UNIT-2

✳ Transformation ⇒

In a graphics system, the effects that we desire include changing the size of the object, its position on the screen etc. The implementation of such a change is called transformation.

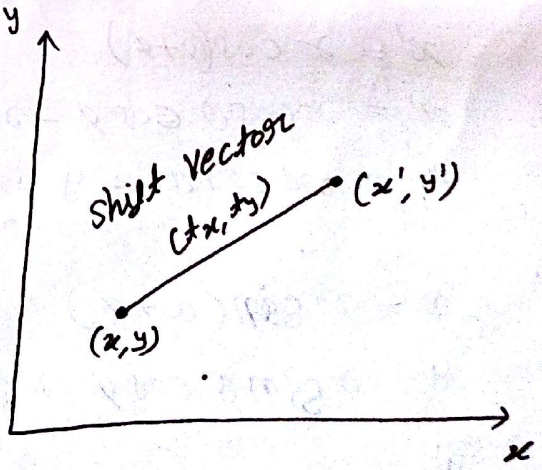
The various transformations possible on an object are as follows:

1) Rotation
2) Translation
3) Scaling
4) Reflection
5) Shearing

1) Translation :-

In translation, an object is repositioning along a straight line path from one coordinate location to another. The translation distance pair (t_x, t_y) is called translation vector or shift vector.

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



Matrix Representation -

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

$$\boxed{P' = P + T}$$

Filename: Anoop Resume.docx
Directory: H:
Template: C:\Users\pradeep sharma\AppData\Roaming\Microsoft\Templates\Normal.dotm
Title:
Subject:
Author: anoop
Keywords:
Comments:
Creation Date: 09/10/2012 7:43:00 PM
Change Number: 35
Last Saved On: 01/10/2016 12:53:00 PM
Last Saved By: Prince-PC
Total Editing Time: 258 Minutes
Last Printed On: 08/10/2016 1:58:00 PM
As of Last Complete Printing
Number of Pages: 2
Number of Words: 466 (approx.)
Number of Characters: 2,658 (approx.)

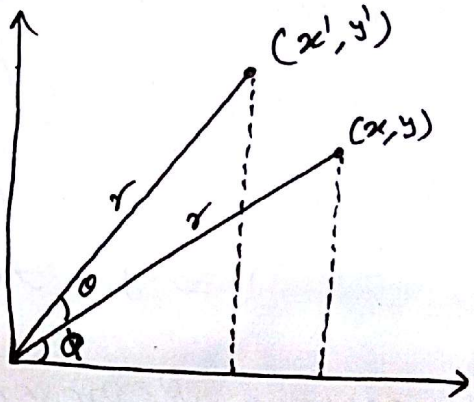
---

2) Rotation :-

In rotation, an object is repositioning along a circular path. The rotation point (Pivat point) has a rotation angle (θ) and position (x, y); about which object is to be rotated.

The original coordinate of the point -

$$x = r\cos\phi$$
$$y = r\sin\phi$$



$$x' = r\cos(\theta+\phi)$$
$$x' = r\cos\theta\cos\phi - r\sin\theta\sin\phi$$
$$x' = x\cos\theta - y\sin\theta$$

$$y' = r\sin(\theta+\phi)$$
$$y' = r\sin\theta\cos\phi + r\sin\phi\cos\theta$$
$$y' = x\sin\theta + y\cos\theta$$

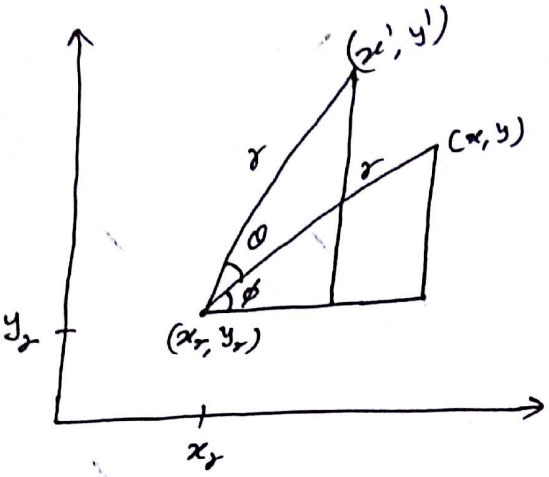
Matrix Representation :-

$$P' = R \cdot P$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

* Rotation about any specified Position (x_r, y_r) –

$$x' = x_r + (x - x_r)\cos\theta - (y - y_r)\sin\theta$$
$$y' = y_r + (x - x_r)\sin\theta + (y - y_r)\cos\theta$$



3) Scaling :-

A scaling transformation alters the size of object. This operation can be carried out by Scaling factor S_x & S_y; where S_x scale the object in x-direction & S_y scale the object in y-direction.

