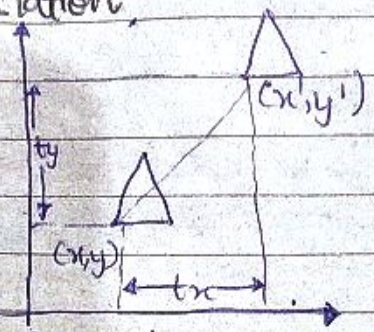
UNIT-3
2D GRAPHICS :-

Transformation -
Changing position, shape, size or orientation of an object on display is known as transformation.

Basic Tranformation :-
three basic tranformation Translation, Rotation & Scaling

Translation -



It is transformation that used to reposition the object along the straight line path from one coordinate location to another.

→ It is rigid body transformation.
$x' = x + t_x$ & $y' = y + t_y$
→ Translation distance pair (t_x, t_y) is called a Translation Vector or Shift Vector.
→ We can represent it into single matrix equation in column vector as; $P' = P + T$

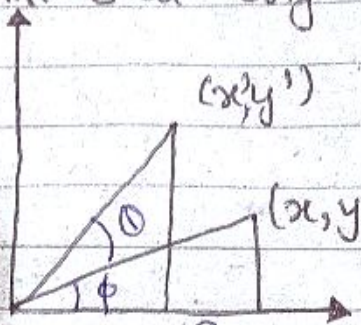
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

→ We can also represent in row vector form :-
$P' = P + T$ $\quad [x'\ y'] = [x\ y] + [t_x\ t_y]$
→ Graphics Package like GKS & PHIGS uses column vector representation.

Rotation - It is a transformation that used to reposition the object along the circular path in the XY - Plane.
To generate a rotation we specify a rotation angle θ & the position of Rotation Point (Pivot Point) (x_r, y_r) about which object is rotated.

→ +ve value of rotation angle define counter clockwise rotation & -ve value for clockwise rotation.

1) When pivot point is at origin (0,0)



$x = r\cos\phi \qquad y = r\sin\phi \quad \text{—(1)}$

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

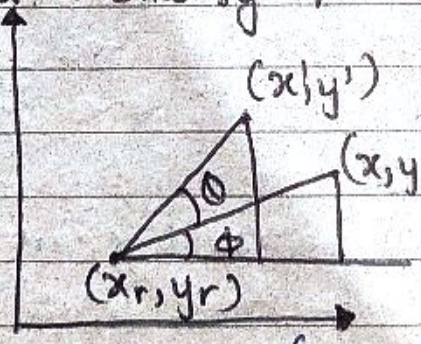
$\Rightarrow x' = r\cos\phi\cos\theta - r\sin\phi\sin\theta \;,\; y' = r\sin\phi\cos\theta + r\cos\phi\sin\theta$

$\Rightarrow x' = x\cos\theta - y\sin\theta \;,\; y' = x\sin\theta + y\cos\theta$

+ We can write it in the form of column vector matrix equation as $P' = R \cdot P$

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

2) Rotation about arbitrary point :-



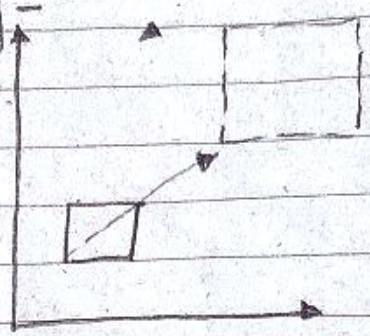
$x' = x_r + (x - x_r)\cos\theta - (y - y_r)\sin\theta$

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

→ Rotation is also rigid body transformation so we need to rotate each point of object.

→ Its matrix equation can be obtained by simple method that we will discuss later.

