Chapter Three

2D Transformation & Viewing

Introduction To Transformation
Types of Transformation
- **Translation**
- **Scaling**
- **Rotation**
- **Reflection**
- **Shear**

Matrix Representation of Transformation
2D Viewing
Window to Viewport Transformation (Mapping)
Window to Viewport Transformation N
Clipping and windowing
- **Clipping window**
- **Point clipping**
- **Line clipping**
- **The Cohen–Sutherland algorithm**
- **Intersection points**

Chapter Three

2D Transformation & Viewing

3.1 Introducton To Transformation

One of the most common and important tasks in computer graphics is to transform (changing) the coordinates (position, orientation, size and shape) of either object within the graphical scene or the camera that is viewing the scene. It is also frequently necessary to transform coordinates from one coordinate system to another, (e.g. world coordinates to viewpoint coordinates to screen coordinates). All of these transformations can be efficiently and sufficiently handled using some simple matrix representations, which we will see can be particularly useful for combining multiple transformations into a single composite transform matrix.

The **advantage** of used the transformation:

1. Details appear more clearly.
2. Reduces a picture more of if is visible.
3. Change the scale of a symbol.
4. Rotate it through some angle.

3.2 Types of Transformations

Three basic types of transformations that can perform in two dimensions:

a. Translation (shift OR move).
b. Scaling.
c. Rotation

These basic **transformations** can also be combined to obtain more complex transformations.

Some package provides few additional transformations which are useful in certain application. Two such transformation are **Reflection** and **Shear**.

3.2.1 Translation

Translation is a transformation that 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').Figure 3.1 show the translation and **Mathematically** this can be represented as:

$$X' = X + tx \qquad \& \qquad Y' = Y + ty$$

Note: Using coordinate system the translating factor are

If $tx > 0$ then point moves to the right.

If $tx < 0$ then point moves to the left.

If $ty > 0$ then point moves to the up.

If $ty < 0$ then point moves to the down.

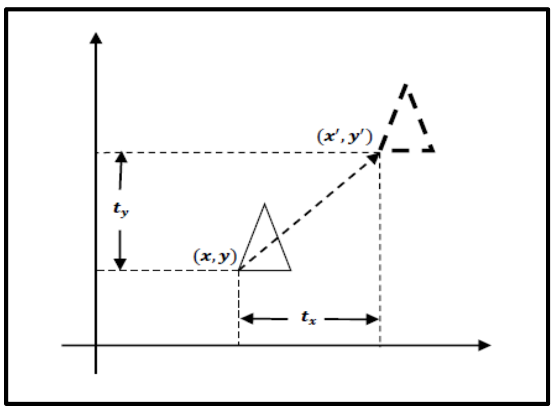


Figure 3.1 Translation

Example 1: Translate the point **A(10,10)**, 2 unit in x direction and 1 unit in y direction? (using **mathematical equation**)

Solution

X =10, Y=10,

tx=2 , ty=1

X'= X+ tx

=10+2=12

Y'= Y + ty

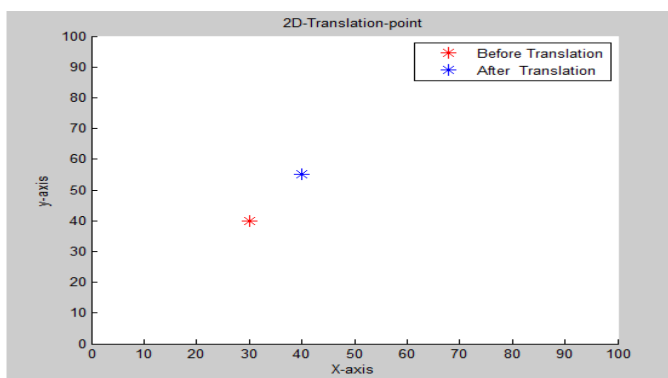
=10+1=11 , the coordinate after translation is **A'(12,11)**.

Program 1: Matlab program to Translate point.

```
% 2D-Translation Transformation for One Point
clc;
clear all;
close all
% enter x-value & y-value for point
x=input ('enter x-value ');
y=input ('enter y-value ');
%enter Translating factor tx & ty
tx=input ('enter Tx: ');
ty=input ('enter Ty: ');
x1=x+tx;
y1=y+ty;
axis ([0 100 0 100]);
hold on
plot (x, y,'r*','markersize',10)
plot (x1, y1,'bo','markersize',10)
legend ('Before Translation’, ‘After Translation')
xlabel('X-axis')
ylabel('y-axis')
title('2D-Translation-point')
```

The Output of program:

enter x-value 30
enter y-value 40
enter Tx: 10
enter Ty: 15



3.2.2 Scaling

Scaling is a transformation used to change the size of an object. 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 by the scaling factor (S) to get the desired result.

Notes:

If the scaling factor (S < 1) ; then we can reduce the size of the object.

If the scaling factor (S > 1) ; then we can increase the size of the object.

If the scaling factor (S = 1) ; then no change.

Let us assume that the original coordinates are (x, y), the scaling factors are (S_x, S_y), and the produced coordinates are (x', y'). This can be **mathematically** represented as shown below :

x' = x . Sx & y' = y . Sy

If $S_x = S_y \rightarrow$ No change in the shape the object ($uniform\ scaling$)

If $S_x \neq S_y$ change in the shape the object (distortion $in\ the\ original\ object$)

The scaling process is shown in figure 3.2 as the following.

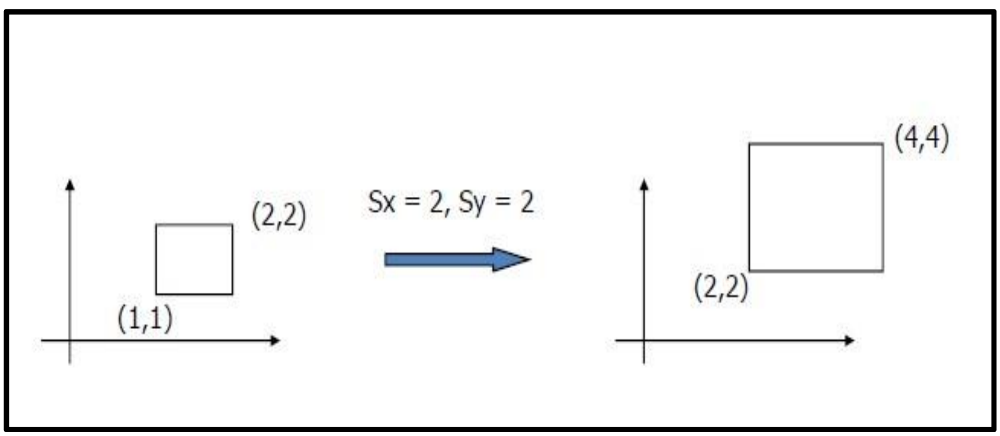


Figure 3.2 Scaling

Whenever the scaling process is performed there is one point still no change (same position), this point is called fixed point of the scaling transformation.

There are **two types of scaling** depending on fixed point:

1. If **the fixed point at the origin**, then the point (x,y) can be scaled by a scale factor S_x , S_y in the x-axis and y-axis direction respectively to the new point (x',y').

$$x' = x \cdot S_x \qquad \& \qquad y' = y \cdot S_y$$

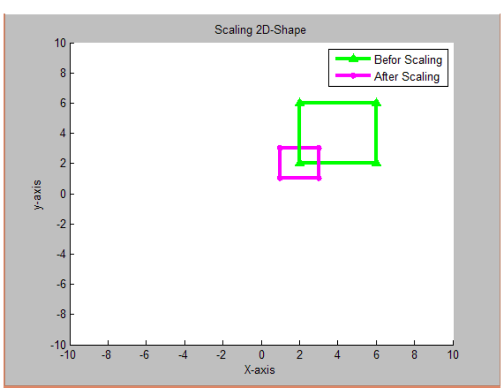
Example 2: consider square with left -bottom corner at (2,2) and height-top corner at (6,6) apply transformation which makes its size half.

Solution

As we want size half so value scale factor are Sx= 0.5, Sy =0.5 ,and the coordinates of square are A(2,2),B(6,2),C(6,6),D(2,6).

<u>**A(2,2)**</u>	<u>**B(6,2)**</u>	<u>**C(6,6)**</u>	<u>**D(2,6)**</u>
$x' = x \cdot S_x$	$x' = x \cdot S_x$	$x' = x \cdot S_x$	$x' = x \cdot S_x$
= 2*0.5=1	= 6*0.5=3	= 6*0.5=3	= 2*0.5=1
$y' = y \cdot S_y$	$y' = y \cdot S_y$	$y' = y \cdot S_y$	$y' = y \cdot S_y$
=2*0.5=1	=2*0.5=1	=6*0.5=3	=6*0.5=3
A'**(1,1)**	B'**(3,1)**	C'**(3,3)**	D'**(1,3)**

The final coordinates of square are A'**(1,1)** ,B'**(3,1)**,C'**(3,3)**,D'**(1,3).**



2. if **fixed point is Arbitrary Point**

Arbitrary Point is a point that is based a random choice or personal rather than a reason or system and figure 3.3 show the types of Arbitrary Point.

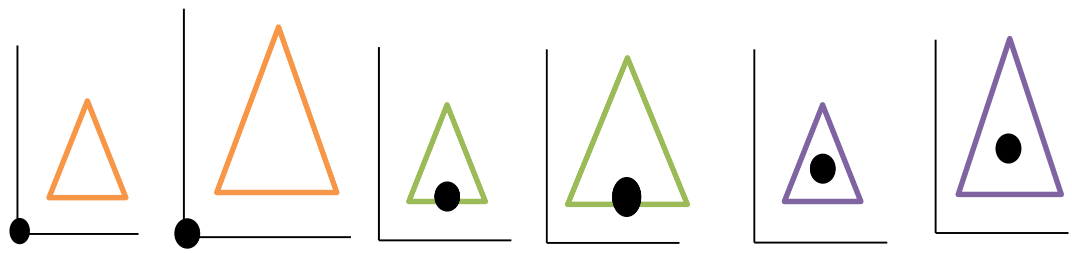


Figure 3.3 Types of Arbitrary Point

The **scaling** is performed with respect to any point (x_f, y_f) as fixed point according to the **three steps**:

1. We must first **translate** the object so that the fixed point is coincide with the origin as follows: every object point (x, y) is moved to the new position (x', y') such as:

$$x' = x - x_f , \qquad y' = y - y_f$$

2. We then **scaled** these translated points by scale factors S_x and S_y ,so that :

$$x'' = x'.\ S_x , \qquad y'' = y'.\ S_y$$

3. Then perform the inverse of the original translation to translate **(move)** the fixed point back to its original position.

$$x''' = x'' + x_f , \qquad y''' = y'' + y_f$$

These three steps can be combined in the following equation that scales a point (xf, yf).

the final equations of scaling object about arbitory point are

$$\boldsymbol{x''' = (x - x_f).\ S_x + x_f ,} \qquad \boldsymbol{y''' = (y - y_f).\ S_y + y_f}$$

Example 3: Consider a tringle defined by its three vertices (1,0), (4,0), (3,2) been scaled 3 units to the Sx and 3 units to the Sy with respect to a fixed point (3,0) . Find the new coordinates of this tringle after Scaling.

