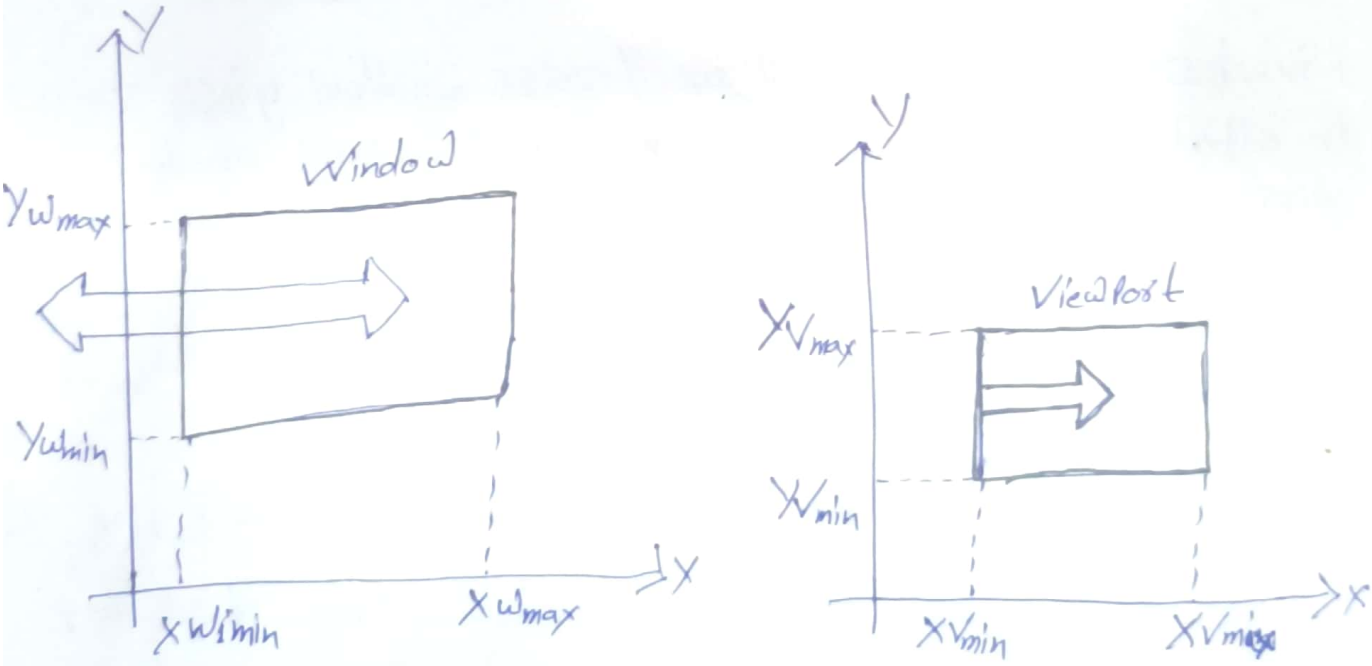
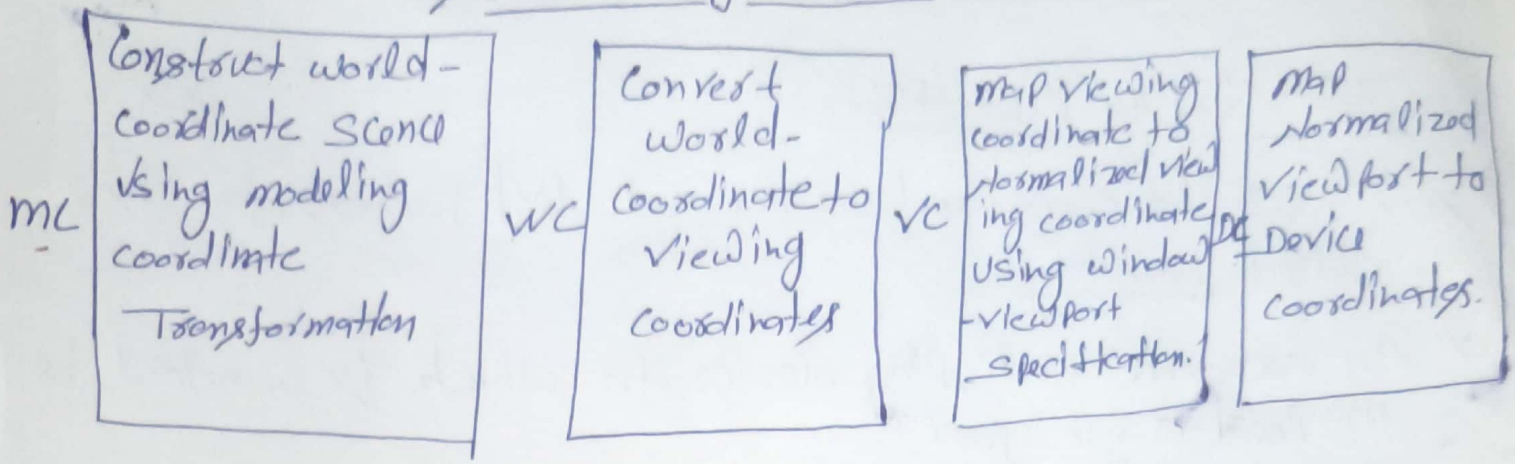
The Viewing Pipeline:-

* The world coordinate area selected for display is called window.
* An area on a display device to which is window is mapped a viewport.
* Windows and viewports are rectangular in standard position.
* In general finding device coordinates of viewport from world coordinate of window is called as viewing Transformation.
* A viewing transformation using standard rectangles for the window and viewport.



→ How we see steps involved in viewing pipeline.

⇒ 2D-Viewing Pipeline



→ Finally device-coordinate is used to display image on display screen.

→ By changing the viewport position on screen we can see image at different place on the screen.

→ By changing the size of the window and viewport we can obtain zoom in and zoom out effect as per requirement.

→ Fixed size viewport and small size window gives zoom in effect, and fixed size viewport and larger window gives zoom out effect.

⇒ Viewing coordinate Reference Frame:-

Set-up the viewing coordinate.

