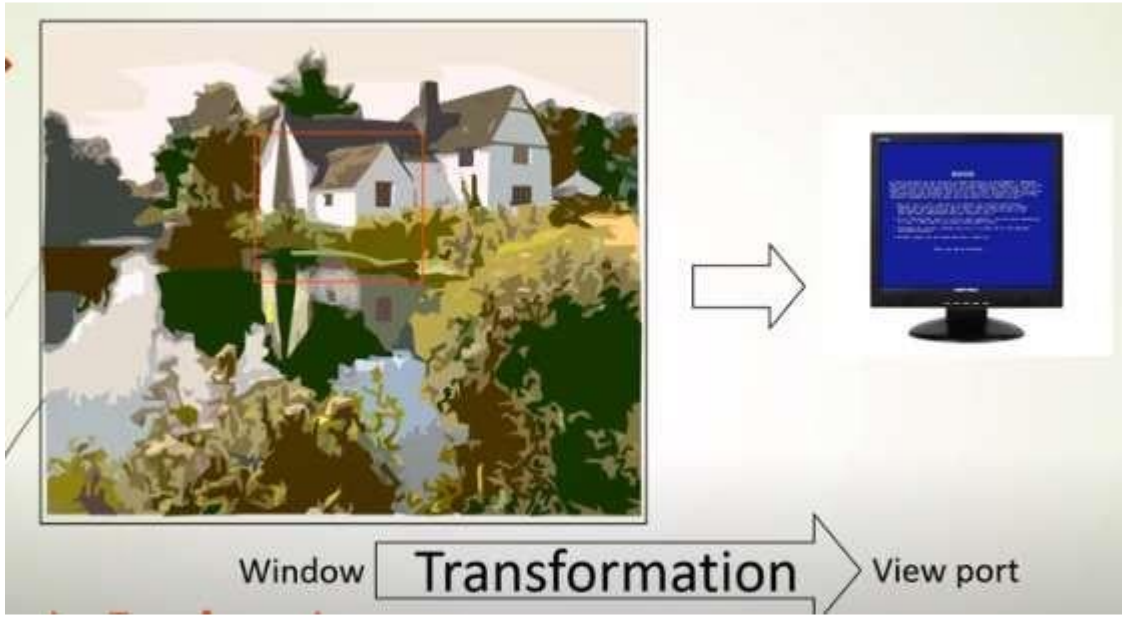
COMPUTER GRAPHICS

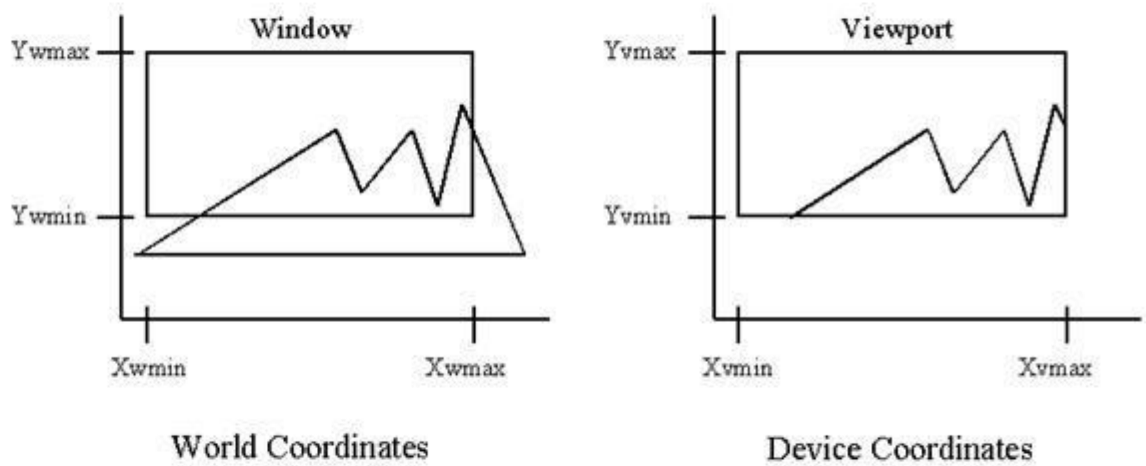
MODULE 3

THE VIEWING PIPELINE

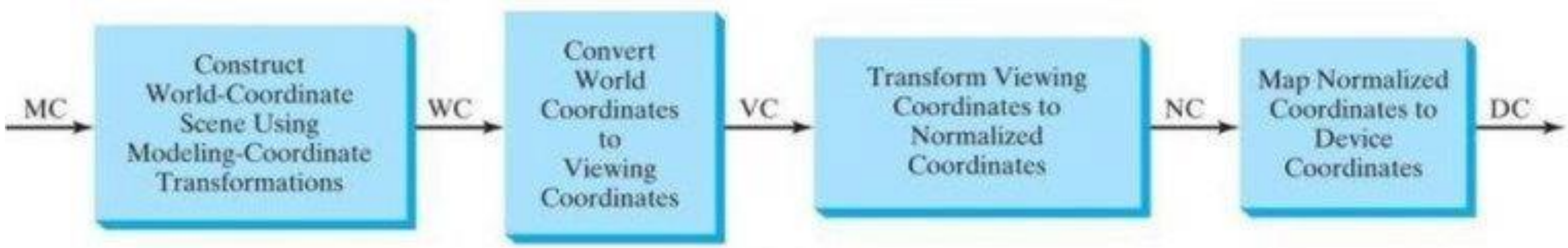
- **Window**: A world-coordinate area selected for display. The window defines what is to be viewed
- **Viewport** : An area on a display device to which a window is mapped. The **viewport** defines where it is to be displayed.
- Windows and viewports are **rectangles** in standard position, with the rectangle edges parallel to the coordinate axes.
- **Viewing transformation**:
  - The mapping of a part of a world-coordinate scene to device coordinates.
  - 2D viewing transformation is also called **window-to-viewport transformation or the windowing transformation.**


Window
Transformation
View port


Window
Transformation
View port



Steps in 2D viewing pipeline



The viewing transformation is carried out in several steps.

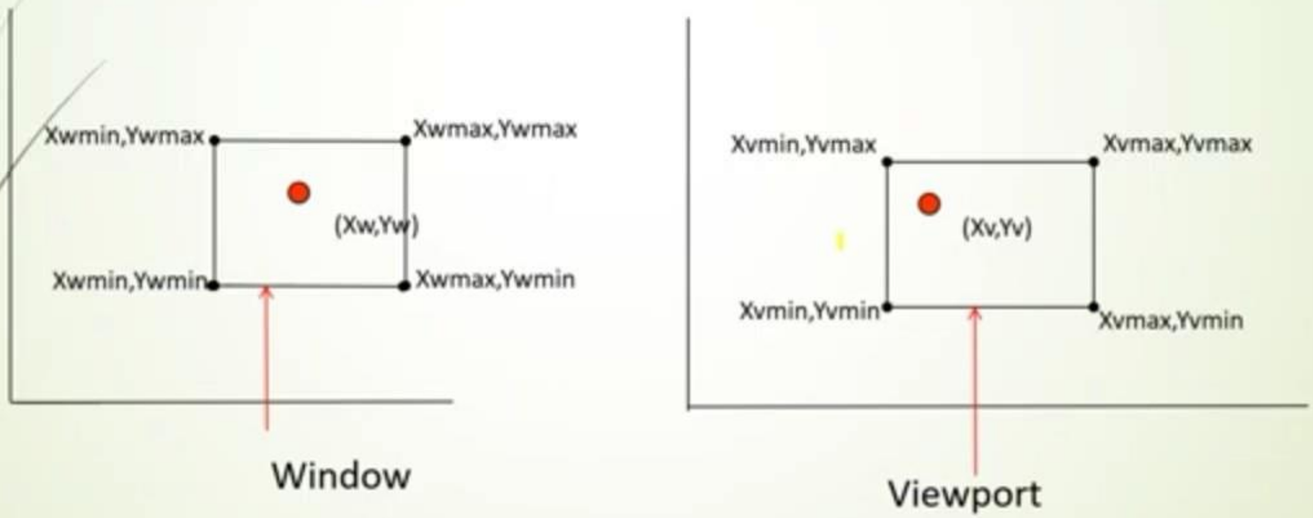
- First, we construct the scene in world coordinates using the output primitives and attributes.
- Next. to obtain a particular orientation for the window, we can set up a two-dimensional viewing-coordinate system in the world-coordinate plane, and define a window in the viewing-coordinate system.
- The viewing coordinate reference frame is used to provide a method for setting up arbitrary orientations for rectangular windows.
- Once the viewing reference frame is established, we can transform descriptions in world coordinates to viewing coordinates.
- We then define a viewport in normalized coordinates (in the range from 0 to 1) and map the viewing-coordinate description of the scene to normalized coordinates.
- At the final step, all parts of the picture that are outside the viewport are clipped, and the contents of the viewport are transferred to device coordinates.

