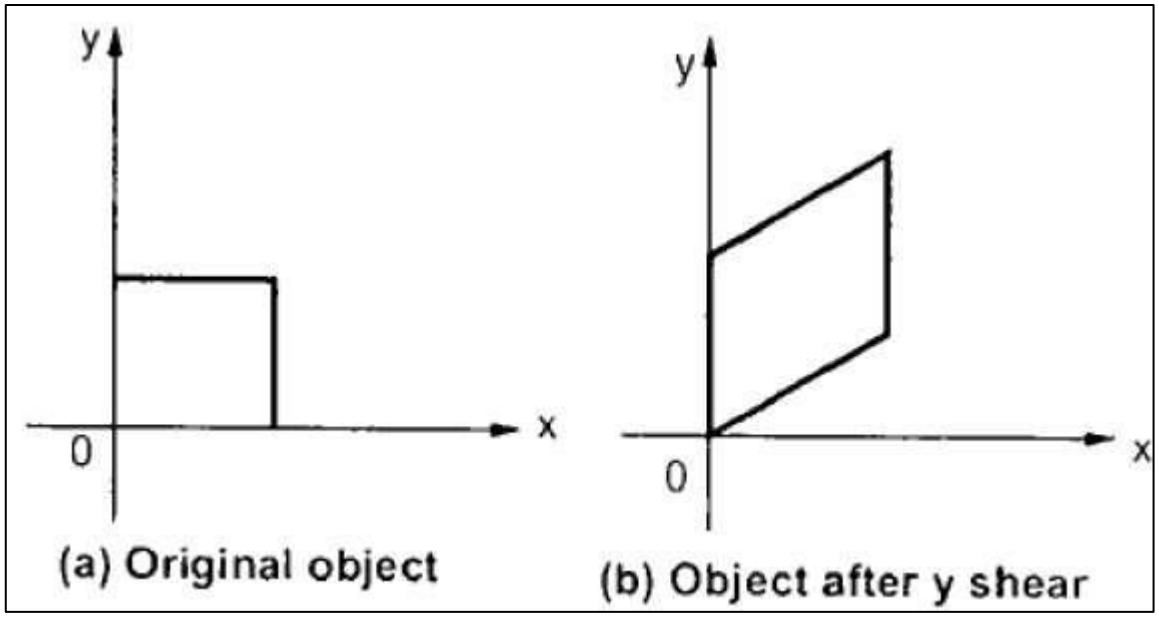
$$X_{sh}=\begin{bmatrix} 1 & 0 & 0 \\ shx & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} X_{sh}=\begin{bmatrix} 1 & shx & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

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

$$X' = X$$

ii) 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.



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

$$Y_{sh}\begin{bmatrix} 1 & sh_y & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad Y_{sh}[100sh_y10001]$$

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

$$Y' = Y$$

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.

Conclusion:

Successfully implemented 2D transformation .

Code :-

Rotation:

```
#include <GL/freeglut.h>
#include <GL/gl.h>
#include <iostream>
using namespace std;

struct Color//declare color stucture
{
	float r,g,b;
};

Color getPixelcolor(float x,float y)//get pixelcolor
{
	Color c;
	glReadPixels(x,y,1,1,GL_RGB,GL_FLOAT,&c);//get color in 'c'
	return c;//return c
}//end

void setPixelcolor(float x,float y)
{
	glBegin(GL_POINTS);//draw point
		glColor3f(1.0,0.0,0.0);//set point color to red
		glVertex2f(x,y);
	glEnd();
}//end

void floodfill(float x,float y)
{
	Color c = getPixelcolor(x,y);//gets color of current pixel
	Color old = {1.0,1.0,1.0};

	if(c.r == old.r && c.g == old.g && c.b == old.b)//if color of current pixel if white
	{
		setPixelcolor(x,y);//set pixel color to red
		floodfill(x+1,y);//call floodfill recursively for four-connected points
		floodfill(x,y+1);
		floodfill(x-1,y);
		floodfill(x,y-1);
	}
	return;
}//end

void render()
{
	glClearColor(1.0,1.0,1.0,0.0);//clear color to white
```