Solution :

Sx=3 ; Sy=3 ; xf=3 ; yf=0 ;

P1=(1,0)

1- x'=x-xf ; x'=1-3 ; x'=-2 ; y'=y-yf ; y'=0-0 ; y'=0;

2- x"=x'*Sx ; x"=-2*3 ;x"=-6 ; y"=y'*Sy ; y"=0*3 ; y"=0;

3- x'"=x"+xf ; x'"=-6+3; x'"=-3 ; y'"=y"+yf ;y'"=0+0; y'"=0;

P1=(1,0) → P'1(-3,0)

So, the new coordinated of the tringle are :

p(x,y)	P'(x',y')
(1,0)	(-3,0)
(4,0)	(6,0)
(3,2)	(3,6)

OR

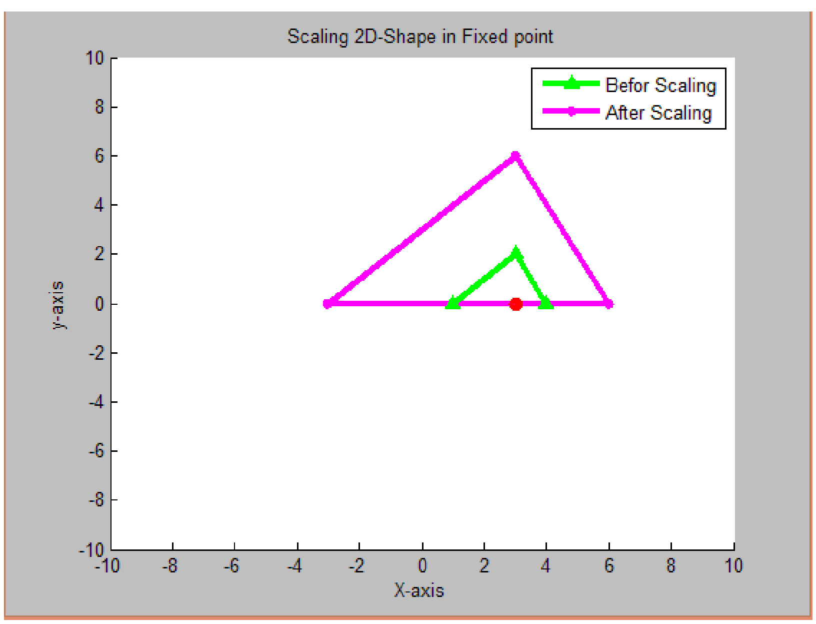
Solution :

Sx=3 ; Sy=3 ; xf=3 ; yf=0 ;

P1=(1,0)

1- x'"= (x-xf)* Sx +xf ; x'"=(1-3)*3+3; x'"=-3 ; y'"= (y-yf)*Sy+yf ;y'"=(0-0)*3+0; y'"=0;

P1=(1,0) → P'1(-3,0)

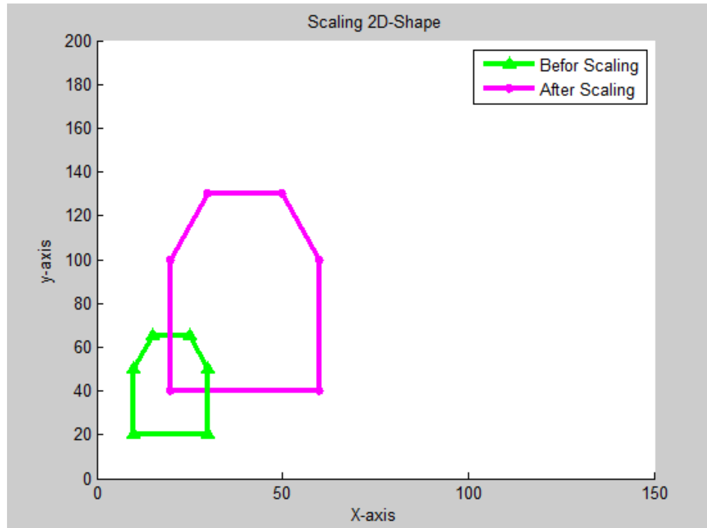


Program 2: Matlab program Scaling Transformation for 2D shape.

```
% SCALING TRANSFORMATION PROGRAM for 2D-shape
clc; clear all; close all
%Enter number of shape or object vertices
g=input ('enter no. of Shape vertices: ');
%enter SCALING factor sx & sy
sx=input ('enter Sx: ');
sy=input ('enter Sy: ');
% enter x-value & y-value for All Shape vertices
for i=1: g
x(i)=input ('enter x-value: ');
y(i)=input ('enter y-value: ');
x1(i)=x(i)*sx;
y1(i)=y(i)*sy;
end
axis ([0 150 0 200]);
hold on
for i=1: g-1
plot([x(i) x(i+1)], [y(i) y(i+1)], ...
'g^-','LineWidth',3,'markersize',5)
plot([x1(i) x1(i+1)], [y1(i) y1(i+1)], ...
'm*-','LineWidth',3,'markersize',5)
end
plot([x(i+1) x(g-i)], [y(i+1) y(g-i)], ...
'g^-','LineWidth',3,'markersize',5)
plot([x1(i+1) x1(g-i)], [y1(i+1) y1(g-i)], ...
'm*-','LineWidth',3,'markersize',5)

Legend ('Before Scaling’, ‘After Scaling')
xlabel('X-axis')
ylabel('y-axis')
title ('Scaling 2D-Shape')
```

The Output of program:

enter no. of Shape vertices: 6
enter Sx: 2
enter Sy: 2
enter x-value 10
enter y-value 20
enter x-value 30
enter y-value 20
enter x-value 30
enter y-value 50
enter x-value 25
enter y-value 65
enter x-value 15
enter y-value 65
enter x-value 10
enter y-value 50



3.2.3 Rotation

Rotation is a transformation that used to reposition the object along the circular path in the XY -plane. You can rotate an object about the origin or about a pivot point.

It is possible to rotate one or more objects or the entire image about any point in world space in either negative oriented a (clockwise) where angle is negative oriented or positive oriented (Anti-clockwise) where angle is positive.

There are **two types of Rotate** :

1. Rotate about the origin

Any point (x,y) can be represented by its radial distance (r) from the origin and its angle $\emptyset$ of the x-axis.

x= r*cos ($\emptyset$)

y=r*sin($\emptyset$) …….. (1)

If (x,y) is rotated an angle θ in the Anti-clockwise direction. The transformed point $(\bar{x}, \bar{y})$ is represented as:

$\bar{x} = r * \cos(\emptyset + \theta)$

$\bar{y} = r * \sin(\emptyset + \theta)$ .... **(2)**

The Figure 3.4 show the Rotate about the origin.

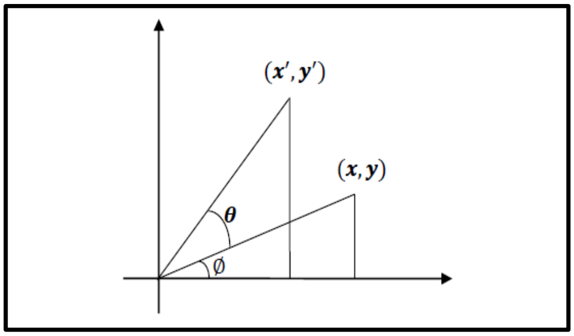


Figure 3.4 Rotate about the origin

Using the laws of sines and cosines we get:

$$\boldsymbol{x} = \boldsymbol{x}\cos(\boldsymbol{\theta}) - y\sin(\boldsymbol{\theta})$$

$$\boldsymbol{y} = \boldsymbol{x}\sin(\boldsymbol{\theta}) + \boldsymbol{y}\cos(\boldsymbol{\theta}) \quad \text{............(3)}$$

Equation (3) are the transformation that rotate a point an angle (θ) about the origin in the ***Anti-clockwise*** direction.

To rotate an object each point defining that object must be transformed using equation (3). The object is then drawn using the list of transformed points.

To rotate in ***clockwise*** change the angle θ to - θ where:

cos (-θ) = cos (θ)

sin(-θ) = -sin (θ)

So to rotate a point (x,y) through a ***clockwise*** angle θ about the origin of the coordinate system we write:

$$\bar{\boldsymbol{x}} = \boldsymbol{x}\cos(\boldsymbol{\theta}) + y\sin(\boldsymbol{\theta})$$

$$\bar{\boldsymbol{y}} = -\boldsymbol{x}\sin(\boldsymbol{\theta}) + y\cos(\boldsymbol{\theta}) \quad \text{............(4)}$$

Equation (4) are the transformation that rotate a point an angle (θ) about the origin in the ***clockwise*** direction.

Example 4: Consider a triangle defind by its three vertices (20,0), (60,10),and (40,100) been rotated 30o counter clockwise. Find the new coordinates of this triangle after Rotation.

Solution

$\bar{x} = x\cos(\theta) - y\sin(\theta)$

$\bar{y} = x\sin(\theta) + y\cos(\theta)$

Cos(30)=0.866; sin(30)=0.5

p1=(20,0)

x'=20*cos(30)-0*sin(30)

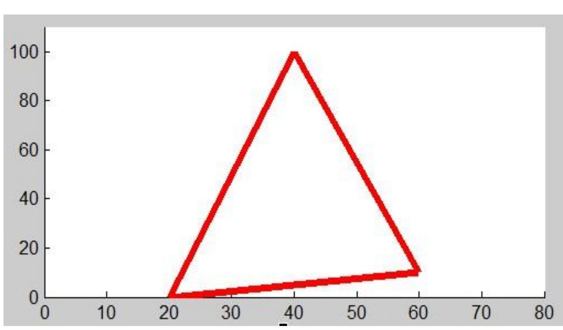
x'=20*0.866 -0= 17.32

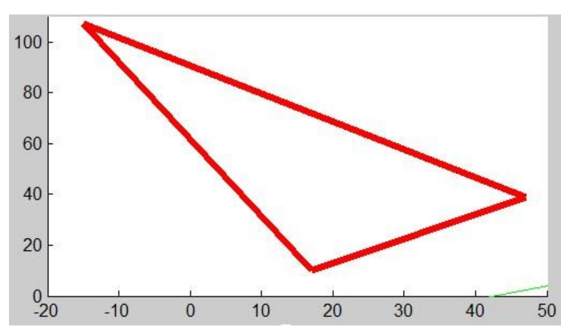
y'= 20*sin(30)+0*cos(30)

y'=20*0.5+0 = 10

p1(20,0) → p'1(17,10)

p(x,y)	P'(x',y')
(20,0)	(17,10)
(60,10)	(47,39)
(40,100)	(-15,107)



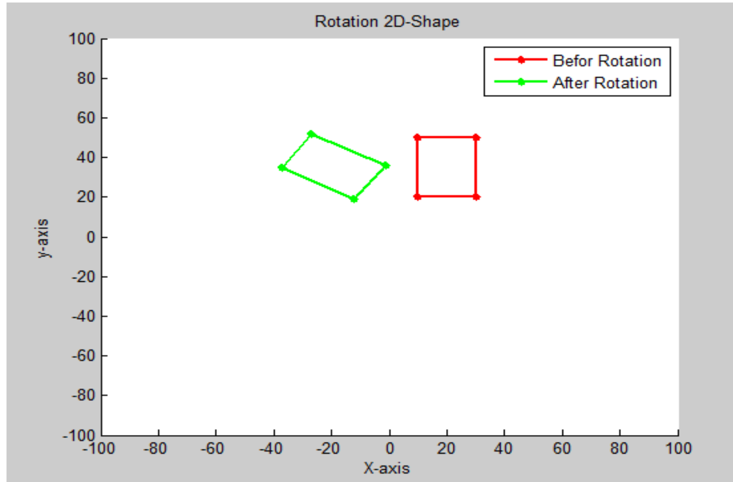


Program 3: Matlab Program to Rotation 2D -Shape (Anti-Clockwise)

```
% ROTATION TRANSFORMATION (Anti-clockwise) PROGRAM %for 2D-shape
clc;
clear all;
close all
%Enter number of shape or object vertices
g=input ('enter no. of object vertices: ');
%Enter Rotation angle
t=input ('enter the angle: ');
% enter x-value & y-value for All Shape vertices
for i=1: g
   x(i)=input ('enter x-value ');
   y(i)=input ('enter y-value ');
   x1(i)=round ((x(i)) *cos(t)-(y(i)) *sin (t));
   y1(i)=round ((y(i)) *cos(t)+(x(i)) *sin(t));
end
axis ([-100 100 -100 100]);
hold on
for i=1: g-1
   plot([x(i) x(i+1)], [y(i) y(i+1)], ...
      'r*-','linewidth',2,'markersize',5)
   plot([x1(i) x1(i+1)], [y1(i) y1(i+1)], ...
      'g*-','linewidth',2,'markersize',5)
end
plot([x(i+1) x(g-i)], [y(i+1) y(g-i)], ...
   'r*-','linewidth',2,'markersize',5)
plot([x1(i+1) x1(g-i)], [y1(i+1) y1(g-i)], ...
   'g*-','linewidth',2,'markersize',5)

Legend ('Before Rotation', 'After Rotation')
xlabel('X-axis')
ylabel('y-axis')
title ('Rotation 2D-Shape')
```

The Output of program:

enter no. of object vertices: 4
enter the angle: 45
enter x-value 10
enter y-value 20
enter x-value 30
enter y-value 20
enter x-value 30
enter y-value 50
enter x-value 10
enter y-value 50



2. Rotate about a pivot point

After an object is rotated about a specified pivot point, it is still the same distance away from the pivot point but its orientation has been changed,Figure 3. 5 Show the types a pivot point :

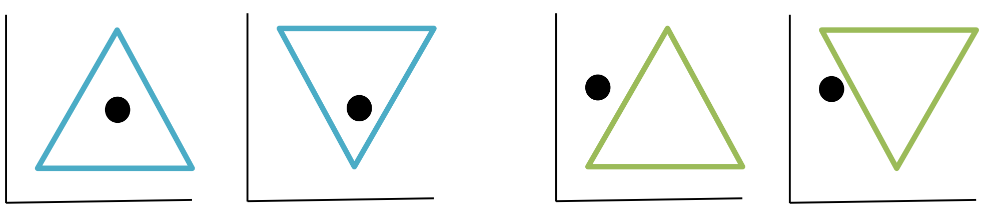


Figure 3. 5 the types a pivot point

To rotate an object an angle (θ) about a pivot point **three steps** are required:

1. **Translate** the pivot point (, y_p) to the origin. Every point (x,y) defining the object is translated to a new point ($\bar{x}$, $\bar{y}$) where:

$$\bar{x} = x - x_p$$

$$\bar{y} = y - y_p$$

2. **Rotate** these translated points $(\bar{x}, \bar{y})$ θ degree about the origin to obtain the new point $(\bar{\bar{x}}, \bar{\bar{y}})$

$$\bar{\bar{x}} = \bar{x} * \cos(\theta) - \bar{y} * \sin(\theta)$$
$$\bar{\bar{y}} = \bar{y} * \cos(\theta) + \bar{x} * \sin(\theta)$$

Substituting for $\bar{x}$ and $\bar{y}$

$$\bar{\bar{x}} = (x - x_p) * \cos(\theta) - (y - y_p) * \sin(\theta)$$
$$\bar{\bar{y}} = (y - y_p) * \cos(\theta) + (x - x_p) * \sin(\theta)$$

3. **Translate** the center of rotation back to the pivot point (x_p , y_p)

$$\bar{\bar{\bar{x}}} = \bar{\bar{x}} + x_p$$
$$\bar{\bar{\bar{y}}} = \bar{\bar{y}} + y_p$$

The final equation of rotate object about a pivot point are:

$$\bar{\bar{\bar{x}}} = (x - x_p) * \cos(\theta) - (y - y_p) * \sin(\theta) + x_p$$

$$\bar{\bar{\bar{y}}} = (y - y_p) * \cos(\theta) + (x - x_p) * \sin(\theta) + y_p$$

Figure 3.6 show rotate about a pivot point.