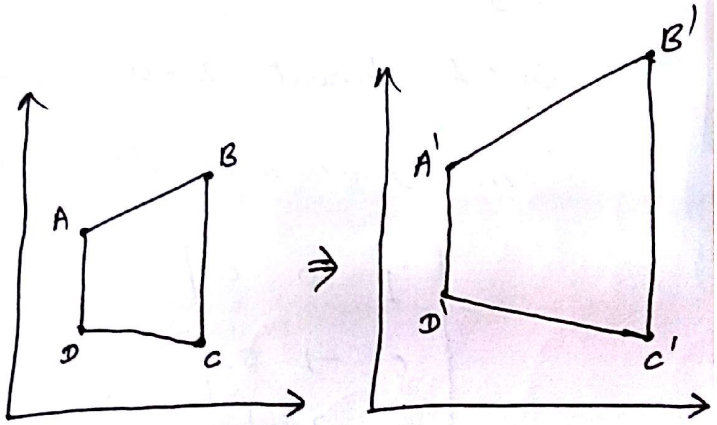
To get

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

Matrix Representation –

$$P' = S\,P$$

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



An positive numeric value can be assigned to the scaling factor S_x & S_y.

If $S_x = S_y \Rightarrow$ Uniform Scaling

If $S_x \neq S_y \Rightarrow$ Different Scaling

$r_x = 6$

$r_y = 6$

Filename: download_Application_format_final_06092016_052516PM.doc
Directory: C:\Users\pradeep sharma\Documents
Template: C:\Users\pradeep sharma\AppData\Roaming\Microsoft\Templates\Normal.dotm
Title: 1
Subject:
Author: Dr.N.V.Vasani
Keywords:
Comments:
Creation Date: 29/06/2007 2:24:00 PM
Change Number: 64
Last Saved On: 06/09/2016 5:28:00 PM
Last Saved By: est-server
Total Editing Time: 139 Minutes
Last Printed On: 08/10/2016 1:56:00 PM
As of Last Complete Printing
Number of Pages: 3
Number of Words: 338 (approx.)
Number of Characters: 1,933 (approx.)

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

---

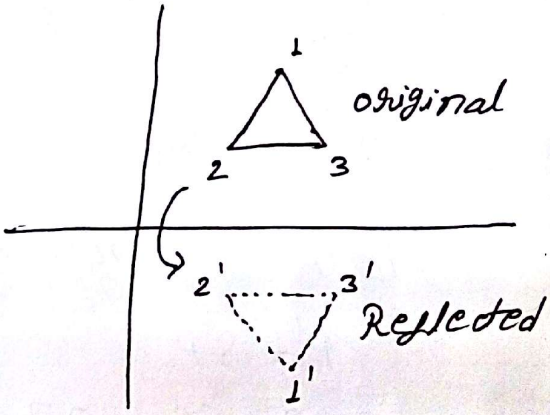
4) Reflection :-

A reflection is a transformation that produce a mirror image of an object. The mirror ~~for~~ is generated by rotating the object 180° about the reflection axis.

Case 1: About x-axis ($y = 0$) :-

The transformation matrix is -

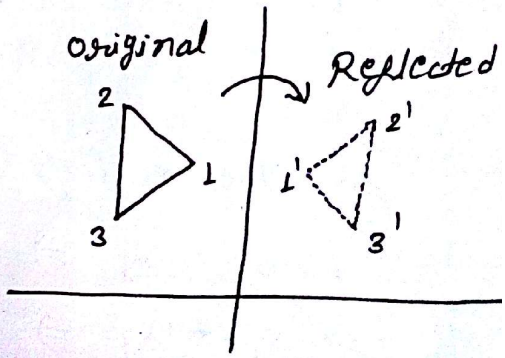
$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



case 2: About y-axis ($x = 0$) :-

The transformation matrix is -

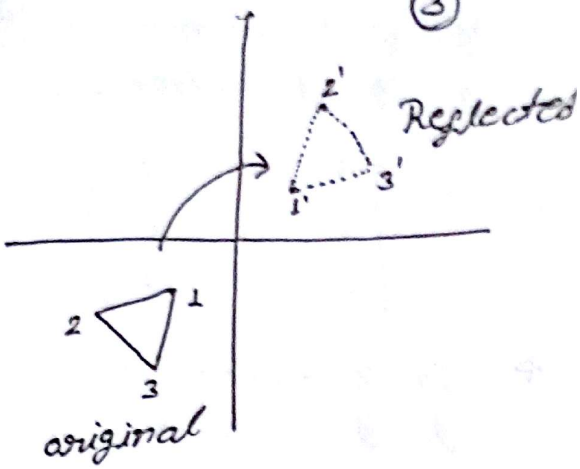
$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