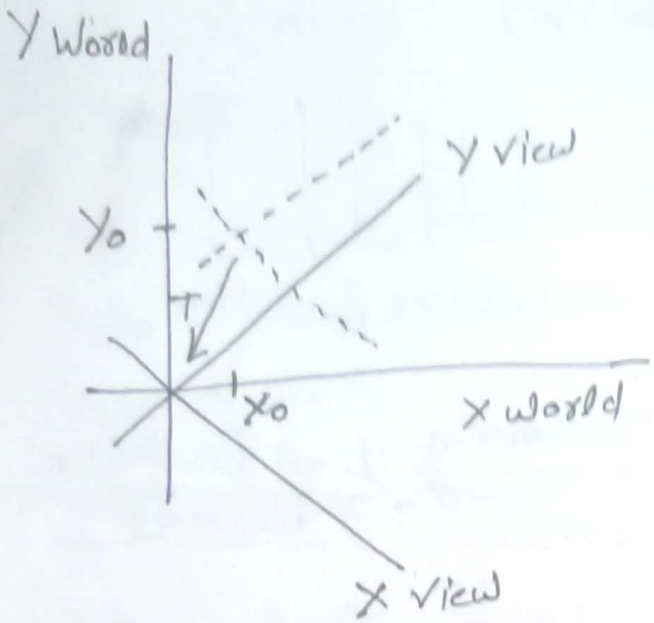
1) origine is selected at some world position $P_0 = (x_0, y_0)$

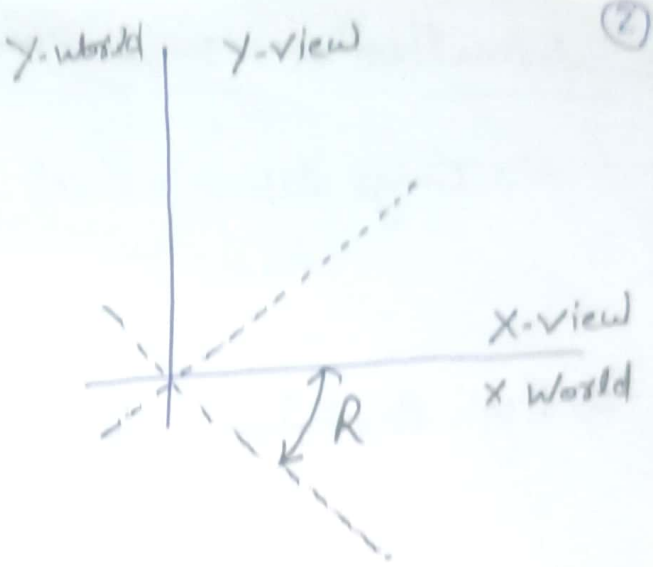
2) Established the orientation. specify a world vector ✓ that defines the viewing y direction

3) obtain the matrix for converting world to viewing coordinates

$$M_{wc,vc} = R\,T$$

T = Translate, R = rotate





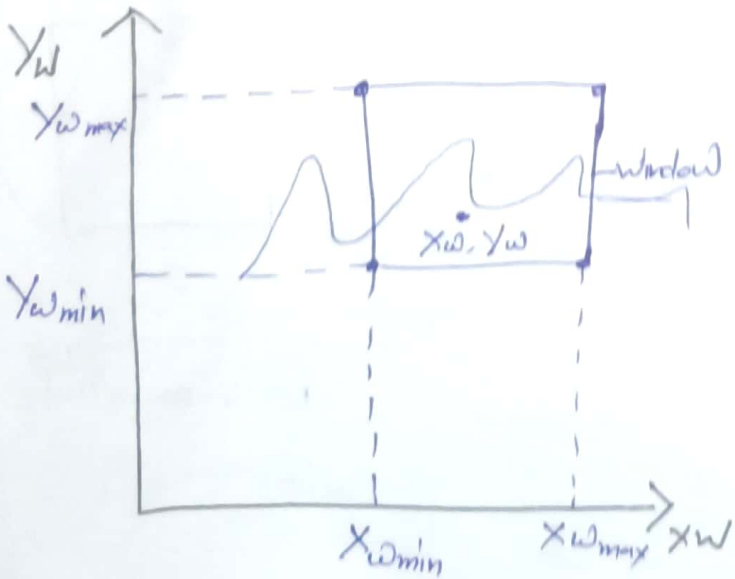
⇒ Window to viewport coordinate Transformation:-

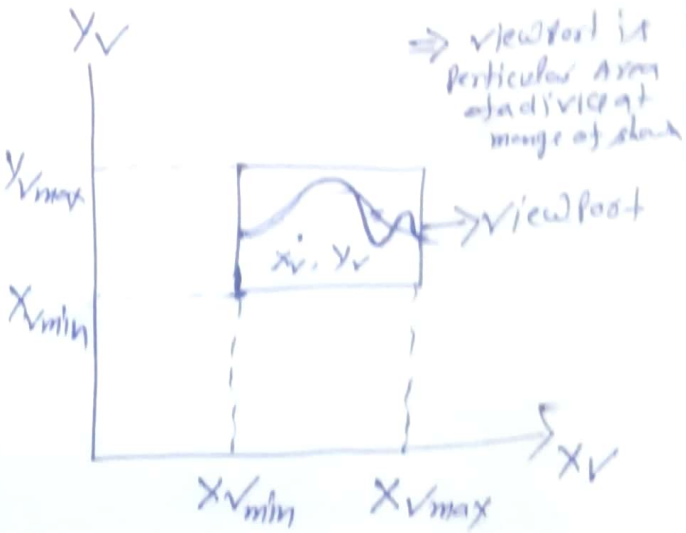
* Select the viewport in normalized coordinate, and then object description transferred to normalized device coordinate

⇒ Used to Process ⇒

1) Normalization Transformation

2) Work station Transformation





⇒ viewport is particular area of a device at which image of object is displayed

$$X = X_n * X_w$$
$$y = Y_n * Y_w$$
} 0-1 Scaling

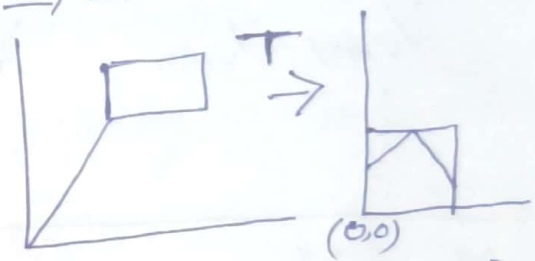
→ x, y → Actual device coordinate
x_n, y_n → Normalized coordinate → scaling
x_w, y_w → window coordinate

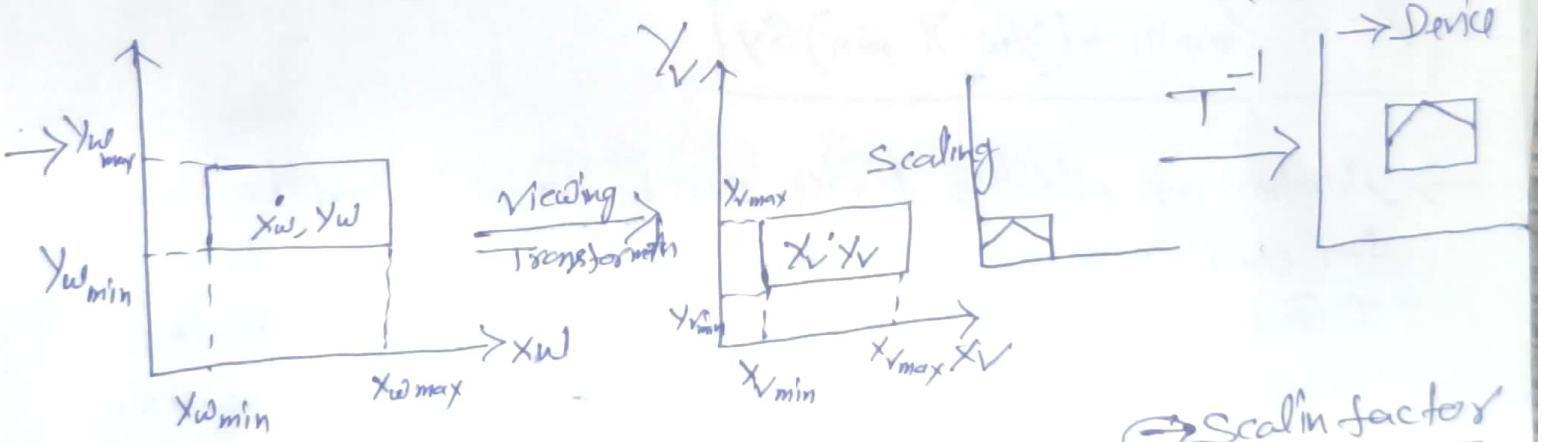
→ work station Transformation:

World coordinate → (normalize) → Normalized coordinate → (work station) → Device coordinate

→ window

$$\Rightarrow V = N \cdot W$$
$$W = T S T^{-1}$$





→ $\dfrac{X_v - X_{v_{min}}}{X_{v_{max}} - X_{v_{min}}} = \dfrac{X_w - X_{w_{min}}}{X_{w_{max}} - X_{w_{min}}}$

→ $X_v - X_{v_{min}} = \left(\dfrac{X_w - X_{w_{min}}}{X_{w_{max}} - X_{w_{min}}}\right) * X_{v_{max}} - X_{v_{min}}$

→ Scaling factor

$$S_x = \frac{X_{v_{max}} - X_{v_{min}}}{X_{w_{max}} - X_{w_{min}}}$$

$$S_y = \frac{Y_{v_{max}} - Y_{v_{min}}}{Y_{w_{max}} - Y_{w_{min}}}$$

→

$$x_v - x_{v_{min}} = \left(\frac{x_{v_{max}} - x_{v_{min}}}{x_{w_{max}} - x_{w_{min}}}\right) * x_w - x_{w_{min}}$$

$$x_v = S_x \cdot (x_w - x_{w_{min}}) + x_{v_{min}}$$

$$\boxed{x_v = x_{v_{min}} + (x_w - x_{w_{min}}) S_x}$$

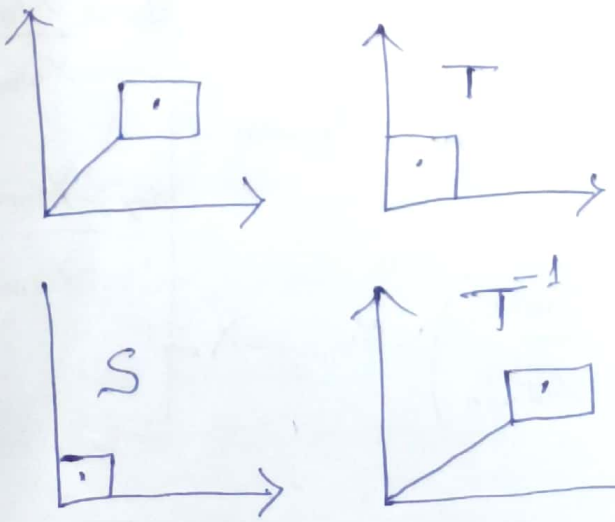
→ Now →

$$= \frac{y_v - y_{v_{min}}}{y_{v_{max}} - y_{v_{min}}} = \frac{y_w - y_{w_{min}}}{y_{w_{max}} - y_{w_{min}}}$$

$$\boxed{y_v = y_{v_{min}} + (y_w - x_{w_{min}}) S_y}$$

→ Number of display device can be used in application and for each we can use different window-to-viewport Transformation. This maping is called the workstation Transformation.

$$W = T\ S\ T^{-1}$$



matrix Representation.

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x_{w_{min}} & -y_{w_{max}} & 1 \end{bmatrix}$$

$$S = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \Rightarrow$$

$$T^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x_{w_{min}} & y_{w_{max}} & 1 \end{bmatrix}$$

CLIPT Display window:

(4)

$$T \cdot S \cdot T^{-1} = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ x_{v_{min}} - x_{w_{min}} \cdot S_x & y_{v_{min}} - y_{w_{min}} \cdot S_y & 1 \end{bmatrix}$$

$$\begin{bmatrix} A' \\ B' \\ C' \end{bmatrix} = \begin{bmatrix} A \\ B \\ C \end{bmatrix} \begin{bmatrix} \text{Transformation} \end{bmatrix}$$

⇒ Viewing function:

→ openGL Two-Dimensional Viewing functions.

* We must set the parameters for the clipping window as part of the projection transformation.

* Functions.

```
glMatrixMode (GL_Projection);
```

* We can also set the initialization as

```
glLoadIdentity ();
```

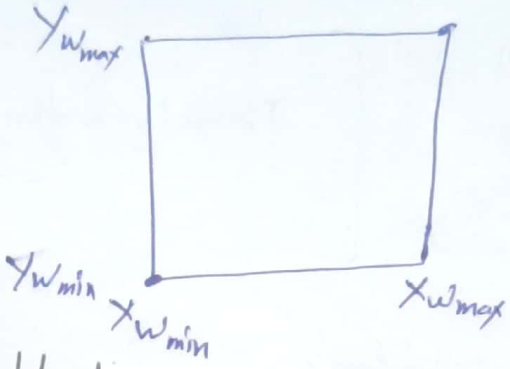
⇒ This ensures that each time we enter the projection mode, the matrix will be reset to the identity matrix so that the new viewing parameters are not combined with the previous ones.

→ GLU clipping_window Functions.

* To define a two-dimsional clipping window, we can use the GLU function:

```
gluOrtho2D (Xwmin, Xwmax, Ywmin, Ywmax);
```

* This function specifies an orthogonal projection for mapping the scene to the screen. The orthogonal projection has no effect on our two-dimensional scene other than to convert object positions to normalized coordinate.



→ OpenGL Viewport function:-

glViewport($x_{v_{min}}$, $y_{v_{min}}$, v_p width, v_p Hight);

Where-

→ $x_{v_{min}}$ and $y_{v_{min}}$ specify the position of the lower left corner of the viewport relative to the lower_left corner of the display window.

→ v_p width and v_p Hight are pixel width and hight of the viewport

→ coordinate for the upper-right corner of the viewport are calculated for this transformation matrix in terms of the viewport width & hight.