Scaling -



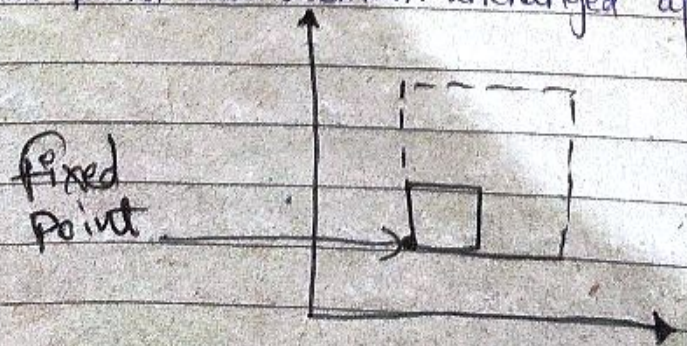
→ It is a transformation that used to alter the size of an object.
→ this operation is carried out by multiplying coordinate value (x, y) with scaling factor (S_x, S_y).

→ $x' = x \cdot S_x \quad y' = y \cdot S_y$
→ Column vector matrix equation as :

$$P' = S \cdot P$$

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

→ Values less than 1 reduces size & greater than 1 enlarge the size of object.
→ Same values of S_x & S_y will produce Uniform Scaling and different values will give Differential Scaling
→ Scaling factor with value less than 1 will move toward origin & greater than 1 will move object away from origin.
→ We can control the position of object after scaling by keeping one position fixed called Fix Point (x_f, y_f) that point will remain unchanged after scaling.



Eqⁿ for Scaling with fixed point posⁿ as (x_f, y_f) is:

$x' = x_f + (x - x_f)S_x$ $\quad y' = y_f + (y - y_f)S_y$

$x' = x_f + xS_x - x_fS_x$ $\quad y' = y_f + yS_y - y_fS_y$

$x' = xS_x + x_f(1 - S_x)$ $\quad y' = yS_y + y_f(1 - S_y)$

→ Matrix eqⁿ for same will discuss in later section.

HOMOGENEOUS MATRIX REPRESENTATION

While designing the application, we perform transformation to fit the picture components into their proper position.

∴ We are constructing the matrices, so that these transformations sequence can be efficiently process.

Each transformation can be expressed in general form as:-

$$P' = M_1 \cdot P + M_2 \quad -①$$

where P & P' are column vectors

M_1 → 2×2 Matrix containing Multiplicative factors

M_2 → 2 element column matrix containing translation terms

FOR Translation

M_1 → Identify Matrix

For Rotation / Scaling

M_2 → Contains translational terms associated with pivot point or scaling point

We cannot do combine transformations using 2×2 matrix with initial coordinates. A more efficient way is to combine the transformation so that the final coordinate point are obtained directly from initial coordinates by using a 3×3 matrix, thereby eliminating the calculation of intermediate coordinate values. Therefore,

$$P' = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \quad P = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad T = \begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix} \quad R = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

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

COMPOSITE TRANSFORMATION

We can set up a matrix for any sequence of transformations as a composite transformation matrix by calculation the Matrix product of the individual transformations.

Forming products of transformation matrics is often referred to as a concatenation or composition of Matrices.

For column matrix representation of coordinate positions, we form composite transformation by multiplying matrices in order from right to left.

Translation

$$\begin{bmatrix} 1 & 0 & tx_1 + tx_2 \\ 0 & 1 & ty_1 + ty_2 \\ 0 & 0 & \end{bmatrix}$$

Scaling

$$\begin{bmatrix} Sx_1 Sx_2 & 0 & 0 \\ 0 & Sy_1 Sy_2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

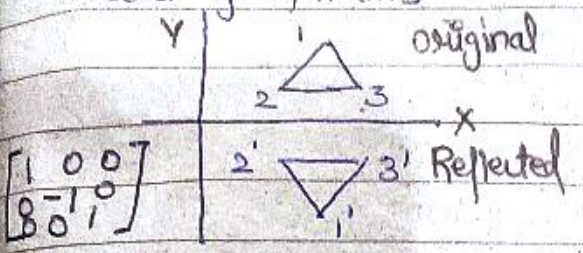
Rotation

$$R(\theta_2)\,R(\theta_1) = R(\theta_1 + \theta_2)$$

REFLECTION :-

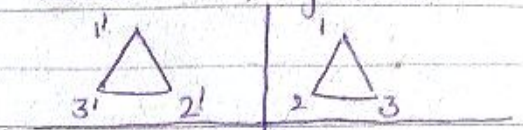
A reflection is a transformation that produces a mirror image of a object