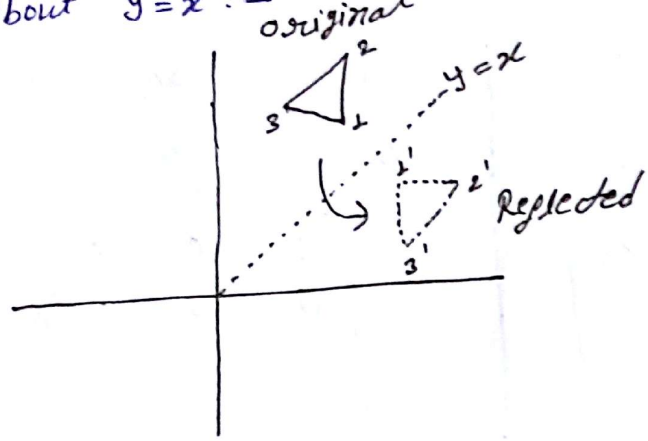
Case 3: About origin :-

The transformation matrix is-

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



case 4: About $y = x$:-



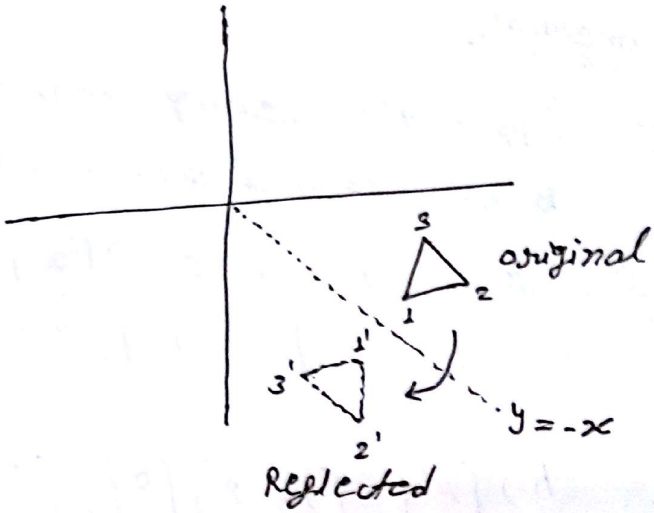
The transformation matrix is-

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

case 5: About $y = -x$:-

The transformation matrix is -

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



5) Shearing :-

Shearing is a transformation that changes the shape of an object.

x-direction relative to x-axis -

$$x' = x + Sh_x \cdot y$$
$$y' = y$$

y-direction relative to y-axis -

$$x' = x$$
$$y' = y + Sh_y \cdot x$$

The transformation matrix is -

In x-direction -

$$\begin{bmatrix} 1 & Sh_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

In y-direction -

$$\begin{bmatrix} 1 & 0 & 0 \\ Sh_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

★ In x-direction relative to line $y = y_{ref}$ -

$x' = x + Sh_x (y - y_{ref})$

$y' = y$

The transformation matrix -

$$\begin{bmatrix} 1 & Sh_x & -Sh_x y_{ref} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

★ In y-direction relative to the line $x = x_{ref}$ -

$x' = x$

$y' = y + Sh_y (x - x_{ref})$

The transformation matrix -

$$\begin{bmatrix} 1 & 0 & 0 \\ Sh_y & 1 & -Sh_y x_{ref} \\ 0 & 0 & 1 \end{bmatrix}$$

Numerical

Q. Apply the shearing transformation to square with A(0,0), B(1,0), C(0,1) & D(1,1) with shear parameter $Sh_x = 2$.

Sol^n :

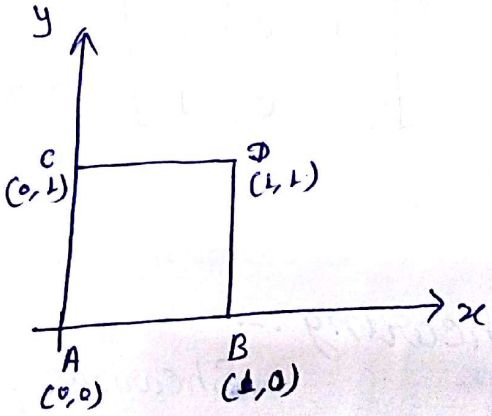
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & Sh_x \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

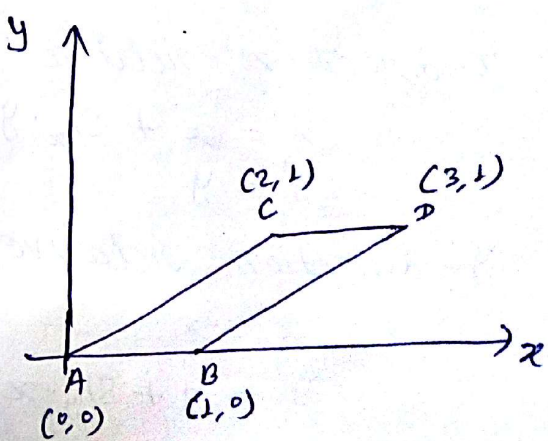
$$A \Rightarrow \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