$x_{v_{max}} = x_{v_{min}} + v_p$ width, $y_{v_{max}} = y_{v_{max}} + v_p$ Hight

→ glGetIntegerv(GL_VIEWPORT, vpArray)

→ vpArray is a single-subscript, four element array

→ Relocating and Resizing a GLUT Display window → (6)

* glut PositionWindow(x NewTopLeft, y NewTopLeft)
* glut Reshape window (dw NewWight, dw NewHeight);
* glut FullScreen();
* glut ReshapeFunc(winReshapeFen);

→ Managing multiple GLUT Display window →

* glut Iconify window();
* glutSetIconTitle("Icon Name");
* glut setwindowtitle("New window Name");
* glut setwindow(window ID);
* glut Pop window();
* glut set window(window ID);
* glut Push window();
* glut Hide window();
* glut show window();

→ GLUT Sub window:-

* glutCreateSubWindow(windowID, xBottomLeft, yBottomLeft, width, height);

→ Selecting a Display-window screen-cursor shape

* glutSetCursor(shape);

where
→ GLUT_CURSOR_UP_DOWN → up-down arrow
→ GLUT_CURSOR_CYCLE → rotating arrow is chosen
→ GLUT_CURSOR_WAIT → a wristwatch shape.
→ GLUT-CURSOR_DESTROY → a skull and crossbones.

→ Viewing Graphics objects in a GLUT Display window

* glutDisplayFunc(pictureDescrip);

* glutPostRedisplay();

⇒ Cohen-Sutherland line cliping algorithm:-

→ In this Algorithm, we given 9 region on the screen out of which one region is of the window and the rest 8 regions are around it given by 4 digit binary. The division of the region are based on

$(X_{max}\ Y_{max})$ & (X_{min}, Y_{min})

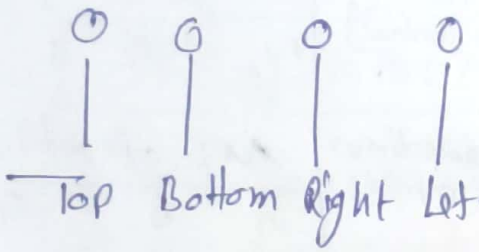
→ The central part is the vieiwing region or window. all the lines which lie within the region are completly visible.

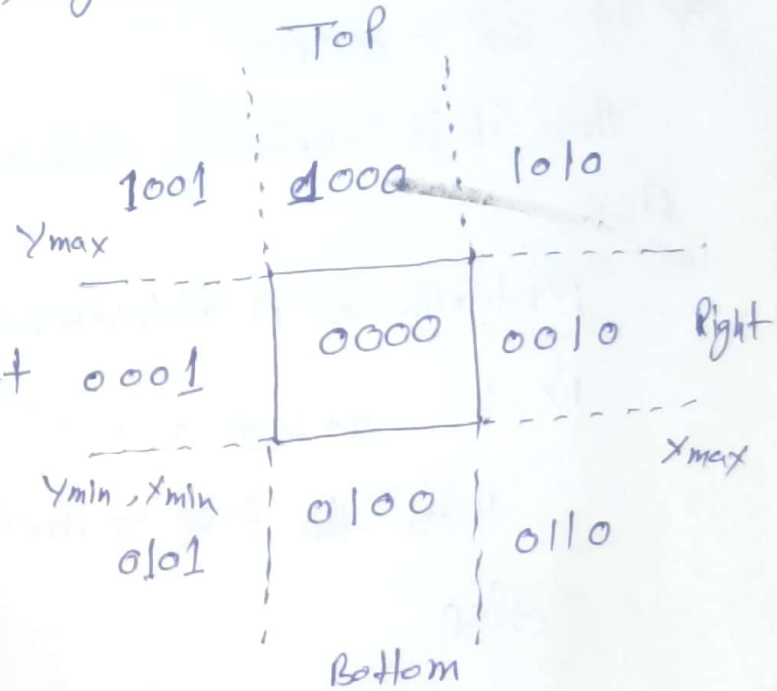
→ A region code is always assigned to the endpoints of the given line.

→ To check whether the line is visible or not.

⇒ Formula to check binary digits:- T B R L which can be defined as Top, bottom, Right and Left.

Nao → TBLR Rule





→ Rule:-

Accept 1) Line Inside

A & B code is "0000"

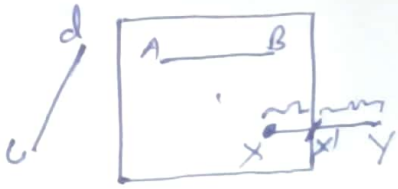
Reject 2) out side

c=0001, d=0001

c = 0 0 0 1

d = 0 0 0 1

logical AND = 0 0 0 1 → Not 0000



3) Partially inside the Window

X = 0 0 0 0

Y = 0 0 1 0

AND = 0 0 0 0

→ Find Intersection Point X'.

→ Algorithm:-

Steps:-

1) Assign the region codes to both end points.

2) Perform OR operation on both of these end points

3) if OR = 0000

then it is completely visible (inside the window)

else

Perform AND operation on both ~~side~~ these end points

i) if AND ≠ 0000

then the line is invisible

else

AND = 0000

Line is considered the clipped case

{partially inside the window}

OR operation			
A	B	⇒	O
0	0	⇒	0
0	1	⇒	1
1	0	⇒	1
1	1	⇒	1

AND operation			
A	B	⇒	Q
0	0	⇒	0
0	1	⇒	0
1	0	⇒	0
1	1	⇒	1

→ clipping

4) After confirming that the line is partially inside the window, then we find Intersection with the boundary of the window.

$$\text{slope} = m = \frac{Y_2 - Y_1}{X_2 - X_1}$$

a) If the line passes through the left region or the line intersects with the left boundary of the window_

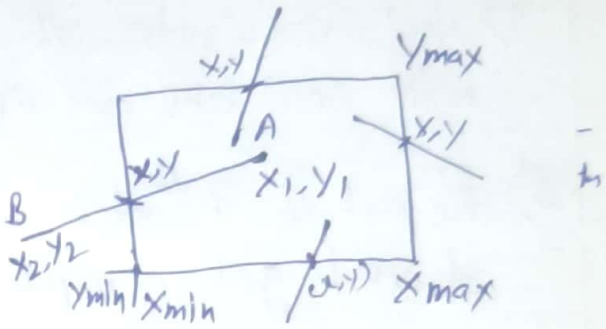
Left: $x = x_{min}$

Left. $m = \frac{y_2 - y_1}{x_2 - x_1}$

$$m = \frac{y - y_1}{x - x_1} = \frac{y - y_1}{x_{min} - x_1}$$

$$y - y_1 = m(x_{min} - x_1)$$

$$\boxed{y = y_1 + m(x_{min} - x_1)}$$



b) If the line passes through the Right region or the line intersect with the right boundary of the window.

