# UNIT 3 2-D VIEWING AND CLIPPING

**Structure** **Page No.**



## 3.1 INTRODUCTION

In the earlier two units of this block, we discussed the basic components of computer graphics, i.e., the software and hardware used for graphic systems along with the graphic primitives used to create graphic images. Now, we are in a position to discuss technicalities related to the display of any graphic primitive or image. Let us begin with, the concept of clipping, for both point and line clipping, followed by the concerned algorithms used to perform line clipping. We shall then examine the concept and algorithm related to polygon clipping, and end with a discussion on window to viewport transformations.

**Clipping** may be described as the procedure that identifies the portions of a picture lie inside the region, and therefore, should be drawn or, outside the specified region, and hence, not to be drawn. The algorithms that perform the job of clipping are called clipping algorithms there are various types, such as:

- Point Clipping
- Line Clipping
- Polygon Clipping
- Text Clipping
- Curve Clipping

Further, there are a wide variety of algorithms that are designed to perform certain types of clipping operations, some of them which will be discussed in unit.

Line Clipping Algorithms:
- Cohen Sutherland Line Clippings
- Cyrus-Beck Line Clipping Algorithm

Polygon or Area Clipping Algorithm
- Sutherland-Hodgman Algorithm

There are various other algorithms such as, Liang – Barsky Line clipping, Weiler-Atherton Polygon Clipping, that are quite efficient in performing the task of clipping images. But, we will restrict our discussion to the clipping algorithms mentioned earlier.
Before going into the details of point clipping, let us look at some basic terminologies used in the field of clipping, such as, window and viewport.

*Window* may be described as the world coordinate area selected for display.

*Viewport* may be described as the area on a display device on which the window is mapped.

So, it is the window that specifies what is to be shown or displayed whereas viewport specifies where it is to be shown or displayed.

Specifying these two coordinates, i.e., window and viewport coordinates and then the transformation from window to viewport coordinates is very essential from the point of view of clipping.

**Note:**

- Assumption: That the window and viewport are rectangular. Then only, by specifying the maximum and the minimum coordinates *i.e.*, ($Xw_{max}$, $Yw_{max}$) and ($Xw_{min}$, $Yw_{min}$) we can describe the size of the overall window or viewport.
- Window and viewport are not restricted to only rectangular shapes they could be of any other shape (Convex or Concave or both).

For better understanding of the clipping concept refer to *Figure 1*:

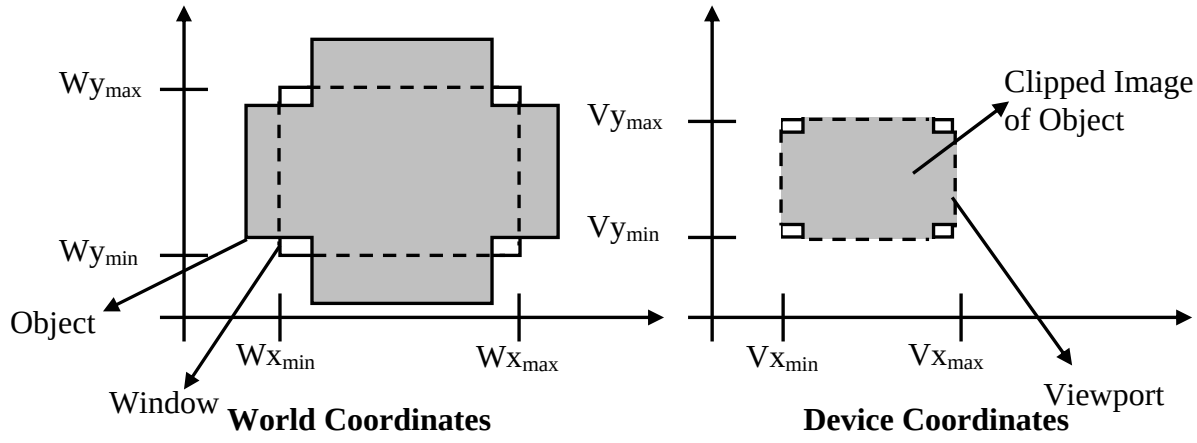



**Figure 1: Clipping alongwith Viewing Transformation through rectangular window and viewport**

## 3.2 OBJECTIVES

After going though this unit, you should be able to:

- explain the concept of clipping,
- examine how line clipping is performed,
- understand and implement the algorithms that work behind the concept of line clipping,
- explain how polygon clipping is performed,
- understand and implement the algorithms that work behind the concept of polygon clipping, and
- describe window to viewport transformation.

## 3.3 POINT CLIPPING

Point clipping is the technique related to proper display of points in the scene, although, this type of clipping is used less frequently in comparison to other types, i.e., line and polygon clipping. But, in some situations, e.g., the scenes which involve particle movements such as explosion, dust etc., it is quite useful. For the sake of simplicity, let us assume that the clip window is rectangular in shape. So, the minimum and maximum coordinate value, i.e., ($Xw_{max}$, $Yw_{max}$) and ($Xw_{min}$, $Yw_{min}$)

are sufficient to specify window size, and any point (X,Y), which can be shown or displayed should satisfy the following inequalities. Otherwise, the point will not be visible. Thus, the point will be clipped or not can be decided on the basis of following inequalities.

$Yw_{max}$

$Xw_{min} \leq X \leq Xw_{max}$

$Yw_{min} \leq Y \leq Yw_{max}$

$Yw_{min}$

$Xw_{min}$ $Xw_{max}$

**Figure 2: Point Clipping**

It is to be noted that ($Xw_{max}$, $Yw_{max}$) and ($Xw_{min}$, $Yw_{min}$) can be either world coordinate window boundary or viewport boundary. Further, if any one of these four inequalities is not satisfied, then the point is clipped (not saved for display).

# 3.4 LINE CLIPPING

Line is a series of infinite number of points, where no two points have space in between them. So, the above said inequality also holds for every point on the line to be clipped. A variety of line clipping algorithms are available in the world of computer graphics, but we restrict our discussion to the following Line clipping algorithms, name after their respective developers:

1) Cohen Sutherland algorithm,
2) Cyrus-Beck of algorithm

## 3.4.1 Cohen Sutherland Line Clippings

This algorithm is quite interesting. The clipping problem is simplified by dividing the area surrounding the window region into four segments Up, Down, Left, Right (U,D,L,R) and assignment of number 1 and 0 to respective segments helps in positioning the region surrounding the window. How this positioning of regions is performed can be well understood by considering *Figure 3*.

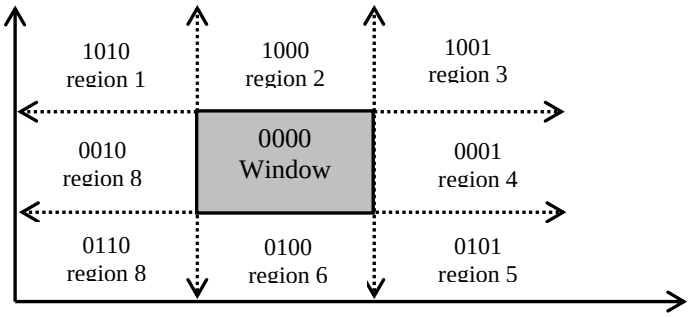



**Figure 3: Positioning of regions surrounding the window**

In *Figure 3* you might have noticed, that all coding of regions U,D,L,R is done with respect to window region. As window is neither Up nor Down, neither Left nor Right, so, the respective bits UDLR are 0000; now see region 1 of *Figure 3*. The positioning code UDLR is 1010, i.e., the region 1 lying on the position which is upper left side of the window. Thus, region 1 has UDLR code 1010 (Up so U=1, not Down so D=0, Left so L=1, not Right so R=0).

The meaning of the UDLR code to specify the location of region with respect to window is:

1st bit ⇒ Up(U) ; 2nd bit ⇒ Down(D) ;3rd bit ⇒ Left(L) ; 4th bit ⇒ Right(R),

Now, to perform Line clipping for various line segment which may reside inside the window region fully or partially, or may not even lie in the widow region; we use the tool of logical ANDing between the UDLR codes of the points lying on the line.

Logical ANDing (^) operation between respective bits implies => 1 ^ 1 = 1; 1 ^ 0 = 0; 0 ^ 1 = 0; 0 ^ 0 = 0

**Note:**

- UDLR code of window is 0000 always and w.r.t. this will create bit codes of other regions.
- A line segment is visible if both the UDLR codes of the end points of the line segment equal to 0000 i.e. UDLR code of window region. If the resulting code is not 0000 then, that line segment or section of line segment may or may not be visible.

Now, let us study how this clipping algorithm works. For the sake of simplicity we will tackle all the cases with the help of example lines $l_1$ to $l_5$ shown in *Figure 4*. Each line segment represents a case.

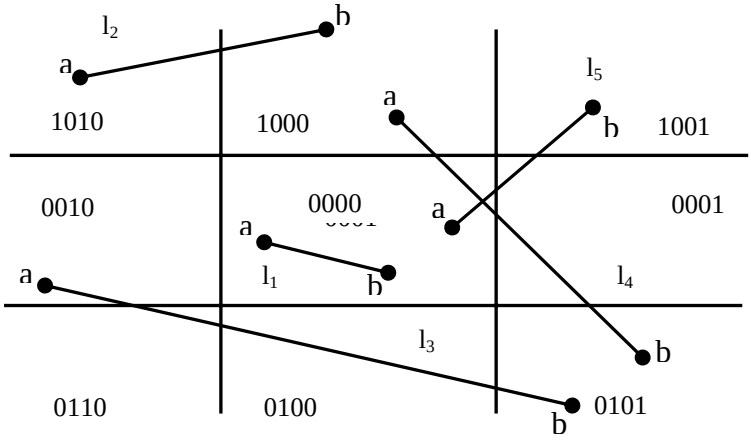



**Figure 4: Various cases of Cohen Sutherland Line Clippings**

Note, that in *Figure 4*, line $l_1$ is completely visible, $l_2$ and $l_3$ are completely invisible; $l_4$and $l_5$ are partially visible. We will discuss these out comings as three separate cases.

**Case 1:** $l_1 \rightarrow$ Completely visible, i.e., Trival acceptance ($\therefore$ both points lie inside the window)

**Case 2:** $l_2$ and $l_3 \rightarrow$ Invisible , i.e., Trival acceptance rejection

**Case 3:** $l_4$ and $l_5 \rightarrow$ partially visible ($\therefore$ partially inside the window)
Now, let us examine these three cases with the help of this algorithm:

**Case 1:** (Trivial acceptance case) *if the UDLR bit codes of the end points P,Q of a given line is 0000 then line is completely visible.* Here this is the case as the end points a and b of line $l_1$ are: a (0000), b (0000). If this trival acceptance test is failed then, the line segment PQ is passed onto Case 2.

**Case 2:** (Trivial Rejection Case) *if the logical intersection (AND) of the bit codes of the end points P, Q of the line segment is $\neq$0000 then line segment is not visible or is rejected.*

Note that, in *Figure 4*, line 2 is completely on the top of the window but line 3 is neither on top nor at the in bottom plus, either on the LHS nor on the RHS of the

window. We use the standard formula of logical ANDing to test the non visibility of the line segment.

So, to test the visibility of line 2 and 3 we need to calculate the logical intersection of end points for line 2 and line 3.

**line l2:** bit code of end points are 1010 and 1000
logical intersection of end points = (1010) ^ (1000) = 1000
as logical intersection ≠ 0000. So line 2 will be invisible.

**line l3:** end points have bit codes 0010 and 0101 now logical intersection = 0000, i.e., 0010 ^ 0101 = 0000 from the *Figure 4*, the line is invisible. Similarly in line 4 one end point is on top and the other on the bottom so, logical intersection is 0000 but then it is partially visible, same is true with line 5. These are special cases and we will discuss them in case 3.