Reflection about y=0, x axis



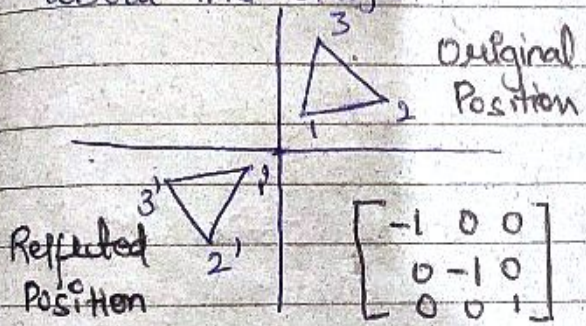
$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

about x=0, y axis



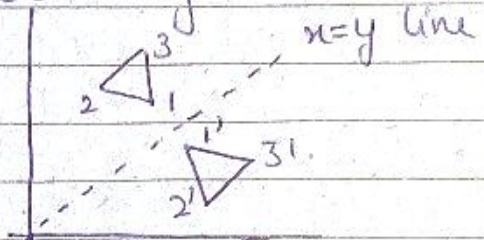
$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

about the origin



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

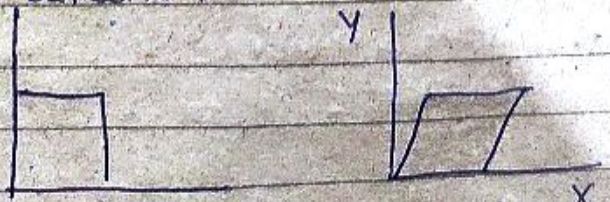
About x=y



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

SHEAR - transformation that distort the shape of an object such that the transformed shape appears as if the object were composed of internal layer that had been caused to slide over each other is called shear.

Shear in x-direction



Shear relative to x axis that is y=0 line can be produced by :

$x' = x + sh_x \cdot y \qquad y' = y$ ← shear parameter

Transformation matrix :

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

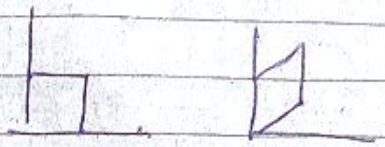
We can generate x-direction shear relative to other reference line $y = y_{ref}$ with following eqn:

$x' = x + sh_x \cdot (y - y_{ref})$ $\quad y' = y$

Transformation matrix:

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

Shear in y direction



$x' = x$ $\quad y' = y + sh_y \cdot x$

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

$x' = x$ $\quad y' = y + sh_y \cdot (x - x_{ref})$

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

VIEW ~~PIPELINE~~ : PIPELINE :-

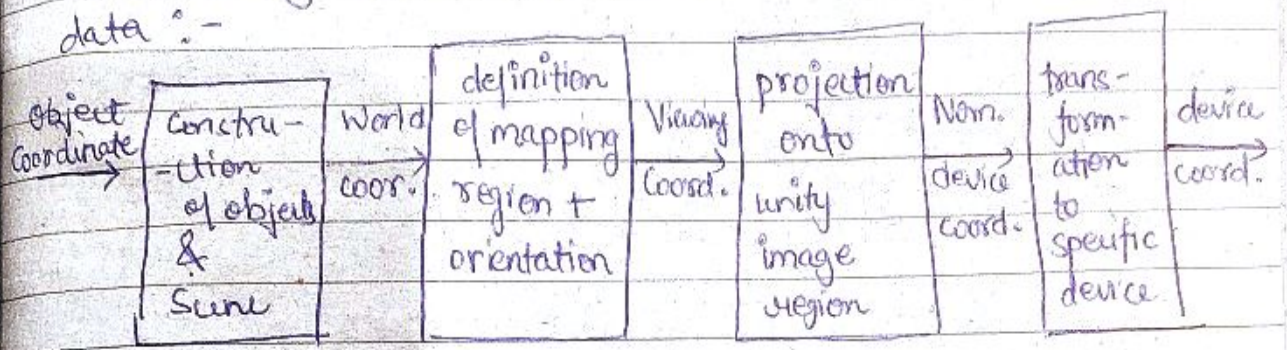
Window - Area selected in world-coordinate for display is called window

Viewport - Area on display device in which window image is display (mapped) is called viewport.