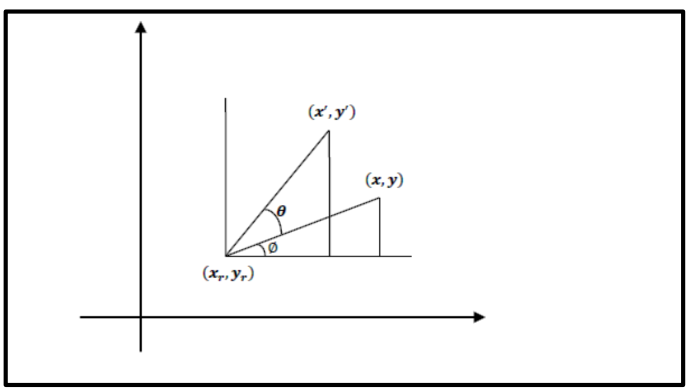


Figure 3.6 Rotate about a pivot point

The Figure 3.7 show the three steps of rotate about a pivot point.

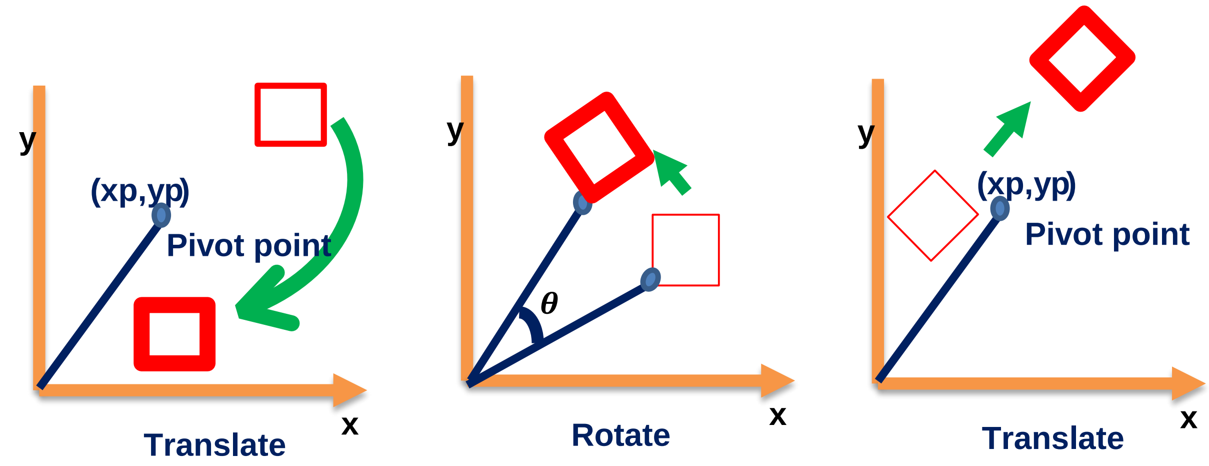


Figure 3.7 The Three Steps of Rotate about a pivot point.

Example 5: Consider a triangle defind by its three vertices (20,0), (60,10) and (40,100) been rotated 30° counterclockwise about a pivot point (15,20). Find the new coordinates of this triangle after Rotation.

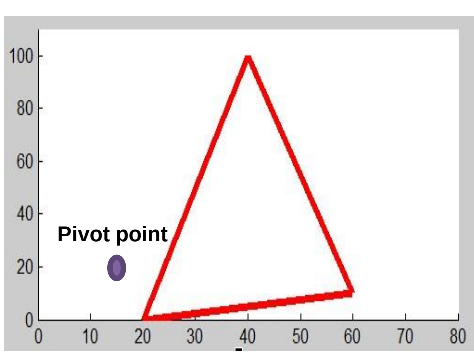
Solution

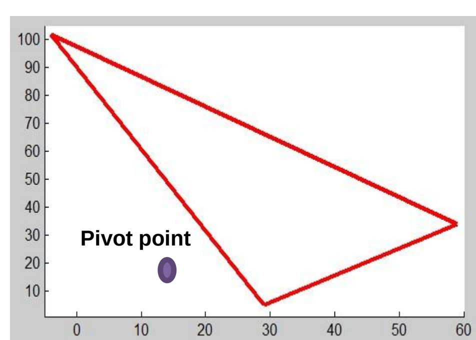
$$x'' = (x - x_p) * \cos(\theta) - (y - y_p) * \sin(\theta) + x_p$$

$$y'' = (y - y_p) * \cos(\theta) + (x - x_p) * \sin(\theta) + y_p$$

cos(30)=0.866 , sin(30)=0.5 ;
xp=15 , yp= 20
p1=(20,0)
x'"=(20-15)*0.866-(0-20)*0.5+15=29
y'"=(0-20)*0.866+(20-15)*0.5+20=5
p1(20,0) → p'1(29, 5)

p(x,y)	p'(x',y')
(20,0)	(29, 5)
(60,10)	(59,34)
(40,100)	(-4,102)





3.2.4 Reflection

A reflection is a Transformation that produces a mirror image of an object**.**
Reflection is a kind of rotation where the angle of rotation is 180 degree,The size of reflected object is same as the size of original object.Consider a point object O has to be reflected in a 2D plane.

Let-

- Initial coordinates of the object O = (Xold, Yold)
- New coordinates of the reflected object O after reflection = (Xnew, Ynew)

Types of Reflection:

1. Reflection about the x-axis
2. Reflection about the y-axis
3. Reflection about an axis perpendicular to xy plane and passing through the origin
4. Reflection about line y=x

1. Reflection On X-Axis:

This reflection is achieved by using the following reflection equations:

Xnew = Xold
Ynew = -Yold

In this transformation value of x will remain same where as the value of y will become negative. Following figure 3. 8 shows the reflection of the object axis. The object will lie another side of the x-axis.

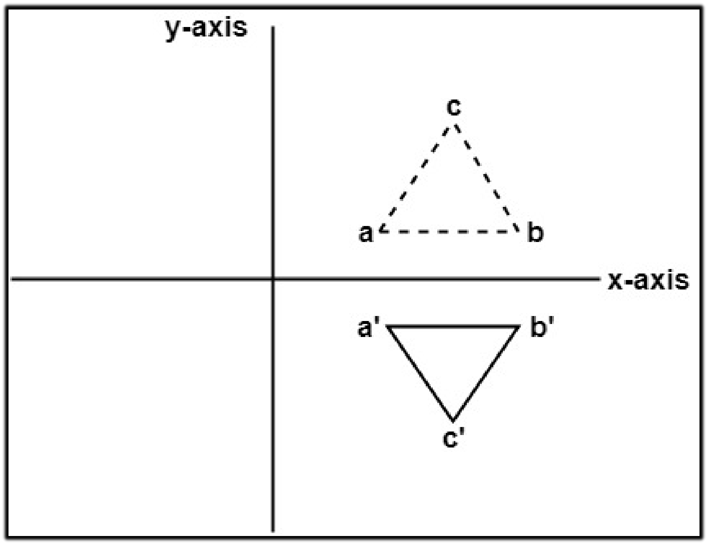


Figure 3. 8 Reflection of the object on X-axis

2. Reflection On Y-Axis:

This reflection is achieved by using the following reflection equations:

Xnew = -Xold

Ynew = Yold

Here the values of x will be reversed, whereas the value of y will remain the same. The object will lie another side of the y-axis.The following figure 3.9 shows the reflection about the y-axis

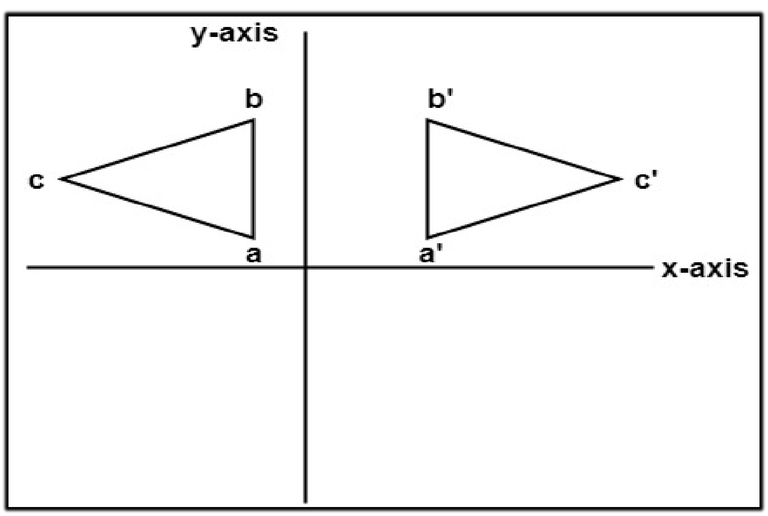


Figure 3.8 the Reflection about the y-axis

3. Reflection about an axis perpendicular to xy plane and passing through origin: In this value of x and y both will be reversed. This is also called as half revolution about the origin. The following figure 3.9 shows the reflection about xy plane.

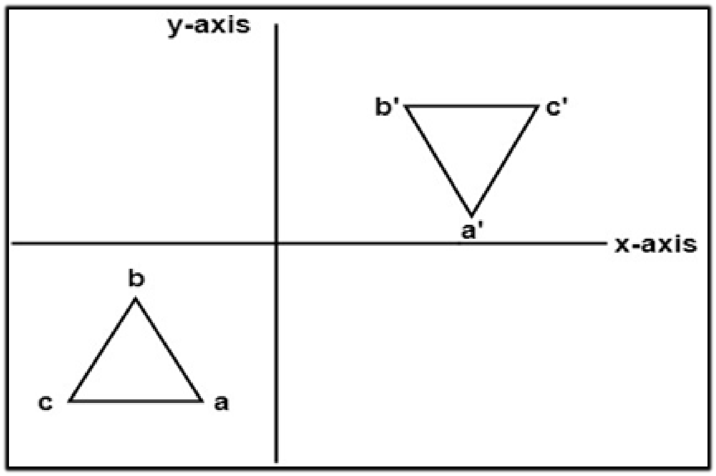


Figure 3.9 shows the reflection about xy plane.

4. Reflection about line y=x:

The object may be reflected about line y = x with the help of following transformation matrix, Figure 3.10 show the reflection about y=x plane.

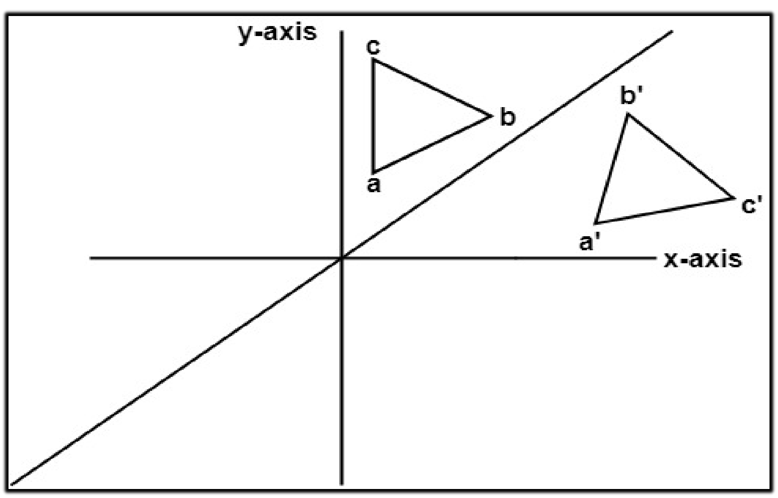


Figure 3.10 the reflection about y=x plane.

First of all, the object is rotated at 45°. The direction of rotation is clockwise. After it reflection is done concerning x-axis. The last step is the rotation of y=x back to its original position that is counterclockwise at 45°.

Example 6:Given a triangle with coordinate points A(3, 4), B(6, 4), C(5, 6). Apply the reflection on the X axis and obtain the new coordinates of the object.