WINDOW-TO-VIEWPORT COORDINATE TRANSFORMATION

- Once object descriptions have been transferred to the viewing reference frame, we choose the window extents in viewing coordinates and select the viewport limits in normalized coordinates.
- Object descriptions are then transferred to normalized device coordinates.
- We do this using a transformation that maintains the same relative placement of objects in normalized space as they had in viewing coordinates.
- If a coordinate position is at the center of the viewing window, for instance, it will be displayed at the center of the viewport.

Window to view port transformation

For every point (Xw,Yw) in the window there is a corresponding or equivalent or relative point (Xv, Yv) in view port



$$\frac{Xw - Xwmin}{Xwmax - Xwmin} = \frac{Xv - Xvmin}{Xvmax - Xvmin} \longrightarrow 1$$

$$\frac{Yw - Ywmin}{Ywmax - Ywmin} = \frac{Yv - Yvmin}{Yvmax - Yvmin} \longrightarrow 2$$

From (1) $Xv = \dfrac{\mathbf{Xvmax - Xvmin}}{\mathbf{Xwmax - Xwmin}} \times (Xw - Xwmin) + Xvmin$

From (2) $Yv = \dfrac{\mathbf{Yvmax - Yvmin}}{\mathbf{Ywmax - Ywmin}} \times (Yw - Ywmin) + Yvmin$

So, corresponding to a window point (Xw, Yw) , the view port point (Xv, Yv) is

$$\left\{ \underbrace{\left(\frac{Xvmax - Xvmin}{Xwmax - Xwmin} \right)}_{\text{window to view port scaling factor of X, } S_x} * (Xw - Xwmin) + Xvmin \quad , \quad \underbrace{\left(\frac{Yvmax - Yvmin}{Ywmax - Ywmin} \right)}_{\text{window to view port scaling factor of Y, } S_y} * (Yw - Ywmin) + Yvmin \right\}$$

So, $X_v = S_x (X_w - X_{wmin}) + X_{vmin}$

Scaling factor Translation factor

& $Y_v = S_y(Y_w - Y_{wmin}) + Y_{vmin}$

So, window to view port transformation involves
SCALING AND TRANSLATION

$$X_v = S_x (X_w - X_{wmin}) + X_{vmin}$$

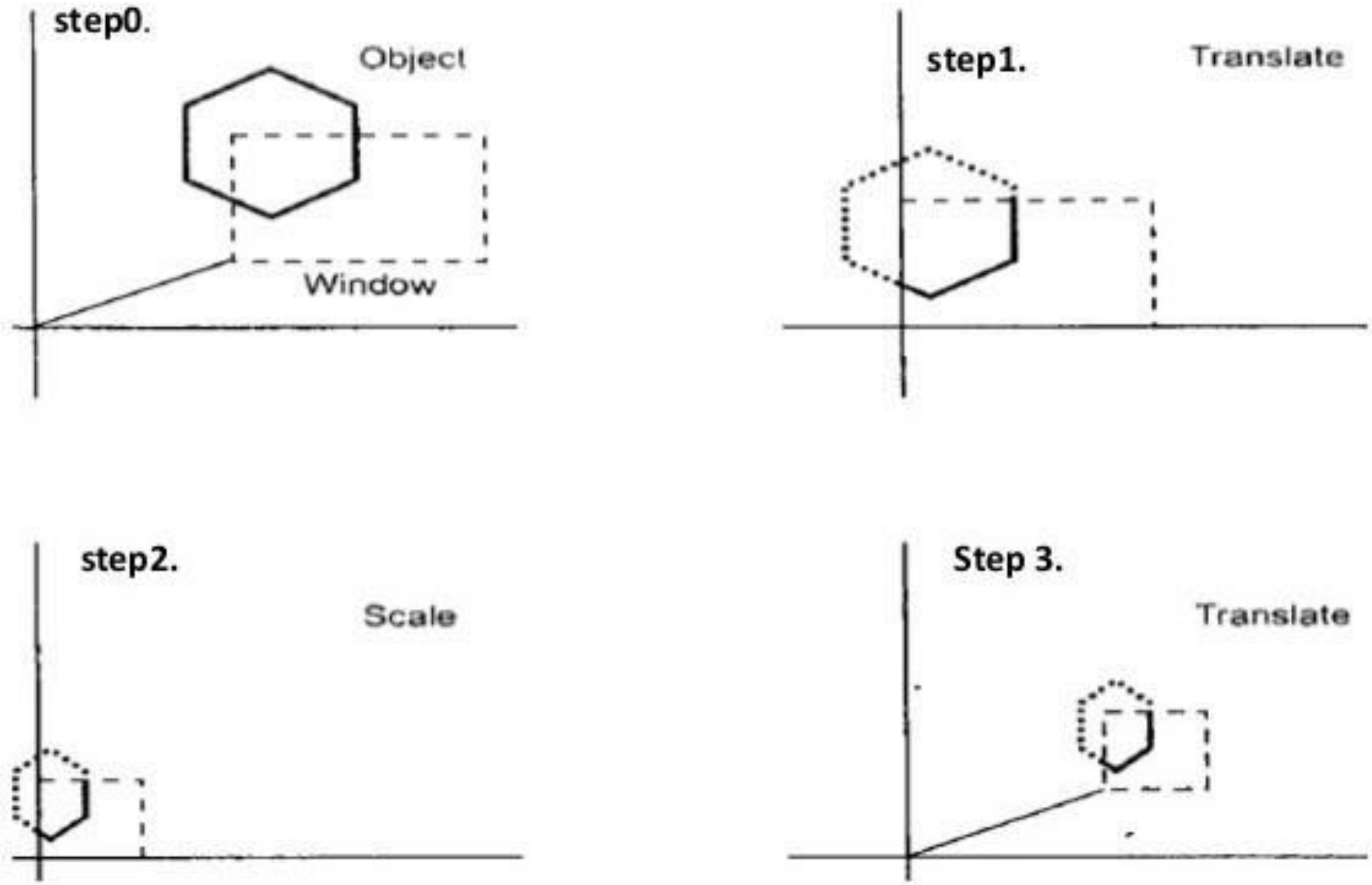
$$Y_v = S_y(Y_w - Y_{wmin}) + Y_{vmin}$$

Contd…

- Given a window and viewport, what is the transformation matrix that maps the window from world coordinates into the viewport in screen coordinates?
- This matrix can be developed by three step transformation composition as:
  - The object together with its widow is translated until the lower left corner of the window is at the origin.
  - Object and window are scaled until the window has the dimension of the viewport.
  - Translate the viewport to its correct position on the screen.