In general finding device coordinates of viewport from world coordinates of window is called as **Viewing transformation**.

Sometimes we consider this viewing transformation as window to viewport transformation but in general it involves more steps.

the term viewpipeline describes a series of transformations, which are passed by geometry data to end up as image data being displayed on a devices. The 2D viewing pipeline describes this process for 2D data :-



WINDOW TO VIEWPORT CO-ORDINATE TRANSFORMATION

It is the process of transforming a 2D world-coordinate objects to device coordinates. Objects inside the world or clipping window are mapped to the viewport which is the area on the screen where world coordinates are mapped to be displayed.

(*)

World coordinate - X_{Wmin}, Y_{Wmin}, X_{WMAX}, Y_{Wmax}

Device Coordinate - X_{Vmin}, X_{VMAX}, Y_{Vmin}, Y_{VMax}

Window - It is area on world coordinate selected for display.

ViewPort - It is the area on device coordinate where graphics to be displayed.

MATHEMATICAL CALCULATION :-

(X_W, Y_W) A Point on window

(X_V, Y_V) Corresponding point on Viewport

→ Normalized Point on window $\left(\frac{X_W - X_{Wmin}}{X_{WMAX} - X_{Wmin}}, \frac{Y_W - Y_{Wmin}}{Y_{WMAX} - Y_{WMIN}}\right)$

Normalized Point on Viewport $\left(\frac{X_V - X_{Vmin}}{X_{VMax} - X_{VMin}}, \frac{Y_V - Y_{Vmin}}{Y_{VMax} - Y_{Vmin}}\right)$

Now the relative pos^n of the object in window & Viewport are same.

for x coordinate,

$$\frac{X_W - X_{WMIN}}{X_{WMAX} - X_{WMIN}} = \frac{X_V - X_{VMIN}}{X_{VMAX} - X_{VMIN}}$$

for y coordinate,

$$\frac{Y_W - Y_{WMIN}}{Y_{WMAX} - Y_{WMIN}} = \frac{Y_V - Y_{VMIN}}{Y_{VMAX} - Y_{VMIN}}$$

So after cal. x & y coordinate we get

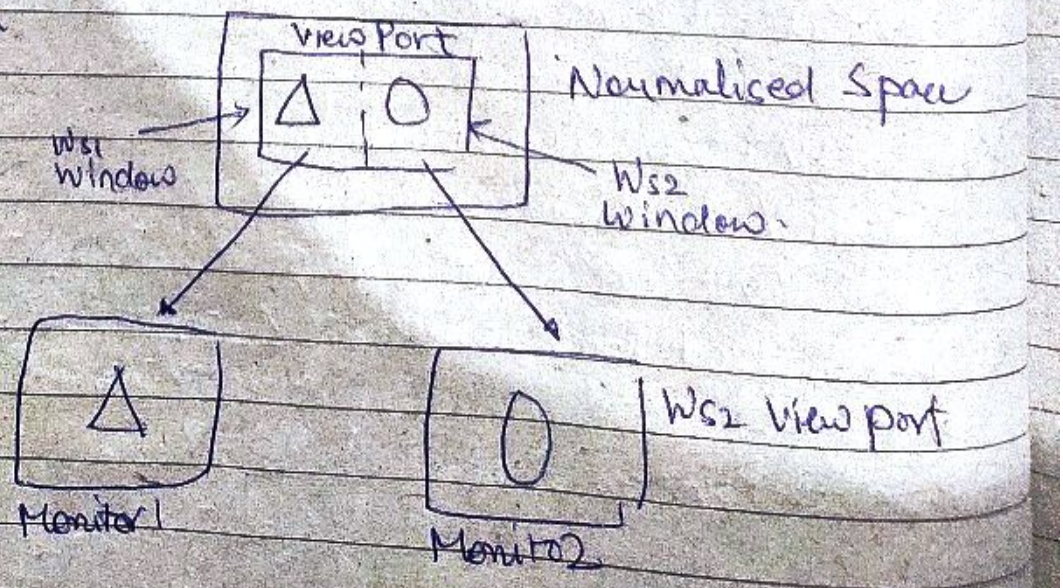
$$X_V = X_{VMIN} + (X_W - X_{WMIN}) S_x$$
$$Y_V = Y_{VMIN} + (Y_W - Y_{WMIN}) S_y$$