Solution:

Applying the reflection equations, we have:

A(3, 4)

Xnew = Xold = 3

Ynew = -Yold = -4

A'(3,-4)

B(6, 4)

Xnew = Xold = 6

Ynew = -Yold = -4

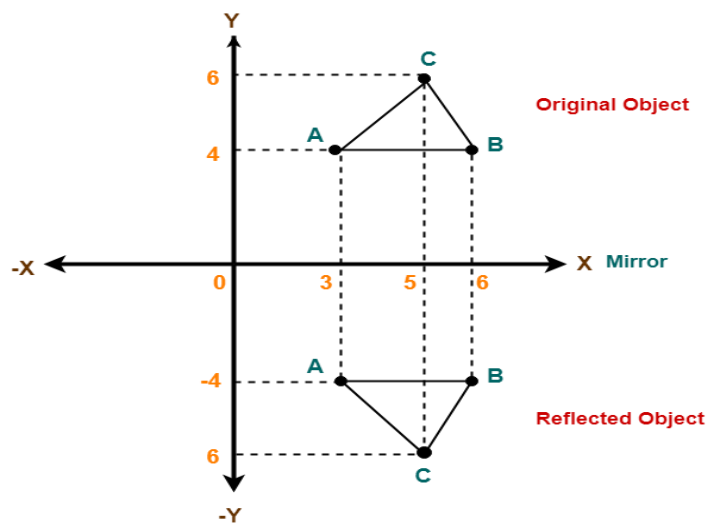
B'(6,-4)

C(5, 6)

Xnew = Xold = 5

Ynew = -Yold = -6

C'(5,-6)



Program 7: Matlab program to Reflection Transformation Program For 2d-Shape

```
% Reflection TRANSFORMATION PROGRAM for 2D-shape
clc; clear all; close all
%Enter number of shape or object vertices
g=input ('enter no. of shape vertices: ');
% enter x-value & y-value for All Shape vertices
for i=1: g
   x(i)=input ('enter x-value:');
   y(i)=input ('enter y-value:');
end
%Reflection Types
disp ('Reflection on x, enter 1');
```

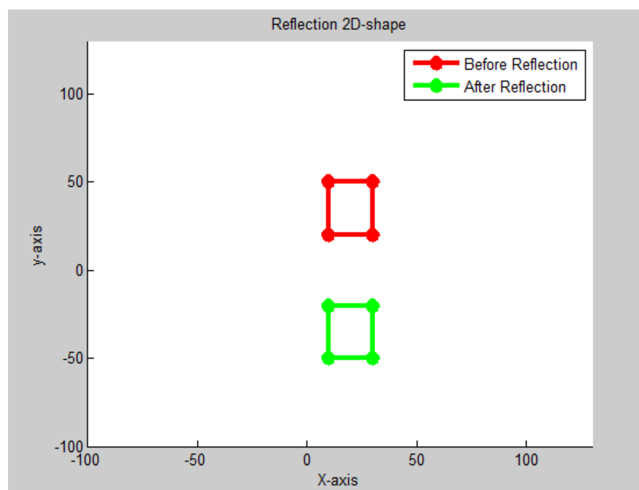
```
disp ('Reflection on y, enter 2');
disp ('Through the origin, enter 3');
disp ('Over the line y = x, enter 4');
disp ('Over the line y = -x, enter 5');
b=input ('enter type of Reflection:');
switch b
   case 1
      for i=1: g
         x1(i)=x(i);
         y1(i)=-y(i);
      end
   case 2
     for i=1: g
         x1(i)=-x(i);
         y1(i)=y(i);
     end
   case 3
     for i=1: g
         x1(i)=-x(i);
         y1(i)=-y(i);
     end
   case 4
     for i=1: g
         x1(i)=y(i);
         y1(i)=x(i);
     end
   case 5
     for i=1: g
         x1(i)=-y(i);
         y1(i)=-x(i);
     end
end
axis ([-100 130 -100 130]);
hold on
for i=1: g-1

   plot([x(i) x(i+1)], [y(i) y(i+1)], ...
```

```
      'r*-','linewidth',3,'markersize',10)
   plot([x1(i) x1(i+1)], [y1(i) y1(i+1)], ...
      'g*-','linewidth',3,'markersize',10)
end
plot([x(i+1) x(g-i)], [y(i+1) y(g-i)], ...
   'r*-','linewidth',3,'markersize',10)
plot([x1(i+1) x1(g-i)], [y1(i+1) y1(g-i)], ...
   'g*-','linewidth',3,'markersize',10)
legend ('Before Reflection', 'After Reflection')
xlabel('X-axis')
ylabel('y-axis')
title ('Reflection 2D-shape')
```

The Output of program:

enter no. of shape vertices: 4 Reflection on x

enter x-value:10enter y-value:20



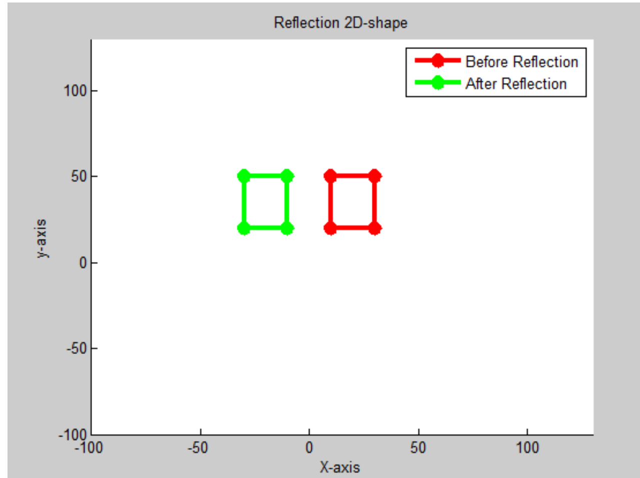
enter x-value:30
enter y-value:20
enter x-value:30
enter y-value:50
enter x-value:10
enter y-value:50
Reflection on x, enter 1
Reflection on y, enter 2
Through the origin, enter 3
Over the line y = x, enter 4
Over the line y = -x, enter 5
enter type of Reflection: 1

The Output of program:

enter no. of shape vertices: 4 Reflection on y

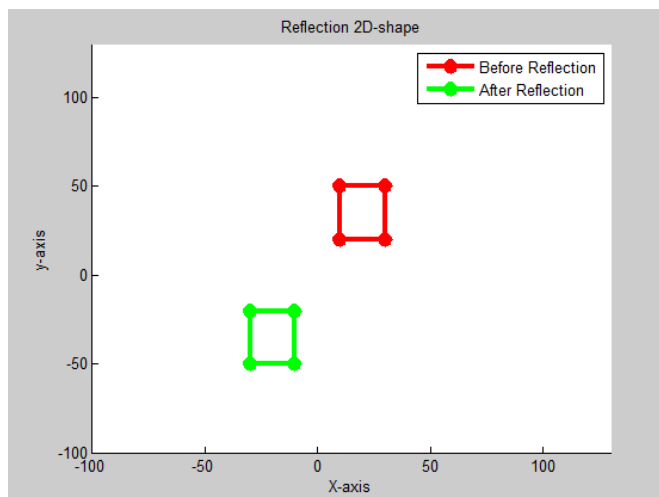
enter x-value:10

enter y-value:20
enter x-value:30
enter y-value:20
enter x-value:30
enter y-value:50
enter x-value:10
enter y-value:50
Reflection on x, enter 1
Reflection on y, enter 2
Through the origin, enter 3
Over the line y = x, enter 4
Over the line y = -x, enter 5
enter type of Reflection: 2



The Output of program:

enter no.of shape vertices:4 Reflection Through the origin
enter x-value:10
enter y-value:20
enter x-value:30
enter y-value:20
enter x-value:30
enter y-value:50
enter x-value:10
enter y-value:50
Reflection on x, enter 1
Reflection on y, enter 2
Through the origin, enter 3
Over the line y = x, enter 4
Over the line y = -x, enter 5
enter type of Reflection: 3



3.2.5 Shear

Distorting or changing the shape of an object by differentially moving some of its vertices as if the object internal layers are sided over each other is called ***Shear***.

Shears either shift coordinates x values or y values, Similar to scaling, the shear transformation requires two parameters (s_x, s_y) not on the main diagonal of the transformation matrix but on the other two positions.

In a two dimensional plane, the object size can be changed along X direction as well as Y direction.

So, there are **two types** of **shearing**:

1. Shearing in X direction
2. Shearing in Y direction

1. Shearing in X direction

Shearing in X axis is achieved by using the following shearing equations:

Xnew = Xold + Shx x Yold

Ynew = Yold

Following Figure 3.11 show Shearing in X direction.

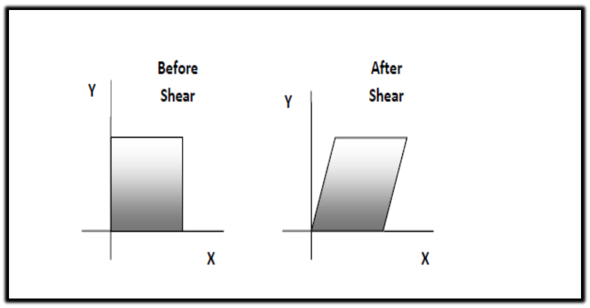


Figure 3.11 show Shearing in X direction

2. Shearing in Y direction

Shearing in Y axis is achieved by using the following shearing equations-

Xnew = Xold

Ynew = Yold + Shy x Xold

Following Figure 3.12 show **Shearing in Y direction**

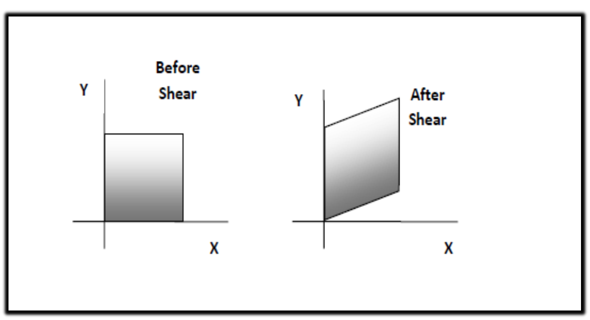


Figure 3.12 show Shearing in Y direction

Example 7:Given a triangle with points A(1, 1), B(0, 0) and C(1, 0). Apply shear parameter 2 on X axis and 2 on Y axis and find out the new coordinates of the object.

solution

1. **Shearing in X Axis**

A(1, 1)

Xnew = Xold + Shx x Yold = 1 + 2 x 1 = 3
Ynew = Yold = 1
A'(3,1)

B(0, 0)

Xnew = Xold + Shx x Yold = 0 + 2 x 0 = 0
Ynew = Yold = 0
B'(0,0)

C(1, 0)

Xnew = Xold + Shx x Yold = 1 + 2 x 0 = 1
Ynew = Yold = 0
C'(1,0)

Thus, New coordinates of the triangle after shearing in X axis = A'(3, 1), B'(0, 0), C'(1, 0).

2. **Shearing in Y Axis**

A(1, 1)

Applying the shearing equations, we have:

$X_{new} = X_{old} = 1$

$Y_{new} = Y_{old} + Sh_y \times X_{old} = 1 + 2 \times 1 = 3$

A'(1,3)

B(0,0)

$X_{new} = X_{old} = 0$

$Y_{new} = Y_{old} + Sh_y \times X_{old} = 0 + 2 \times 0 = 0$

B'(0,0)

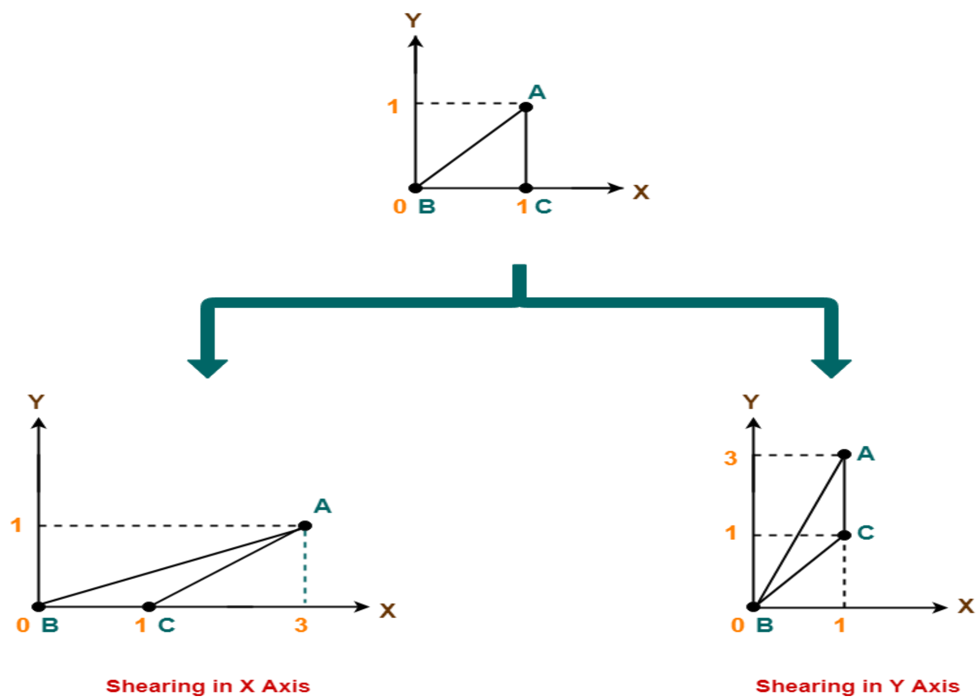
C(1,0)

$X_{new} = X_{old} = 1$

$Y_{new} = Y_{old} + Sh_y \times X_{old} = 0 + 2 \times 1 = 2$

C’(1,2)

New coordinates of the triangle after shearing in Y axis = A’ (1, 3), B’(0, 0), C’(1, 2).