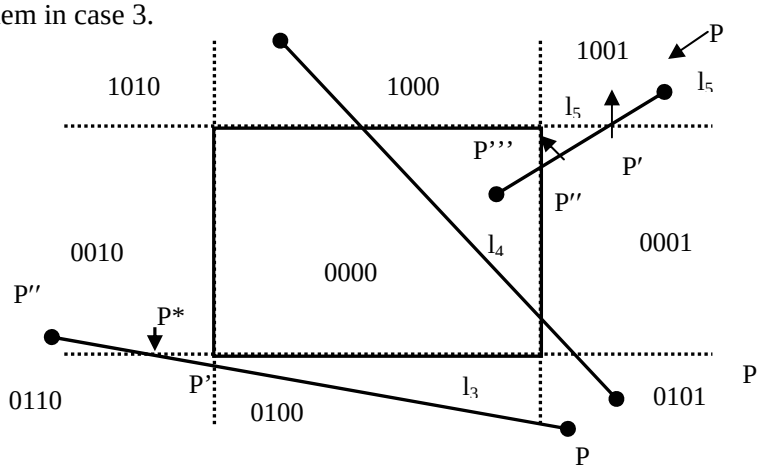



**Figure 5: Cohen Sutherland line clipping**

**Case 3:** Suppose for the line segment PQ, both the trivial acceptance and rejection tests failed (i.e., Case 1 and Case 2 conditions do not hold, this is the case for $l_3$, $l_4$ and $l_5$ line segments) shown above in *Figure 5*. For such non-trivial Cases the algorithm is processed, as follows.

Since, both the bitcodes for the end points P, Q of the line segment cannot be equal to 0000. Let us assume that the starting point of the line segment is P whose bit code is not equal to 0000. For example, for the line segment $l_5$ we choose P to be the bitcodes 1001. Now, scan the bitcode of P from the first bit to the fourth bit and find the position of the bit at which the bit value 1 appears at the first time. For the line segment $l_5$ it appears at the very first position. If the bit value 1 occurs at the first position then proceed to intersect the line segment with the UP edge of the window and assign the first bit value to its point of intersection as 0. Similarly, if the bit value 1 occurs at the second position while scanning the bit values at the first time then intersect the line segment PQ with the Down edge of the window and so on. This point of intersection may be labeled as P'. Clearly the line segment PP' is outside the window and therefore rejected and the new line segment considered for dipping will be P'Q. The coordinates of P' and its remaining new bit values are computed. Now, by taking P as P', again we have the new line segment PQ which will again be referred to Case 1 for clipping.

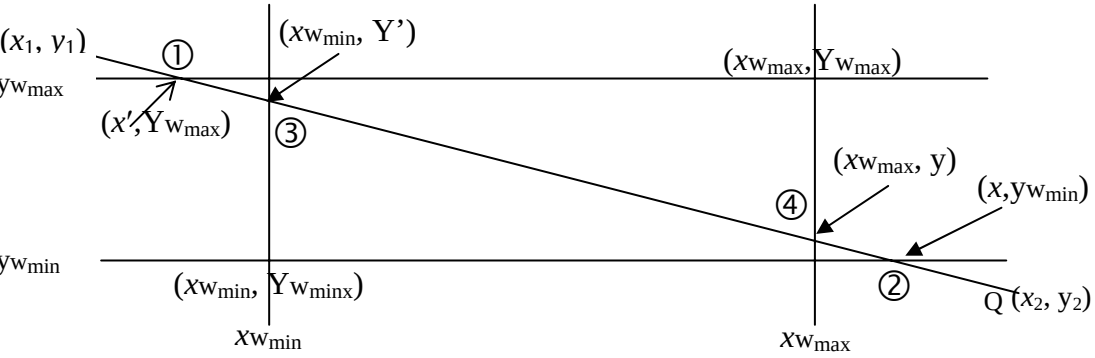


**Figure 6: Line Clipping - Geometrically**

Geometrical study of the above type of clipping (it helps to find point of intersection of line PQ with any edge).

Let $(x_1, y_1)$ and $(x_2, y_2)$ be the coordinates of P and Q respectively.

1) top case/above
   if $y_1 > yw_{max}$ then 1st bit of bit code = 1 (signifying above) else bit code = 0

2) Bottom case/below case
   if $y_1 < yw_{min}$ then 2nd bit = 1 (*i.e.* below) else bit = 0

3) Left case: if $x_1 < xw_{min}$ then 3rd bit = 1 (*i.e.* left) else 0

4) Right case: if $x_1 > xw_{max}$ then 4th bit = 1 (*i.e.* right) else 0

Similarly, the bit codes of the point Q will also be assigned.

1) **Top/above case:**

   equation of top edge is: $y = yw_{max}$ . The equation of line PQ is

   $$y - y_1 = m\,(x - x_1)$$

   where,

   $$m = (y_2 - y_1)/(x_2 - x_1)$$

   The coordinates of the point of intersection will be $(x, yw_{max})$ $\therefore$ equation of line between point P and intersection point is

   $$(yw_{max} - y_1) = m\,(x - x_1)$$

   rearrange we get

   $$x = x_1 + \frac{1}{m}\,(yw_{max} - y_1) \qquad \text{---------- (A)}$$

   Hence, we get coordinates $(x, yw_{max})$ i.e., coordinates of the intersection.

2) **Bottom/below edge** start with $y = yw_{min}$ and proceed as for above case.

   $\therefore$ equation of line between intersection point $(x', yw_{min})$ and point Q *i.e.* $(x_2, y_2)$ is

   $$(yw_{min} - y_2) = m\,(x' - x_2)$$

   rearranging that we get,

   $$\boxed{x' = x_2 + \frac{1}{m}\,(yw_{min} - y_2)} \qquad \text{---------- (B)}$$

   The coordinates of the point of intersection of PQ with the bottom edge will be

   $$\left(x_2 + \frac{1}{m}(yw_{min} - y_2), yw_{min}\right)$$

3) **Left edge:** the equation of left edge is $x = xw_{min}$.

   Now, the point of intersection is $(xw_{min}, y)$.
   Using 2 point from the equation of the line we get

   $$(y - y_1) = m\,(xw_{min} - x_1)$$

   Rearranging that, we get, $y = y_1 + m\,(xw_{min} - x_1)$. ---------- (C)

Hence, we get value of $xw_{min}$ and y both *i.e.* coordinates of intersection point is given by $(xw_{\min}, y_1 + m(xw_{\min} - x_1))$.

4) **Right edge:** proceed as in left edge case but start with x-$xw_{max}$.

Now point of intersection is ($xw_{max}$, y′).

Using 2 point form, the equation of the line is

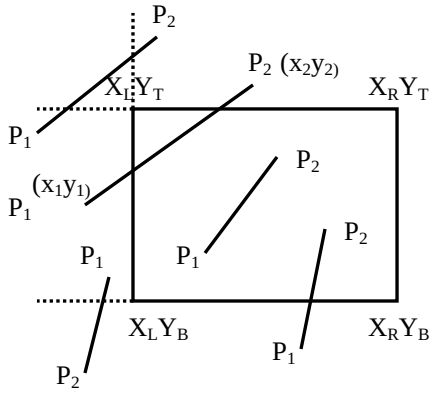
$(y' - y_2) = m\ (xw_{max} - x_2)$

$$y' = y_2 + m\ (xw_{max} - x_2) \quad \text{(D)}$$

---

The coordinates of the intersection of PQ with the right edge will be $(xw_{\max}, y_2 + m(xw_{\max} - x_2))$.

Steps of the Cohen Sutherland Line Clipping Algorithm are




L→Left ;
R→ Right
T→Top ;
B→Bottom

**Figure 7: Steps for Cohen Sutherland Line Clipping**

**STEP 1:**

Input: $x_L, x_R, y_T, y_B, P_1(x_1, y_1), P_2(x_2, y_2)$
Initialise $i = 1$
While $i <= 2$
if $x_i < x_L$ then bit 1 of code –$P_i$ = 1 else 0
if $x_i > x_R$ then bit 2 of code –$P_i$ =1 else 0 : The endpoint codes of the line are set
if $y_i < y_B$ then bit 3 of code –$P_i$ = 1 else 0
if $y_i > y_T$ then bit 4 of code –$P_i$ = 1 else 0
i = i +1
end while
i = 1

**STEP 2:**

Initialise j = 1
While j <= 2
if $x_j < x_L$ then $C_{j\,left} = 1$ else $C_{j\,left} = 0$
if $x_j > x_R$ then $C_{j\,right} = 1$ else $C_{j\,right} = 0$ :Set flags according to the position of the line endpoints w.r.t. window edges
if $y_j < y_B$ then $C_{j\,bottom} = 1$ else $C_{j\,bottom} = 0$

if $y_j > y_T$ then $C_{j\,top} = 1$ else $C_{j\,top} = 0$

end while

**STEP 3:** If codes of $P_1$and $P_2$ are both equal to zero then draw $P_1P_2$ (totally visible)

**STEP 4:** If logical intersection or AND operation of code $-P_1$ and code $-P_2$ is not equal to zero then ignore $P_1P_2$ (totally invisible)

**STEP 5:** If code $-P_1 = 0$ then swap $P_1$ and $P_2$ along with their flags and set i = 1

**STEP 6:** If code $-P_1 <> 0$ then

for i = 1,
{if $C_{1\,left} = 1$ then
find intersection $(x_L, y'_L)$ with left edge vide eqn. (C)
assign code to $(x_L, y'_L)$
$P_1 = (x_L, y'_L)$
end if
i = i + 1;
go to 3
}

for i = 2,
{if $C_{1\,right} = 1$ then
find intersection $(x_R, y'_R)$ with right edge vide eqn. (D)
assign code to $(x_R, y'_R)$
$P_1 = (x_R, y'_R)$
end if
i = i + 1
go to 3
}

for i = 3
{if $C_{1\,bottom} = 1$ then
find intersection $(x'_B, y_B)$ with bottom edge vide eqn. (B)
assign code to $(x'_B, y_B)$
$P_1 = (x'_B, y_B)$
end if
i = i + 1
go to 3
}

for i = 4,
{if $C_{1\,top} = 1$ then
find intersection $(x'_T, y_T)$ vide eqn. (A) with top edge
assign code to $(x'_T, y_T)$
$P_1 = (x'_T, y_T)$
end if
i = i + 1
go to 3
}
end

**Example:** A Clipping window ABCD is located as follows:

A(100, 10), B(160, 10, C(160, 40), D(100, 40). Using Sutherland-Cohen clipping algorithm find the visible portion of the line segments EF, GH and $P_1P_2$. E(50,0), F(70,80), G(120, 20), H(140, 80), $P_1$(120, 5), $P_2$(180, 30).

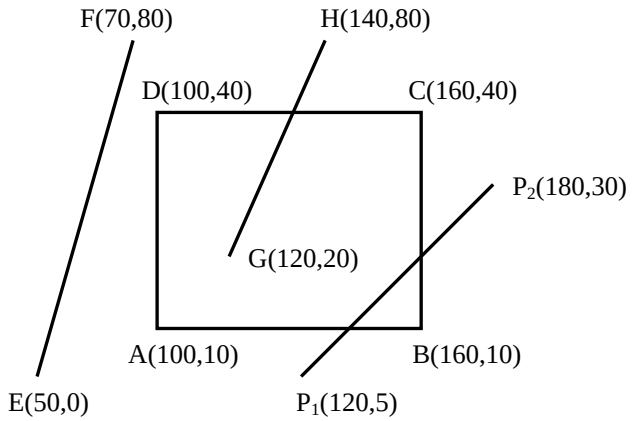



**Figure 8: Example of Cohen Sutherland Line Clipping**

At first considering the line $P_1P_2$

INPUT: $P_1$(120, 5), $P_2$(180, 30)
$x_L = 100$, $x_R = 160$, $y_B = 10$, $y_T = 40$

$x_1 > x_L$ then bit 1 of code $-P_1 = 0$ $C_{1\ left} = 0$
$x_1 < x_R$ then bit 2 of code $-P_1 = 0$ $C_{1\ right} = 0$
$y_1 < y_B$ then bit 3 of code $-P_1 = 1$ $C_{1\ bottom} = 1$
$y_1 < y_T$ then bit 4 of code $-P_1 = 0$ $C_{1\ top} = 0$

code $-P_1 = 0100$,

$x_2 > x_L$ then bit 1 of code $-P_1 = 0$ $C_{2\ left} = 0$
$x_2 > x_R$ then bit 2 of code $-P_1 = 1$ $C_{2\ right} = 1$
$y_2 > y_B$ then bit 3 of code $-P_1 = 0$ $C_{2\ bottom} = 0$
$y_2 < y_T$ then bit 4 of code $-P_1 = 0$ $C_{2\ top} = 0$

code $-P_2 = 0010$.