Contd…

- Above three steps are illustrated in the following figure:



Contd…

- Step1: translation

$$T = \begin{bmatrix} 1 & 0 & -X_{W\min} \\ 0 & 1 & -Y_{w\min} \\ 0 & 0 & 1 \end{bmatrix}$$

Contd…

- Step2: scaling about origin

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

- Where,

$$sx = \frac{xv_{max} - xv_{min}}{xw_{max} - xw_{min}}$$

$$sy = \frac{yv_{max} - yv_{min}}{yw_{max} - yw_{min}}$$

Contd…

- Step 3: inverse translation

$$inverse(T) = \begin{bmatrix} 1 & 0 & X_{v\min} \\ 0 & 1 & Y_{v\min} \\ 0 & 0 & 1 \end{bmatrix}$$

Contd…

- The overall transformation matrix for window to viewport transformation is:

$$=\begin{bmatrix} 1 & 0 & X_{v\min} \\ 0 & 1 & Y_{v\min} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -X_{W\min} \\ 0 & 1 & -Y_{w\min} \\ 0 & 0 & 1 \end{bmatrix}$$

$$=\begin{bmatrix} s_x & 0 & (xv_{\min} - xw_{\min}).sx \\ 0 & s_y & (yv_{\min} - yw_{\min}).sy \\ 0 & 0 & 1 \end{bmatrix}$$

Questions:

1. Derive an equation for window to viewport transformation by specifying the sequence of basic transformations involved. (3)
2. Define the terms window, viewport and windowing transformation in the context of 2D viewing with suitable diagrams. (4)
3. Explain the window to viewport coordinate transformation and also derive the scaling factors during the transformation. (5)

Clipping

- Any procedure that identifies those portions of a picture that are either inside or outside of a specified region of a space is referred to as clipping.
- The region against which an object is to be clipped is called a clip window. Depending on the application a clip window can be polygon or curved boundaries.
- World- coordinate clipping removes the primitives outside the window from further consideration; thus eliminating the processing necessary to transform these primitives to device space.
- **Clipping type-** point ,line, polygon, curved areas and etc.

Contd…

- Applications of clipping include:
  - Extracting part of a defined scene for viewing
  - Identifying visible surfaces in three-dimensional views
  - Antialiasing line segments or object boundaries
  - Creating objects using solid-modeling procedures
  - Displaying a multi-window environment
  - Drawing and painting operations that allow parts of a picture to be selected for copying, moving, erasing, or duplicating.

Point Clipping

- In a rectangular clip window save a point P = (x, y) for display(i.e., not clipped) if the following inequalities are satisfied:
  - $Xw_{min} \leq x \leq xw_{max}$
  - $yw_{min} \leq y \leq yw_{max}$
- If any one of these four inequalities is not satisfied, the point is clipped (not saved for display).
- The equal sign indicates that point on the window boundary are included within the window.

Contd…

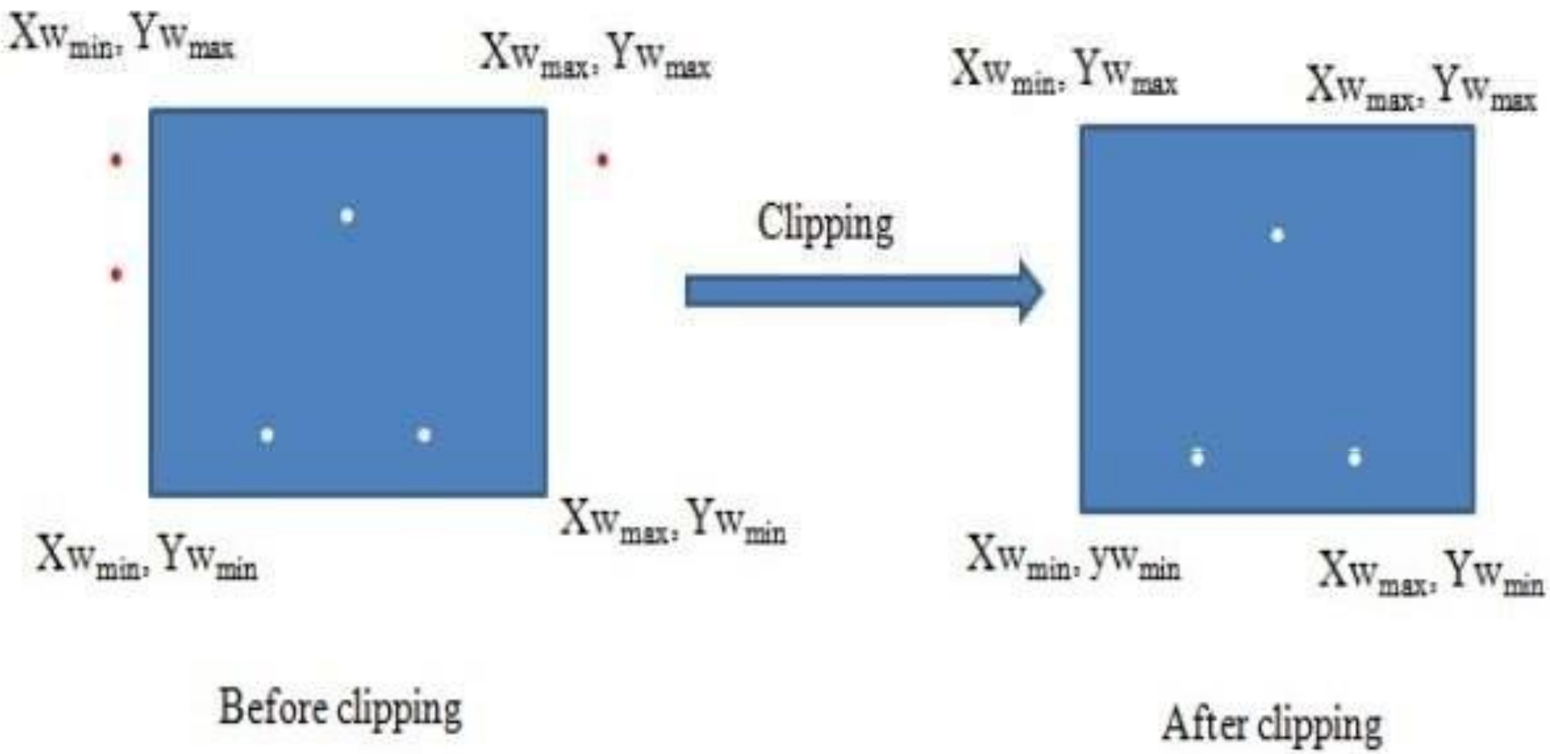


Fig: Point Clipping process

University Questions

- Explain the concept of point clipping in 2D. (2)

Line clipping

- The visible segment of a straight line can be determined by **inside – outside test:**
  - A line with both endpoints **inside** clipping boundary, such as the line from $\mathbf{p_1}$ to $\mathbf{p_2}$, is saved.