where S_x is scaling factor of x & S_y of y coordinate

$$S_x = \frac{X_{VMAX} - X_{VMIN}}{X_{WMAX} - X_{WMIN}}$$

$$S_y = \frac{Y_{VMAX} - Y_{VMIN}}{Y_{WMAX} - Y_{WMIN}}$$

Work Station Transformation
→ No of display devices



CLIPPING

Clipping in Computer Graphics is to remove objects, lines, or line segments that are outside the viewing pane. clipping Algorithm- Any procedure that identifies parts of an image lying inside OR OUTSIDE OF A specific region is called a clipping algorithm.
CLIPPING WINDOW- The part of the image that is selected to be displayed on computer screen, is called a WINDOW
The window against which object is clipped is called a Clip window. It can be curved or rectangle in shape.

POINT CLIPPING :-

Removing those points that lie outside the window.
→ We consider clipping window with edge (X_{Wmin}, X_{WMAX}, Y_{Wmin}, Y_{WMax})
→ So for finding wheter given point is inside or outside the clippind window we use following inequality:

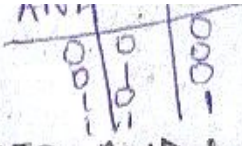
$$X_{Wmin} \leq x \leq X_{WMax}$$
$$Y_{Wmin} \leq y \leq Y_{WMax}$$

→ If above both inequality is satisfied then the point is inside otherwise outside.

LINE CLIPPING :-

→ It involves several possible cases.
1. Completely Inside the Clipping Window (Visible lines)
2. Completely outside (Invisible Lines)
3. Partially Inside & Partially Outside (Partially Visible)

↓ Clipping is required
Intersection point



COHEN SUTHERLAND:-

Nine regions are created eight "outside" & one "inside" region.

1001	1000	1010
0001	0000	0010
0101	0100	0110

4bit code
TBLR
top / Left / Bottom / Right

Algorithm:-

1. Assign a region code for two endpoints of given line.
2. If both end points have a region code 0000 then given line is completely inside
3. Else perform logical AND operation for both codes.
   - 3.1 If result is not 0000, then given line is completely outside
   - 3.2 Else line is partially inside
     - 3.2.1 : Choose an endpoint of the line that is outside the given rectangle
     - 3.2.2 : Find the intersection point of the rectangular boundary
     - 3.2.3 : Replace the endpoint with the intersection point & Update the region code
     - 3.2.4 : Repeat Step 2 until we find a clipped line either trivally accepted or trivially rejected