Both code $-P_1$ <> 0 and code $-P_2$ <> 0
then $P_1P_2$ not totally visible

code $-P_1$ AND code $-P_2 = 0000$
hence (code $-P_1$ AND code $-P_2 = 0$)
then line is not totally invisible.

As code –P <> 0

for i = 1
{
$C_{1\ left}$ (= 0) <> 1 then nothing is done.
i = i + 1 = 2
}
code $-P_1$ <> 0 and code $-P_2$ <> 0
then $P_1P_2$ not totally visible.

code $-P_1$ AND code $-P_2 = 0000$
hence (code $-P_1$ AND code $-P_2 = 0$)

then line is not totally invisible.

for i = 2
{

$C_{1\ right}$ (= 0) <> 1 then nothing is to be done.
i = i + 1 = 2 + 1 = 3
}
code $-P_1$ <> 0 and code $-P_2$ <> 0
then $P_1P_2$ not totally visible.

code $-P_1$ AND code $-P_2$ = 0000
Hence, (code $-P_1$ AND code $-P_2$ = 0)
then the line is not totally invisible.

for i = 3
{

$C_{1\ bottom}$ = 1 then find intersection of $P_1P_2$ with bottom edge
$y_B = 10$
$x_B = (180-120)(10-5)/(30-5) + 120$
=132
then $P_1 = (132,10)$

$x_1 > x_L$ then bit 1 of code $-P_1 = 0$ $C_{1\ left} = 0$
$x_1 < x_R$ then bit 2 of code $-P_1 = 0$ $C_{1\ right} = 0$
$y_1 = y_B$ then bit 3 of code $-P_1 = 0$ $C_{1\ bottom} = 0$
$y_1 < y_T$ then bit 4 of code $-P_1 = 0$ $C_{1\ top} = 0$

code $-P_1$ = 0000
i = i + 1 = 3 + 1 = 4
}

code $-P_1$ <> 0 but code $-P_2$ <> 0
then $P_1P_2$ not totally visible.

code $-P_1$ AND code $-P_2$ = 0000
Hence, (code $-P_1$ AND code $-P_2$ = 0)
then line is not totally invisible.

As code $-P_1 = 0$

Swap $P_1$ and $P_2$ along with the respective flags

$P_1 = (180, 30)$
$P_2 = (132, 10)$
code $-P_1$ = 0010
code $-P_2$ = 0000

| | |
|---|---|
| $C_{1\ left} = 0$ | $C_{2\ left} = 0$ |
| $C_{1\ right} = 1$ | $C_{2\ right} = 0$ |
| $C_{1\ bottom} = 0$ | $C_{2\ bottom} = 0$ |
| $C_{1\ top} = 0$ | $C_{2\ top} = 0$ |

Reset i = 1
for i = 1
{

$C_{1\ left}$ (= 0) <> 1 then nothing is to be done.
i = i + 1 = 1 + 1 = 2
}

code $-P_1$ <> 0, and code $-P_2$ <> 0
then $P_1P_2$ is not totally visible.

code $-P_1$ AND code $-P_2$ = 0000

Hence, (code $-P_1$ AND code $-P_2 = 0$)
then line is not totally invisible.

for i = 2
{
$C_{1\ right}$ = 1 then find intersection of $P_1P_2$ with right edge
$x_R = 160$
$y_R = (30 - 5)(160 - 120)/(180 - 120) + 5$
$= 21.667$
$= 22$
then $P_1 = (160, 22)$
$x_1 > x_L$ then bit 1 of code $-P_1 = 0$ $C_{1\ left} = 0$
$x_1 = x_R$ then bit 2 of code $-P_1 = 0$ $C_{1\ right} = 0$
$y_1 > y_B$ then bit 3 of code $-P_1 = 0$ $C_{1\ bottom} = 0$
$y_1 < y_T$ then bit 4 of code $-P_1 = 0$ $C_{1\ top} = 0$
code $-P_1 = 0000$, i = i + 1 = 2 + 1 = 3
}

As both code $-P_1 = 0$ and code $-P_2 = 0$ then the line segment $P_1P_2$ is totally visible.

So, the visible portion of input line $P_1P_2$ is $P'_1P'_2$ where, $P_1 = (160, 22)$ and $P_2 = (132, 10)$.

Considering the line EF

1) The endpoint codes are assigned
code – E $\rightarrow$ 0101
code – F $\rightarrow$ 1001
2) Flags are assigned for the two endpoints
$E_{left} = 1$ (as x coordinate of E is less than $x_L$)
$E_{right} = 0$, $E_{top} = 0$ $E_{bottom} = 1$

Similarly,
$F_{left} = 1$, $F_{right} = 0$, $F_{top} = 1$ $F_{bottom} = 0$

3) Since codes of E and F are both not equal to zero the line is not totally visible.

4) Logical intersection of codes of E and F is not equal to zero. So, we may ignore EF line and declare it as totally invisible.

Considering the line GH

1) The endpoint codes are assigned
code – G $\rightarrow$ 0000
code – H $\rightarrow$ 1000

2) Flags are assigned for the two endpoints
$G_{left} = 0$, $G_{right} = 0$, $G_{top} = 0$, $G_{bottom} = 0$.

Similarly,
$H_{left} = 0$, $H_{right} = 0$, $H_{top} = 1$, $H_{bottom} = 0$.

3) Since, codes of G and H are both not equal to zero so the line is not totally visible.

4) Logical intersection of codes of G and H is equal to zero so we cannot declare it as totally invisible.

5) Since, code – G = 0, Swap G and H along with their flags and set i = 1

Implying $G_{left} = 0$, $G_{right} = 0$, $G_{top} = 1$, $G_{bottom} = 0$.

$H_{left} = 0$, $H_{right} = 0$, $H_{top} = 0$, $H_{bottom} = 0$.
as $G \rightarrow 1000$, $H \rightarrow 0000$

6) Since, code – G <> 0 then

for i = 1, {since $G_{left} = 0$
i = i + 1 = 2
go to 3
}

The conditions 3 and 4 do not hold and so we cannot declare line GH as totally visible or invisible.

for i = 2, {since $G_{right} = 0$
i = i + 1 = 3
go to 3
}

The conditions 3 and 4 do not hold and so we cannot declare line GH as totally visible or invisible.

for i = 3, {since $G_{bottom} = 0$
i = i + 1 = 4
go to 3
}

The conditions 3 and 4 do not hold and so we cannot declare line GH as totally visible or invisible.

for i = 4, {since $G_{top} = 1$
Intersection with top edge, say P(x, y) is found as follows:

Any line passing through the points G, H and a point P(x, y) is given by
$y – 20 = \{(180 – 20) /(140 – 120)\}(x – 120)$
or, $y – 20 = 3x – 360$
or, $y – 30 = -340$

Since, the y coordinate of every point on line CD is 40, so we put y = 40 for the point of intersection P(x, y) of line GH with edge CD.
$40 – 3x = -340$
or, $- 3x = - 380$
or $x = 380/3 = 126.66 \approx 127$

So, the point of intersection is P(127, 40). We assign code to it.

Since, the point lies on edge of the rectangle so the code assigned to it is 0000.

Now, we assign G = (127, 40); i = 4 + 1 = 5. Conditions 3 and 4 are again checked.

Since, codes G and H are both equal to 0, so, the line between H(120, 20) and G(127, 40) is totally visible.

Limitation of Cohen Sutherland line Clipping Algorithm

The algorithm is only applicable to rectangular windows and not to any other convex shaped window.

So, a new line-clipping algorithm was developed by Cyrus, and Beck to overcome this limitation. This Cyrus Beck line-clipping Algorithm is capable of clip lining segments irrespective of the shape of the convex window.

You might wonder as to what a convex window? In general, on the basis of the shapes the windows are classified into two categories:

i) **Convex shaped windows:** Windows of a shape such that if we take any two points inside the window then the line joining them will never cross the window boundary, such shaped windows are referred as convex shaped windows.

ii) **Concave or non Convex shaped windows:** Windows of a shape such that if we can choose a pair of points inside the window such that the line joining them may cross the window boundary or some section of the line segment may lie outside the window. Such shaped windows are referred to as non-convex or concave shaped windows.

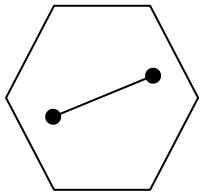


Figure (a)
Convex Polygonal Window

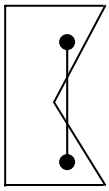


Figure (b)
Non-convex Polygonal window

**Figure 9: Types of Windows**

### 3.4.2 Cyrus-Beck Algorithm

Cyrus Beck Line clipping algorithm is infact, a parametric line-clipping algorithm. The term parametric implies that we need to find the value of the parameter t in the parametric representation of the line segment for the point at which the segment intersects the clipping edge. For better understanding, consider the *Figure 9(a),* where P Q is a line segment, which is intersecting at the two edges of the convex window.

**Note:** The algorithm is applicable to the "convex polygonal window".

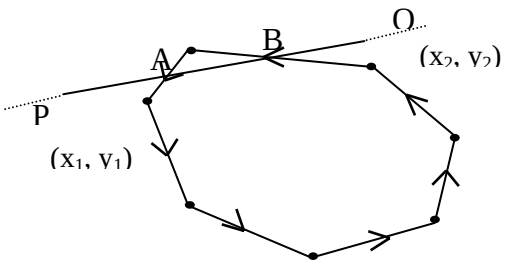



**Figure 9(a): Interaction of line PQ and Window**

Now, just recall the parametric equation of line segment PQ, which we have studied in the course CS-60.

It is simply $P + t(Q - P)$ $0 \leq t \leq 1$

Where, $t \rightarrow$ linear parameter continuously changes value.

$\therefore P + t(Q - P) \Rightarrow (x_1, y_1) + t(x_2 - x_1, y_2 - y_1) = (x, y)$ be any point on PQ. ------ (1)

For this equation (1) we have following cases:

1) When $t = 0$ we obtain the point P.
2) When $t = 1$ we get the point Q.
3) When t varies such that $0 \leq t \leq 1$ then line between point P and Q is traced.

For t = ½ we get the mid-point of PQ.

4) When t < 0 line on LHS of P is traced.
5) When t > 1 line on RHS of Q is traced.

So, the variation in parameter t is actually generating line in point wise manner .The range of the parameter values will identify the portion to be clipped by any convex polygonal region having n-vertices or lattice points to be specified by the user. One such clipping situation is shown in *Figure 9(a).*

**Remark:**

- **How to specify the window region:** a convex polygonal region having n-vertices $\{P_0, P_1, P_2\ldots, P_{n-1}, P_n, P_0\}$ or lattice points to be specified by the user encloses the convex window region. To be specific about the window we may say that each edge between two points contributes to the boundary of the window under the situation that (when we traverse each edge in anticlockwise manner), the window region lies on left hand side (LHS) of edge. This orientation is preserved and while giving input, the user takes care of the orientation to specify the window region of any arbitrary convex shape.

- **Potentially entering and leaving points ($P_E$ and $P_L$ )**
  The point of intersection of the line and window may be classified either as Potentially entering or leaving. Before going into other details, let us describe the actual meaning of Potentially entering and leaving points ($P_E$ and $P_L$ ). $P_E$ and $P_L$ are dependent on the edge orientation, *i.e.* direction. Let us understand **how to find $P_E$ and $P_L$** (we know as we move anticlockwise across the window boundary then region of LHS encloses the window).