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

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

$$D \Rightarrow \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ 1 \end{bmatrix}$$





* 3X3 Matrix Representation :-

1) Translation :-

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$P' = T(t_x, t_y) \cdot P$$

To perform ~~any~~ transformation of any object, object matrix 'X' is multiplied by the transformation matrix 'T'.

$$\begin{bmatrix} \text{Transformed} \\ \text{object} \end{bmatrix} = \begin{bmatrix} \text{Transformation} \\ \text{matrix} \end{bmatrix} X \begin{bmatrix} \text{original object} \\ \text{matrix} \end{bmatrix}$$

$$X' = T \cdot X$$

2) Rotation :-

Rotation about the origin are-

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$P' = R(\theta) \cdot P$$

3) Scaling :- Scaling relative to origin is -

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$P' = S(S_x, S_y) \cdot P$$

✳ Composite Transformation :-

The basic purpose of composite transformation is gain efficiency by applying a single composed transformation to a point, rather than applying a series of transformations one after the other.

1) Translation :-

If two successive translation vectors (t_{x_1}, t_{y_1}) & (t_{x_2}, t_{y_2}) are applied to a coordinate $P(x, y)$, the final position P'' is calculated as -

$$P' = T(t_{x_1}, t_{y_1}) \cdot P$$

$$P'' = T(t_{x_2}, t_{y_2}) \cdot P'$$

$$P'' = T(t_{x_2}, t_{y_2}) \cdot \{T(t_{x_1}, t_{y_1}) \cdot P\}$$

where $T(t_x, t_y)$

$$\Downarrow$$

$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_{x_1} + t_{x_2} \\ 0 & 1 & t_{y_1} + t_{y_2} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\boxed{P'' = T(t_{x_1} + t_{x_2}, t_{y_1} + t_{y_2}) \cdot P}$$

2) Rotation :-

Two successive rotation to point P.

$$P'' = R(\theta_2) \cdot \{R(\theta_1) \cdot P\}$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & 0 \\ \sin\theta_1 & \cos\theta_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta_2 & -\sin\theta_2 & 0 \\ \sin\theta_2 & \cos\theta_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta_1 + \theta_2) & -\sin(\theta_1 + \theta_2) & 0 \\ \sin(\theta_1 + \theta_2) & \cos(\theta_1 + \theta_2) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\boxed{P'' = R(\theta_1 + \theta_2) \cdot P}$$

3) Scaling :-

Two successive scaling to point P, produce the resultant transformed position as -

$$P'' = S'(S_{x_2}, S_{y_2}) \cdot \{S(S_{x_1}, S_{y_1}) \cdot P\}$$

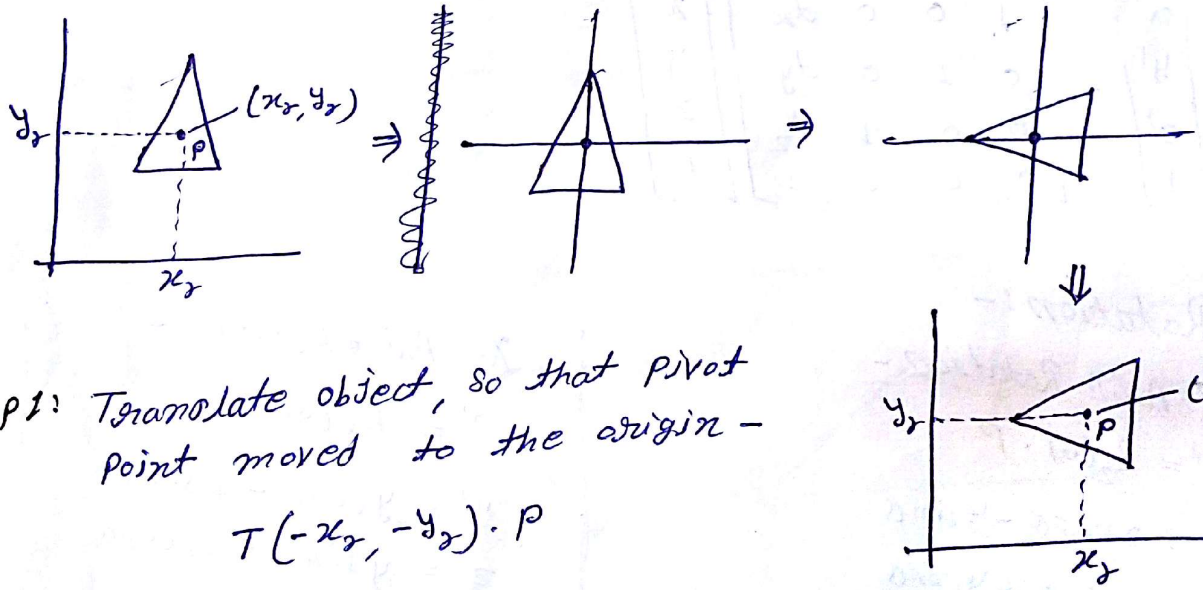
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} S_{x_2} & 0 & 0 \\ 0 & S_{y_2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} S_{x_1} & 0 & 0 \\ 0 & S_{y_1} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} S_{x_2} \cdot S_{x_1} & 0 & 0 \\ 0 & S_{y_2} \cdot S_{y_1} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\boxed{P'' = S(S_{x_2} \cdot S_{x_1}, S_{y_2} \cdot S_{y_1}) \cdot P}$$

✳ Pivot Point Rotation :-

We can generate rotations about any selected pivot point (x_r, y_r).