Step 4: Repeat Step 1 for other lines

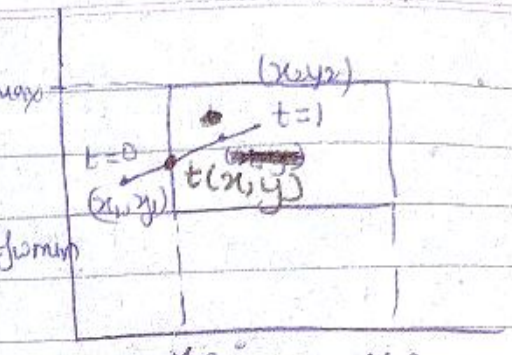
LIANG BERSKY :-

$0 \le t \le 1$

Parametric Equation of Line

$x = (1-t)x_1 + t x_2$

$y = (1-t)y_1 + t y_2$



$x = x_1 - t x_1 + t x_2$

$x = x_1 + t(x_2 - x_1)$ $\quad = x_1 + t\Delta x$

$y = y_1 - t y_1 + t y_2$

$y = y_1 + t(y_2 - y_1)$ $\quad = y_1 + t\Delta y$

As per point clipping algorithm

$$\begin{cases} x_{wmin} \le x \le x_{wmax} \\ y_{wmin} \le y \le y_{wmax} \end{cases}$$

$$\begin{cases} x_{wmin} \le x_1 + t\Delta x \le x_{wmax} \\ y_{wmin} \le y_1 + t\Delta x \le y_{wmax} \end{cases}$$

$$\begin{cases} x_1 + t\Delta x \ge x_{wmin} \\ x_1 + t\Delta x \le x_{wmax} \\ y_1 + t\Delta y \ge y_{wmin} \\ y_1 + t\Delta y \le y_{wmax} \end{cases}$$

$$\begin{cases} t\Delta x \ge x_{wmin} - x_1 \quad -① \\ t\Delta x \le x_{wmax} - x_1 \\ t\Delta y \ge y_{wmin} - y_1 \quad -② \\ t\Delta y \le y_{wmax} - y_1 \end{cases}$$

Multiply eq ① & ② by -ve sign

$$\begin{cases} -t\Delta x \le (x_{wmin} - x) \text{ OR } -t\Delta x \le x_1 - x_{wmin} \\ t\Delta x \le x_{wmax} - x_1 \\ -t\Delta y \le y_1 - y_{wmin} \\ t\Delta y \le y_{wmin} - y_1 \end{cases}$$

$x = x_1 + t\Delta x$

$y = y_1 + t\Delta y$

General Equation form

$tp_k \leq q_k \quad \{k = 1, 2, 3, 4\}$

$P_1 = -\Delta x$	$q_1 = x_1 - x_{wmin}$
$P_2 = \Delta x$	$q_2 = x_{wmax} - x_1$
$P_3 = -\Delta y$	$q_3 = y_1 - y_{wmin}$
$P_4 = \Delta y$	$q_4 = y_{wmax} - y_1$

If $P_k = 0$ Line parallel with clipping window for $k = 1, 2, 3, 4$

If $q_k < 0$ Line Outside

If P_k ~~other than~~ non zero

{ If $P_k < 0$ then t_1 value find

the new value of t

$$t = \max\left(0, \frac{q_k}{P_k}\right)$$

else means $P_k > 0$

then t_2 find :-

$$t = \min\left(1, \frac{q_k}{P_k}\right)$$

→ now check $t_1 > t_2$ line

Completely outside & Reject it

If $t_1 < t_2$ then we will use

$x = x_1 + t\Delta x$

$y = y_1 + t\Delta y$

Condition	Position of Line
$P_k = 0$	Parallel to the clipping boundaries
$P_k = 0$ & $q_k < 0$	Completely outside the boundary
$p_k = 0$ & $q_k >= 0$	inside the parallel clipping boundary
$P_k < 0$	Line proceeds from outside to inside
$P_k > 0$	Line proceeds from inside to outside