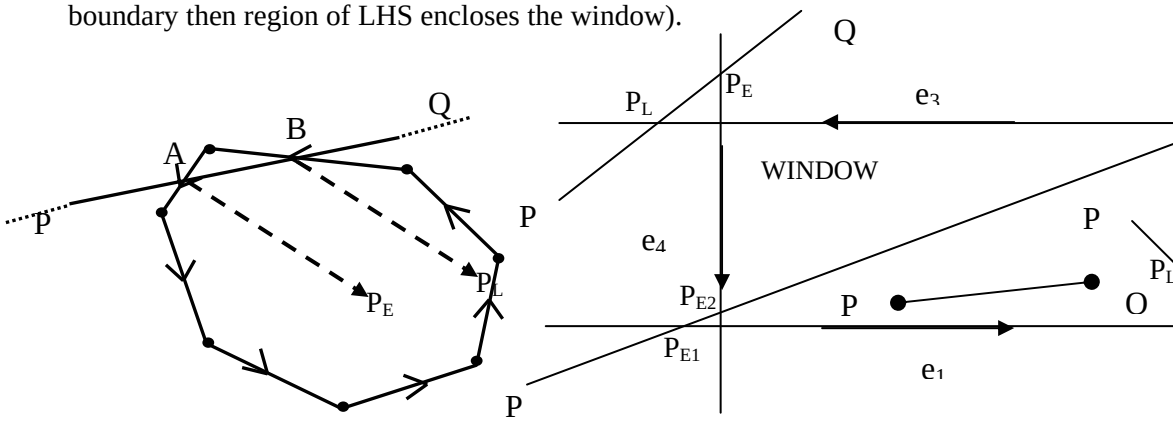



**Figure 10(a): Potentially Entering $P_E$ and Potentially Leaving $P_L$ points**

Say, we are moving from point P to point Q, and we know that while traversing the windows edges in anticlockwise manner the region lying on the left hand side is referred to as the window region. So, from the *Figure 10 (a)*, you may notice that at point $P_E$ we are moving from P to Q we are entering the window region , thus this point of intersection should be referred to as the Potentially entering point($P_E$). Now, refer to other intersection shown in the *Figure 10 (a)*, here, also the movement of points on the line are determined by varying value of parameter t is from point P to Q and w.r.t ., the orientation of window boundary, we are exiting the window region. So, this point of intersection is known as potentially leaving point ($P_L$).

**Potentially entering point ($P_E$)** while moving towards the window, the point of intersection between line PQ and the window edges faced are known as **$P_E$**.

**Potentially leaving point ($P_L$)** These are the point of intersection encountered while we move away from window.

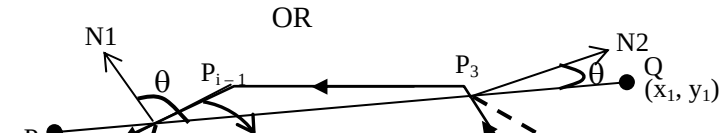


**Figure 10 (b): Potentially Entering $P_E$ and Potentially Leaving $P_L$ points**

Other way round you may detect the point of intersection as Potentially leaving point ($P_L$) or Potentially entering point ($P_E$) by studying the angle between the line to be clipped from the outward normal to the intersecting edge, at the point of intersection. From *Figure 10 (b)*, it is the angle between PQ and N1 or N2. You may notice that, while moving from P to Q the angle between line PQ and N1 is obtuse whereas the angle between PQ and N2 is acute; so, you may say, if the angle between line PQ and N1 is obtuse the point of intersection is potentially entering ($P_E$) whereas, if the angle between PQ and N2 is acute, then the point of intersection is potentially leaving ($P_L$).

OR

$\mathbf{P_E}$ => $\mathbf{N_i \cdot PQ < 0}$ ; **(angle θ greater than 90° or Obtuse )**

$\mathbf{N_i \cdot (Q-P) < 0}$

$\mathbf{P_L}$ => $\mathbf{N_i \cdot PQ > 0}$ ; **(angle θ is less than 90° or Acute)**

$\mathbf{N_i \cdot (Q-P) > 0}$

- **Where $N_i$ is the outward normal to the $i^{th}$ edge.**

**Note:** when $(\overline{Q} - \overline{P}) \cdot \overline{N_i} = 0$ then it implies that:

1) $\overline{Q} - \overline{P} = 0$ ↓ not possible ∵ if $\overline{Q} - \overline{P} = 0$ then $\overline{PQ}$ is a point and not a line

2) $\overline{N_i} = 0$ ↓ not possible

3) $\theta = 90°$ ↓ possible i.e. $(\overline{Q} - \overline{P}) \perp \overline{N_i}$ ⇩ line segment PQ is || to $i^{th}$ edge then only $\overline{N_i}$ will be ⊥ to both PQ and $i^{th}$ edge.

- **How to find the normal :**

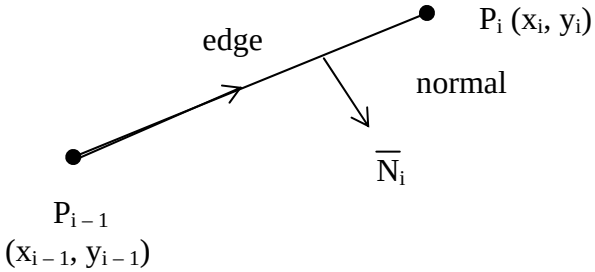



**Figure 11 : Normal Determination**

$\overrightarrow{P_{i-1}P_i} = (x_1 - x_{i-1}, y_i - y_{i-1}) = (s_1, s_2)$

if $\overrightarrow{N_i} = (n_{1i}, n_{2i})$, then

$\overrightarrow{N_i} \cdot \overrightarrow{P_{i-1}P_i} = 0 \Rightarrow (s_1, s_2) \cdot (n_{1i}, n_{2i}) = 0$

$$\Rightarrow s_1 n_{1i} + s_2 n_{2i} = 0$$
$$\Rightarrow s_1 n_{1i} = - s_2 n_{2i} = 0$$
$$\Rightarrow n_{1i} = \frac{-s_2}{s_1} n_{2i}$$

If $n_{2i} = 1$ then, $n_{1i} = \frac{s_2}{s_1}$

Therefore, $\overrightarrow{N_i} = (n_{1i}, n_{2i}) = (\frac{-s_2}{s_1}, 1) \longrightarrow$ NORMAL

if $\left(\frac{-s_2}{s_1}, 1\right)$ is outward normal then $-\left(\frac{-s_2}{s_1}, 1\right)$ i.e. $\left(\frac{s_2}{s_1}, -1\right)$ is inward normal.

Now, we have learned some basic things which will be required in the working of the algorithm, so, we can start using the concepts learned so far to clip the line segment passing through the window.

We know that the parametric equation of line PQ is

$$P + t\,(Q - P) \;\; ; \;\; 0 \le t \le 1$$

Now, to find the point of intersection of line PQ and $i^{th}$ edge we need to find the appropriate value of parameter t, for that we firstly find normal to the $i^{th}$ edge. Say $\overrightarrow{Ni}$ is the normal to $i^{th}$ edge ($P_{i-1}$ to $P_i$), then to find the appropriate parameter t, we should determine the dot product of the vector from $P_{i-1}$ to the point of intersection and the normal Ni. Let us do this exercise.

The vector joining edge point $P_{i-1}$ and point of intersection of the line PQ with the $i^{th}$ edge for some t will be given by:

$$\{ [\overline{P} + t(\overline{Q} - \overline{P})] - \overline{P}_{i-1} \}$$

As $\overrightarrow{Ni}$ is normal to the $i^{th}$ edge , so, the dot product of the vector joining edge point $P_{i-1}$ and point of intersection of the line PQ with the $i^{th}$ edge for some t and Ni will be given by:

$$\{[\overline{P} + t(\overline{Q} - \overline{P})] - \overline{P}_{i-1}\}.\overrightarrow{N_i} = 0 \qquad \text{----------(1)}$$

Rearranging (1) we get,

$$t = \frac{-(\overline{P} - \overline{P}_{i-1}).\overline{N_i}}{(\overline{Q} - \overline{P}).\overline{N_i}} \qquad \text{----------(2)}$$

Using this value of t from (2) in parametric form of line we will get the point of intersection of line PQ and $i^{th}$ edge.

**Note**: By the above discussion of $P_E$ and $P_L$ we came to know that if $N_i . (Q-P) < 0$ => $P_E$ point and $N_i . (Q-P) > 0$ => $P_L$ point. If the value of t calculated using (2) lies in the interval 0 to 1 (in magnitude) then, the point of intersection will be considered otherwise it will not be considered.

**Steps to clip a line segment PQ:**

- Firstly, find all the points of intersections of the line segment PQ with the edges of the polygonal window and classify them either as $P_E$ and $P_L$ points. Also

determine the value of parmeter t , using equation (2) for respective PE's and PL's.

Or

If value of the normals to respective edges are not given then, find the value of normal to every edge of the window, then, determine the value of parmeter t , using equation (2) for respective point of intersection between line segment and window edge then on the basis of the vaue of parameter t mark them as $\mathbf{P_E}$ and $\mathbf{P_L}$ provided the value of t lies from 0 to 1 in magnitude.

- Secondly, out of the various values of t for $\mathbf{P_E}$'s determine the maximum value of t say it be $t_{max}$. Similarly, out of the various values of t for $\mathbf{P_L}$'s determine the minimum value of t say it be $t_{min}$ . Note that for clipping to be possible $t_{min} > t_{max}$.

- Finally, vary the parameter t from $t_{max}$ to $t_{min}$ and determine the clipped line as outcome.

For better understanding of steps consider *Figure 12*.

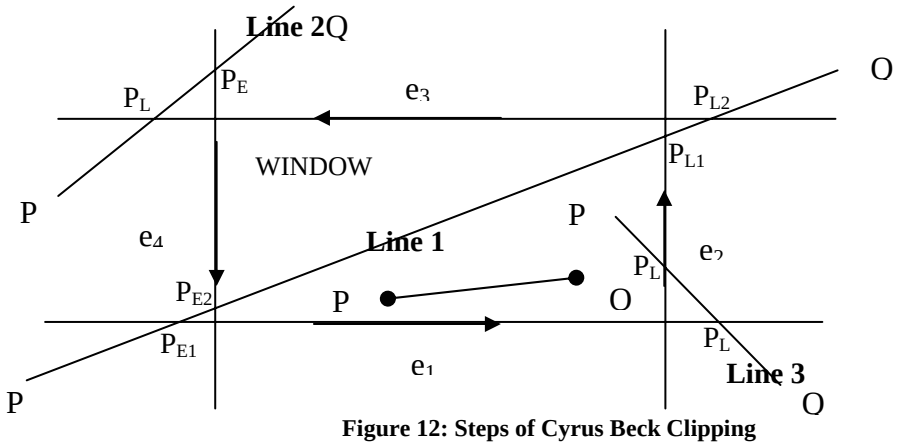



**Figure 12: Steps of Cyrus Beck Clipping**

**In case of Line 1: (PQ)**

1) Point 1 is potentially entering ($P_{E1}$) because as we move along PQ, the LHS of $e_1$ is window and hence, it seems that we are entering the window.
2) Point 2 is again $P_{E2}$ similarly as point 1.
3) Point 3 is potentially leaving ($P_4$) ∵ as we move along PQ, the LHS of $e_2$ is window and hence, it seems that we are leaving the window.
4) Similarly, point 4 is also $PL_2$.

**Line 2 and 3 (PQ)**:

Using same logic as for line 1 we find $\bar{P}_L$ and $P_E$. Now, it is to be noted that for each point of intersection we have some value of t.

say $t_1$ is value of t for $P_{E1}$
say $t_2$ is value of t for $P_{E2}$
say $t_3$ is value of t for $P_{L1}$
say $t_4$ is value of t for $P_{L2}$

*As the portion visible to us is image what laying inside the window* for line 1 the line visible is from $P_{E2}$ to $P_{L1}$ and for these points there is some value of t. With this initialisation let us study the following cases:

**Case 1:** Say we get a new value of $t_E$ (i.e value of parameter t for any potentially entering ($P_E$) point) we choose $t_{max}$ as: $t_{max} = \max\{t_{max}, t_E\}$. The initial value of $t_{max} = 0$ is taken.

**Case 2:** Say we get a new value of $t_L$ (i.e value of parameter t for any potentially leaving($P_L$) point) then $t_{max}$ value is updated by $t_{min} = \min \{t_{min}, t_L\}$. The initial value of $t_{min} = 1$ is taken.

Finally when value of t for all edges are found and if $t_{max} < t_{min}$ then line is visible else not. And line is visible from

$P + t_{max} (Q – P)$ to $P + t_{min} (Q – P)$ *i.e.*

$P + t (Q – P)$ such that $t_{max} \leq t \leq t_{min}$

$t_{max} < t_{min}$ (draw line)

$t_{max} \not< t_{min}$ (reject line)

**Example:** How does the Cyrus Beck line clipping algorithm, clip a line segment if the window is non convex?

**Solution:** Consider the *Figure 13*, here, the window is non convex in shape and PQ is a line segment passing through this window .Here too the condition of visibility of the line is $t_{max} < t_{min}$ and the line is visible from $P + t_{max} (Q – P)$ to $P + t_{min} (Q – P)$, if $t_{max} \not< t_{min}$ then reject the line segment. Now, applying this rule to the *Figure 13,* we find that when PQ line segment passes through the non convex window, it cuts the edges of the window at 4 points. $1 \rightarrow P_E$; $2 \rightarrow P_L$; $3 \rightarrow P_E$; $4 \rightarrow P_L$ . In this example, using the algorithm we reject the line segment PQ but it is not the correct result.