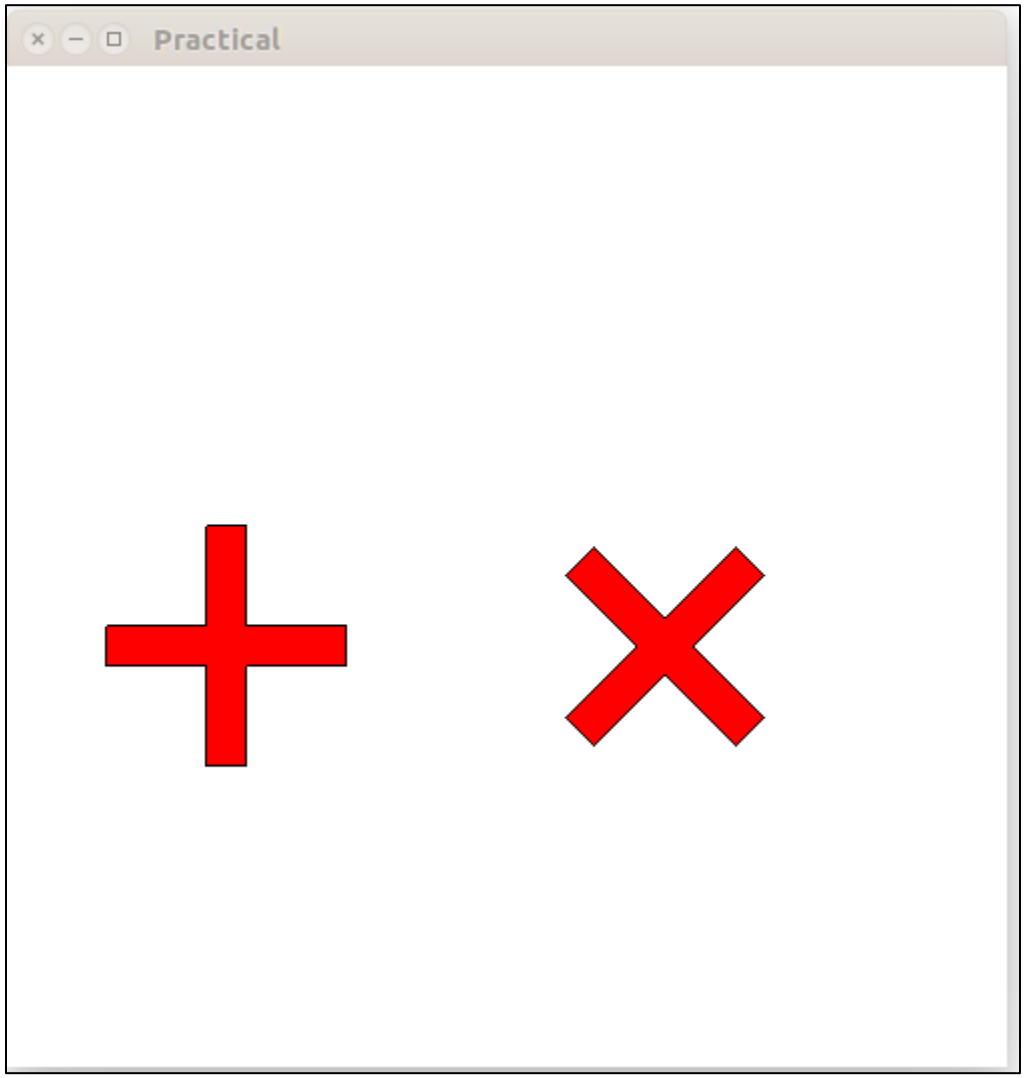
```
	glClear(GL_COLOR_BUFFER_BIT);//set color
	glMatrixMode(GL_PROJECTION);//set matrix mode
	glLoadIdentity();//load identity matrix
	gluOrtho2D(0,400,0,400);//sets axis length
	glFlush();//flush buffer and execute all command
}//end



void draw()//draw '+' Diagram
{
	glBegin(GL_LINES);
		glColor3f(0.0,0.0,0.0);//sets black color
		glVertex2d(100,150);
		glVertex2d(100,200);

		glVertex2d(100,200);
		glVertex2d(50,200);

		glVertex2d(50,200);
		glVertex2d(50,220);

		glVertex2d(50,220);
		glVertex2d(100,220);

		glVertex2d(100,220);
		glVertex2d(100,270);

		glVertex2d(100,270);
		glVertex2d(120,270);

		glVertex2d(120,270);
		glVertex2d(120,220);

		glVertex2d(120,220);
		glVertex2d(170,220);

		glVertex2d(170,220);
		glVertex2d(170,200);

		glVertex2d(170,200);
		glVertex2d(120,200);

		glVertex2d(120,200);
		glVertex2d(120,150);

		glVertex2d(120,150);
		glVertex2d(100,150);
```

```
    glEnd();//end
    floodfill(110,210);//fill '+' Diagram
    glFlush();//flush buffer and execute all command
}//end

void rotate()//draw '+' Diagram
{
    float y_tra = 210 - (320/1.41);//y-translation to have same level
    glBegin(GL_LINES);
        glColor3f(0.0,0.0,0.0);//sets black color
        glVertex2d(400+(-50/1.41),250/1.41 + y_tra);
        glVertex2d(400+(-100/1.41),300/1.41 + y_tra);

        glVertex2d(400+(-100/1.41),300/1.41 + y_tra);
        glVertex2d(400+(-150/1.41),(250/1.41) + y_tra);

        glVertex2d(400+(-150/1.41),(250/1.41) + y_tra);
        glVertex2d(400+(-170/1.41),270/1.41 + y_tra);

        glVertex2d(400+(-170/1.41),270/1.41 + y_tra);
        glVertex2d(400 + (-120/1.41),320/1.41 + y_tra);

        glVertex2d(400 + (-120/1.41),320/1.41 + y_tra);
        glVertex2d(400 +(-170/1.41),370/1.41 + y_tra);

        glVertex2d(400 +(-170/1.41),370/1.41 + y_tra);
        glVertex2d(400+(-150/1.41),390/1.41 + y_tra);

        glVertex2d(400+(-150/1.41),390/1.41 + y_tra);
        glVertex2d(400+(-100/1.41),340/1.41 + y_tra);