Algorithm -

1. Set $t_{min} = 0$ $t_{max} = 1$

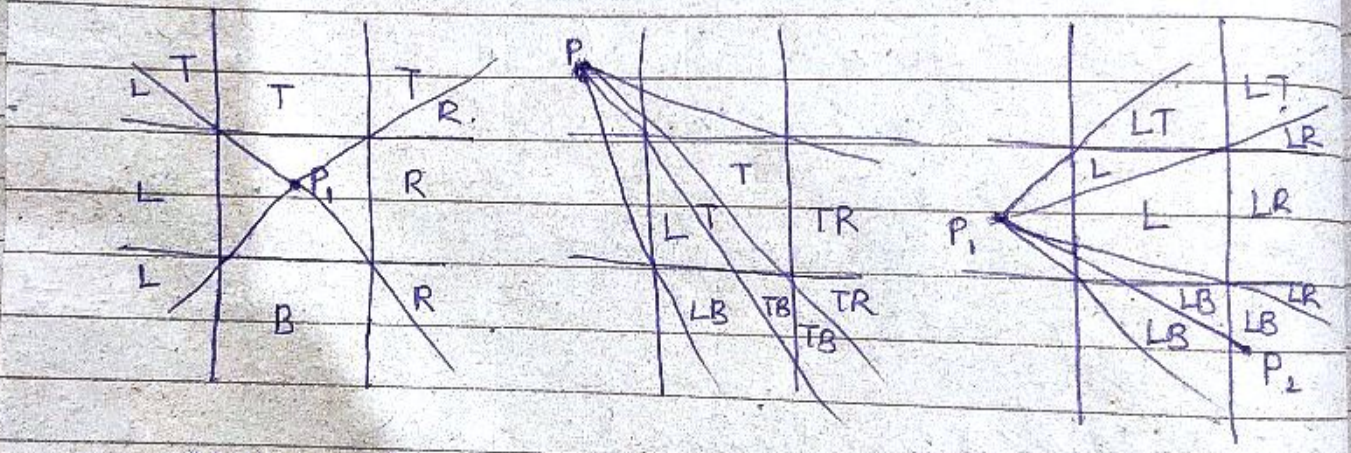
2. Calculate the values of t (t(left), t(right), t(top), t(bottom))
   (i) If $t < t_{min}$ ignore that & move to the next edge
   (ii) else seperate the values as entering or exiting values using the inner product
   (iii) If t is entering value, set $t_{min} = t$, if exiting $t_{max} = t$

3. If $t_{min} < t_{max}$, draw a line from
   $(x_1 + t_{min}(x_2 - x_1), y_1 + t_{min}(y_2 - y_1))$ to
   $(x_1 + t_{max}(x_2 - x_1), y_1 + t_{max}(y_2 - y_1))$

4. If the line crosses over the window,
   $(x_1 + t_{min}(x_2 - x_1), y_1 + t_{min}(y_2 - y_1))$
   & $(x_1 + t_{max}(x_2 - x_1), y_1 + t_{max}(y_2 - y_1))$ are intersection point of line & edge

Nicholl-Lee-Nicholl :-

T - ray intersect with top boundary
L - " " " left boundary
B - " " " Bottom "
R - " " " Right "
LT - Ray intersects with left & top boundary
LR - " " " Left & Right "
LB - " " " Left & Bottom "
TR - " " " Top & Right "
TB - " " " Tob & bottom "



Slope condition are checked
&
Intersection point is calculated using paramitric equation

POLYGON CLIPPING :-

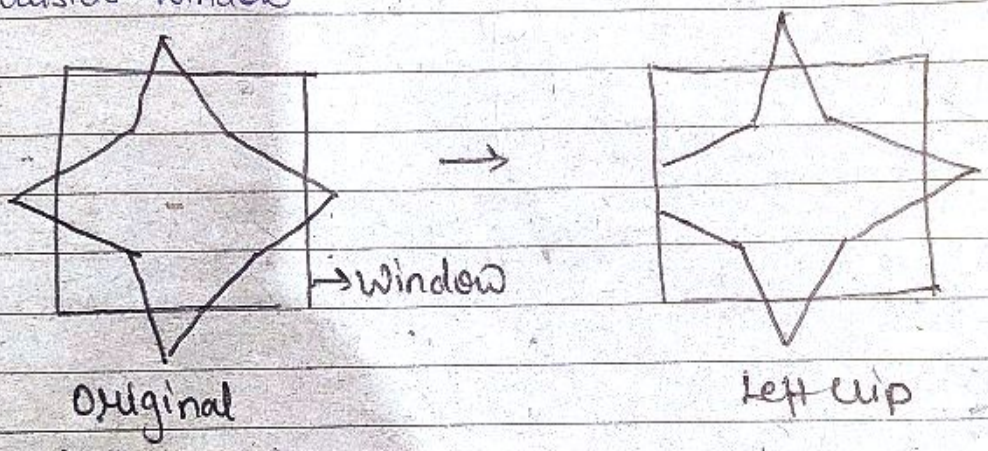
It is a process in which we only consider the part which is inside the view port or window. We will remove or clip the part outside the window.