Right: $x = x_{max}$

$$\boxed{y = y_1 + m(x_{max} - x_1)}$$

c) If the line passes through top or the line intersects with the top boundary of the window

Top: $y = y_{max}$

$$m = \frac{y - y_1}{x - x_1} = \frac{y_{max} - y_1}{x - x_1}$$

$$\boxed{x = x_1 + (y_{max} - y_1)/m}$$

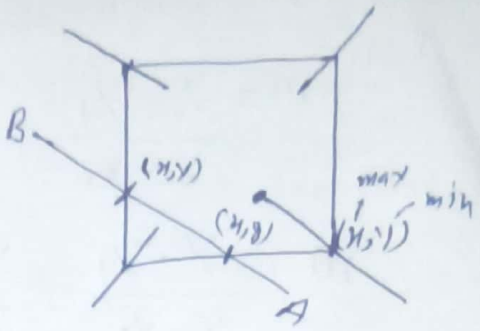
D) If the line passes the bottom or the intersect with bottom boundary

Bottom: $y = y_{min}$

$$\boxed{x = x_1 + (y_{min} - y_1)/m}$$

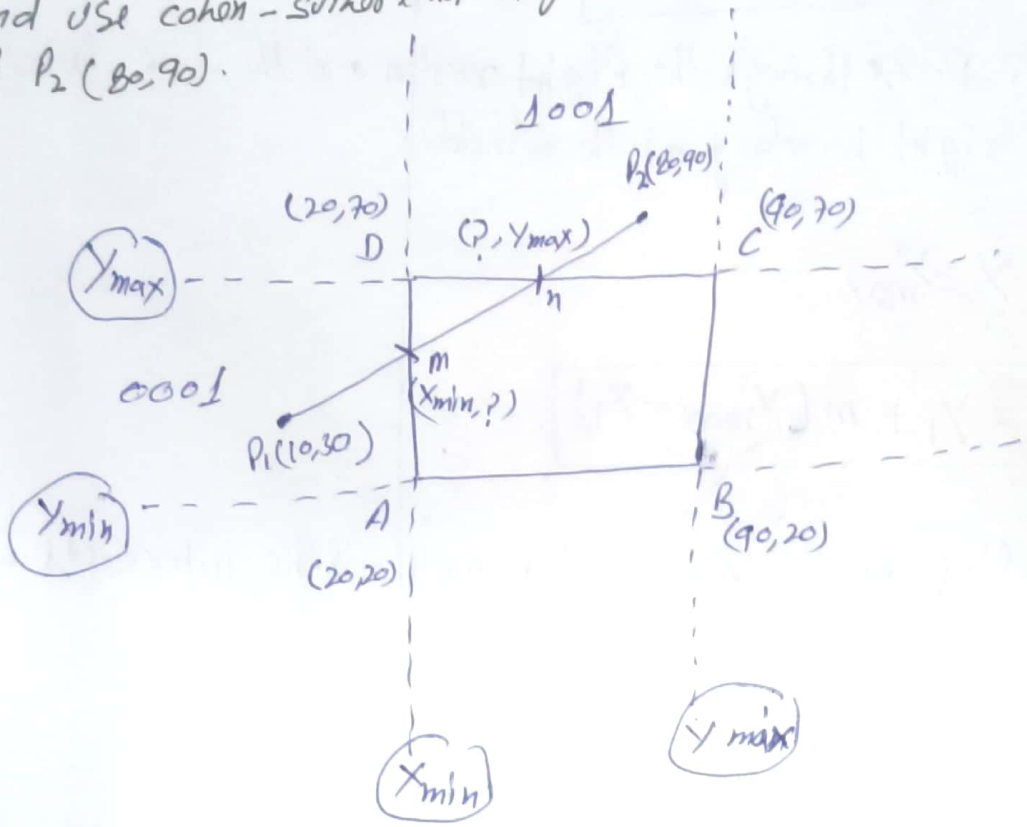
5) Now, over write the end points with new one and update it.

6) Repeat the 4th step till your line doesn't get completly clipped.



Ex: Let ABCD be the rectangular window with A (20,20), B (90,20) C (90,70), and D (20,70). Find region codes for the end points and use cohen-sutherland Algo. to clip the line $P_1 P_2$ with P_1(10,30) & P_2 (80,90).

→



→ Finding Region code:

① $P_1 = 0001$

$P_2 = 1001$

AND = 0000 → Line is Partially visible (clipping)

P_1 (10,30) → X_1, Y_1

P_2 (80,90) → X_2, Y_2

② slop $m = \frac{Y_2 - Y_1}{X_2 - X_1} \Rightarrow \frac{90-30}{80-10} \Rightarrow \frac{6}{7} \Rightarrow 0.857$

→ For point of intersection (m):-

$y = m(x_{min} - x_1) + y_1$

$= 0.857(20-10) + 30$

$= 8.57 + 30$

$= 38.57$

$= 39$

m(20, 39)

$x_{min} = 20$

$y_{max} = 70$

→ For point n :- (Intersection)

$x = \frac{1}{m}(y_{max} - y_1) + x_1$

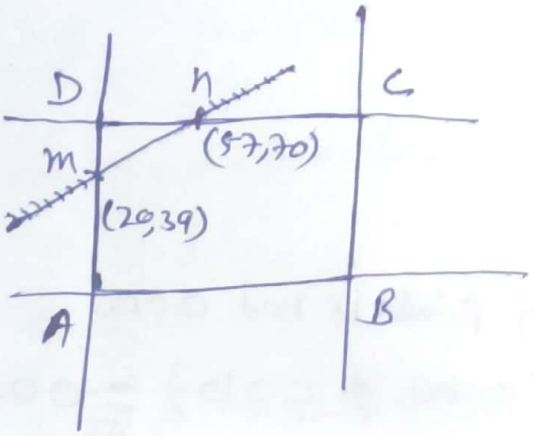
$x = \frac{1}{m}(70-30) + 10$

$= \frac{1}{0.857}(40) + 10$

$= 56.67$

$x \simeq 57$

n(57, 70)



~~Cohen Sutherland line clipping Algorithm~~

→ Formula to calculate 4b bit code:-

* Bit 1 = sign of $(y - y_t)$
* Bit 2 = sign of $(y_B - y)$
* Bit 3 = sign of $(x - x_R)$
* Bit 4 = sign of $(x_L - x)$

* sign = 1 → +ve
* sign = 0 → -ve

Q) Use the Cohen Sutherland algorithm to determine the visibility of a line P_1P_2 $P_1(70,20)$, $P_2(100,10)$ against a window with lower left hand corner at (50,10) and upper right hand corner at (80,40).

→

$P_1(70,20)$ → x, y

→ outcode for P_1:

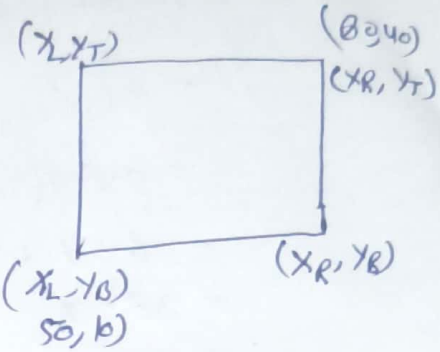
Bit1 = S(20-40) = -20 = 0

Bit2 = S(10-20) = -10 = 0

Bit3 = S(70-80) = -10 = 0

Bit4 = S(50-70) = -10 = 0

(0000)



window

(10,10)

$P_2(100,10)$ → x, y

→ outcode to P_2:

Bit1 = S(10-40) = -30 = 0

Bit2 = S(10-10) = 0

Bit3 = S(100-80) = -20 ⇒ 1

Bit4 = S(50-100) = -50 = 0

(0010)