        glVertex2d(400+(-100/1.41),340/1.41 + y_tra);
        glVertex2d(400+(-50/1.41),390/1.41 + y_tra);

        glVertex2d(400+(-50/1.41),390/1.41 + y_tra);
        glVertex2d(400+(-30/1.41),370/1.41 + y_tra);

        glVertex2d(400+(-30/1.41),370/1.41 + y_tra);
        glVertex2d(400+(-80/1.41),320/1.41 + y_tra);

        glVertex2d(400+(-80/1.41),320/1.41 + y_tra);
        glVertex2d(400+(-30/1.41),270/1.41 + y_tra);

        glVertex2d(400+(-30/1.41),270/1.41 + y_tra);
        glVertex2d(400+(-50/1.41),250/1.41 + y_tra);

    glEnd();//end
    floodfill(400+(-100/1.41),320/1.41 + y_tra);//fill rotated Diagram
    glFlush();//flush buffer and execute all command
```

```
}//end

void mouse(int button,int state,int x,int y)//On only when menu is commented
{
	if(button == GLUT_LEFT_BUTTON && state == GLUT_UP)//if left button and up
	{
		render();//draw '+' diagram
		draw();
	}
	else if(button == GLUT_RIGHT_BUTTON && state == GLUT_UP)//if right button and up
	{
		render();//draw '+' and rotated diagram
		draw();
		rotate();
	}
}//end

int main(int argc,char **argv)//taking command line orguments
{
	int e;
	glutInit(&argc,argv);//initialise glut with libraries
	glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);//initalise mode
	glutInitWindowPosition(1000,200);//sets position of window
	glutInitWindowSize(400,400);//sets size of window
	glutCreateWindow("Practical");//create window
	glutMouseFunc(mouse);//activate mouse function(activated only when menu is commented)
	render();//call to function
	do
	{
		cout<<"*Menu = \n1 : Given Diagram\n2 : Rotated Diagram\n3 : Exit\n";//create menu
		cout<<"Enter Your Choice = ";
		cin>>e;
		switch(e)
		{
			case 1://Draw '+' Diagram
				render();//clear screen
				draw();
				break;

			case 2://Draw rotated '+' Diagram
				rotate();
				break;

			case 3://exit from program
				exit(1);
		}
	}while(e);

	glutMainLoop();//infinite loop untill user closes window
```

```
return 0;
}//end of program
```