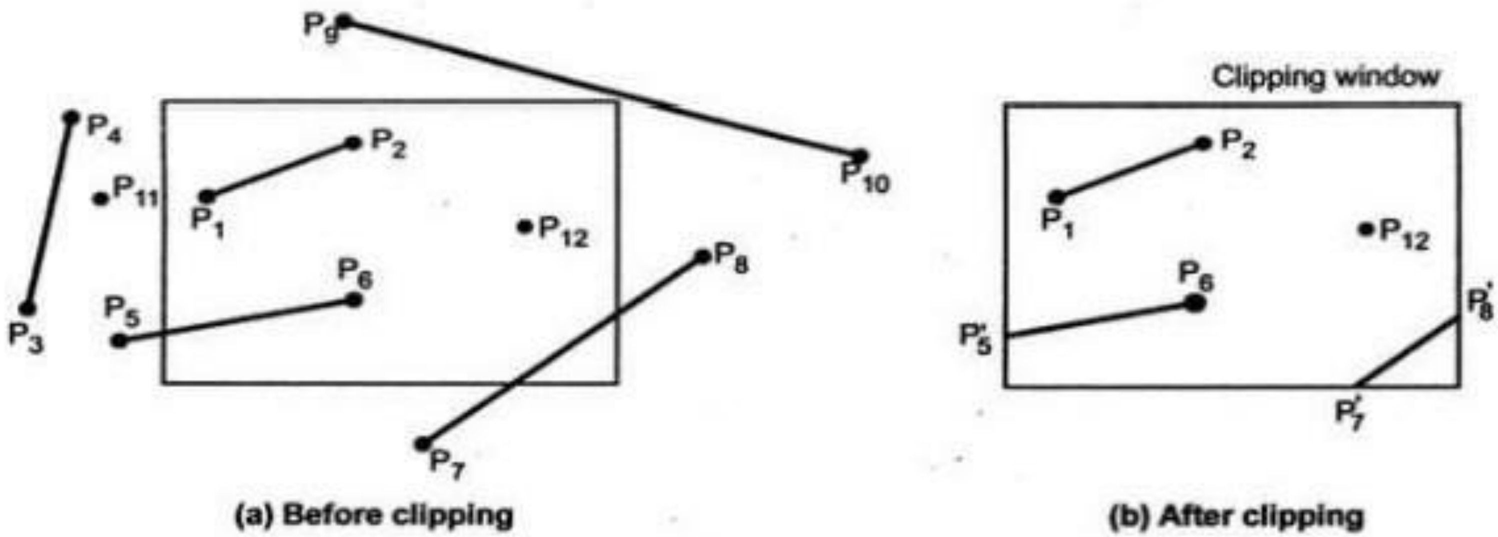


Fig. (e)

Contd…

- A line with both endpoints **outside** the clip boundary, such as the line from $\mathbf{p_3}$ to $\mathbf{p_4}$, is not saved.

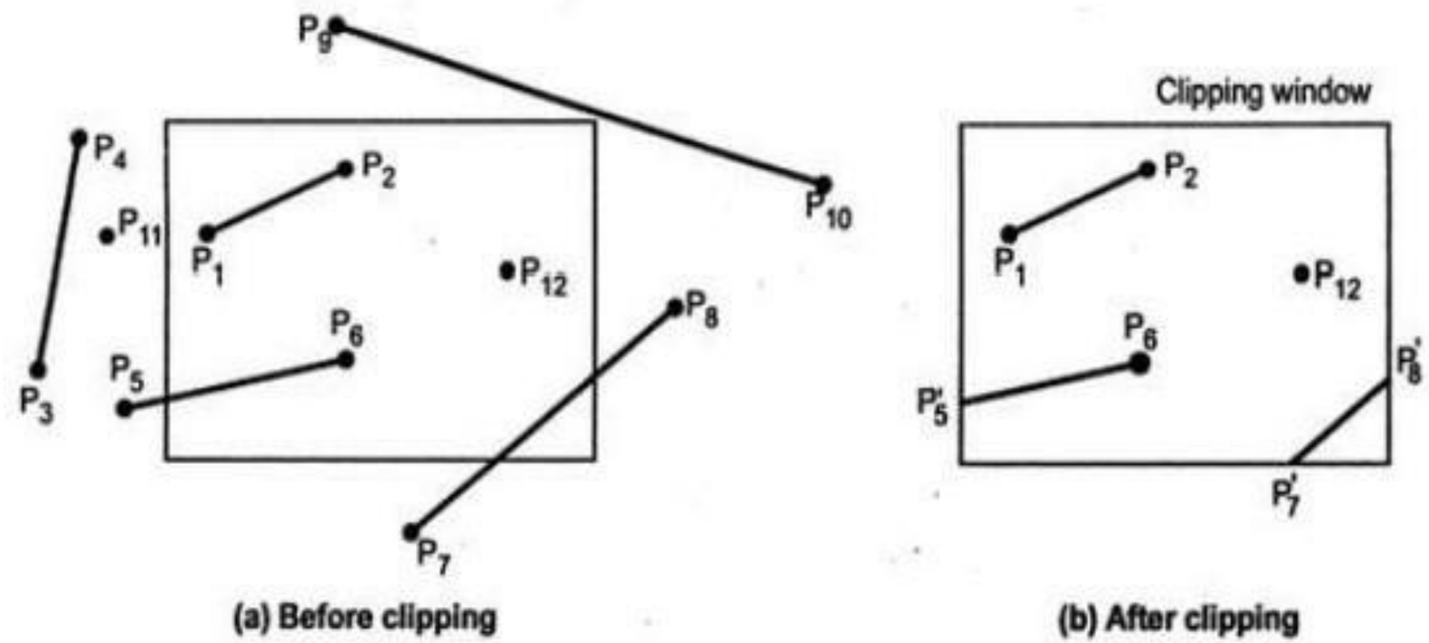


Fig. (e)

Contd…

- If the line is not completely inside or completely outside (e.g., p7 to p8), then perform intersection calculations with one or more clipping boundaries.

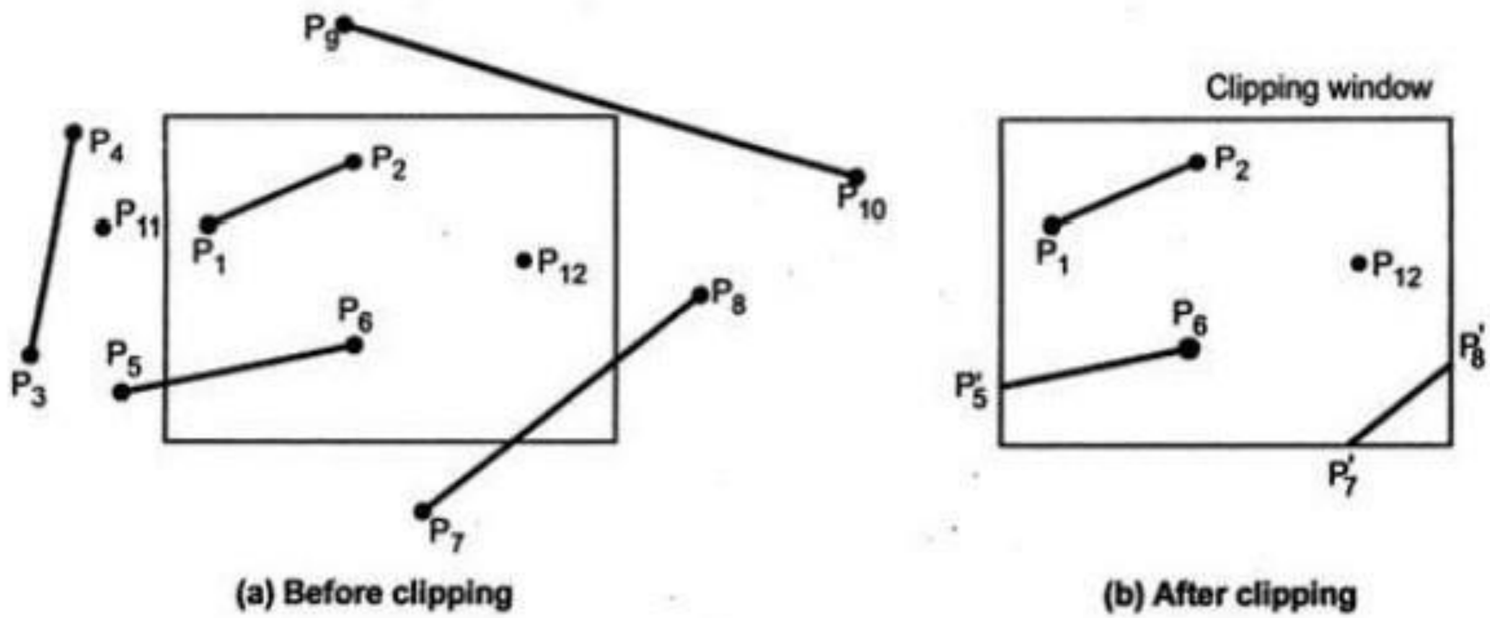


Fig. (e)