→ As the outcode for both the end point is not 0000

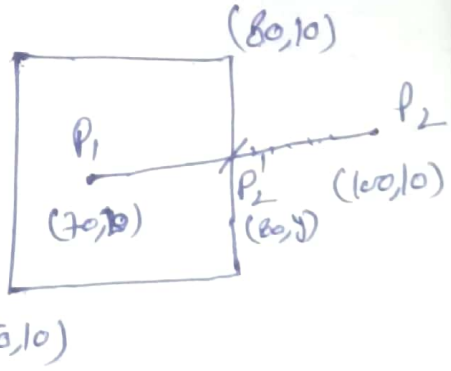
∴ the line is partially visible (0000 & 0010) = 0000

→ Now determine the visible portion of this line

TBRL

As the outcome for P_2 is 0010

∴ It is intersecting the right edge of window



→ Determine the intersection point P_2'

Slope of the line $P_1P_2 = \frac{10-20}{100-70} = \frac{-10}{30} = -1/3$

Now $P_2'P_1 \Rightarrow m = \frac{y-20}{80-70} \Rightarrow \frac{-1}{3} = \frac{y-20}{10}$

$\Rightarrow 3(y-20) = -10$

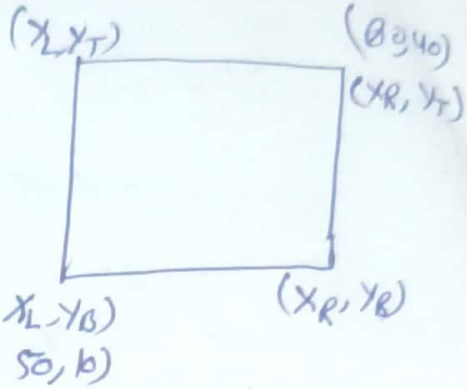
$3y = 60 - 10$

$y = \left(\frac{50}{3}\right) \Rightarrow 16.67 \cong 17$

→ P_1P_2' is the visible portion of line with coordinate (70,20) & (80,17)

→ Outcode for P_1:

$P_1(70, 20)$
x, y



Bit1 = S(20-40) = -20 = 0
Bit2 = S(10-20) = -10 = 0
Bit3 = S(70, -80) = -10 = 0
Bit4 = S(50-70) = -10 = 0

(0000)

→ $P_2(100, 10)$
x, y

→ Outcode to P_2:

Bit1 = S(10-40) = -30 = 0
Bit2 = S(10-10) = 0
Bit3 = S(100-80) = 20 ⇒ 1
Bit4 = S(50-100) = -50 = 0

(0010)

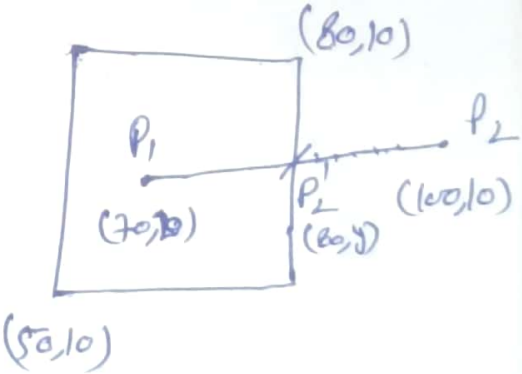
→ As the outcode for both the end point is not 0000
∴ the line is partially visible (0000 & 0010) = 0000

→ How determine the visible portion of this line

TBRL

→ As the outcome for P_2 is 0010

∴ It is intersecting the right edge of window



→ Determine the Intersection Point P_2'

Slope of the line $P_1P_2 = \frac{10-20}{100-70} = \frac{-10}{30} = -1/3$

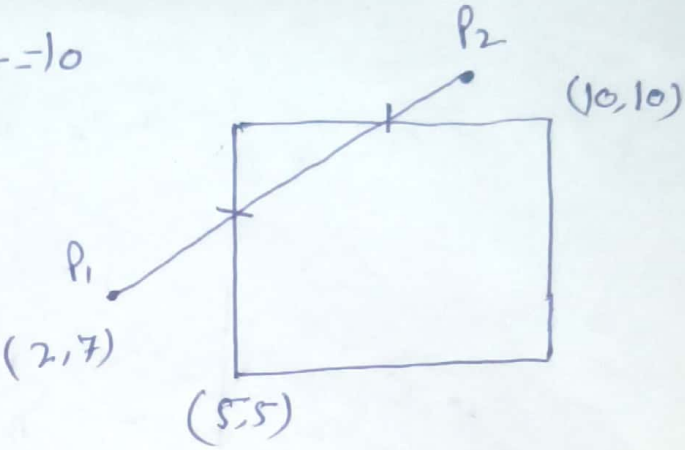
Now $P_2'P_1 \Rightarrow m = \frac{y-20}{80-70} \Rightarrow \frac{-1}{3} = \frac{y-20}{10}$

$\Rightarrow 3(y-20) = -10$
$3y = 60 - 10$
$y = \left(\frac{50}{3}\right) \Rightarrow 16.67 \cong 17$

→ P_1P_2' is the visible portion of line with coordinate (70,20) & (80,17)

Q:- Use the clipping Algorithm for calculating the visible portion of the line $P_1(2,7)$, $P_2(8,12)$ against a window where $(X_{min}, Y_{min}) = (5,5)$ & $(X_{max}, Y_{max}) = (10,10)$. (10)

→ $X_L = 5$, $X_R = 10$, $Y_B = 5$, $Y_T = 10$



Bit code $P_1(2,7)$	Bitcode $P_2(8,12)$
$B_1 = 7-10 = 0$	$B_1 = 12-10 = 1$
$B_2 = 5-7 = 0$	$B_2 = 5-12 = 0$
$B_3 = 2-10 = 0$	$B_3 = 8-10 = 0$
$B_4 = 5-2 = 1$	$B_4 = 5-8 = 0$

→ Perform And operation

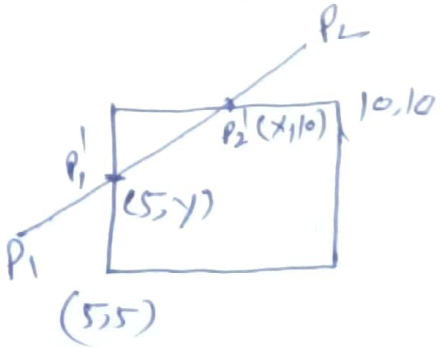
P_1 - 0 0 0 1

P_2 - 1 0 0 0

AND - 0 0 0 0 → Line is partially visible

→ As a result of AND in operation for bitcode of both the end point 0000, so the line is Partially visible and need to be clipped.