Program 8: Matlab Program to Shearing Transformation Program For 2d-Shape

```
% SHEARING TRANSFORMATION PROGRAM for 2D-shape
clc; clear all; close all
%Enter number of shape or object vertices
g=input ('enter no. of shape vertices: ');
% enter x-value & y-value for All Shape vertices
for i=1: g
   x(i)=input ('enter x-value:');
   y(i)=input ('enter y-value:');
end
```

```
%enter Shearing factor sx & sy
sx=input ('enter Sx: ');
sy=input ('enter Sy: ');

disp ('Shear in the x direction, enter 1');
disp ('Shear in the y direction, enter 2');
disp ('Shear in the both direction, enter 3');
b=input ('enter type of Shearing: ');
switch b
   case 1
      for i=1: g
         x1(i)=x(i)+sx*y(i);
         y1(i)=y(i);
      end
   case 2
      for i=1: g
         x1(i)=x(i);
         y1(i)=y(i)+sy*x(i);
      end
   case 3
      for i=1: g
         x1(i)=x(i)+sx*y(i);
         y1(i)=y(i)+sy*x(i);
      end
end
```

```
axis ([0 50 0 50]);
hold on
for i=1: g-1
   plot([x(i) x(i+1)], [y(i) y(i+1)], ...
      'r^-','linewidth',3,'markersize',10)
   plot([x1(i) x1(i+1)], [y1(i) y1(i+1)], ...
      'g^-','linewidth',3,'markersize',10)
end

plot([x(i+1) x(g-i)], [y(i+1) y(g-i)], ...
```

```
   'r^-','linewidth',3,'markersize',10)
plot([x1(i+1) x1(g-i)], [y1(i+1) y1(g-i)], ...
   'g^-','linewidth',3,'markersize',10)
legend ('Before Shear', 'After Shear')
xlabel('X-axis')
ylabel('y-axis')
title ('Shearing 2D-shape')
```

The Output of program:

enter no. of shape vertices: 4

enter x-value:5

enter y-value:5

enter x-value:10

enter y-value:5

enter x-value:10

enter y-value:15

enter x-value:5

enter y-value:15

enter Sx: 2

enter Sy: 3

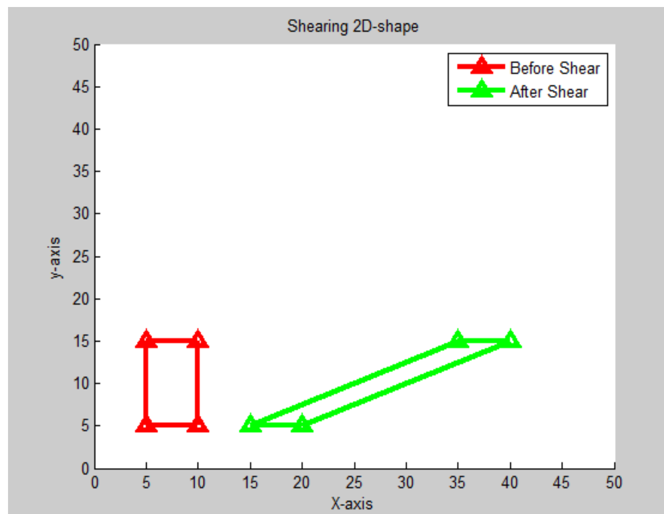
Shear in the x direction, enter 1

Shear in the y direction, enter 2

Shear in the both direction, enter 3

enter type of Shearing: 1

Shear in the x direction



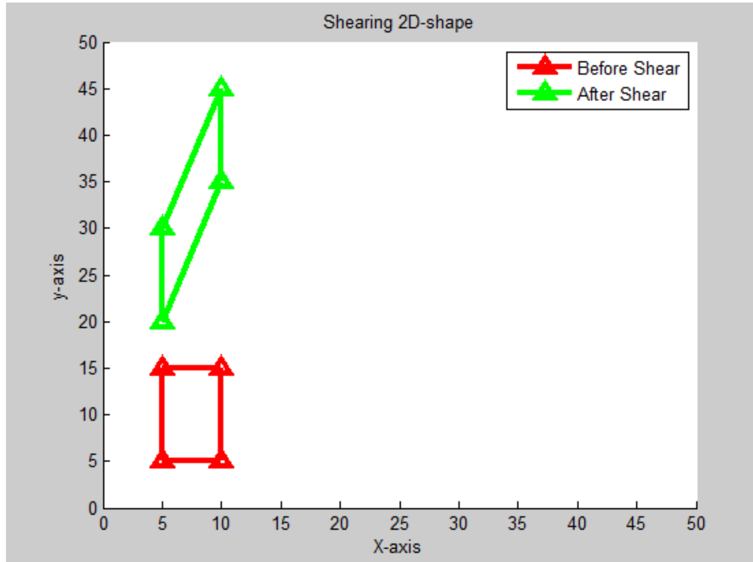
The Output of program:

enter no. of shape vertices :4

enter x-value:5

Shear in the y direction

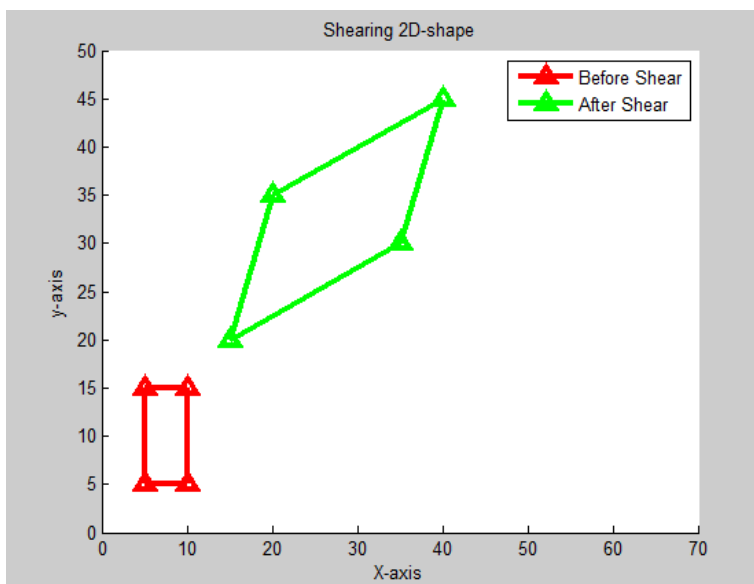
enter y-value:5
enter x-value:10
enter y-value:5
enter x-value:10
enter y-value:15
enter x-value:5
enter y-value:15
enter Sx: 2
enter Sy: 3
Shear in the x direction, enter 1
Shear in the y direction, enter 2
Shear in the both direction, enter 3
enter type of Shearing: 1



The Output of program:

Shear in the both direction

enter no. of shape vertices: 4
enter x-value:5
enter y-value:5
enter x-value:10
enter y-value:5
enter x-value:10
enter y-value:15
enter x-value:5
enter y-value:15
enter Sx: 2
enter Sy: 3
Shear in the x direction, enter 1
Shear in the y direction, enter 2
Shear in the both direction, enter 3 enter type of Shearing: 1



3.3 Matrix Representation of Transformation

Many graphic applications involve sequence of geometric transformations. For example, animation transformation which is require an object to be translated and rotated at each increment of the motion.

- Transformation can be represented as a product of the row vector [x,y] and a 2x2 matrix accept for the translation.
- Transformations can be combined using matrix multiplication

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix} \begin{bmatrix} i & j \\ k & l \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

- Matrices are convenient to represent a sequence of transformations

1- Translation Matrix T(tx , ty)

Wecanrepresent the translation transformation as follows:

P’ = P+T,

$$P = \begin{bmatrix} x \\ y \end{bmatrix}, \quad T = \begin{bmatrix} t_x \\ t_y \end{bmatrix} \dashrightarrow \quad P' = \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

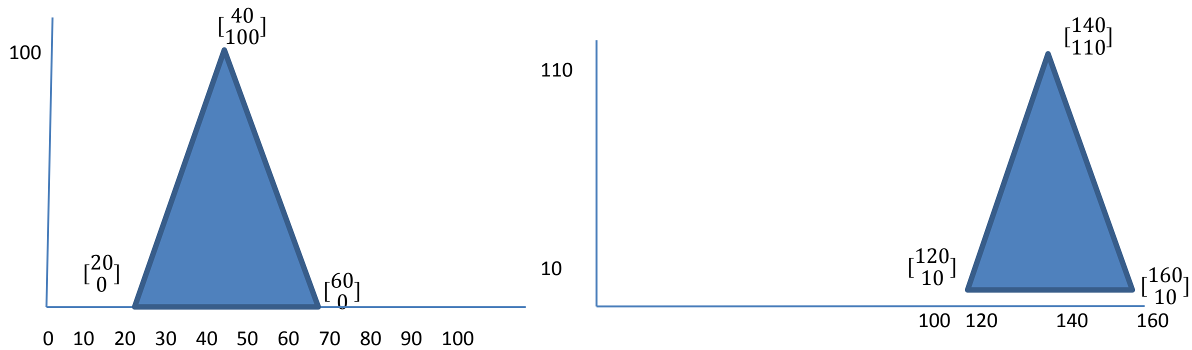
$$P' = \begin{bmatrix} x + tx \\ y + ty \end{bmatrix}$$

Example 8: Consider a triangle defind by its three vertices (20,0), (60,0), (40,100) been moved 100 units to the right and 10 units up. Find the new coordinates of this triangle after translation. (Using **Matrix**)

So, the new coordinated of the triangle are :

$$T=\begin{bmatrix} 100 \\ 10 \end{bmatrix}, P=\begin{bmatrix} 20 & 60 & 40 \\ 0 & 0 & 100 \end{bmatrix}, P'=\begin{bmatrix} 20 + tx & 60 + tx & 40 + tx \\ 0 + ty & 0 + ty & 100 + ty \end{bmatrix}$$

So, the new coordinated of the triangle are : $P' = \begin{bmatrix} 120 & 160 & 140 \\ 10 & 10 & 110 \end{bmatrix}$

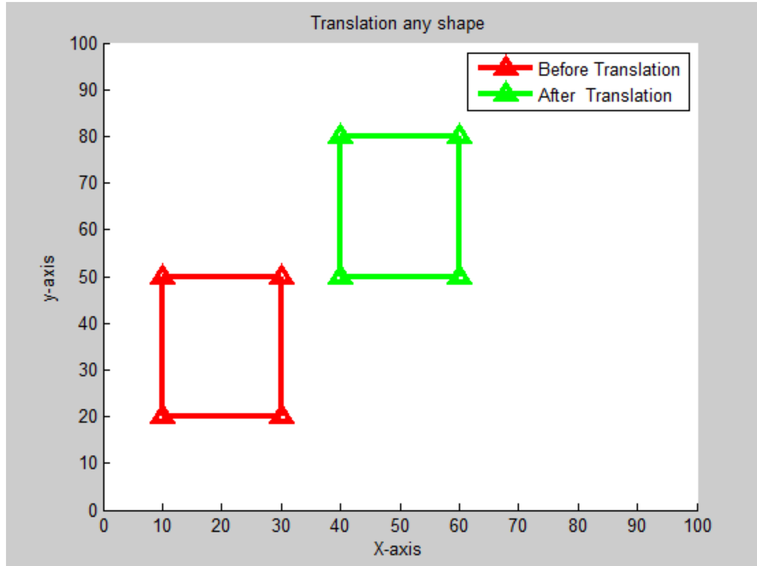


Program 7 : Matlab program to Translate 2D-shape.(using Matrix)

```
% translation transformation program for 2D-shape
clc;clear all; close all
%Enter number of shape or object vertices
g=input ('enter no. of object vertices: ');
%enter Translating factor tx & ty
tx=input ('enter Tx: ');
ty=input ('enter Ty: ');
% enter x-value & y-value for All Shape vertices
for i=1: g
   x(i)=input ('enter x-value:');
   y(i)=input ('enter y-value:');
   x1(i)=x(i)+tx;
   y1(i)=y(i)+ty;
end
axis [0 100 0 100]);
hold on
for i=1: g-1
   plot([x(i) x(i+1)], [y(i) y(i+1)], ...'r^- ','linewidth',3,'markersize',10)
   plot([x1(i) x1(i+1)], [y1(i) y1(i+1)], ...'g^-','linewidth',3,'markersize',10)
end
plot([x(i+1) x(g-i)], [y(i+1) y(g-i)], ...'r^-','linewidth',3,'markersize',10)
plot([x1(i+1) x1(g-i)], [y1(i+1) y1(g-i)], ...'g^-
','linewidth',3,'markersize',10)
legend ('Before Translation', 'After Translation')
xlabel('X-axis')
ylabel('y-axis')
title ('Translation 2D-shape')
```

The Output of program:

enter no. of object vertices: 4
enter Tx: 30
enter Ty: 30
enter x-value 10
enter y-value 20
enter x-value 30
enter y-value 20
enter x-value 30
enter y-value 50
enter x-value 10
enter y-value 50



2- Scaling Matrix

If a point P is $\begin{bmatrix} x \\ y \end{bmatrix}$ being a 2x1 vector. If we multiply it by 2x2 matrix

S= $\begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix}$

We will obtain another 2x1 vector which we can interpret as another point: P'= S . P

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} . \begin{bmatrix} x \\ y \end{bmatrix}$$

What will happen if we transfer every point by means of multiplication by S and display the result:

1- If S is the Identity matrix: S= $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ No change

2- If S= $\begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}$ then

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} . \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 2x \\ y \end{bmatrix}$$

That mean:

- every new x coordinate would be twice as large as the old value of vertical lines.
- x coordinate would be twice as width and the same tall.