- For a line segment with endpoints (x1,y1) and (x2, y2) and one or both endpoints outside the clipping rectangle, the parametric representation

  x=x1+u (x2-x1)

  y=y1+u(y2-y1) where 0<=u<=1

  could be used to determine values of parameter *u* for intersections with clipping boundary coordinates.
- If the **value of** u for an intersection with a rectangle boundary edge is **outside the range 0 to 1**, the line **does not enter the interior** of the window at that boundary.
- If the **value of u** is **within the range from 0 to 1**, the line segment **does indeed cross into the clipping area**.

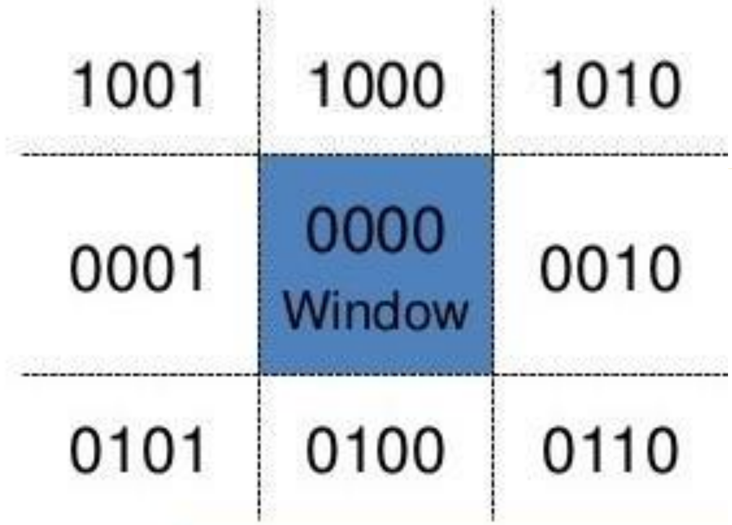
- This method can be applied to each clipping boundary edge in turn to determine whether any part of the line segment is to be displayed.
- Line segments that are **parallel to window e**dges can be handled as **special cases**.
- Clipping line segments with these parametric tests requires a good deal of computation, and faster approaches to clipping are possible.

Line clipping algorithms

- Cohen Sutherland algorithm
- Midpoint Subdivision algorithm

Cohen-Sutherland Line Clipping

- This algorithm clips the line by performing following steps:
- Step1: Region code assignment
  - divide the whole picture region into nine regions by extending the window boundaries as shown in figure below and then assign a 4-bit region code to each region as follows:



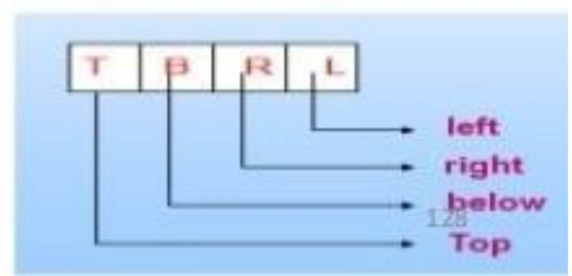
Rules For assigning region code:

T=1 , if the region is above the window,
= 0 , otherwise.

B= 1, if the region is below the window,
= 0, otherwise.

R=1, if the region is right of window,
= 0, otherwise.

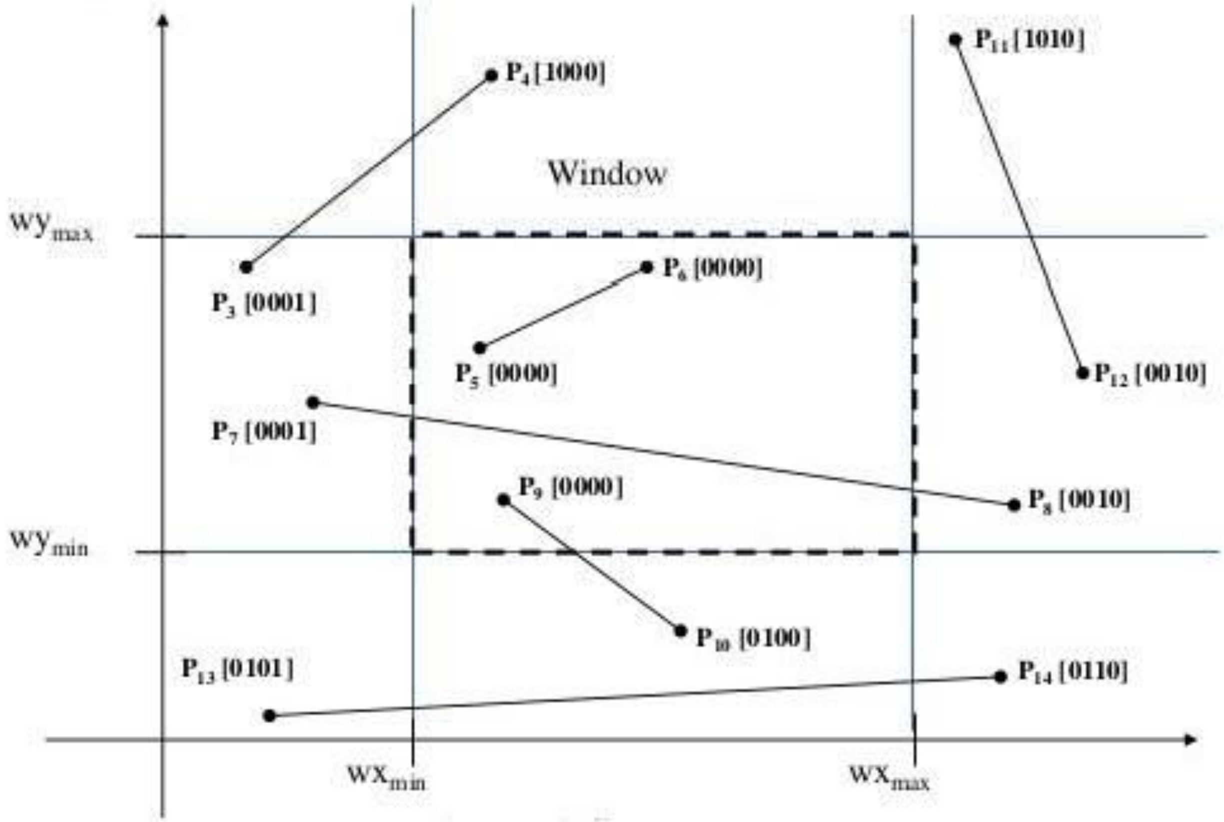
L= 1, if the region is left of window,
= 0, otherwise.