Condition of visibility is satisfied in region 1-2 and 3-4 only so the line exists there but in region 2-3 the condition is violated so the line does not exists.

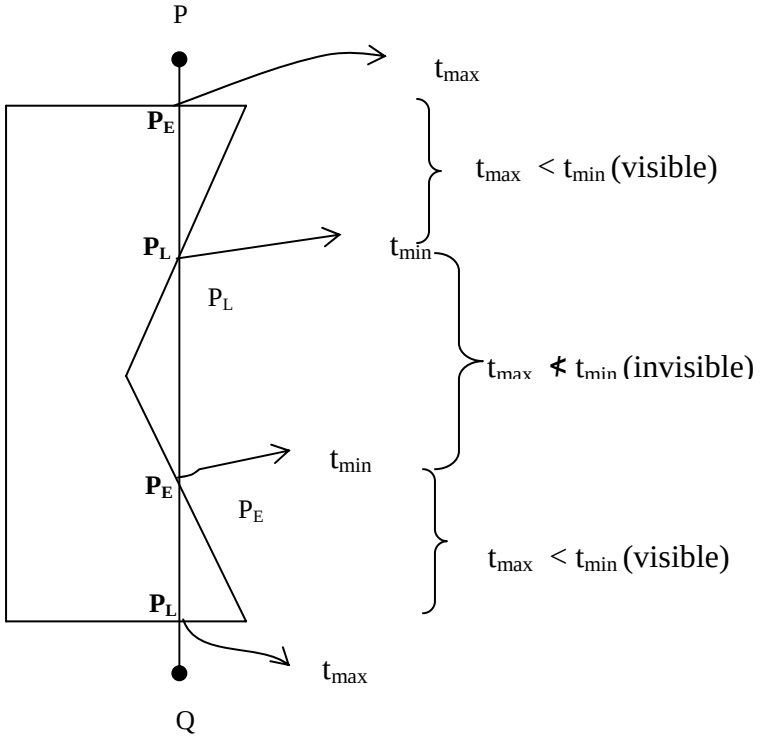



**Figure 13: Example Cyrus Beck Clipping**

## ☞ Check Your Progress 1

1) Suppose a rectangular window ABCD is defined such that A = (–1, –2) and C(3, 1) using generalised geometrical approach , clip the line segment joining the points P(–20, 0) and Q(20, 30).

……

……

……

……

……

2) A clipping window is given by $P_0(10,10)$, $P_1(20,10)$, $P_2(25,15)$, $P_3(20,20)$, $P_4(10,15)$, $P_5 = P_0$. Using Cyrus Beck algorithm clip the line A(0,0) and B(30,25).

……

……

……

……

3) How will you compute the outward normal to a clipping in the case of Cyrus-Back algorithm?

……

……

……

……

4) Compare Cohen Sutherland Line clipping with Cyrus Beck Line clipping algorithm?

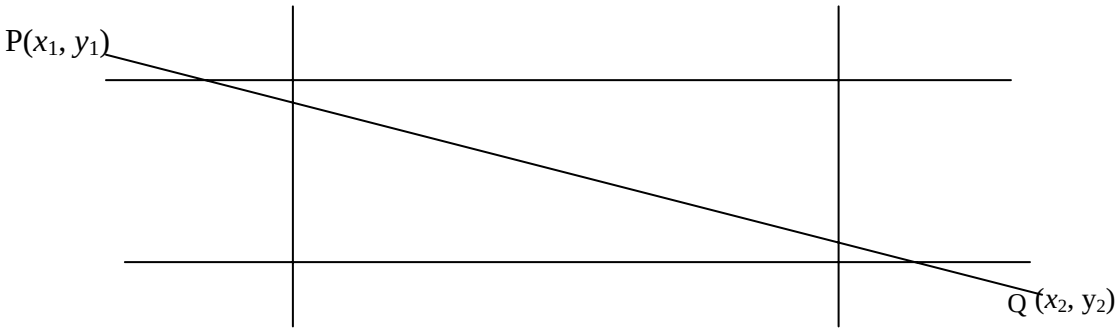
……

……

……

……

5) Clip the line shown in Figure below using Cohen Sutherland Line clipping algorithm.




……

……

……

……

6) Which type of clipping windows can't be handled by Cyrus Beck clipping algorithm, how can such cases be handled?

……

……

……

……

# 3.5 POLYGON CLIPPING

After understanding the concept of line clipping and its algorithms, we can now extend the concept of line clipping to polygon clipping, because polygon is a surface enclosed by several lines. Thus, by considering the polygon as a set of line we can divide the problem to line clipping and hence, the problem of polygon clipping is simplified. But it is to be noted that, clipping each edge separately by using a line clipping algorithm will certainly not produce a truncated polygon as one would expect. Rather, it would produce a set of unconnected line segments as the polygon is exploded. Herein lies the need to use a different clipping algorithm to output truncated but yet bounded regions from a polygon input. Sutherland-Hodgman algorithm is one of the standard methods used for clipping arbitrary shaped polygons with a rectangular clipping window. It uses divide and conquer technique for clipping the polygon.

## 3.5.1 Sutherland-Hodgman Algorithm

Any polygon of any arbitrary shape can be described with the help of some set of vertices associated with it. When we try to clip the polygon under consideration with any rectangular window, then, we observe that the coordinates of the polygon vertices satisfies one of the four cases listed in the table shown below, and further it is to be noted that this procedure of clipping can be simplified by clipping the polygon edgewise and not the polygon as a whole. This decomposes the bigger problem into a set of subproblems, which can be handled separately as per the cases listed in the table below. Actually this table describes the cases of the Sutherland-Hodgman Polygon Clipping algorithm.

Thus, in order to clip polygon edges against a window edge we move from vertex $V_i$ to the next vertex$V_{i+1}$ and decide the output vertex according to four simple tests or rules or cases listed below:

**Table: Cases of the Sutherland-Hodgman Polygon Clipping Algorithm**

| Case | $V_i$ | $V_{i+1}$ | Output Vertex |
|---|---|---|---|
| A | Inside window) | Inside | $V_{i+1}$ |
| B | Inside | Outside | $V'_i$ i.e intersection of polygon and window edge |
| C | Outside | Outside | None |
| D | Outside | Inside | $V'_I$ ; $V_{i+1}$ |

In words, the 4 possible Tests listed above to clip any polygon states are as mentioned below:

1) If both Input vertices are inside the window boundary then only 2nd vertex is added to output vertex list.
2) If 1st vertex is inside the window boundary and the 2nd vertex is outside then, only the intersection edge with boundary is added to output vertex.
3) If both Input vertices are outside the window boundary then nothing is added to the output list.
4) If the 1st vertex is outside the window and the 2nd vertex is inside window, then both the intersection points of the polygon edge with window boundary and 2nd vertex are added to output vertex list.

So, we can use the rules cited above to clip a polygon correctly. The polygon must be tested against each edge of the clip rectangle; new edges must be added and existing edges must be discarded , retained or divided. Actually this algorithm decomposes

the problem of polygon clipping against a clip window into identical subproblems, where a subproblem is to clip all polygon edges (pair of vertices) in succession against a single infinite clip edge. The output is a set of clipped edges or pair of vertices that fall in the visible side with respect to clip edge. These set of clipped edges or output vertices are considered as input for the next sub problem of clipping against the second window edge. Thus, considering the output of the previous subproblem as the input, each of the subproblems are solved sequentially, finally yielding the vertices that fall on or within the window boundary. These vertices connected in order forms, the shape of the clipped polygon.

For better understanding of the application of the rules given above consider the *Figure 14*, where the shaded region shows the clipped polygon.

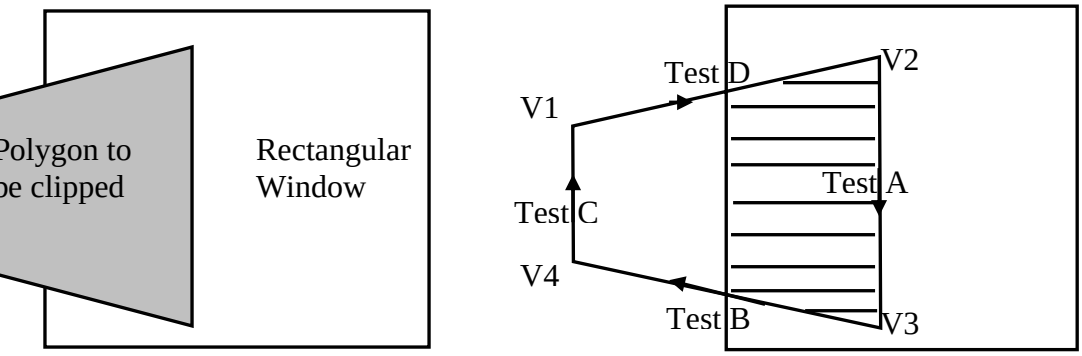



**Figure 14 :Sutherland-Hodgman Polygon Clipping**

## Pseudocode for Sutherland – Hodgman Algorithm

***Define variables***

inVertexArray is the array of input polygon vertices
outVerteArray is the array of output polygon vertices
Nin is the number of entries in inVertexArray
Nout is the number of entries in outVertexArray
n is the number of edges of the clip polygon
ClipEdge[x] is the xth edge of clip polygon defined by a pair of vertices
s, p are the start and end point respectively of current polygon edge
i is the intersection point with a clip boundary
j is the vertex loop counter

***Define Functions***

**AddNewVertex(newVertex, Nout, outVertexArray)**
: Adds newVertex to outVertexArray and then updates Nout

**InsideTest(testVertex, clipEdge[x])**
: Checks whether the vertex lies inside the clip edge or not;
retures TRUE is inside else returns FALSE

**Intersect(first, second, clipEdge[x])**
: Clip polygon edge(first, second) against clipEdge[x],
outputs the intersection point

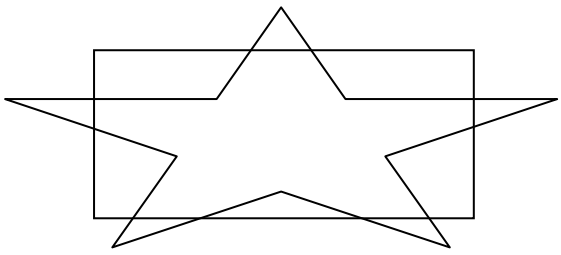
```
{                          :  begin main
x = 1
while (x ≤ n)              : Loop through all the n clip edges
{
Nout = 0                   : Flush the outVertexArray
s = inVertexArray[Nin] : Start with the last vertex in inVertexArray
for j = 1 to Nin do        : Loop through Nin number of polygon vertices (edges)
        {
                p = inVertexArrray[j]
```

```
        if InsideTest(p, clipEdge[x] = = TRUE then          : Case A
and D
        if InsideTest(s, clipEdge[x] = = TRUE then
        AddNewVertex(p, Nout, outVertexArray)               : Case A
          else
        i = Intersect(s, p, clipEdge[x])                    : Case D
        AddNewVertex(i, Nout, outVertexArray)
        AddNewVertex(p, Nout, outVertexArray)
        end if
        else          : i.e. if InsideTest(p, clipEdge[x] = = FALSE
                        (Cases 2 and 3)
        if InsideTest(s, clipEdge[x]) = =TRUE then          : Case B
        {
          Intersect(s, p, clipEdge[x])
          AddNewVertex(i, Nout, outVertexArray)
          end if                          : No action for case C

          s = p                           : Advance to next pair of vertices
          j = j + 1
         end if                           :  end {for}
           }
        x = x + 1                         : Proceed to the next ClipEdge[x +1]
        Nin = Nout
        inVertexArray = outVertexArray    :  The ouput vertex array for
                                             the current clip edge becomes the input
                                             vertex array for the next clip edge
        }                                 : end while
        }                                 : end main
```

### ☞ Check Your Progress 2

1) Using Sutherland-Hodgeman polygon clipping algorithm clip on the polygon given below.