Step 1: Translate object, so that pivot point moved to the origin -

$$T(-x_r, -y_r) \cdot P$$

Step 2: Rotate the object about the origin -

$$R(\theta) \cdot T(-x_r, -y_r) \cdot P$$

Step 3: Translate the object, so that pivot point is returned to its original position -

$$P' = T(x_r, y_r) \cdot R(\theta) \cdot T(-x_r, -y_r) \cdot P$$

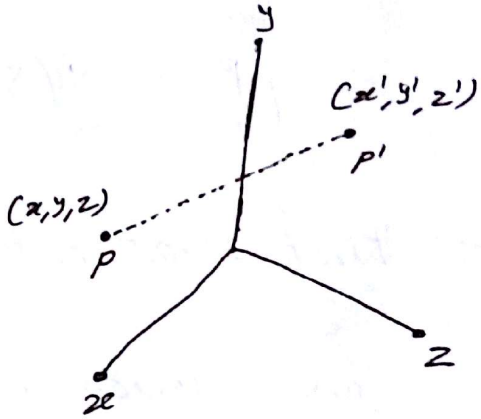
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & x_r \\ 0 & 1 & y_r \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_r \\ 0 & 1 & -y_r \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_r \\ y_r \\ 1 \end{bmatrix}$$

✳ 3-D Transformation :-

2D transformation can be represented by 3×3 matrix using homogeneous coordinates, so the 3D transformations ↓ by 4×4 matrices.
represented

1) Translation :-

Translate a point with translation vector $T = (t_x, t_y, t_z)$



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

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

2) Rotation :-

Z-Axis Rotation -

$$P' = R_z(\theta) \cdot P$$
$$x' = x\cos\theta - y\sin\theta$$
$$y' = x\sin\theta + y\cos\theta$$
$$z' = z$$

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

X- Axis Rotation -

$$P' = R_x(\theta) \cdot P$$
$$y' = y\cos\theta - z\sin\theta$$
$$z' = y\sin\theta + z\cos\theta$$
$$x' = x$$

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

∴ Y-Axis Rotation -

$$P' = R_y(\theta) \cdot P$$

$$z' = z\cos\theta - x\sin\theta$$
$$x' = z\sin\theta + x\cos\theta$$
$$y' = y$$

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3) Scaling :-

The scaling transformation relative to the origin, where scaling parameters S_x, S_y, S_z.

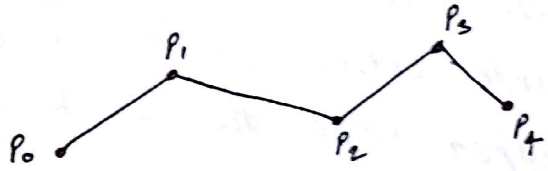
$$x' = S_x \cdot x$$
$$y' = S_y \cdot y$$
$$z' = S_z \cdot z$$
$$P' = S(S_x, S_y, S_z) \cdot P$$

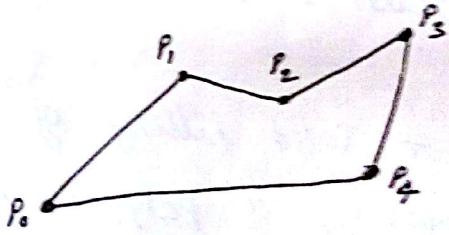
$$S(S_x, S_y, S_z) = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

* POLYGON :-

A polygon can be represented as a group of connected edges, forming a close figure. The polygon may be of any shape.

A polyline is a chain of connected line segments, when starting point and terminal point of any polyline is same i.e. when polyline is closed then it is called polygon.