→ SUTHERLAND - HODGEMAN POLYGON CLIPPING ALGO
→ WEILER - ATHERTON POLYGON CLIPPING ALGO

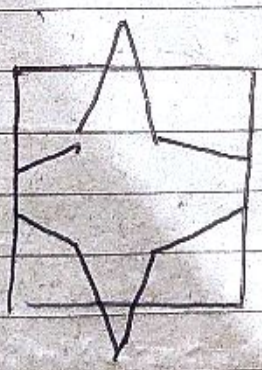
1.) SUTHERLAND - HODGEMAN POLYGON CLIPPING ALGO

→ It deals with four different clipping case. The o/p of each case is i/p for the next case.

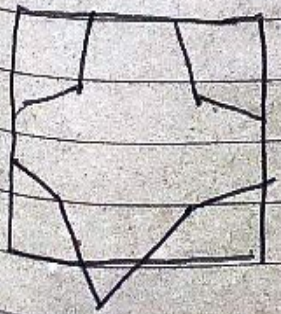
Case 1) Left Clip - Clip left part of polygon, which is outside window



Case 2) Right Clip



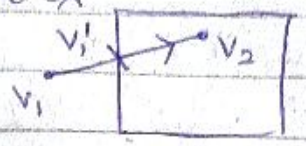
Case 3) TOP CLIP



Case 4) Bottom clip

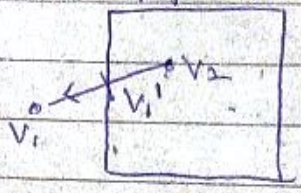
There should be following conditions while we clip a polygon.

Condition 1) (Out-In) If first vertex of polygon is outside & second is inside, then the o/p will be intersection point & second vertex



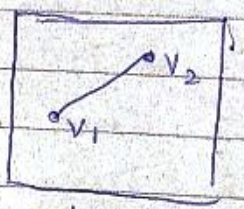
O/P → $V_1' V_2$

Condition 2) (In-Out) If first vertex inside & 2nd Outside then, then o/p will be intersection point



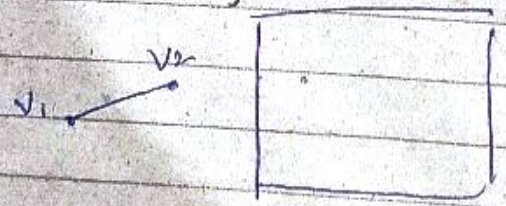
Output → V_1'

Condition 3) (In-In)



O/P → V_2

Condition 4) (Out-Out)



Output → Nothing

WEILER-ATHERTON POLYGON CLIPPING

In this algo, we consider the view pane boundaries instead of edges and vertices of the polygon.

Algo :-

1) First, create a list of intersection points that are in starting or ending state $(I_1, I_2 \ldots I_n)$
2) Now create two more list, one for subject polygon & other for clip polygon. Fill both lists with intersection points & vertices of polygon.
3) Insert the vertices in both lists in such a way that the intersection point exists b/w the correct vertices.
4) Now, start from the first vertix of the polygon. Select the first intersection point as an entering point & follow the same process until we reach the exiting point
5) We can move from clip polygon list to subject polygon list & search for the finishing intersection point. Repeat the process until we find the entering point.
6) Now, the polygon is being clipped. Repeat the same process until each point has been visited once.
7. Stop.

Ex :-