3- If S= $\begin{bmatrix} 0.5 & 0 \\ 0 & 1 \end{bmatrix}$ shrink all x coordinate (shrink the width with the same tall)

Example 9: Stretch the image/object to twice and then compress it to one half of the new width?

P'= (S_1S_2). P

S_1S_2 $=\begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}\begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}=\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\cdot\begin{bmatrix} x \\ y \end{bmatrix}$

identity matrix then **no change**.

3.Rotation Matrix

There are **two types of Rotation:**

1. Anti-clockwise direction :

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} \times \begin{bmatrix} \cos(x) & \sin(x) & 0 \\ -\sin(x) & \cos(x) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

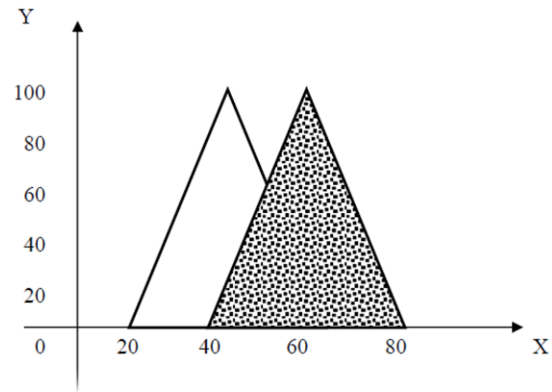
2. Clockwise direction :

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} \times \begin{bmatrix} \cos(x) & -\sin(x) & 0 \\ \sin(x) & \cos(x) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Example 10 : Consider a triangle defined by it three vertices (40 , 100), (20 , 0), (60 , 0) be translated 20 units to the right, using matrix representation.

Solution

$$\begin{bmatrix} 40 & 100 & 1 \\ 20 & 0 & 1 \\ 60 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 20 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 60 & 100 & 1 \\ 40 & 0 & 1 \\ 80 & 0 & 1 \end{bmatrix}$$



4.Reflection Matrix

There are different types of Reflection:

1 - Reflection about X – axis:

$$[x' \quad y' \quad 1] = [x \quad y \quad 1] \times \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

2-Reflection about Y – axis:

$$[x' \quad y' \quad 1] = [x \quad y \quad 1] \times \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

3 - Reflection about the origin (0, 0):

$$[x' \quad y' \quad 1]=[x \quad y \quad 1]\times\begin{bmatrix}-1 & 0 & 0\\ 0 & -1 & 0\\ 0 & 0 & 1\end{bmatrix}$$

4 - Reflection about the line y = x :

$$[x' \quad y' \quad 1]=[x \quad y \quad 1]\times\begin{bmatrix}0 & 1 & 0\\ 1 & 0 & 0\\ 0 & 0 & 1\end{bmatrix}$$

Example 11: Reflect the shape (20, 70), (40, 50), (60, 70), (40, 90), about:

1- X – axis
2- Y- axis
3- origin (0,0)
4- y = x

by used matrix representation, and draw the result.

Solution:

1- X – axis:

x'=x

y'=-y

$$\begin{bmatrix}20 & 70 & 1\\ 40 & 50 & 1\\ 60 & 70 & 1\\ 40 & 90 & 1\end{bmatrix}\times\begin{bmatrix}1 & 0 & 0\\ 0 & -1 & 0\\ 0 & 0 & 1\end{bmatrix}=\begin{bmatrix}20 & -70 & 1\\ 40 & -50 & 1\\ 60 & -70 & 1\\ 40 & -90 & 1\end{bmatrix}$$

2- Y- axis:

x'=-x

y'=y

$$\begin{bmatrix}20 & 70 & 1\\ 40 & 50 & 1\\ 60 & 70 & 1\\ 40 & 90 & 1\end{bmatrix}\times\begin{bmatrix}-1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix}=\begin{bmatrix}-20 & 70 & 1\\ -40 & 50 & 1\\ -60 & 70 & 1\\ -40 & 90 & 1\end{bmatrix}$$

3- origin (0,0):

x'=-x

y'=-y

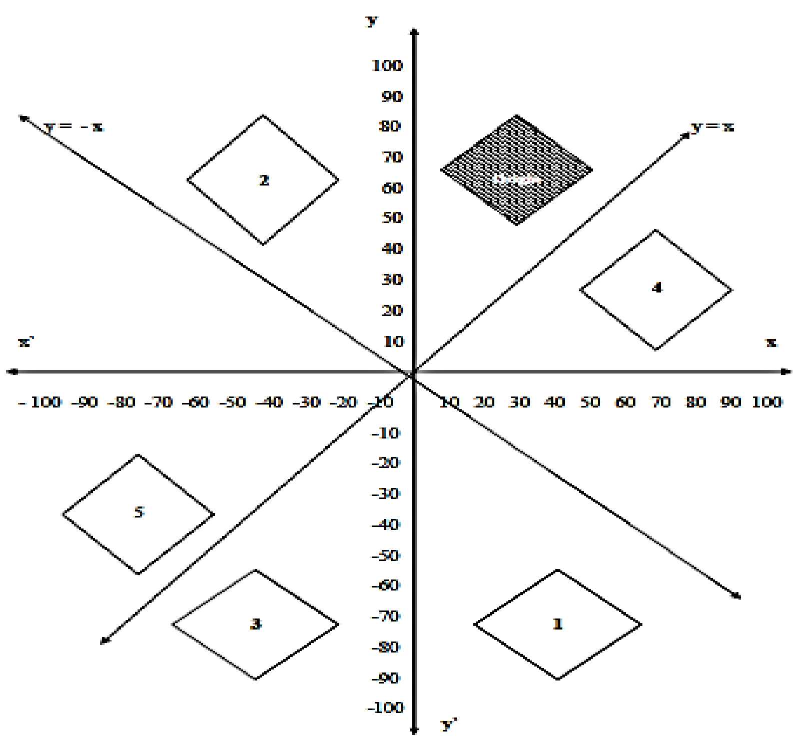
$$\begin{bmatrix} 20 & 70 & 1 \\ 40 & 50 & 1 \\ 60 & 70 & 1 \\ 40 & 90 & 1 \end{bmatrix} \times \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -20 & -70 & 1 \\ -40 & -50 & 1 \\ -60 & -70 & 1 \\ -40 & -90 & 1 \end{bmatrix}$$

4.y = x

x'=y

y'=x

$$\begin{bmatrix} 20 & 70 & 1 \\ 40 & 50 & 1 \\ 60 & 70 & 1 \\ 40 & 90 & 1 \end{bmatrix} \times \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 70 & 20 & 1 \\ 50 & 40 & 1 \\ 70 & 60 & 1 \\ 90 & 40 & 1 \end{bmatrix}$$



3.4 2D Viewing

Viewing is the process of drawing a view of a model on a 2-dimensional display. The geometric description of the object or scene provided by the model, is converted into a set of graphical primitives, which are displayed where desired on a 2D display.The same abstract model may be viewed in many different ways:
e.g. faraway, near, looking down, looking up.

3.4.1 Real World Coordinates

It is logical to use dimensions which are appropriate to the object for example:

- meters for buildings
- nanometers or microns for molecules, cells, atoms
- light years for astronomy

The **objects** are described with respect to their actual physical size in the **real world**, and then mapped onto screen co-ordinates. It is therefore possible to view an object at various sizes by zooming in and out, without actually having to change the model.

3.4.2 How do we convert Real-world coordinates into screen coordinates?

We could have a model of a whole room, full of objects such as chairs, tablets and students.We may want to view the whole room in one go, or zoom in on one single object in the room. We may want to display the object or scene on the full screen, or we may only want to display it on a portion of the screen.Once a model has been constructed, the programmer can specify a view.**2-Dimensional view consists of *two* rectangles**:

1. A ***Window***, given in **real-world** coordinates, which defines the portion of the model that is to be drawn.

2. A ***Viewport*** given in **screen** coordinates,which defines the portion of the screen on which the contents of the window will be displayed.
Figure 3. show the window and viewport.

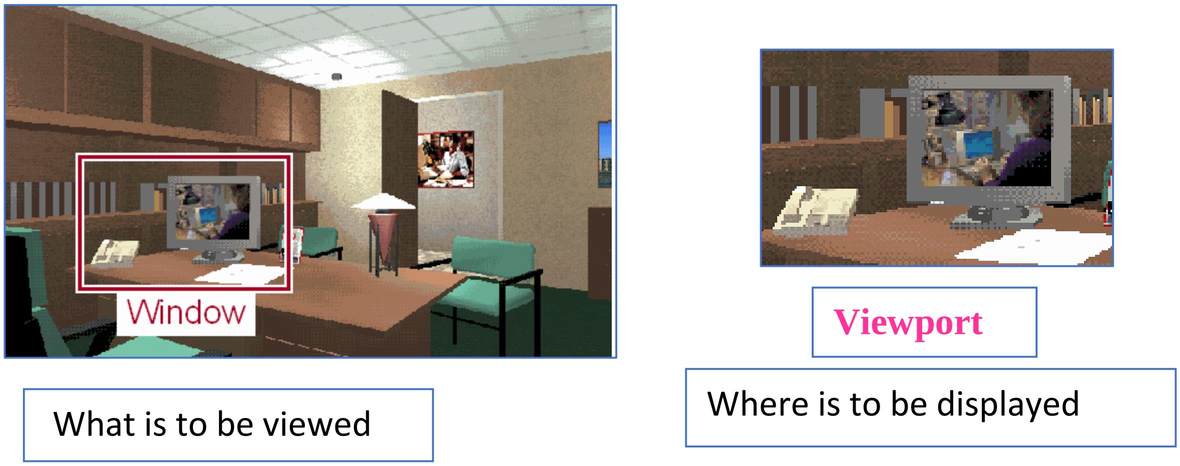


Figure 3.13 the window and viewport.

3.5 Window to Viewport Transformation (Mapping)

The window to viewport mapping is a process of transforming or mapping the two dimensional or world coordinate view into device coordinate. The object which is available inside of the clipping window or world is mapped into the viewport and is displayed on the interface window screen, or the clipping window selects the piece of the scene from the display and view port positions it in the output device.The following figure 3.14 show Window to Viewport Mapping.

Window:

1- A world-coordinate area selected for display is called awindow.

2- In computer graphics, a window is a graphical control element.

3- It consists of a visual area containing some of the graphicaluser interface of the program it belongs to and is framed by awindow decoration.

4- A window defines a rectangular area in world coordinates. Youcan define the window to be larger than, the same size as, orsmaller than the actual range of data values, depending onwhether you want to show all of the data or only part of thedata.

5- Window defines what is to be viewed .

Viewport:

1- An area on a display device to which a window is mapped iscalled a viewport.

2- A viewport is a polygon viewing region in computer graphics.The viewport is an area expressed in rendering-device-specificcoordinates, e.g. pixels for screen coordinates, in which theobjects of interest are going to be rendered.

3- A viewport defines in normalized coordinates a rectangulararea on the display device where the image of the data appears.You can have your graph take up the entire display device orshow it in only a portion, say the upper-right part.

4- Viewport defines where the window to be displayed.

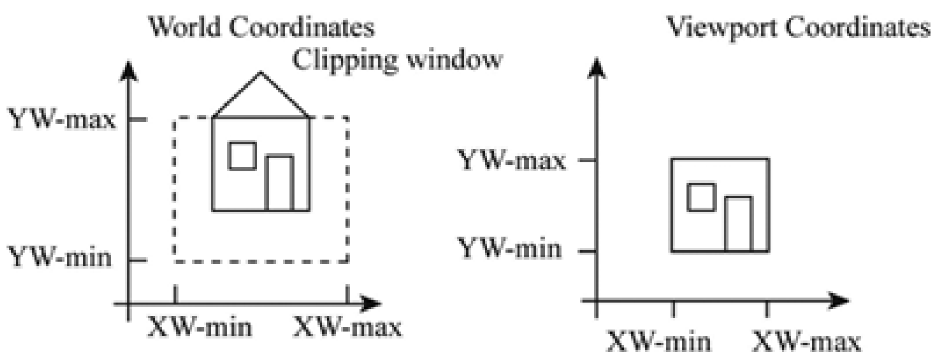
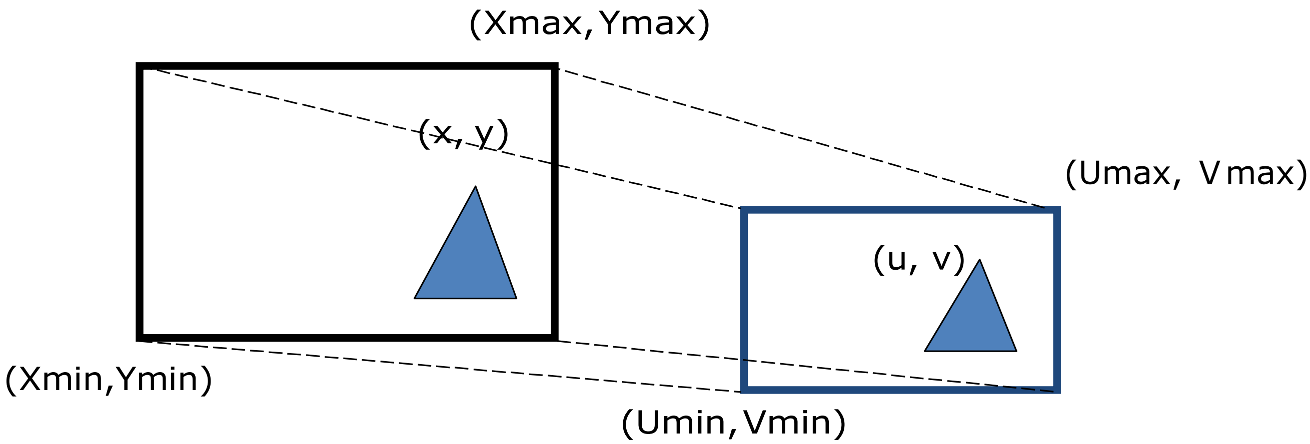


Figure 3.14 Window to Viewport Mapping

This transformation involves developing formulas that start with a point in the world window, say (x, y).