Output :-



Scaling and Reflection :

```
#include <GL/freeglut.h>
#include <GL/gl.h>
```

```
#include <math.h>
#include <stdio.h>

struct Pt
{
	int x,y;
};

int numv,cnt;
bool inp;
Pt points[10];

void initGlobalVars(){
	inp=false;
	cnt=0;
	numv=0;
}

/* ---------------------- DDA LINE ALGORITHM ---------------------- */
void LineDDA(int x1,int y1,int x2,int y2)
{
	float dx,dy,incx,incy;
	float x,y;
	int steps,i;
	dx=x2-x1;
	dy=y2-y1;
	steps=(abs(dx)>abs(dy))?abs(dx):abs(dy);

	incx=dx/float(steps);
	incy=dy/float(steps);

	x=x1;y=y1;

	glBegin(GL_POINTS);

	glVertex2f(x,y);

	for(i=0;i<steps;i++)
	{
		x+=incx;
		y+=incy;
		glVertex2f(x,y);
	}
	glFlush();
	glEnd();
}
/*-------------- INITIALISE DRAWING WINDOW --------------*/
void init()
{
```

```
   glClearColor(0.0, 0.0, 0.0, 0.0);
   glClear(GL_COLOR_BUFFER_BIT);
   glColor3f(1.0, 1.0, 1.0);
	gluOrtho2D(0,500,0,500);
   //glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);

   LineDDA(0,250,500,250);
	LineDDA(250,0,250,500);

	glRasterPos3f(0,246,1);
	glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,'<');
	glRasterPos3f(490,246,1);
	glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,'>');
	glRasterPos3f(246,0,1);
	glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,'v');
	glRasterPos3f(246,487,1);
	glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,'^');

	glColor3f(1.0,1.0,0.0);
}

void clrScr()
{
	glClear(GL_COLOR_BUFFER_BIT);
   glColor3f(1.0, 1.0, 1.0);
   LineDDA(0,250,500,250);
	LineDDA(250,0,250,500);
	glRasterPos3f(0,246,1);
	glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,'<');
	glRasterPos3f(490,246,1);
	glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,'>');
	glRasterPos3f(246,0,1);
	glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,'v');
	glRasterPos3f(246,487,1);
	glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,'^');
	glColor3f(1.0,1.0,0.0);
}

void drawFig(){
	int i;
	//clrScr();
	for(i=0;i<numv-1;i++)
	{
		LineDDA(points[i].x,points[i].y,points[i+1].x,points[i+1].y);
	}
	LineDDA(points[0].x,points[0].y,points[i].x,points[i].y);
}
/*--------------------------- INPUT VERTICES ---------------------------*/
void input(){
```

```
	int ch,i,x,y;
	if(inp==true){
		printf("\nAlready Input\n");
		return;
	}
	printf("\nENTER NUMBER OF VERTICES :: ");
	scanf("%d",&numv);
	printf("Input points using :-\n1.Keyboard\n2.Mouse\nENTER CHOICE ::");
	scanf("%d",&ch);
	if(ch!=1)
		return;
	for(i=0;i<numv;i++)
	{
		printf("\nENTER POINT (X Y) :: ");
		scanf("%d %d",&x,&y);
		points[i].x=250+x;
		points[i].y=250+y;
	}
	inp=true;
}
/*--------------------------- MOUSE FUNCTION ---------------------------*/
void mouseinp(int button,int action,int xMouse,int yMouse){
	if(inp==false)
	{
		if(cnt<numv)
		{
			if(button==GLUT_LEFT_BUTTON && action==GLUT_DOWN)
			{
				//printf("%d %d",xMouse,yMouse);
				points[cnt].x=xMouse;
				points[cnt].y=500-yMouse;
				cnt++;
			}
		}
		else
		{
			inp=true;
			drawFig();
		}
	}
}
/*--------------------- TRANSLATE -------------------------*/
void translate(float tx,float ty)
{	int i;
	for(i=0;i<numv;i++)
	{
		points[i].x+=tx;
		points[i].y+=ty;
	}
```

```
}
/*---------------------- ROTATE -------------------------*/
void rotate(){
	float ang,angsin,angcos,x,y;
	int i;

	printf("\nENTER ANGLE OF ROTATION :: ");
	scanf("%f",&ang);
	ang=(ang*3.141)/180;
	angsin=sin(ang);
	angcos=cos(ang);

	for(i=0;i<numv;i++)
	{

		x=points[i].x-250;
		y=points[i].y-250;

		//printf("\n%f %f\n",x,y);

		x=(x*angcos)-(y*angsin);
		y=((points[i].x-250)*angsin)+(y*angcos);

		points[i].x=x+250;
		points[i].y=y+250;

		//printf("\n%d %d\n",points[i].x,points[i].y);
	}
	glColor3f(1.0,0.0,0.0);
}
/*---------------------- SHEAR -------------------------*/
void shear(){
	int ch,i;
	float x,y;
	printf("\n1.X - SHEAR\n2.Y - SHEAR\nENTER CHOICE :: ");
	scanf("%d",&ch);
	if(ch==1)
	{
		printf("\nENTER X - SHEAR FACTOR :: ");
		scanf("%d",&ch);
		for(i=0;i<numv;i++)
		{
			x=points[i].x-250;
			y=points[i].y-250;
			x=x+(y*ch)-((points[0].y-250)*ch);
			points[i].x=x+250;
			points[i].y=y+250;
		}
	}
```

```
	else if(ch==2)
	{