## 3.6 WINDOWING TRANSFORMATIONS

From section 3.1,we understood the meaning of the term window and viewport which could again be understood as:

**Window:** A world coordinate area selected for display (i.e. area of picture selected for viewing).

**Viewport:** An Area or a display device to which a window is mapped.

**Note:**

- Window defines what is to be viewed and viewpoint defines where it is to be displayed.
- Often window and viewpoints are rectangles in standard position with edges parallel to coordinate axes. Generalised shapes like polygon etc., take long to

process, so we are not considering these cases where window or viewport can have general polygon shape or circular shape.

The mapping of a part of a world coordinate scene to device coordinates is referred to as Viewing Transformation. In general 2D viewing transformations are referred to as window to viewport transformation or windowing transformation.

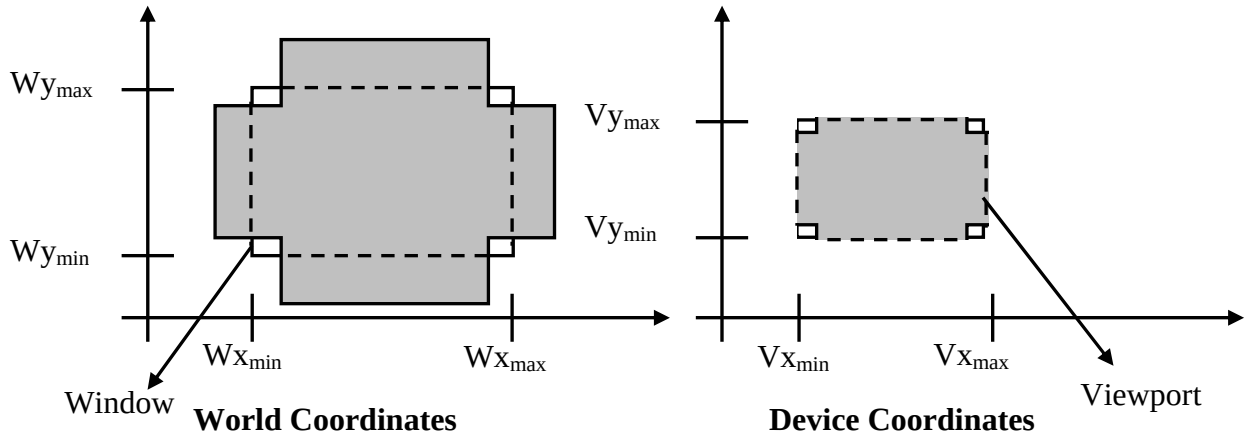



**Figure 15: Windowing Transformation**

You may notice in *Figure 15*, that all parts of the picture that lie outside the window are clipped and the contents which lie inside the widow are transferred to device coordinates. Secondly, you may also notice that while window selects a part of the scene, viewport displays the selected part at the desired location on the display area. When window is changed we see a different part of the scene at the same portion (viewport) on display. If we change the viewport only, we see the same part of the scene drawn at a different scale or at a different place on the display. By successively increasing or decreasing the size of the window around a part of the scene the viewport remain fixed, we can get the effect of zoom out or zoom in respectively on the displayed part. Mathematically, viewing transformation can be expressed as V=W.N

Where,

- **V** refers Viewing transformation which maps a part of world coordinate scene to device coordinates;
- **W** refers to workstation transformation which maps normalised device coordinates to physical device coordinates;
- **N** refers to Normalisation transformation used to map world coordinates to normalized device coordinates.

Window to Viewpoint Coordinates transformation:

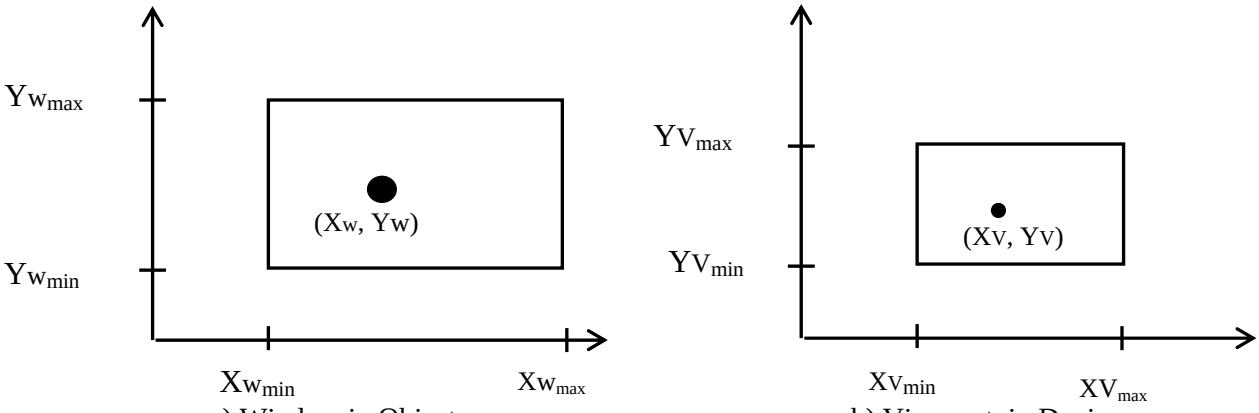



**Figure 16: Window to Viewport Transformation**

*Figure 16*, illustrates window-viewpoint mapping. Here, it is depicted that a point at position ($X_w$, $Y_w$) in window is mapped into position ($X_v$, $Y_v$) in the associated viewpoint.

So, as to maintain the same relative placement in the viewpoint as in the window we require

$$\frac{xv - xv_{min}}{xv_{max} - xv_{min}} = \frac{xw - xw_{min}}{xw_{max} - xw_{min}} \quad \text{----------(a)}$$

$$\frac{yv - yv_{min}}{yv_{max} - yv_{min}} = \frac{yw - yw_{min}}{yw_{max} - yw_{min}} \quad \text{----------(b)}$$

----------(1)

rearranging equation (a) and (b) of (1) we get viewpoint position ($x_v$, $y_v$) i.e.,

$$\begin{aligned} xv &= xv_{min} + (xw - xw_{min})\, S_x \\ yv &= yv_{min} + (yw - yw_{min})\, S_y \end{aligned} \quad \text{----------(2)}$$

where

$S_x$ scaling factor along $x$ axis = $\dfrac{xv_{max} - xv_{min}}{xw_{max} - xw_{min}}$

$S_y$ scaling factor along y axis = $\dfrac{yv_{max} - yv_{min}}{yw_{max} - yw_{min}}$ ----------(3)

Note, if $S_x = S_y$ then the relative proportions of objects are maintained else the world object will be stretched or contracted in either $x$ or $y$ direction when displayed on output device.

In terms of Geometric transformations the above relation can be interpreted through the following two steps:

**Step 1** Scaling the window area to the size of the viewport with scale factors $s_x$ and $s_y$ w.r.t a fixed point ($xw_{min}$, $yw_{min}$).

$$\Rightarrow [T_1] = \begin{pmatrix} s_x & 0 & xw_{min}(1 - s_x) \\ 0 & s_y & yw_{min}(1 - s_y) \\ 0 & 0 & 1 \end{pmatrix}$$

**Step 2** Translating the scaled window to the position of the viewport so that,

$$\Delta x = xv_{min} - xw_{min} \quad \text{and}$$
$$\Delta y = yv_{min} - yw_{min}$$

$$\Rightarrow [T_2] = \begin{pmatrix} 1 & 0 & \Delta x \\ 0 & 1 & \Delta y \\ 0 & 0 & 1 \end{pmatrix}$$

Concatenating $[T_1]$ and $[T_2]$ we get,

$$[T] = [T_1]\,[T_2] = \begin{pmatrix} s_x & 0 & \Delta x + xw_{min}(1 - s_x) \\ 0 & s_y & \Delta y + yw_{min}(1 - s_y) \\ 0 & 0 & 1 \end{pmatrix}$$

Replacing the values of $\Delta x$ and $\Delta y$ in the above transformation matrix we finally get,

$$[T] = \begin{pmatrix} s_x & 0 & -s_x xw_{min} + xv_{min} \\ 0 & s_y & -s_y yw_{min} + yv_{min} \\ 0 & 0 & 1 \end{pmatrix} \quad (1)$$

The aspect ratio of a rectangular window or viewport is defined by

$$a = \frac{x_{max} - x_{min}}{y_{max} - y_{min}}$$

If $s_x = s_y$ then $\frac{xv_{max} - xv_{min}}{xw_{max} - xw_{min}} = \frac{yv_{max} - yv_{min}}{yw_{max} - yw_{min}}$

$$\Rightarrow \frac{xv_{max} - xv_{min}}{yv_{max} - yv_{min}} = \frac{xw_{max} - xw_{min}}{yw_{max} - yw_{min}} \Rightarrow a_v = a_w$$

So, it can be said that if the aspect ratio $a_v$ of the viewport equals the aspect ratio $a_w$ of the window, then $s_x = s_y$ and no distortion of displayed scene occurs other than uniform magnification or compression. If $a_v \neq a_w$ then the displayed scene in the viewport gets somewhat distorted w.r.t the scene captured by the window.

**Example** Find the normalisation transformation *N* which uses the rectangle W (1, 1), X (5, 3), Y (4, 5) and Z (0, 3) as a window and the normalised device screen as the viewport.

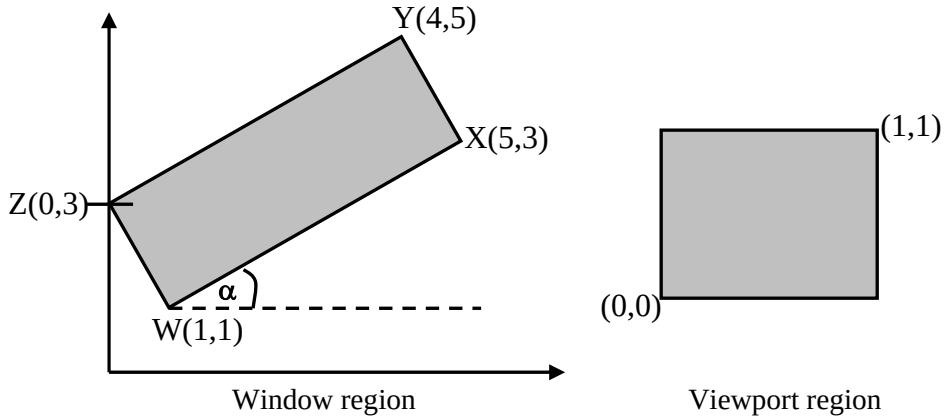



**Figure 17: Example Transformations**

Here, we see that the window edges are not parallel to the coordinate axes. So we will first rotate the window about W so that it is aligned with the axes.

$$\text{Now, } \tan\alpha = \frac{3-1}{5-1} = \frac{1}{2}$$

$$\Rightarrow \sin\alpha = \frac{1}{\sqrt{5}} \quad \cos\alpha = \frac{2}{\sqrt{5}}$$

Here, we are rotating the rectangle in clockwise direction. So $\alpha$ is ( – )ve i.e., $-\alpha$

The rotation matrix about W (1, 1) is,

$$[T_{R.\theta}]_W = \begin{bmatrix} \mathbf{Cos\ \alpha} & \mathbf{-Sin\ \alpha} & \mathbf{(1\text{-}Cos\ \alpha)\ x_p + Sin\ \alpha\ y_p} \\ \mathbf{Sin\ \alpha} & \mathbf{Cos\ \alpha} & \mathbf{(1\text{-}Cos\ \alpha)\ y_p - Sin\ \alpha\ x_p} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix}$$

$$[T_{R.\theta}]_W = \begin{pmatrix} \frac{2}{\sqrt{5}} & \frac{1}{\sqrt{5}} & \left(\frac{1-3}{\sqrt{5}}\right) \\ \frac{-1}{\sqrt{5}} & \frac{2}{\sqrt{5}} & \left(\frac{1-1}{\sqrt{5}}\right) \\ 0 & 0 & 1 \end{pmatrix}$$

The *x* extent of the rotated window is the length of WX which is $\sqrt{(4^2 + 2^2)} = 2\sqrt{5}$