Contd…

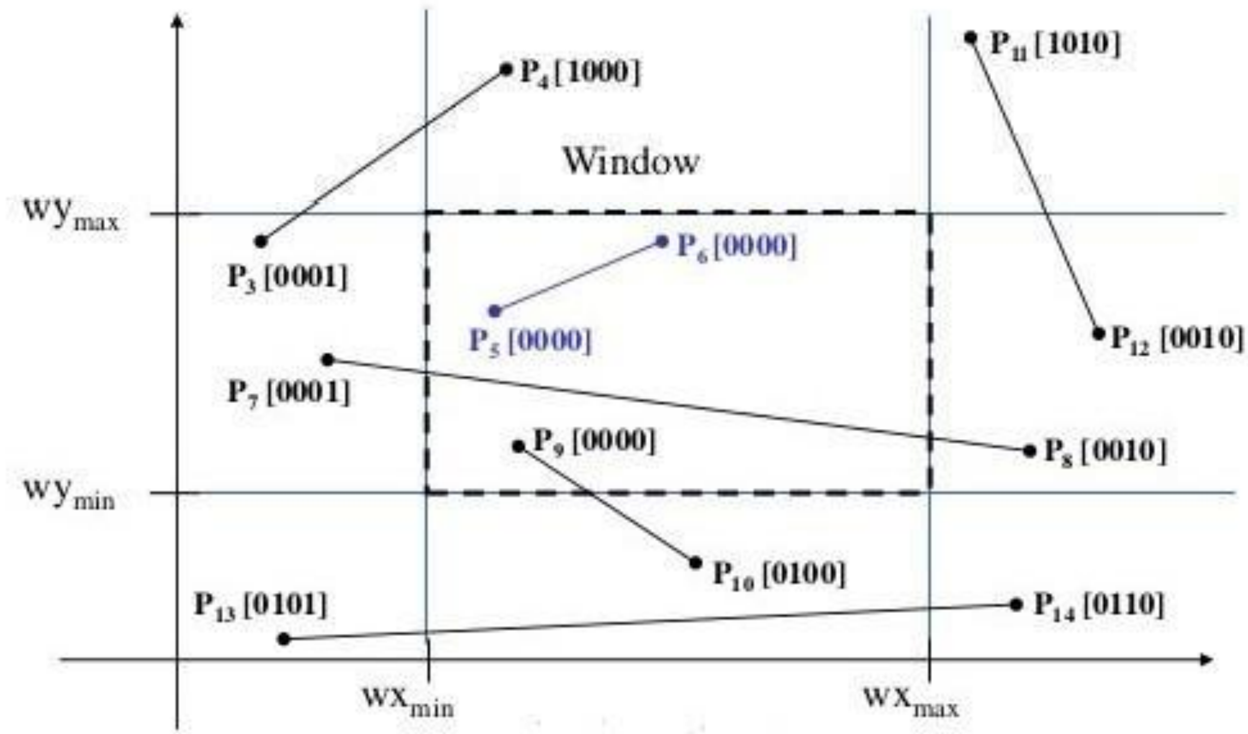
- Every end-point is labelled with the appropriate region code

- For example:



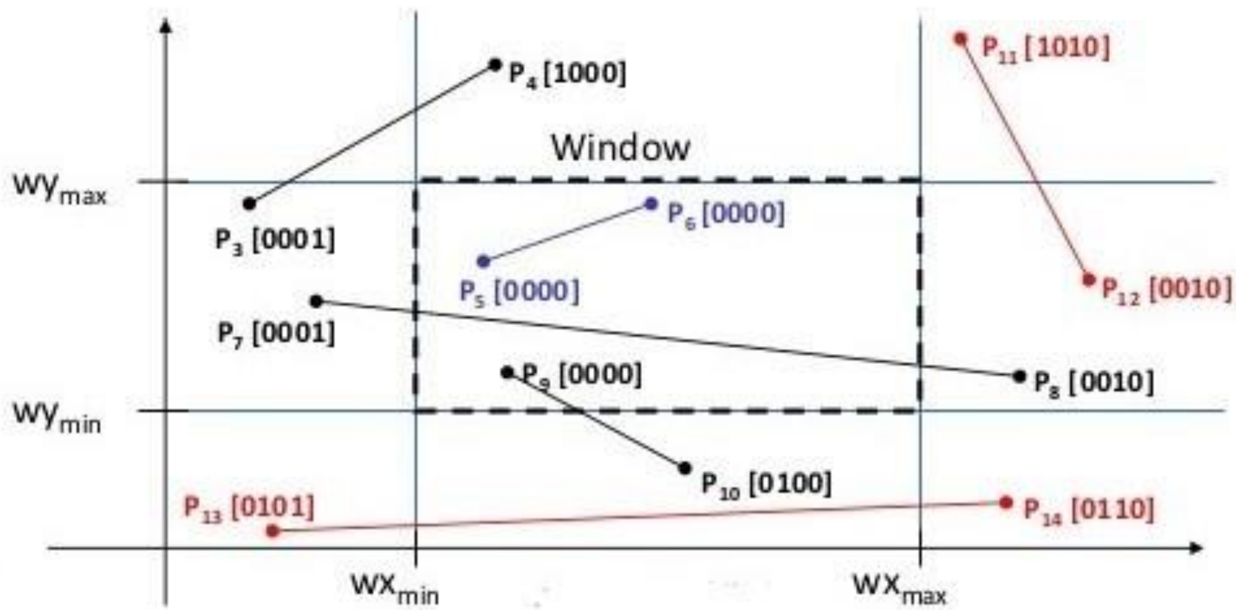
Contd…

- Step2: Trivial acceptance of line segment
  - Lines completely contained within the window boundaries have region code [0000] for both end-points so are not clipped. i.e., accept these lines trivially.
  - *E.g., line p5p6*



Contd…

- Step3: Trivial rejection and clipping
  - 3.1 : Any lines that have a **1** in the same bit position in the region-codes for each endpoint are **completely outside** and we reject these lines.
    - The AND operation can efficiently check this: If the logical AND of both region codes result is **not 0000**, the line is **completely outside** the clipping region so clipped.



AND operation

INPUT		OUTPUT
A	B	Y = A.B
0	0	0
0	1	0
1	0	0
1	1	1

Contd…

3.2:If the logical AND operation results in 0000 then

a) Choose an endpoint of the line that is outside the window.

b) Find the intersection point at the window boundary by using the following formula:

Intersection with vertical boundary

$$y = y_1 + m(x - x_1)$$

Where

$$x = xw_{min} \text{ or } xw_{max}$$

Intersection with horizontal boundary

$$x = x_1 + (y - y_1)/m$$

Where

$$y = yw_{min} \text{ or } yw_{max}$$

c) Replace endpoint with the intersection point and update the region code.

Above process is repeated until we find a clipped line either trivially accepted or trivially rejected.

Algorithm

Step 1 – Assign a region code for each endpoints.

Step 2 – If both endpoints have a region code **0000** then accept this line.

Step 3 – Else, perform the logical **AND** operation for both region codes.

- **Step 3.1** – If the result is not **0000,** then reject the line.
- **Step 3.2** – Else you need clipping.
  - **Step 3.2.1** – Choose an endpoint of the line that is outside the window.
  - **Step 3.2.2** – Find the intersection point at the window boundary using set of equations given below.(based on region code and window boundary).