→ The out code P_1 is 0001
∴ we can calculate that the line is extending (0001) left edge of the window



→ Line P_1P_2 : $(2,7)$ $(8,12)$ $m = \frac{5}{6}$

Line $P_1'P_2$: $(5,y)$ $(8,12)$ $m = \frac{12-y}{3} = \frac{5}{6} \Rightarrow 9.5$

→

→ Now, also finding out code of $P_1'(5, 9.5)$
x, y

$B_1 = 9.5 - 10 = 0$

$B_2 = 5 - 9.5 = 0$

$B_3 = 5 - 10 = 0$

$B_4 = 5 - 5 = 0$

→ As its bit code of P_1' is 0000
∴ It is inside the clipping window

$P_1' P_2'$ $m = \frac{10 - 9.5}{X - 5} = \frac{5}{6}$ $x = 5.6$

→ Finding out code of $P_2'(5.6, 10)$

$B_1 = 10 - 10 = 0$ | $B_3 = 5.6 - 10 = 0$

$B_2 = 5 - 10 = 0$ | $B_4 = 5 - 5.6 = 0$

→ As the out code of both the end points P_1', P_2' 0000, ∴ Line is completly inside the clipping window & visible portion (5, 9.5) & (5.6, 10)

⇒ Cyrus-beak line clipping Algorithm:

Region code P_3P_4 - $P_3 - 0101$

$P_4 - 0000$

0000 Partially visible

slope $m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{60-10}{70-10} = \frac{50}{60} = \frac{5}{6} = 0.833.$

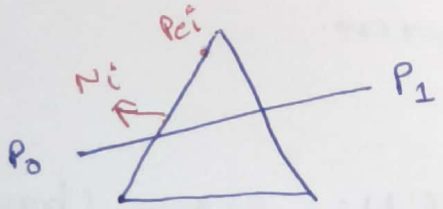
$i_3(x_{min}, y) = i_3(20, y).$ $\left\{ \because i_3(20, 18.33) \right.$

$y = m(x_{min} - x_1) + y_1$

$= \frac{5}{6}(20-10) + 10 = 18.33$

~~$i_4(x_0, y_{max}) =$~~ P_4 is already 0000.

Cyrus-Back line clipping algorithm



1) Dot Product
2) Parametric eq. line.

$P(t) = P_0 + t(P_1 - P_0)$

(1) Normal vector → outward to lin

N_i

i = edge no.

(2) Take a point anywhere on line

Pe_i Point at edge $i = 1, 2, 3 \ldots$

(1) Dot Product $N_i \cdot (P(t) - Pe_i)$

$N_i \cdot (P_0 + t(P_1 - P_0) - Pe_i) = 0$

$N_i \cdot P_0 + N_i t P_1 - N_i t P_0 - N_i Pe_i = 0$

$N_i t P_1 - N_i t P_0 = N_i Pe_i - N_i P_0$

$t = \frac{N_i Pe_i - N_i P_0}{N_i P_1 - N_i P_0} = \frac{N_i (Pe_i - P_0)}{N_i (P_1 - P_0)}$

$t = \frac{(Pe_i - P_0) N_i}{(P_1 - P_0) N_i}$

→ Nor

$N_i(P_1 - P_0) = D$ { Denominator

$D > 0$ leaving point

$D < 0$ entering point

$D = 0$

→ Calculate t_1, t_2, t_3

→ make list of enty or leaving point.

→ Pick values highest in enty point list

→ Pick value lowest in leaving point

→ Put t in.

(2) $P(t) = P_0 + t(P_1 - P_0)$

Parametric eq.

- cyrus back is a line clipping algorithm that is allow clipping for non-rectangular.

Steps ① firstly find the exterior point on the edge and, which is represent in a general way n(normal vector) which is an edge no.

Step 2 Now take a point on every edge or a vertex which is represent by P_{e1} ---- P_{ei} (point at edge)

step 3 After ~~take a point on every~~ calculating N_i or P_{ei} then by using both values we can calculate the time for every edge of polygon.

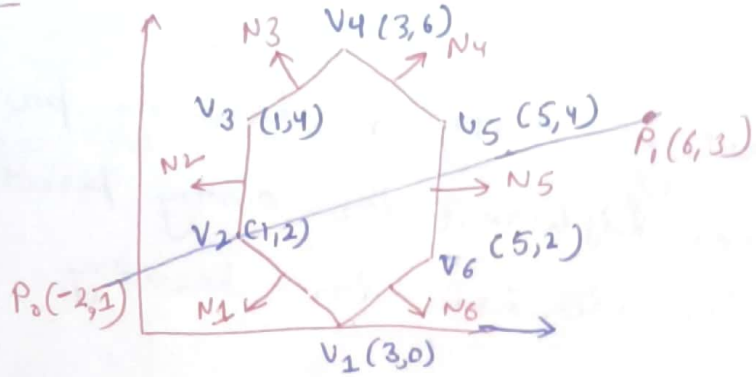
$$t = \frac{N_i(P_{ei} - P_0)}{N_i(P_1 - P_0)}$$

Step 4 After calculating time for all edge by using P_0 and P_1 and use this time bo

formula: $P(t) = P_0 + t(P_1 - P_0)$

find out the value of P_{e_0} and P_1.

Numerical :-



$N_1(-1,-1)$

$N_2(-1,0)$

$N_3(-1,1)$

$N_4(1,1)$

$N_5(1,0)$

$N_6(1,-1)$

$$P(t) = P_0 + t(P_1 - P_0)$$

$$t_i = \frac{N_i (P_{ei} - P_0)}{N_i (P_1 - P_0)}$$

$$t_1 = \frac{(-1,-1)\left((3,0) - (-2,1)\right)}{(-1,-1)\left((6,3) - (-2,1)\right)}$$

$$\Rightarrow \frac{(-1,-1)\left(3-(-2), (0-1)\right)}{(-1,-1)\left((6-(-2), (3-1)\right)}$$

$$\Rightarrow \frac{(-1,-1)(5,-1)}{(-1,-1)(8,2)} = 01.$$

$$\Rightarrow \frac{-5,+1}{-8\ -2} = \frac{-4}{-10}$$

$$t_2 = \frac{(-1,0)[(1,2) - (-2,1)]}{(-1,0)[(6,3) - (-2,1)]} = \frac{(-1,0)[(1+2), (2-1)]}{(-1,0)((6+2), (3-1)]}$$

$$= \frac{(-1,0)(3,1)}{(-1,0)(8,2)} = \frac{-3}{-8}.$$

$t_3 = \frac{8}{-6}$ $t_4 = \frac{10}{10}$ $t_5 = \frac{7}{8}$ $t_6 = \frac{6}{6}$

Entering point = t_1, t_2, t_3

leaving point = t_4, t_5, t_6

$P(t) = P_0 + t(P_1 - P_0)$

$= (-2,1) + \left(\frac{4}{10}\right)((6,3) - (-2,1))$

$= (-2,1) + \frac{4}{10}((6-(-2)), (3-1))$

$= (-2,1) + \left(\frac{4^2}{10_5} \times 8, \frac{4}{10_5} \times 2\right)$

$= (-2,1) + \left(\frac{16}{5}, \frac{4}{5}\right)$

$= \left(-2 + \frac{16}{5} \times 2\right), \left(\frac{4}{5} + 1\right) = \left(\frac{6}{5}, \frac{9}{5}\right)$