Similarly, the y extent is length of WZ which is $\sqrt{(1^2 + 2^2)} = \sqrt{5}$

For scaling the rotated window to the normalised viewport we calculate $s_x$ and $s_y$ as,

$$s_x = \frac{\text{viewport } x \text{ extent}}{\text{window } x \text{ extent}} = \frac{1}{2\sqrt{5}}$$

$$s_y = \frac{\text{viewport } y \text{ extent}}{\text{window } y \text{ extent}} = \frac{1}{\sqrt{5}}$$

As in expression (1), the general form of transformation matrix representing mapping of a window to a viewport is,

$$[T] = \begin{pmatrix} s_x & 0 & -s_x xw_{\min} + xv_{\min} \\ 0 & s_y & -s_y yw_{\min} + yv_{\min} \\ 0 & 0 & 1 \end{pmatrix}$$

In this problem [$T$] may be termed as $N$ as this is a case of normalisation transformation with,

$xw_{\min} = 1$ $\qquad$ $xv_{\min} = 0$

$yw_{\min} = 1$ $\qquad$ $yv_{\min} = 0$

$s_x = \frac{1}{2\sqrt{5}}$ $\qquad$ $s_y = \frac{1}{\sqrt{5}}$

By substituting the above values in [$T$] i.e. N,

$$N = \begin{pmatrix} \frac{1}{2\sqrt{5}} & 0 & \left(\frac{-1}{2}\right)\frac{1}{\sqrt{5}} + 0 \\ 0 & \frac{1}{\sqrt{5}} & \left(\frac{-1}{\sqrt{5}}\right)1 + 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Now, we compose the rotation and transformation $N$ to find the required viewing transformation $N_R$

$$N_R = N\,[T_{R.\theta}]_W = \begin{pmatrix} \frac{1}{2\sqrt{5}} & 0 & \frac{-1}{2\sqrt{5}} \\ 0 & \frac{1}{\sqrt{5}} & \frac{-1}{\sqrt{5}} \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{2}{\sqrt{5}} & \frac{1}{5} & 1 - \frac{3}{\sqrt{5}} \\ \frac{-1}{\sqrt{5}} & \frac{2}{\sqrt{5}} & 1 - \frac{1}{\sqrt{5}} \\ 0 & 0 & 1 \end{pmatrix}$$

## 3.7 SUMMARY

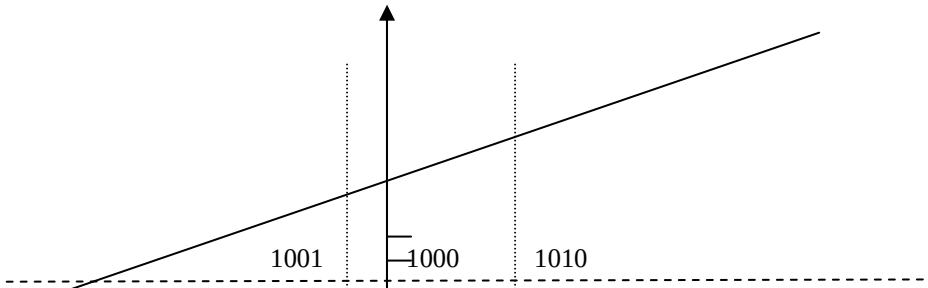
In this unit, we have discussed the concept of clipping, along with the concept of clipping we have also discussed various clipping types. The unit is quite important from the point of view of achieving realism through computer graphics.

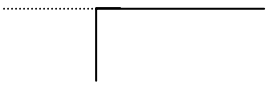
This unit contains algorithms, which are easy to implement in any programming language.

## 3.7 SOLUTIONS/ANSWERS

### Check Your Progress 1

1) Rectangular window ABCD is defined such that A = (–1, –2) and C (3, 1). So, to clip the line segment joining the points P(–20, 0) and Q(20, 30) using generalised geometrical approach we do the following tasks.





Equations of line segment PQ is:

y = mx + c where $m = \frac{y_2 - y_1}{x_2 - x_1}$ i.e $m = \frac{30 - 0}{20 - (-20)} = \frac{30}{40} = 3/4$

$\therefore$ equation of line PQ $\Rightarrow$ y = 3/4x + c ----------(1)

As point P (-20, 0) lies on the line given by equation (1) so it should satisfy equation (1) i.e.,

0 = ¾ * (-20) + c $\Rightarrow$ c = 15 ----------(2)

Using (2) in (1) we get the complete equation of line as

y = 3/4x + 15 ----------(3)

Steps to clip the line segment PQ are:

1) Point P is lying as LHS of window ABCD so by finding the intersection of line PQ with $y = y_{max} = 1$ we will get co-ordinates of P' using equation (3) put $y = y_{max} = 1$ in (3) we get,

$$1 = 3/4x + 15 \Rightarrow x = \frac{-14 * 4}{3} = -18.66$$

i.e., P'(-18.66, 1), which is lying outside the window region ABCDhence reject portion PP' of line segment PQ.

2) Now consider altered line segment P'Q
By finding the pt. of intersection of P'Q with $x = x_{min} = -1$ we will get p'', so put $x = x_{min} = -1$ in (3) we get

Y = ¾(-1) + 15 = -3/4 + 15 = 57/4 = 14.25

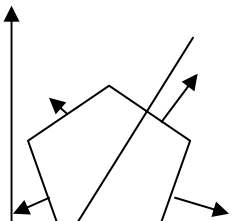
$\therefore$ $\Rightarrow$ P''(-1, 14.25) is also lying out of window region ABCD so reject portion P'P'' of line segment.

3) Now consider line segment P''Q
We will get P''' if we will find the pt. of intersection of P''Q with $x = x_{max} = 3$
$\therefore$ by using $x = x_{max} = 3$ in (3) we get

Y = ¾ * 3 + 15 = 9/4 + 15 = 69/4 = 17.25

$\therefore$ $\Rightarrow$ P'''(3, 17.25) is also lying out of window region ABCD hence Q (20,30) is also lying out $\therefore$ Reject whole line segment P'''Q.
That is how a whole line segment PQ is rejected in small segments.

2) The clipping window is given by $P_0(10,10)$, $P_1(20,10)$, $P_2(25,15)$, $P_3(20,20)$, $P_4(10,15)$, $P_5 = P_0$ and we need to clip the line A(0,0) and B(30,25). Using Cyrus Beck algorithm, we proceed as follows:



Steps to find clipping are:

1) Find $\overline{N}$ and using that find t:

   $t = \frac{-(\overline{P} - \overline{P_{i-1}}).\overline{N}}{(\overline{Q} - \overline{P}).\overline{N}}$, $\overline{P}, \overline{Q}$ end pts. of line segment $\overline{P_{i-1}}$ starting pt of edge check t lies in the interval (0,1)

2) Find pt. of intersection $\overline{P} + t(\overline{Q} - \overline{P})$

3) If $(\overline{Q} - \overline{P}).\overline{N} < 0$ then pt. of intersection is potentially entering ($P_E$) if $(\overline{Q} - \overline{P}).\overline{N} > 0$ then pt. of intersections potentially leaving ($P_L$)

4) Find (max) $t_{max}$ out of the value of $t_{max}$ for $P_E$'s and find(min) $t_{mix}$ out of the value of $t_{min}$ for $P_L$'s condition of clip is $t_{max} < t_{min}$

5) Clipping is available from $\overline{P} + t_{max}(\overline{Q} - \overline{P})$ to $\overline{P} + t_{min}(\overline{Q} - \overline{P})$.

Now, let us do the above steps so as to clip the line $\overline{AB}$ in clipping windows $p_0p_1$,$p_2$,$p_3$,$p_4$.

edge 0: (i.e. edge $P_0P_1$)

Say $\overline{N_0} = (n_{01}, n_{02})$ be normal to this edge $\overline{p_0 p_1}$. So $\overline{N_0}.\overline{p_0 p_1} = 0$

$\overrightarrow{p_0 p_1} = \overline{p_1} - \overline{p_0} = ((20,10) - (10,10) = (10,0)$ ----------(1)

$\overline{N_0}.\overline{p_0 p_1} = (n_{01}, n_{02}).(10,0) = 0$

$= 10n_{01} + 0n_{02} = 0 \quad \Rightarrow \quad n_{01} = \frac{-0}{10} n_{02}$

If we take $n_{02} = -1$ then $n_{01} = 0$ so, $\overline{N_0} = (0,1)$ ----------(2)

Now, to check that the point of intersection of line segment $\overline{AB}$ and edge is potentially entering or leaving find $\overline{AB}.\overline{N}$

$\overline{AB} = \overline{B} - \overline{A} = ((30-0),(25-0)) = (30,25)$

$\overline{AB}.\overline{N_0} = (30,25).(0,1) = -25 < 0$ *so pt.us PE*

$t_{max} = \frac{-(\overline{A} - \overline{P_0}).\overline{N_0}}{(\overline{B} - \overline{A}).N_0} = \frac{-((0,0) - (10,10)).(0,1)}{-25} = \frac{(10,10).(0,1)}{-25} = \frac{2}{5}$

edge 1: (i.e. edge $\overline{P_1P_2}$ )

Say, $\overline{N_1} = (n_{11}, n_{12})$ be normal to this edge $\overline{p_1 p_2}$. So $\overline{N_1}.\overline{p_1 p_2} = 0$

$\overrightarrow{p_1 p_2} = \overline{p_2} - \overline{p_1} = ((25,20) - (15,10) = (5,5)$ --------(1)

$(n_{11}, n_{12}).\overline{p_1 p_2} = (n_{11}, n_{12}).(5,5) = 5n_{11} + 5n_{12} = 0 \quad \Rightarrow \quad n_{11} = -n_{12}$

If $n_{12} = -1$ then $n_{11} = 1 \ \therefore \overline{N_1} = (n_{11}, n_{12}) = (1,-1)$ ----------(2)

Now let's find $\overline{AB}.\overline{N_1}$ to check that point of intersection is potentially entering or leaving

$\overline{AB}.\overline{N_1}$ = (30,25).(1,-1) = 30-25 = 5 > 0 so pt. is $P_L$

So, find $t_{min}$;

$$t_{\min} = \frac{-(\overline{A} - \overline{P_1}).\overline{N_1}}{(\overline{B} - \overline{A}).N_1} = \frac{-((0,0) - (20,10)).(1,-1)}{5} = \frac{(20,10).(1,-1)}{5} = \frac{20-10}{5} = 2;$$

reject this point of intersection

Edge 2: (i.e. edge $\overline{P_2P_3}$ )

Say $\overline{N_2} = (n_{21}, n_{22})$ be normal to this edge $\overline{p_2p_3}$ . So $\overline{N_2}.\overline{p_2p_3} = 0$

$$\overrightarrow{p_2p_3} = \overline{p_3} - \overline{p_2} = ((20-25),(20-15) = (-5,5) \qquad \text{--------(1)}$$

$$\overline{p_1p_2}.\overline{N_2} = (-5,5).(n_{21}, n_{22}) = 0 \Rightarrow -5n_{21} + 5n_{22} = 0 \Rightarrow n_{21} = n_{22}$$

If $n_{21} = 1$ then $n_{22} = 1 \ \therefore \overline{N_2} = (n_{21}, n_{22}) = (1,1)$ ----------(2)

To check that pt. of intersection in $P_L$ or $P_E$, angle sign of $\overline{AB}.\overline{N_2}$

$\overline{AB}.\overline{N_2}$ = (30,25).(1,1) = 30+25 > 0 so $P_L$

So, find $t_{min}$;

$$t_{\min} = \frac{-(\overline{A} - \overline{P_2}).\overline{N_2}}{(\overline{B} - \overline{A}).N_2} = \frac{-((0,0) - (25,15)).(1,1)}{55} = \frac{(25,15).(1,1)}{55} = \frac{25-15}{55} = \frac{40}{55} = \frac{8}{11}$$

Edge 3: (i.e. edge $\overline{P_3P_4}$ )

Say $\overline{N_3}$ be normal to $\overline{p_3p_4}$ , then $\overline{N_3}.\overline{p_3p_4} = 0$

$$\overrightarrow{p_3p_4} = \overline{p_4} - \overline{p_3} = ((10,15) - (20,20) = (-10,-5) \qquad \text{--------(1)}$$

$$\overline{p_1p_2}.\overline{N_3} = (-10,-5)(n_{31}, n_{32}) = 10n_{31} + 5n_{32} = 0 \ \Rightarrow \ n_{31} = -1/2\, n_{32}$$

if $n_{32} = 1$ then $n_{31} = -1/2 \ \therefore \overline{N_3} = (-1/2,-1)$ ----------(2)