    Intersections with a vertical boundary:

    y=y1+m(x-x1) (x is either xmin or xmax)

    Intersections with a horizontal boundary:

    x=x1+(y-y1)/m (y is either ymin or ymax)

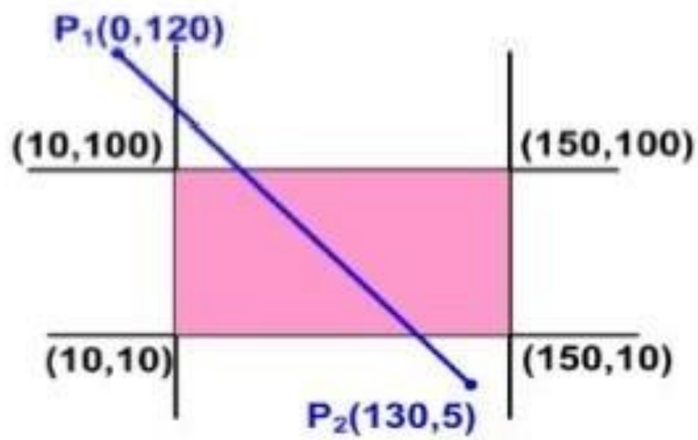
  - **Step 3.2.3** – Replace endpoint with the intersection point and update the region code.

Step 4 – Repeat step 1 for other lines.

Contd..

- Example: Use the Cohen-Sutherland line clipping algorithm to clip line defined with end points p1(0,120) and (130,5) against a window defined by the following four points : (10,10), (10,100), (150,10) and (150,100)

Obtain the endpoints of line P_1P_2 after cohen-sutherland clipping



Contd…

- Solution:

1. $\mathbf{P_1}$=1001 $\mathbf{P_2}$=0100
2. Both (0000)→NO
3. And Operation

   1001

   0100

Result→0000

3.1 not(0000)→NO

3.2 0000→ yes

3.2.1 choose $\mathbf{P_1}$

3.2.2 Intersection with **LEFT** boundary

$m=(5-120)/(130-0)= -0.8846$

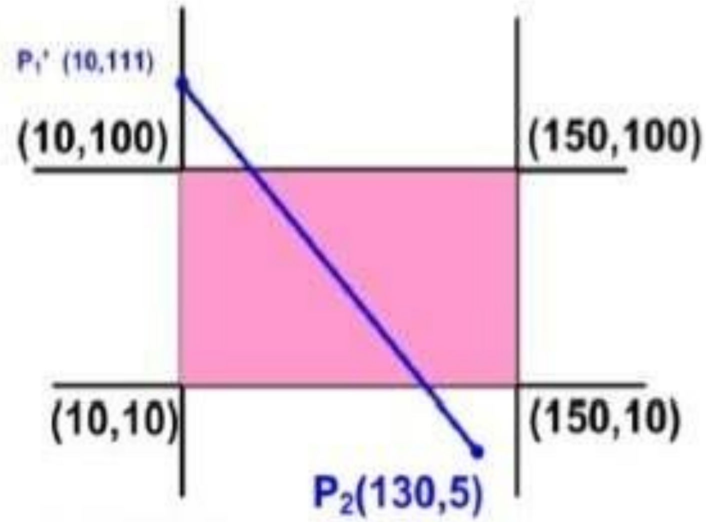
$y = y_1 + m(x-x_1)$ where $x=10$

$y=120-0.8846(10-0)=111.15\approx111$

$\mathbf{P_1}'= (10, 111)$

3.2.3 Update region code $\mathbf{P_1}'$= 1000 **(TOP)**

3.2.4 Repeat step 2.



Contd…

1. $\mathbf{P_1}$'=1000 $\mathbf{P_2}$=0100
2. Both (0000)→NO

And Operation

1000

0100

Result→0000

3.1 not(0000)→NO

3.2 0000→ yes

3.2.1 choose $\mathbf{P_1}$'

3.2.2 Intersection with **TOP** boundary

$m=(5-120)/(130-0)= -0.8846$

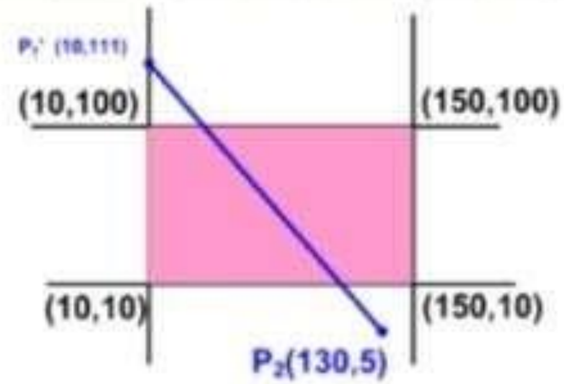
$x = x_1 + (y-y_1)/m$ where $y= 100$

$x=10+ (100-111)/(-0.8846)=22.44 \approx 22$

$\mathbf{P_1}'' = (22, 100)$

3.2.3 Update region code $\mathbf{P_1}''$= 0000

3.2.4 Repeat step 2.



Contd…

1. P_1''=0000 P_2=0100

2. Both (0000)→NO

And Operation

0000

0100

Result→0000

3.1 not(0000)→NO

3.2 0000→ yes

3.2.1 choose P_2

3.2.2 Intersection with **BOTTOM** boundary

$m=(5-120)/(130-0)= -0.8846$

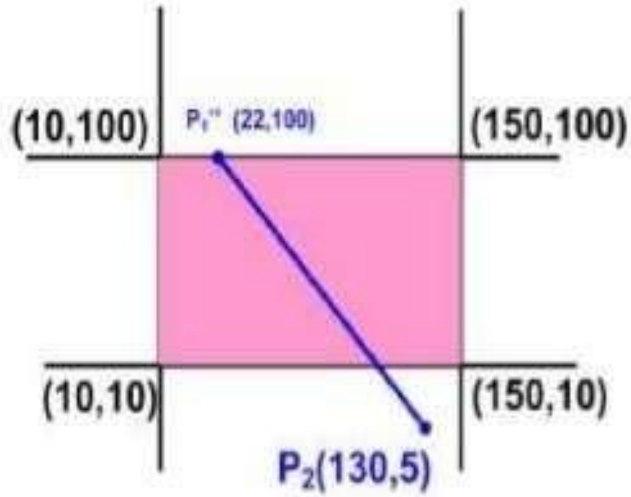
$x = x_1 + (y-y_1)/m$ where $y= 10$

$x=130+ (10-5)/(-0.8846)=124.35 \approx 124$

$P_2' = (124, 10)$

3.2.3 Update region code P_2'= 0000

3.2.4 Repeat step 2.



Contd…

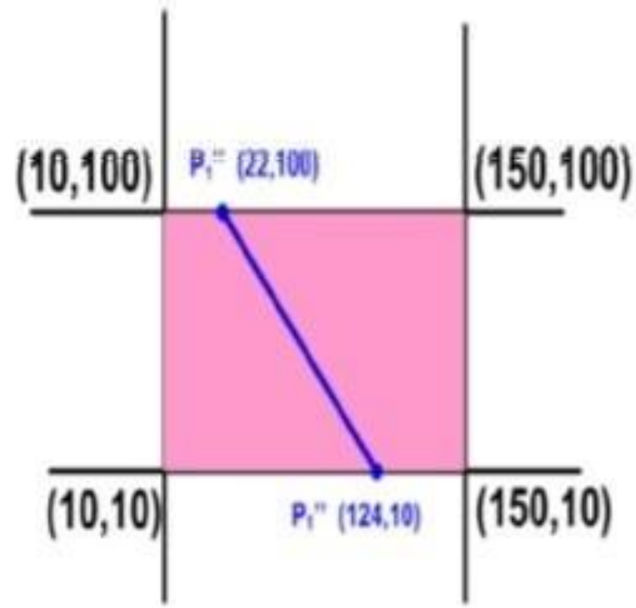
Both (0000)→YES

ACCEPT & DRAW

Thus endpoints after clipping

$\mathbf{P_1}$''= (22, 100)

$\mathbf{P_2}$'= (124, 10)



Homework

- **Example1:** Let ABCD be the rectangular window with A(20, 20), B(90, 120), C(90, 70), and D(20,70). Find the region codes for end points and use Cohen- Sutherland algorithm to clip the line p q with p(10,30) and q(80, 90).