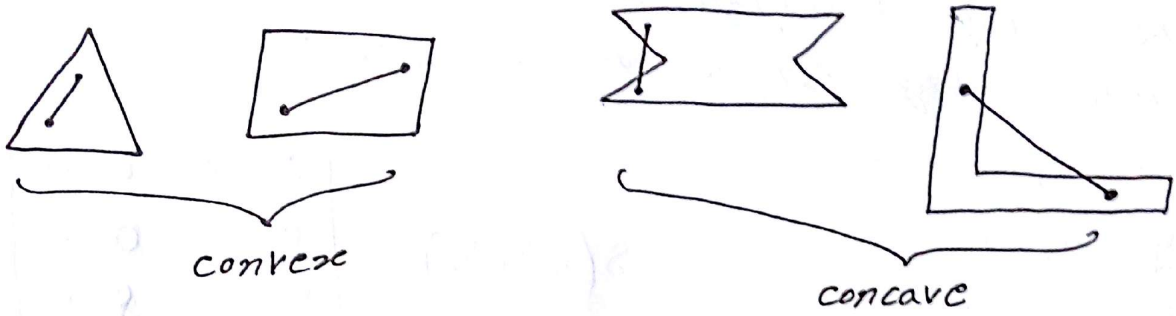
Types of Polygon :-

This classification is based on where the line segment joining two points, is going to lie.

1) Convex
2) Concave

A Polygon is said to be convex if the line joining any two internal points of the polygon lies completely inside the polygon.

A Polygon is said to be concave if the line joining any two internal points of the polygon is not going to lie completely within the polygon.

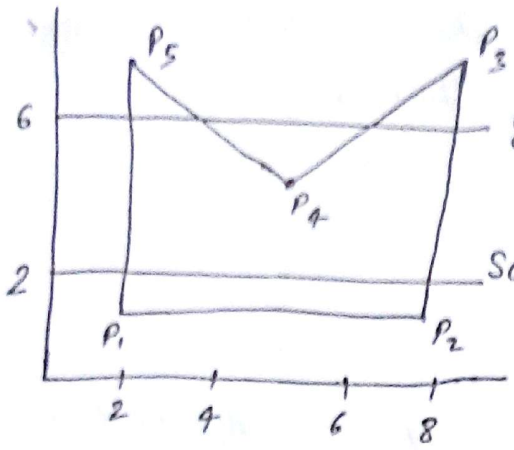


* Polygon Filling :-

Polygon filling is the process of coloring the area of polygon. Polygon filling algorithm assume that at least one pixel is interior to the polygon is known. Then the algorithm tries to find all other pixels interior to the polygon and subsequently color them.

1) Scan Line Polygon Fill Algorithm :-

The scan line procedure for solid filling of polygon areas for each scan line crossing a polygon. The intersection points are marked as a seed pixel and pushed onto the stack. Each scan lines examine immediately above and immediately below scan line, and process is complete if each scan line have either boundary pixel or previously filled pixels.



The scan line 1 is divided into the three region –

$x < 2$ outside the polygon

$2 \leq x \leq 8$ inside the polygon

$x > 8$ outside the polygon

Similarly, the scan line 2 is divided into the five region –

$x < 2$ outside the polygon

$2 \leq x \leq 4$ inside the polygon

$4 < x < 6$ outside the polygon

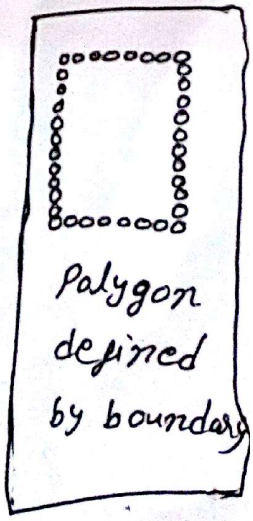
$6 \leq x \leq 8$ inside the polygon

$x > 8$ outside the polygon

2) Boundary Fill Algorithm :-

If Polygon is defined by the bounding pixels, then boundary fill algorithm is used to fill the area. It is start at the point inside a region, if a boundary is specified in a single color, the fill algorithm proceeds outward pixels by pixels until the boundary is encounter.

```
Void boundary fill (int x, int y, int fill, int boundary)
{
  int current = getpixel (x, y);
  if ((current != boundary) AND (current != fill))
  {
    Putpixel (x, y, fill);
    boundary fill (x+1, y, fill, boundary);
    boundary fill (x, y+1, fill, boundary);
```



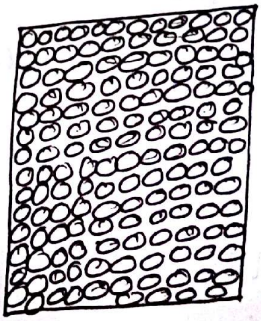
```
        boundary fill (x-1, y, fill, boundary);
        boundary fill (x, y-1, fill, boundary);
    }
}
```

3) Flood Fill Algorithm :-

If the polygon defined by the total number of pixels, then flood fill algorithm is used to fill the area. This algorithm begins with a starting pixel inside the region, if area has more than one interior color, we reassign pixel values that all interior points have same color.