The formula for window to viewport mapping is:

$$(x, y) \longrightarrow (u, v)$$

$$\frac{x - x_{min}}{x_{max} - x_{min}} = \frac{u - u_{min}}{u_{max} - u_{min}}$$

$$\frac{y - y_{min}}{y_{max} - y_{min}} = \frac{v - v_{min}}{v_{max} - v_{min}}$$

By rewriting this relationship, we get the following formula:

$u = c_1 x + c_2$ $\qquad c_1 = \frac{u_{max} - u_{min}}{x_{max} - x_{min}}$

$c_2 = u_{min} - c_1 x_{min}$

$v = d_1 y + d_2$ $\qquad d_1 = \frac{v_{max} - v_{min}}{y_{max} - y_{min}}$

$d_2 = v_{min} - d_1 y_{min}$

Example 12: A normalized window has left and right boundaries of (-0.05 to +0.05) and lower and upper boundaries of (0.1 to 0.2). the viewport window left and right is (250,550) and lower to upper is (100,400),find the coordinate of any point (u,v) in the viewport window.

Solution :
Window(xmin=-0.05 , xmax=+0.05 , ymin=0.1, ymax=0.2)
Viewport (umin=250, umax=550, vmin=100, vmax=400)

$$u = c_1\, x + c_2$$

$$c_1 = \frac{umax - umin}{xmax - xmin}$$

$$c_1 = \frac{(550 - 250)}{0.05 - (-0.05)} = 300/0.1 = 3000$$

$$c_2 = u_{min} - c_1 x_{min}$$

=250-3000(-0.05) =250+150 =400

u=3000x+400

$$v = d_1 y + d_2$$

$$d_1 = \frac{vmax - vmin}{ymax - ymin}$$

$$d_1 = \frac{(400 - 100)}{(0.2 - 0.1)} = 300/0.1 = 3000$$

$$d_2 = v_{min} - d_1 y_{min}$$

=100-3000(0.1)
=-200

v =3000y-200

3.6 Window to Viewport Transformation N

We can express these two formula for computing (u,v) from (x,y) by term:

(translate-scale-translate)

$$\begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{v} \\ \boldsymbol{1} \end{bmatrix} = \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \\ \boldsymbol{1} \end{bmatrix} . \boldsymbol{N}$$

$N = T_2 \quad S \quad T_1$

1. T_1 is the translation matrix about window origin :

$$T_1 = \begin{bmatrix} 1 & 0 & -x_{min} \\ 0 & 1 & -y_{min} \\ 0 & 0 & 1 \end{bmatrix}$$

2. is the scaling transformation matrix:

$$S = \begin{bmatrix} \frac{u_{max} - u_{min}}{x_{max} - x_{min}} & 0 & 0 \\ 0 & \frac{v_{max} - v_{min}}{y_{max} - y_{min}} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

3. T2 is the translation matrix position of the viewport :

$$T_2 = \begin{bmatrix} 1 & 0 & u_{min} \\ 0 & 1 & v_{min} \\ 0 & 0 & 1 \end{bmatrix}$$

Example 13: A normalized window has left and right boundaries of (-0.05 to +0.05) and lower and upper boundaries of (0.1 to 0.2). the viewport window left and right is (250,550) and lower to upper is (100,400),find the transformation N.

Solution N=T_2ST_1

$$T_1 = \begin{bmatrix} 1 & 0 & -x_{min} \\ 0 & 1 & -y_{min} \\ 0 & 0 & 1 \end{bmatrix}$$

$$S = \begin{bmatrix} \frac{u_{max}-u_{min}}{x_{max}-x_{min}} & 0 & 0 \\ 0 & \frac{v_{max}-v_{min}}{y_{max}-y_{min}} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$T_1 = \begin{bmatrix} 1 & 0 & -(-0.05) \\ 0 & 1 & -0.1 \\ 0 & 0 & 1 \end{bmatrix}$$

$$S = \begin{bmatrix} 3000 & 0 & 0 \\ 0 & 3000 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$T_2 = \begin{bmatrix} 1 & 0 & u_{min} \\ 0 & 1 & v_{min} \\ 0 & 0 & 1 \end{bmatrix}$$

$$T_2 = \begin{bmatrix} 1 & 0 & 250 \\ 0 & 1 & 100 \\ 0 & 0 & 1 \end{bmatrix}$$

$$N = \begin{bmatrix} 1 & 0 & 250 \\ 0 & 1 & 100 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 3000 & 0 & 0 \\ 0 & 3000 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -(-0.05) \\ 0 & 1 & -0.1 \\ 0 & 0 & 1 \end{bmatrix}$$

3.7 Clipping and windowing

Many graphics application programs give the user the impression of looking through a window at a very large picture.
To display an enlarged portion of a picture we must not only apply the appropriate scaling and translation but identify the visible parts of the picture for inclusion in the displayed image. The correct way to select visible information for display is to use **clipping** (a process which divides each element of the picture into its visible and invisible portions, allowing the invisible portion to be discarded) . Clipping can be applied to a variety of different types of picture elements:
vectors, curves of various kinds, and even polygons. The basis for these clipping operations is a simple pair of inequalities that determine whether a point (x,y) is visible or not.

$$xleft \leq x \leq xright\ ,\ ybottom \leq y \leq ytop$$

Where xleft, xright, ybottom, ytop are the positions of the edges of the screen. These inequalities provide us with a very simple method of clipping pictures on a point by point basis; we substitute the coordinates of each point for x and y and if the point fails to satisfy either inequality; it is invisible. It would be quite inappropriate to clip pictures by converting all picture elements into points and using these inequalities; the clipping process would take far too long and would leave the picture in a form no longer suitable for a line drawing display. We must attempt to clip larger elements of the picture. This involves developing more powerful clipping algorithms that can be determine the visible and invisible portions of such picture elements.

3.7.1 Clipping window

It is refer to a rectangular region whose sides are aligned with the coordinates axes. The x extent is measured from xmin to xmax and the y extent is measured from ymin to ymax.

3.7.2 Point clipping

The basis for these clipping operations is a simple pair of inequalities that determine whether a point (x,y) is visible or not:

$$x_{min} \leq x \leq x_{max}\ ,\ y_{min} \leq y \leq y_{max}$$

Where x_{min}, x_{max}, y_{min}, y_{max} are the positions of the edges of the window.

3.7.3 Line clipping

Lines that do not intersect the clipping window are either completely inside the window or completely outside the window.

On the other hand a line that intersects the clipping window is divided by the intersection point (s) into segments that are either inside or outside the window. The following algorithm provide efficient way to decide the relationship between an arbitrary line and the clipping window to find intersection point (s).Figure 3.15 show the type of line clipping.

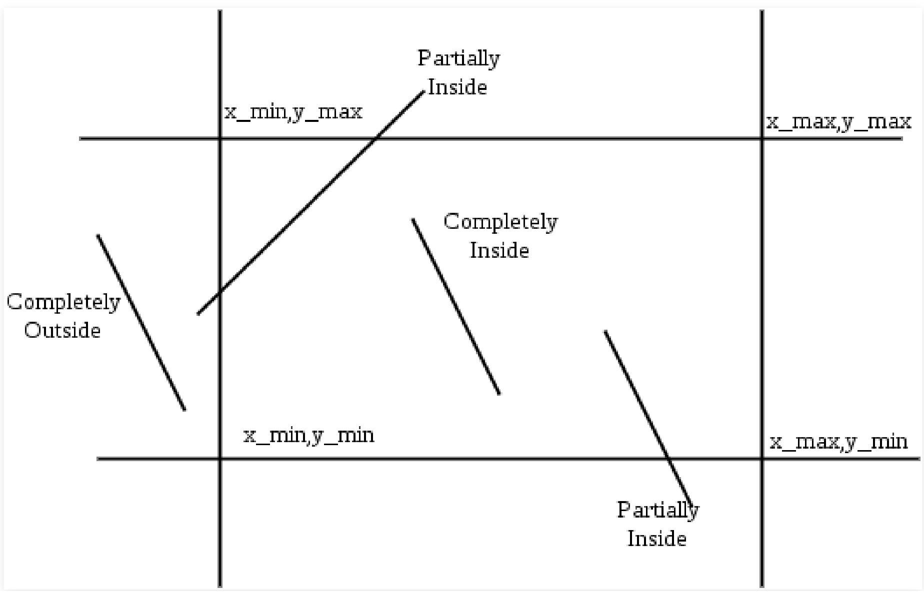
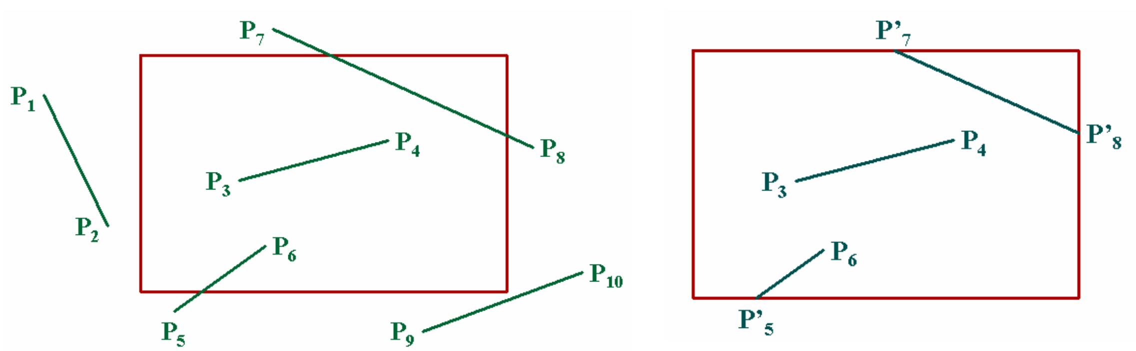


Figure 3.15 the type of line clipping.

Figure 3.16 show Possible relationship between line position and a standard clipping region.



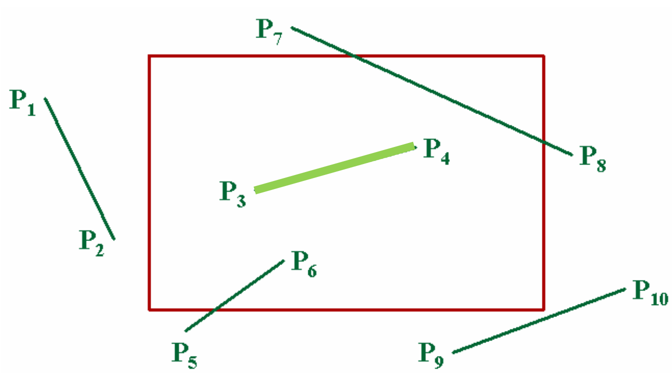
Before Clipping **After Clipping**

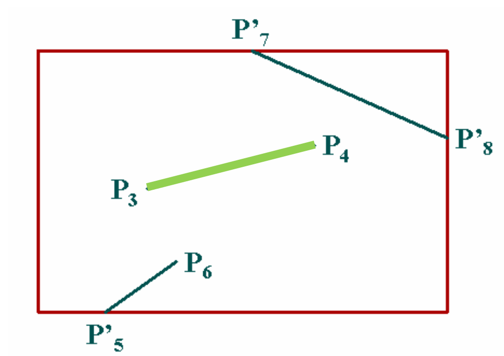
Figure 3.16 Possible relationship between line position and a standard clipping region.

A line clipping procedure involves several parts:

1. Determine whether line lies completely inside the clipping window.
2. Determine whether line lies completely outside the clipping window.
3. Perform intersection calculation with one or more clipping boundaries.