- **Example2**: Use the Cohen-Sutherland algorithm to clip line defined with end points p1(40,15) and p2(75,45) against a window A(50,10), B(80,10), C(80,40) and D(50,40).

- **Example3**: Use the Cohen-Sutherland algorithm to clip line defined with end points p1(70, 20) and p2(100,10) against a window A(50,10), B(80,10), C(80,40) and D(50,40).

University Questions

- Given a clipping window A(-20,-20), B(40,-20), C(40,30) and D(-20,30). Using Cohen Sutherland line clipping algorithm, find the visible portion of the line segment joining the points P(-30,20) and Q(60,-10). (6)
- Explain the Cohen Sutherland line clipping algorithm with suitable examples. (6)
- How does Cohen Sutherland algorithm determine whether a line is visible, invisible or a candidate for clipping based on the region codes assigned to the end points of the line? (4)

Midpoint Subdivision algorithm

- This algorithm is mainly used to compute visible areas of lines that are present in the view port are of the sector or the image.
- It is based on bisection method.
- The line is divided at its midpoint into two shorter line segments using the endpoint values.
- The midpoint coordinates (xm, ym) of a line joining (x1, y1) and (x2, y2) are given by

$$xm = (x1 + x2)/2 \qquad ym = (y1 + y2)/2$$

- Initially the line is tested for visibility.
- If line is completely visible it is drawn and if it is completely invisible it is rejected.
- If line is partially visible then it is subdivided in two equal parts.
- The visibility tests are then applied to each half.
- This subdivision process is repeated until we get completely visible and completely invisible line segments.

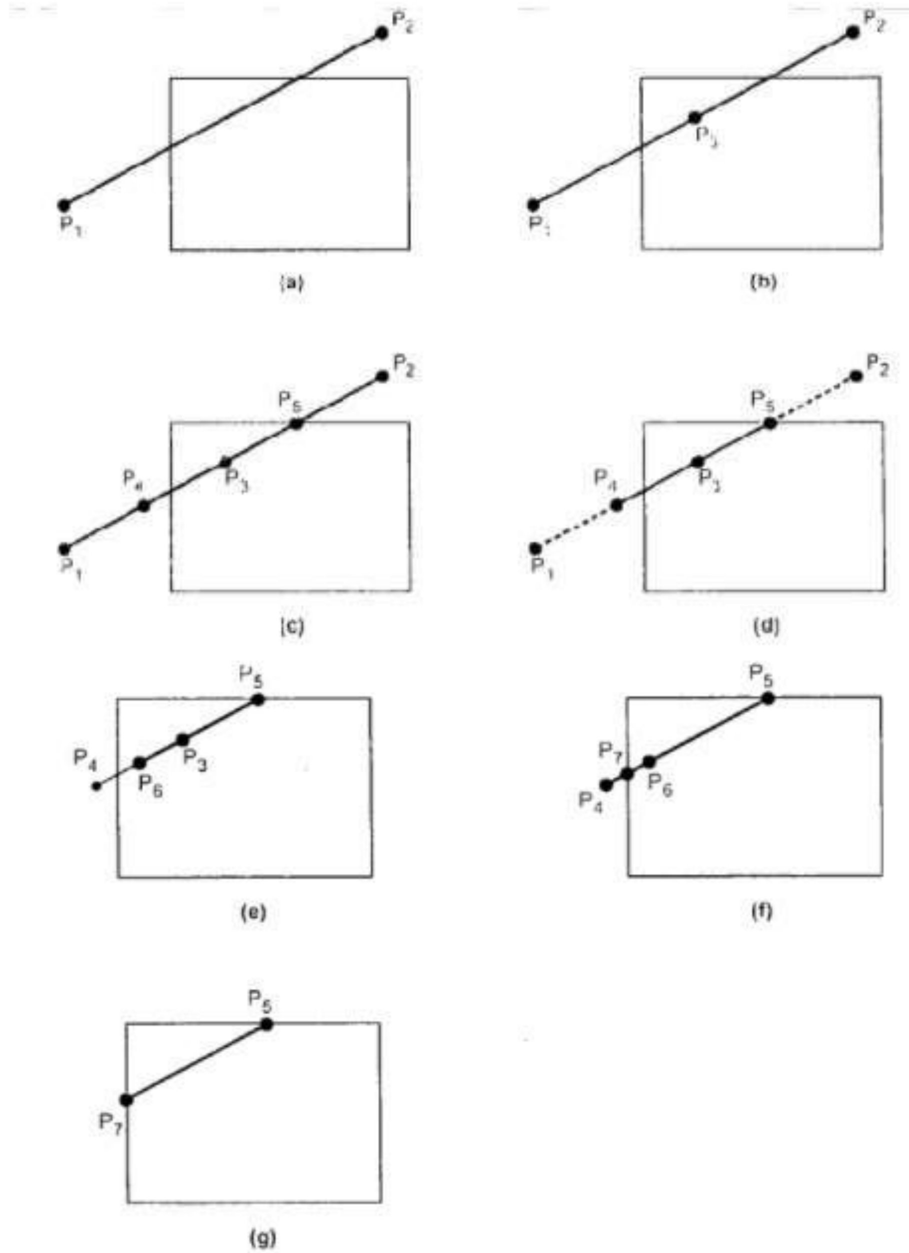


Fig. (k) Clipping line with midpoint subdivision algorithm

- As shown in the figure (k), line P1 P2 is partially visible.
- It is subdivided in two equal Parts P1 P3 and P3 P2 (see Fig. k (b)).
- Both the line segments are tested for visibility and found to be partially visible.
- Both line segments are then subdivided in two equal parts to get midpoints P4 and P5 (see Fig. k (c)).
- It is observed that line segments P1 P4 and P5 P2 are completely invisible and hence rejected.
- However, line segment P3 P5 is completely visible and hence drawn.
- The remaining line segment P4 P3 is still partially visible.
- It is then subdivided to get midpoint P6.

- It is observed that P6 P3 is completely visible whereas P4 P6 is partially visible.
- Thus P6 P3 line segment is drawn and P4 P6 line segment is further subdivided into equal parts to get midpoint P7.
- Now, it is observed that line segment P4 P7 is completely invisible and line segment P7 P6 is completely visible (see Fig. k (f)), and there is no further partially visible segment.

Midpoint Subdivision Algorithm :

1. Read two endpoints of the line say $P_1(x_1, y_1)$ and P_2 (x_2, y_2).
2. Read two corners (left-top and right-bottom) of the window, say $(Wx_1, Wy_1$ and $Wx_2, Wy_2)$.
3. Assign region codes for two end points using following steps :

   Initialize code with bits 0000

   Set Bit 1 - if $(x < Wx_1)$

   Set Bit 2 - if $(x > Wx_2)$

   Set Bit 3 - if $(y < Wy_1)$

   Set Bit 4 - if $(y > Wy_2)$

4. Check for visibility of line
   a) If region codes for both endpoints are zero then the line is completely visible. Hence draw the line and go to step 6.
   b) If region codes for endpoints are not zero and the logical ANDing of them is also nonzero then the line is completely invisible, so reject the line and go to step 6.
   c) If region codes for two endpoints do not satisfy the conditions in 4a) and 4b) the line is partially visible.
5. Divide the partially visible line segment in equal parts and repeat steps 3 through 5 for both subdivided line segments until you get completely visible and completely invisible line segments.
6. Stop.

Thank You