$P(t) = (-2,1) + \frac{7}{8}(8,2)$

$= (-2,1) + \frac{7}{8} \times 8, \frac{7}{8} \times 2$

$= (-2,1) + \left(7, \frac{7}{4}\right) \Rightarrow (-2+7), 1 + \frac{7}{4}$

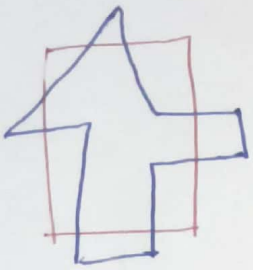
$= \left(5, \frac{11}{4}\right)$

Sutherland - Hodgeman Polygon clipping al

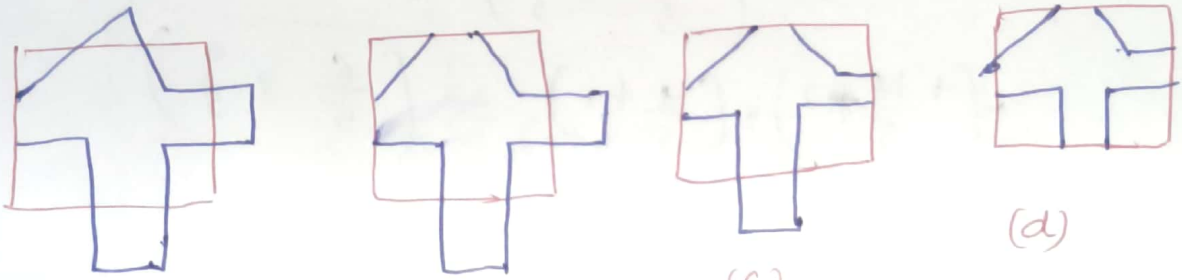
The Sutherland-Hodgeman algorithm perfor... clipping of a polygon against each window ... in turn.

The order to clip the polygon should be:

1. clipping against the left side of the windo...
2. Clipping against the top side of the window.
3. clipping against the right side of the window.
4. clipping against the bottom side of the window



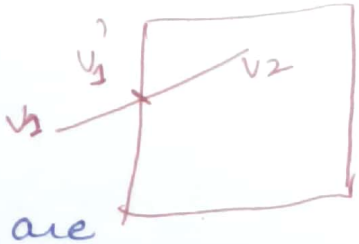
Before clipping



(a) (b) (c) (d)

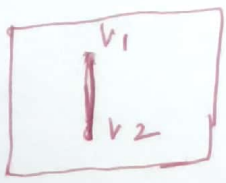
It consists of four cases :-

(1) If the first vertex is outside the window, second vertex is inside then the intersection point of the polygon edge with the window boundary and second vertex are added to the output vertext list.



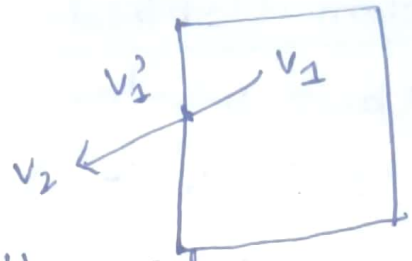
Save $\{v_1'$ and $v_2\}$

(2) If both vertices of edge are inside the window. Then only second vertex is added to the output vertex list.



Save $\{v_2\}$

(3) If the first vertex of the edge is inside and second is outside then only the intersection point of the polygon edge with the window boundary is added to the output vertex list.

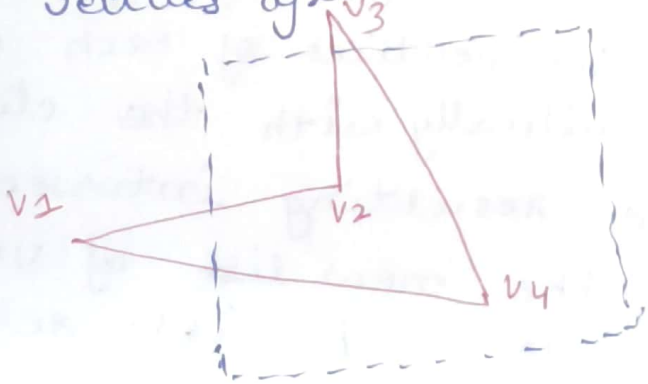


Save $\{ v_1' \}$

(4) If both vertices of the edge are outside the window. Then nothing is added.

Save $\{ \phi \}$

Que for a polygon and clipping window. give the list of vertices after each boundary clipping.

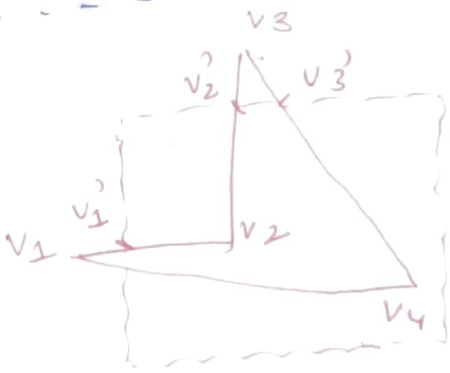


Solution

(1) Left clipped

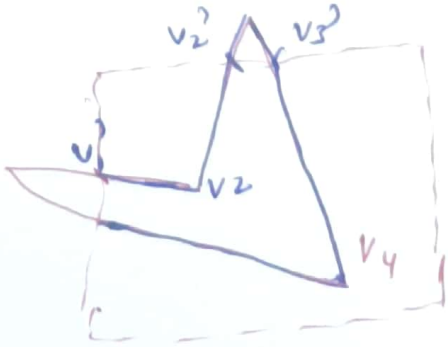
output vertex:-

$\{ v_1', v_2, v_2', v_3, v_3', v_4, v_4' \}$



(2) Top clipped

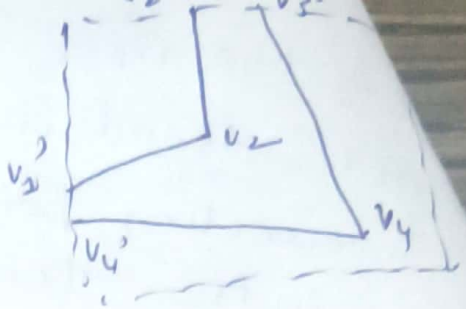
$\{ v_1', v_2, v_2', v_3', v_4, v_4' \}$



③ Right clipped :-

output vertex =

$\{V_1', V_2, V_2', V_3', V_4, V_4'\}$



④ Bottom clipped :-

output vertex =

$\{V_1', V_2, V_2', V_3', V_4, V_4'\}$

final clipped polygon :- $\{V_1', V_2, V_2', V_3', V_4, V_4'\}$

Algorithm :- ① Read coordinates of all vertices of the polygon.

② Read coordinates of the clipping window.

③ consider the left edge of the window.

④ compare the vertices of each edge of the polygon, individually with the clipping plane.

⑤ save the resulting intersections and vertices in the new list of vertices according to four possible relationships between the edge and clipping boundary.

⑥ Repeat step ④ & ⑤ for remaining edges of the clipping window.

⑦ stop.