Now, let us find that pt. of intersection is $P_E$ or $P_L$, lets check sign of $\overline{AB}.\overline{N_3}$

$\overline{AB}.\overline{N_3}$ = (30,25).(-1/2,-1) = 30(-1/2)+25(1) = -15+25 =10 > 0 so $P_L$

$$t_{\min} = \frac{-(\overline{A} - \overline{P_3}).\overline{N_3}}{(\overline{B} - \overline{A}).N_3} = \frac{-((0,0) - (20,20)).(-1/2,1)}{10} = \frac{(20,20).(-1/2,1)}{10} = \frac{-10+20}{10} = 1$$

Edge 4: (i.e. edge $\overline{P_4P_5}$ or $\overline{P_4P_0}$ ($\therefore$ $P_5 = P_0$)

Say $\overline{N_4}(n_{41}, n_{42})$ be normal to the edge $\overline{p_4p_0}$ , so $\overline{N_4}.\overline{p_4p_0} = 0$

$$\overrightarrow{p_4p_0} = \overline{p_0} - \overline{p_4} = ((10,10) - (10,15) = (0,-5) \qquad \text{--------(1)}$$

$$\overline{p_1p_2}.\overline{N_4} = (0,-5)(n_{41}, n_{42}) = 0n_{41} - 5n_{42} = 0 \ \Rightarrow \ n_{42} = \frac{0n_{41}}{5}$$

If $n_{41} = -1$ then $n_{42} = 0 \ \therefore \overline{N_4} = (-1,0)$ ----------(2)

Now to find whether that point of intersection is potentially entering or leaving find and check sign of $\overline{AB}.\overline{N_4}$

$\overline{AB}.\overline{N_4}$ = (30,25).(-1,0) = -30+0 < 0 so $P_E$

Now find that ($t_{max}$);

$$t_{\max} = \frac{-(\overline{A} - \overline{P_4}).\overline{N_4}}{(\overline{B} - \overline{A}).N_4} = \frac{-((0,0) - (10,15)).(-1,0)}{-30} = \frac{(10,15).(-1,0)}{-30} = \frac{-10}{-30} = \frac{1}{3}$$

Now out of all $t_{max}$ find max. value of $t_{max}$ and
out of all $t_{min}$ find min. value of $t_{min}$ and
$\max[t_{max}] = 2/5 = t_{max}$, $\min[t_{min}] = 8/11 = t_{min}$.

as $[t_{max}] < [t_{min}]$. So the line segment AB is clipped from

A + $t_{max}$(B-A) to A + $t_{min}$(B-A)

$\overline{A} + t_{\max}(\overline{B} - \overline{A})$

= (0,0) + 2/5[(30,25) - (0,0)] = (12, 10)

$\overline{A} + t_{\min}(\overline{B} - \overline{A})$

= (0,0) + 8/11[(30,25) - (0,0)] = (240/11, 200/11)

i.e., line is dipped from (12,10) to (240/11, 200/11), means

line AB is visible from (12,10) to (240/11, 200/11)

3) Let us compute the outward normal to a clipping in the case of Cyrus-Back algorithm. Consider a convex window, say it is hexagonal in shape; through which a line AB is passing and say $\overline{N_i}$ is normal at the edge of the hexagon where the line AB intersects i.e.,

$\overrightarrow{P_{i-1}P_i} = \overline{P_i} - \overline{P_{i-1}}$ is the edge with which the line segment AB intersects so,

$\overrightarrow{P_{i-1}P_i} = \overline{P_i} - \overline{P_{i-1}} = (x_i, y_i) - (x_{i-1}, y_{i-1})$ ----------(1)

$= (x_i - x_{i-1}, y_i - y_{i-1}) = (s_1, s_2)$

If normal $\overline{N_i} = (n_{1i}, n_{2i})$ ----------(2)

then, $\overrightarrow{P_{i-1}P_i}.\overrightarrow{N_i} = 0$

$(s_1, s_2).(n_{1i}, n_{2i}) = (s_1 n_{1i} + s_2 n_{2i}) = 0$

$n_{1i} = \frac{-s_2}{s_1} n_{2i}$

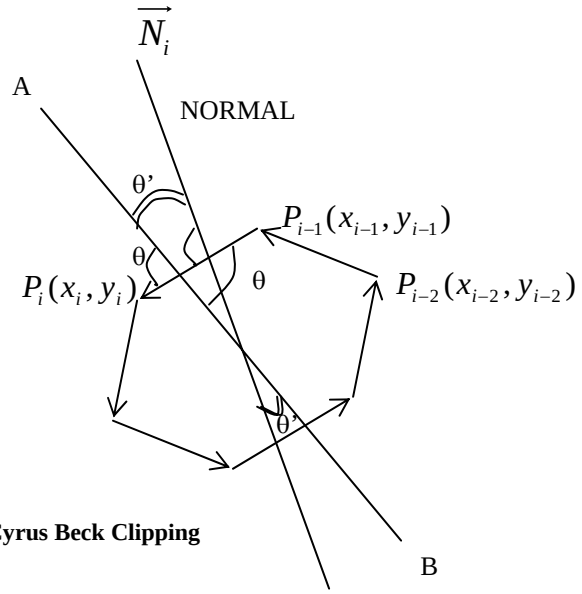



**Figure 18: Example Cyrus Beck Clipping**

Case1: If $n_{2i} = 1$ then $n_{1i} = \frac{-s_2}{s_1}$ $\therefore \overline{N_i} = (-s_2/s_1, 1)$

Case2: If $n_{2i} = -1$ then $n_{1i} = \frac{s_2}{s_1}$ $\therefore \overline{N_i} = (s_2/s_1, -1)$

Normal $\overline{N_i}$ in case 1 is opposite of the Normal in case 2 i.e. the direction is opposite.

So, if $\overline{N_i}$ in case is OUTWARD NORMAL then $\overline{N_i}$ in case2 will be INWARD NORMAL

In CYRUS-BECK Algorithm the point of intersection is either potentially entering on potentially leaving, lets study the two with inward/outward normal.

Case A: POTENTIALLY ENTERING CASE (*Figure 19*)

In *Figure19* the point of intersection given by $\overline{AB}$ is potentially entering

i) $\overline{N_i}$ is outward normal then,

$$\overline{AB}.\overline{N_i} = \left|\overline{AB}\right|\left|\overline{N_i}\right|\cos(\theta + 90°) = -\left|\overline{AB}\right|\left|\overline{N_i}\right|\sin\theta < 0$$

ii) $\overline{N_i}$ is inward normal then,

$$\overline{AB}.\overline{N_i} = \left|\overline{AB}\right|\left|\overline{N_i}\right|\cos\theta' = \left|\overline{AB}\right|\left|\overline{N_i}\right|\cos(90° - \theta) = -\left|\overline{AB}\right|\left|\overline{N_i}\right|\sin\theta$$

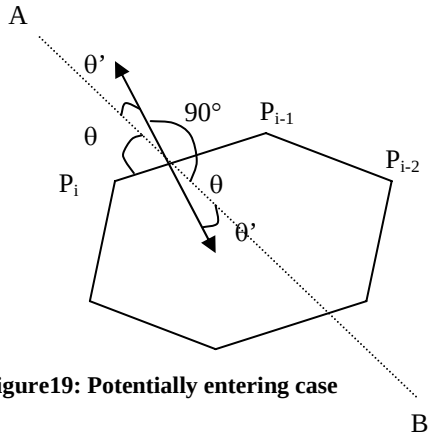



**Figure19: Potentially entering case**

Similarly ;

Case B: POTENTIALLY LEAVING CASE

When the point of intersection of line AB with edge $\overrightarrow{P_{i-1}.P_i}$ is potentially leaving.

i) $\overline{N_i}$ is outward normal then,

$$\overline{AB}.\overline{N_i} = \left|\overline{AB}\right|\left|\overline{N_i}\right|\cos\theta' = \left|\overline{AB}\right|\left|\overline{N_i}\right|\cos(90° - \theta) = \left|\overline{AB}\right|\left|\overline{N_i}\right|\sin\theta > 0$$

ii) $\overline{N_i}$ is inward normal then,

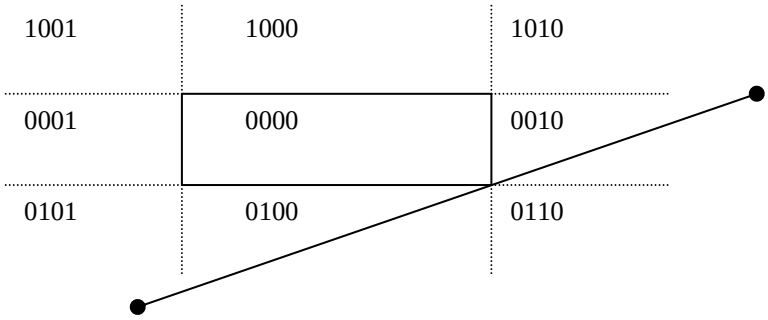
$$\overline{AB}.\overline{N_i} = \left|\overline{AB}\right|\left|\overline{N_i}\right|\cos(90° + \theta) = -\left|\overline{AB}\right|\left|\overline{N_i}\right|\sin\theta < 0$$

i.e., by analyzing the sign of dot product of $\overline{AB}$ with $\overline{N_i}$ in case of potentially entering/leaving pt. we can find that normal $\overline{N_i}$ is outward normal or inward normal.

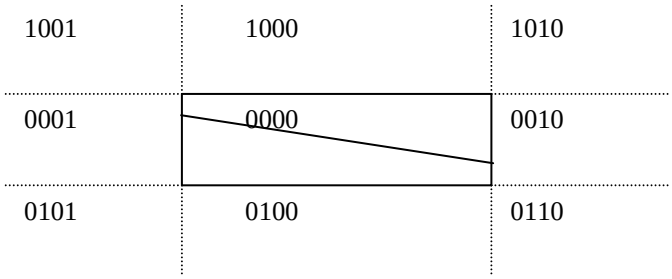
4) Limitations of Cohen-Sutherland clipping algorithm are:

1) Clipping window region can be rectangular in shape only and no other polygonal shaped window is allowed.
2) Edges of rectangular shaped clipping window has to be parallel to the x-axis and y-axis.
3) If end pts of line segment lies in the extreme limits i.e., one at R.H.S other at L.H.S., and on one the at top and other at the bottom (diagonally) then, even

if the line doesn't pass through the clipping region it will have logical intersection of 0000 implying that line segment will be clipped but infact it is not so.




5) Using Cohen Sutherland Line clipping algorithm when we clip the line we get the line shown in *Figure* below as output.




6) Window regions which are Non convex can't be handled by the CYRUS-BECK ALGORITHM without modification.

As we know that in Cyrus Beck algorithm the line is to draw if tmax < tmin else it is to be rejected and as per equation (1) and Figure we find that the whole line LM will be rejected but some portion of LM has to be visible, to achieve this, the algorithm has to be modified so as to handle new convex window regions.

Modifications to be applied to Cyrus Beck algorithm to handle the non convex region case are:

1) The non convex region is to be divided into many convex sub regions.

2) tmax and tmin are to be found w.r.t individual convex window region hence line is to be clipped for different convex window regions.

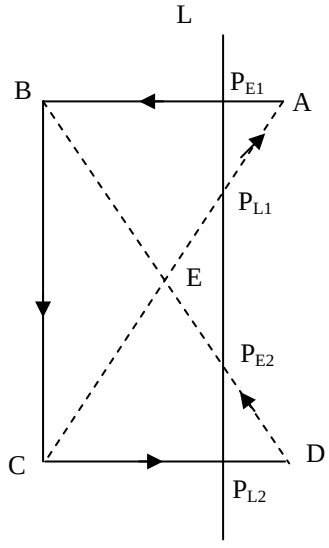
As per the modification the window region ABCDE is to be divided into two convex windows.

AEBA and BCDEB

Or

CDEC and ABCEA

ABCDE → Non convex window3
LM → line to be clipped
$P_E$ → Potentially entering
$P_L$ → Potentially leaving




**Check Your Progress 2**

1) Clipping the given polygon using Sutherland Hodgeman polygon clipping algorithm we will get the result given in the *Figure* below as output.