It checks, if pixel has the region's original color then it fills pixel with new color and uses each of the pixel's neighbors as a new seed in a recursive call.

```
Void flood fill (int x, int y, int fill, int old)
{
    int current = getpixel (x,y);
    if (current == old)
    {
        ~~setcolor~~ putpixel (x,y, fill);
        flood fill (x+1, y, fill, old);
        flood fill (x, y+1, fill, old);
        flood fill (x-1, y, fill, old);
        flood fill (x, y-1, fill, old);
    }
}
```



Polygon defined by total number of pixels

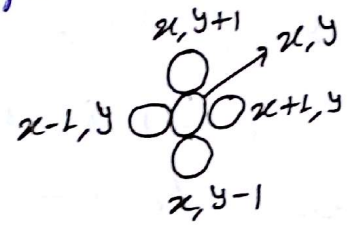
* How to find next pixels :-

For to find the next pixel we have two methods for proceeding to ~~neigh~~ neighboring pixel from the current pixel.

1) 4-connected Approach
2) 8-connected Approach

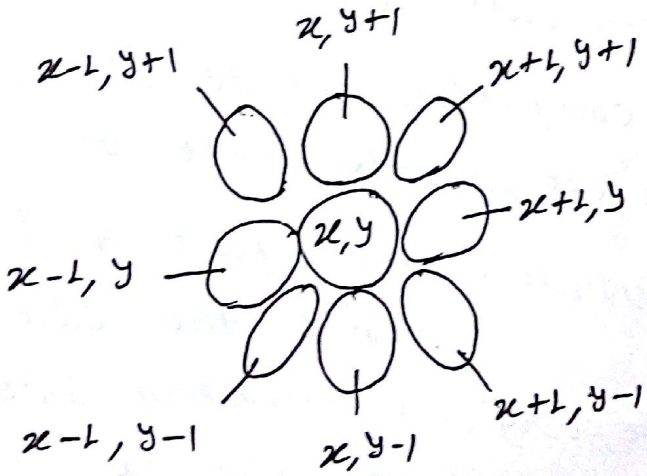
1) 4-Connected Approach :-

In this method choose the four neighboring ~~pix~~ pixels. These are the pixel positions that are left, right, above and below the current pixels.



2) 8-Connected Approach :-

In this method choose the eight neighboring pixels. These are the pixels positions that are left, right, above, below and also includes four diagonal pixels of the current pixel.



* CLIPPING :-

A procedure that identifies those portions of a picture that are either inside or outside of a specified region is referred to as a clipping algorithm. The region against which an ~~algorithm~~ object is to be clipped is called clip window.

1) Point clipping :-

Assuming that the clip window is rectangle, where the edge of clip window (xw_{min}, xw_{max}, yw_{min}, yw_{max}) are boundary. Then a point $P = (x, y)$ is visible, if the following condition hald -

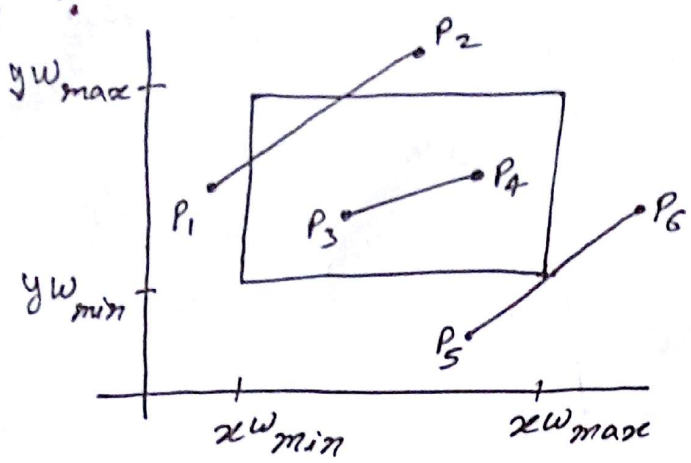
$$xw_{min} \leq x \leq xw_{max}$$
$$yw_{min} \leq y \leq yw_{max}$$

2) Line clipping :-

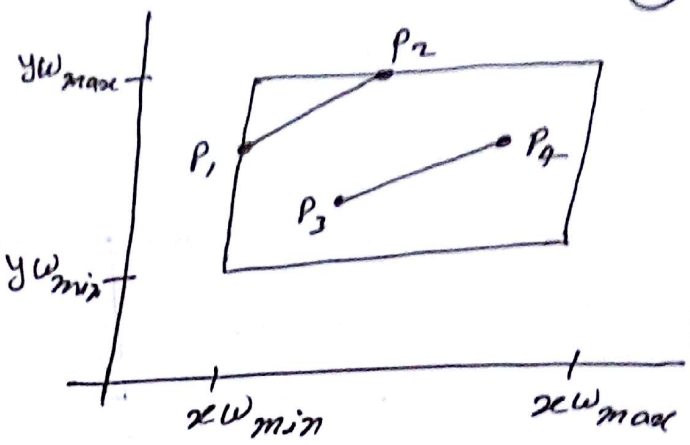
Line clipping algorithm display line which lies inside the clip window. All the line segments fall into one of the following ~~cou~~ categories -

(i) Visible : completely inside the clip window
(ii) Non visible : completely outside the clip window
(iii) Partially visible : some part of line lies inside the clip window.