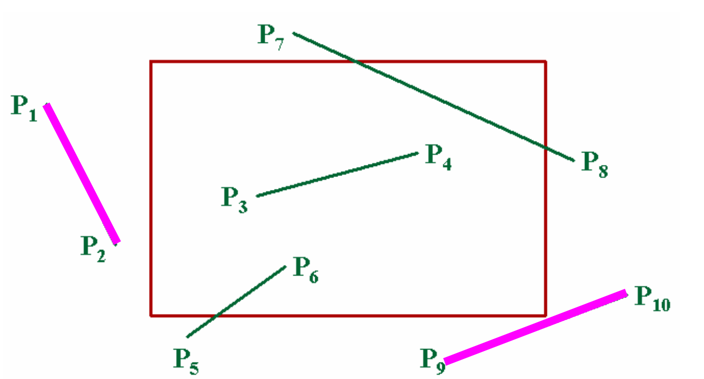
A line with both endpoints inside all clipping boundaries is saved ($\overline{P_3P_4}$)

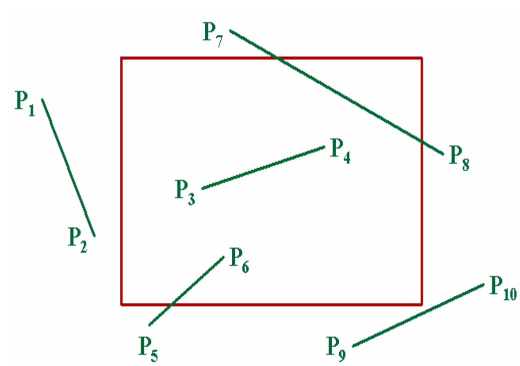




Before Clipping **After Clipping**

A line with both endpoints outside all clipping boundaries is **reject** ($\overline{P_1P_2}$& $\overline{P_9P_{10}}$)

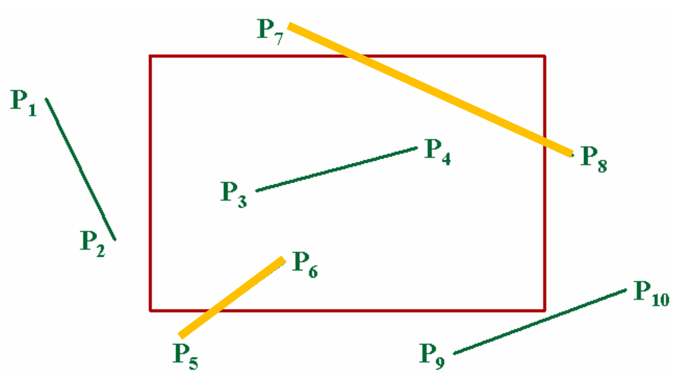




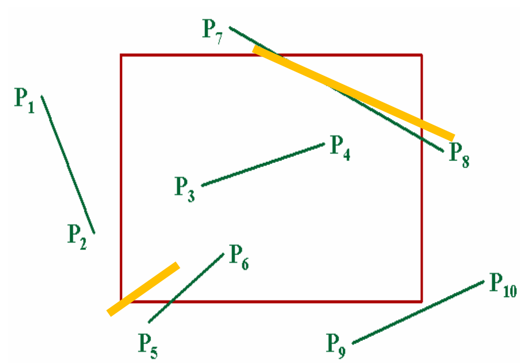
Before Clipping **After Clipping**

If one or both endpoints outside the clipping rectangular, the **parametric representation** could be used to determine values of parameter **u** for intersection with the clipping boundary coordinates.

$$\begin{cases} x = x_1 + u(x_2 - x_1) \\ y = y_1 + u(y_2 - y_1) \end{cases} \quad 0 \le u \le 1$$



Before Clipping



After Clipping

1. If the value of u is outside the range 0 to 1: The line dose not enter the interior of the window at that boundary.

2. If the value of u is within the range 0 to 1, the line segment does cross into the clipping area.

Clipping line segments with these parametric tests requires a good deal of computation, and faster approaches to clipper are possible.

3.7.4 The Cohen–Sutherland algorithm

The **Cohen–Sutherland algorithm** is a computer-graphics algorithm used for line clipping**.**

The Cohen–Sutherland algorithm can be used only on a rectangular clip window.

Given a set of lines and a rectangular area of interest, the task is to remove lines which are outside the area of interest and clip the lines which are partially inside the area.

Cohen-Sutherland algorithm divides a two-dimensional space into 9 regions and then efficiently determines the lines and portions of lines that are inside the given rectangular area.Figure 3.17 show the 9 regions of Cohen-Sutherland algorithm

The algorithm can be outlines as follows:-

Nine regions are created, eight "outside" regions and one "inside" region.

For a given line extreme point (x, y), we can quickly find its region's four bit code. Four bit code can be computed by comparing x and y with four values (x_min, x_max, y_min and y_max).

If x is less than x_min then bit number 1 is set.

If x is greater than x_max then bit number 2 is set.

If y is less than y_min then bit number 3 is set.

If y is greater than y_max then bit number 4 is set.

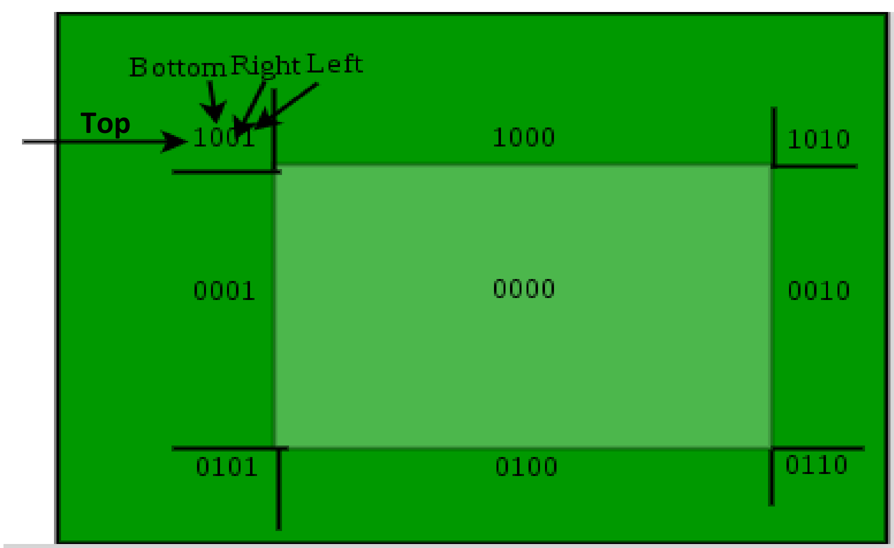


Figure 3.17 the 9 regions of Cohen-Sutherland algorithm

The diagram on the above page is associated with the checking order TBRL, corresponding to Top, Bottom, Right, Left. We assign a 1-bit where the region is strictly outside the boundary (in the half-plane not containing the window), and a 0-bit where the region is on the same side as the window. Thus, only the window itself is assigned all zeros. Since the high-order bit is associated with the top boundary for example, only the three regions above the window (outside the top boundary) have high-order bit equal to 1.

There are three possible cases for any given line:

1. Completely inside the given rectangle : Bitwise OR of regionof two end points of line is 0 (Both points are inside therectangle)

2. Completely outside the given rectangle : Both endpointsshare at least one outside region which implies that the linedoes not cross the visible region. (bitwise AND of endpoints !=0).

3. Partially inside the window : Both endpoints are in differentregions. In this case, the algorithm finds one of the two pointsthat is outside the rectangular region. The intersection of theline from outside point and rectangular window becomes newcorner point and the algorithm repeats.

3.7.5 Intersection points

Intersection points with a clipping boundary can be calculated using the slop-intercept form of the line equation. For a line with endpoint coordinates(x1, y1) and (x2, y2), the y coordinate of the intersection point with a **vertical boundary** can be obtained with the calculation

$$y = y1 + m(x - x1)$$

Where the x value is set either to x_{min} or to x_{max}, and the slop of the line is calculated as

$$m = (y2 - y1) / (x2 - x1).$$

Similarly, if we are looking for the intersection with a **horizontal boundary**, the x coordinate can be calculated as

$$x = x1 + (y - y1) / m$$

Note

1. If the boundary line is vertical then:

$x=x_{min}$ if the line is left

$x=x_{max}$ if the line is right

$\mathbf{y = y1 + m(x - x1)}$

2. If the boundary line is horizontal then:

$y=y_{min}$ if the line is bottom

$y=y_{max}$ if the line is top

$\mathbf{x = x1 + (y - y1) / m}$

Example 14 : Apply the Cohen Sutherland line clipping algorithm to clip the line segment with coordinates (30,60) and (60,25) against the window with (X_{min},Y_{min})= (10,10) and (X_{max},Y_{max})= (50,50).

Solution

Clip bit code

AB 1000 AND

0010

0000 (Partially inside) (clipping)

First ,Find the slop of line AB from the equation:

m = (y2 – y1) / (x2 – x1)

m=(25-60)/(60-30)

=-35/30

=-1.16

Then ,We find the coordinate of intersection point from line A A**-**.

The boundary line A A**-** is horizontal ,so Y_{max}=y=50 and calculate xvalue from this :

x = x1 + (y – y1) / m

= 30+(50-60)/-1.16

=30+-10/-1.16

= 30+8.6

=38.6

the coordinate of intersection point is **A-(38.6,50)**.

We find the coordinate of intersection point form line BB-.

The boundary line BB- is vertical ,so xmax=x=50 and calculate y valuefrom this:

y = y1 + m(x – x1)

=25+(-1.16)(50-60)

= 25+11.6

=36.6

The coordinate of intersection point is B-(50,36.6).

Example 15: Window is defined A(10,20),B(20,20),C(20,10),D(10,10) Find visible portion of line P(15,15),Q(5,5) using Cohen Sutherland line clipping algorithm.

<u>Solution</u>

Clip bit code

PQ 0000 AND

0101

0000 Partially inside (clipping)

Find the slop of line PQ

m = (y2 – y1) / (x2 – x1)

=(5-15)/(5-15)

=-10/-10=1

We fined the coordinate of intersection point from line PP -,

The boundary line PP- is horizontal ,so Ymin=y=10 and find x as follow:

x = x1 + (y – y1) / m

=15+(10-15)/1

=15-5

= 10

the coordinate of intersection point is P-(10,10).

Example 16: Window is defind A(20,20),B(90,20),C(90,70),D(20,70) Find visible portion of
line1 :P1(10,30),P2(80,90)
Line2: Q1(20,10) , Q2(70,60)
using Cohen Sutherland line clipping algorithm.

<u>Solution</u>

Xmin=20 , Xmax=90 , ymin=20 , ymax=70

Clip bit code

P1P2 0001 AND
1000

0000 **(Partially inside) (clipping)**

Q1Q2 0101 AND
0000

0000 **(Partially inside) (clipping)**

First find the slop of line P1P2 from the equation:

m = (y2 – y1) / (x2 – x1)

=(90-30)/(80-10)

=60/70=0.8

Then find the coordinate of intersection point from line P1P1**-**.

The boundary line P1P1**-** is vertical ,so Xmin=x=20 and calculate yvalue from this :

y = y1 + m(x – x1)

=30+0.8(20-10)

=30+8=38

the coordinate of intersection point **P1-(20,38).**

Then find the coordinate of intersection point from line P2P2**-** .

The boundary line P2P2**-** is horizontal ,so ymax=y=70 and find x fromthis equation: **x = x1 + (y – y1) / m**

=80+(70-90)/0.8

=80+(-20)/0.8

=80+(-25)=55

the coordinate of intersection point **P2-(55,70).**
Find the slop of second line Q1Q2
m = (y2 – y1) / (x2 – x1)
=(60-10)/(70-20)=50/50=1
Then find the coordinate of intersection point from line Q1Q1-
The boundary line Q1Q1- is horizontal ,so ymin=y=20 and calculate xvalue from this :
x = x1 + (y – y1) / m
=20+(20-10)/1
=20+10=30
The coordinate of intersection point is Q1- (30,20)

Example 17: Rectangular area of interest (defined by below four values which are coordinates of bottom left and top right)
Xmin=4,ymin=4,xmax=10,ymax=8
A set of lines(defined by two corner coordinates)
Line 1: A(5,5), B(7,7)
Line 2: C(7,9), D(11,4)
Line 3: E(1,5), F(3,2)
Apply the Cohen Sutherland line clipping algorithm to clip the line segment.

Solution:
Clip bit code AB 0000 OR
0000
____
0000 accept (inside)
CD 1000 AND
0110
____
0000 partially inside (clipping)

EF 0001 AND
0101

0001 reject (outside)

Find slop for line CD as follow:
m = (y2 – y1) / (x2 – x1)
=(4-9)/(11-7)
=-5/4=-1.25
We fined the coordinate of intersection point from line CC-,
The boundary line CC- is horizontal ,so Ymax=y=8 and find x as follow:
x = x1 + (y – y1) / m
= 7+(8-9)/-1.25
=7+-1/-1.25
7+0.8=7.8
The coordinate of intersection point is C-(7.8,8).
We fined the coordinate of intersection point form line DD-,
the boundary line DD- is vertical ,so xmax=x=10 and find y as follow:
y = y1 + m(x – x1)
=4+(-1.25)(10-11)
=4+1.25
=5.25
The coordinate of intersection point is D -(10,5.25).