		printf("\nENTER Y - SHEAR FACTOR :: ");
		scanf("%d",&ch);
		for(i=0;i<numv;i++)
		{
			x=points[i].x-250;
			y=points[i].y-250;
			y=y+(x*ch)-((points[0].x-250)*ch);
			points[i].x=x+250;
			points[i].y=y+250;
		}
	}
	else
	{
		printf("\n!! INVALID INPUT !!\n");
	}
}
/*---------- SCALE ----------*/
void scale(float sx,float sy)
{
	float x,y,xi,yi;
	int i;
	xi=points[0].x;
	yi=points[0].y;
	for(i=0;i<numv;i++)
	{
		x=points[i].x-250;
		y=points[i].y-250;
		x=(x*sx);
		y=(y*sy);
		points[i].x=x+250;
		points[i].y=y+250;
	}
	translate(xi-points[0].x,yi-points[0].y);
	glColor3f(0.0,1.0,0.0);
}
/*---------- REFLECT ----------*/
void reflect(){
	int ch,i;
	float x,y;
	printf("REFLECTION ABOUT :-\n1.X - AXIS\n2.Y - AXIS\n3.ORIGIN\nENTER CHOICE :: ");
	scanf("%d",&ch);
	switch(ch)
	{
		case 1:
			for(i=0;i<numv;i++){
				y=(-1)*(points[i].y-250);
				points[i].y=250+y;
```

```
                        }
                        break;
                case 2:
                        for(i=0;i<numv;i++){
                                x=(-1)*(points[i].x-250);
                                points[i].x=250+x;
                        }
                        break;
                case 3:
                        for(i=0;i<numv;i++){
                                x=(-1)*(points[i].x-250);
                                y=(-1)*(points[i].y-250);
                                points[i].y=250+y;
                                points[i].x=250+x;
                        }
                        break;
        }
        glColor3f(1.0,.38,.01);
}

void menu(GLint ch)
{
        float sx,sy;
        switch(ch)
        {
                case 1:// Input
                        input();
                        break;
                case 2:// Translate
                        printf("\nENTER TRANSLATION FACTORS (X Y) :: ");
                        scanf("%f %f",&sx,&sy);
                        translate(sx,sy);
                        break;
                case 3:// Rotate
                        sx=250.0-points[1].x;
                        sy=250.0-points[1].y;
                        translate(sx,sy);
                        rotate();
                        translate(-sx,-sy);
                        break;
                case 4:// Shear
                        shear();
                        break;
                case 5:// Scale
                        printf("\nENTER SCALING FACTORS (X Y) :: ");
                        scanf("%f %f",&sx,&sy);
                        scale(sx,sy);
                        break;
                case 6:// Reflect
```

```
                                reflect();
                                break;
        }
        drawFig();
}

int main(int argc, char** argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_SINGLE);
   glutInitWindowSize(500,500);      // Define Window Size
   glutInitWindowPosition(100,100);// Define Window Position
   glutCreateWindow("2D Transformations");
        init();      // Initialise drawing window
        initGlobalVars(); // Initilaise Global Variables
   glutDisplayFunc(drawFig);          // Declare Drawing Function
   glutMouseFunc(mouseinp);         // Declare Mouse Function

   glutCreateMenu(menu);// Define Menu
        glutAddMenuEntry("Input",1);
        glutAddMenuEntry("Translate",2);
        glutAddMenuEntry("Rotate",3);
        glutAddMenuEntry("Shear",4);
        glutAddMenuEntry("Scale",5);
        glutAddMenuEntry("Reflect",6);
   glutAttachMenu(GLUT_RIGHT_BUTTON); // Attack menu to right mouse button

   glutMainLoop();
   return 0;
}
```

Output:

```
mllab10@mllab10: ~/prac
File  Edit  View  Search  Terminal  Help
mllab10@mllab10:~$ cd prac
mllab10@mllab10:~/prac$ g++ prog.c -lglut -lGL -lGLEW -lGLU -o out
mllab10@mllab10:~/prac$ ./out
mllab10@mllab10:~/prac$ ./out

ENTER NUMBER OF VERTICES :: 3
Input points using :-
1.Keyboard
2.Mouse
ENTER CHOICE ::2

ENTER TRANSLATION FACTORS (X Y) :: -50 -50

ENTER SCALING FACTORS (X Y) :: 2 2

ENTER ANGLE OF ROTATION :: 45
REFLECTION ABOUT :-
1.X - AXIS
2.Y - AXIS
3.ORIGIN
ENTER CHOICE :: 2
mllab10@mllab10:~/prac$
```