In this category (iii), perform intersection calculations with one or more clipping boundaries.



[Before clipping]



[After clipping]

There is no need of clipping, if and only if –

$xw_{min} \leq x_1 \leq xw_{max}$; $xw_{min} \leq x_2 \leq xw_{max}$

$yw_{min} \leq y_1 \leq yw_{max}$; $yw_{min} \leq y_2 \leq yw_{max}$

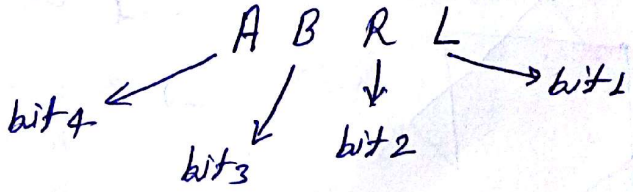
[condition for line which is fully visible]

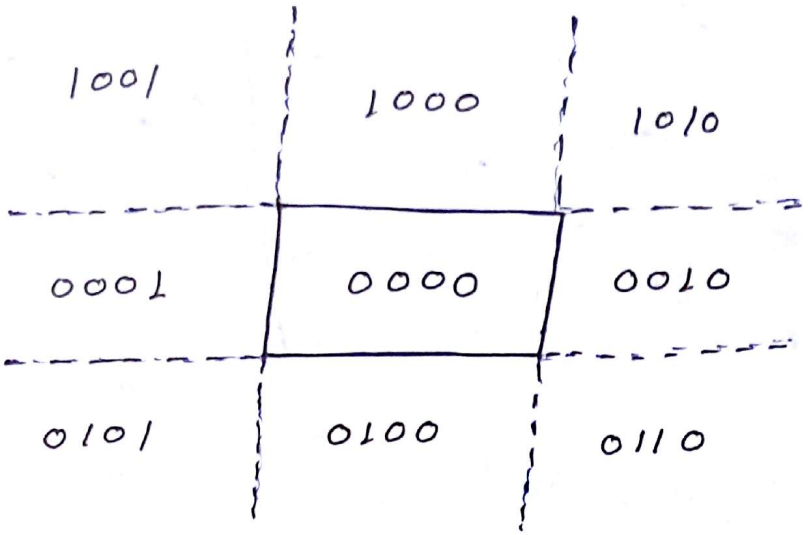
✯ Sutherland Line Clipping Algorithm :–

This line clipping algorithm speed up the processing of line segment by performing intial tests, that reduces the number of intersections that must be calculated.

Every end point is assigned a four digit binary code called region code that identifies the location of the point relative to the boundaries of the clipping rectangle.

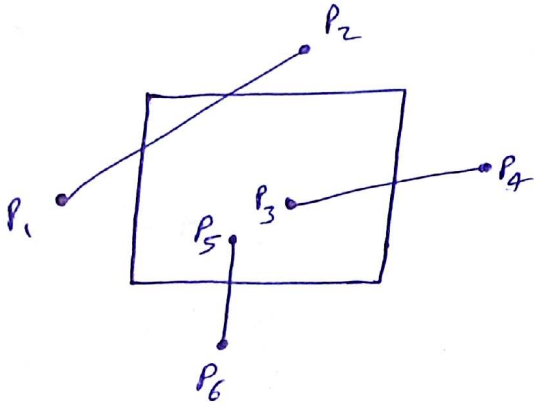
bit1 → Left
bit2 → Right
bit3 → Below
bit4 → Above





A method that can be used to test line for total clipping is to perform the logical AND operation with both region codes.

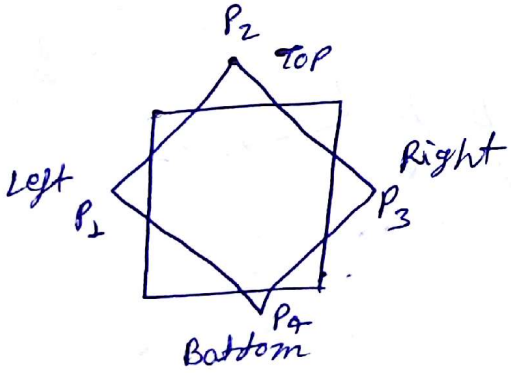
If result is not 0000, then the line is completely outside the clipping region.



$P_1 \rightarrow 0001$
$P_2 \rightarrow 1000$
$P_3 \rightarrow 0000$
$P_4 \rightarrow 0010$
$P_5 \rightarrow 0000$
$P_6 \rightarrow 0100$

3) Polygon Clipping :-

A polygon boundary processed with a line clipper may be displayed as a series of unconnected line segment, depending on the orientation of the polygon to the clipping window.



⇒ If clip left then right then bottom then Top ⇒