Computer Graphics

2D- VIEWING

Introduction

- Clipping refers to the removal of part of a scene. A technique which divides each element of the picture into visible & invisible portions, allowing the invisible portion to be discarded is called clipping.
- Internal clipping removes parts of a picture outside a given region & External clipping removes parts inside a region.
- We'll explore internal clipping, but external clipping can almost always be accomplished as a by-product.
- **Types of Clipping:** *Point* Clipping, *line* clipping and *polygon clipping.*
- A line clipping algorithms takes as input two endpoints of line segment and returns one (or more) line segments.
- A polygon clipper takes as input the vertices of a polygon and returns one (or more) polygons.
- The algorithms that perform the job of clipping are called clipping algorithms

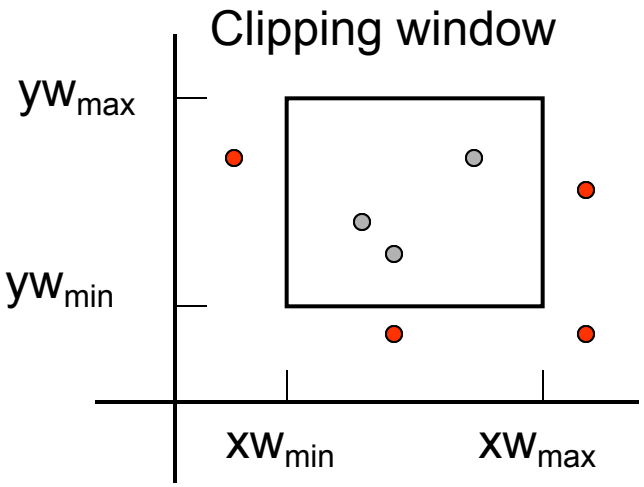
Types of Clipping

- Clipping algorithms are designed to efficiently identify the portions of scene (in viewing coordinates) that lie inside a given viewport.
- Point Clipping
- It 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.
- let us assume that the clip window is rectangular in shape.
- The maximum and minimum coordinate value are 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 satisfying the inequalities. Otherwise, the point will not be visible. Thus, the point will be clipped or not can be decided on the basis of the inequalities.

$$xw_{min} \quad x \quad xw_{max}$$

$$yw_{min} \quad y \quad yw_{max}$$

Point Clipping



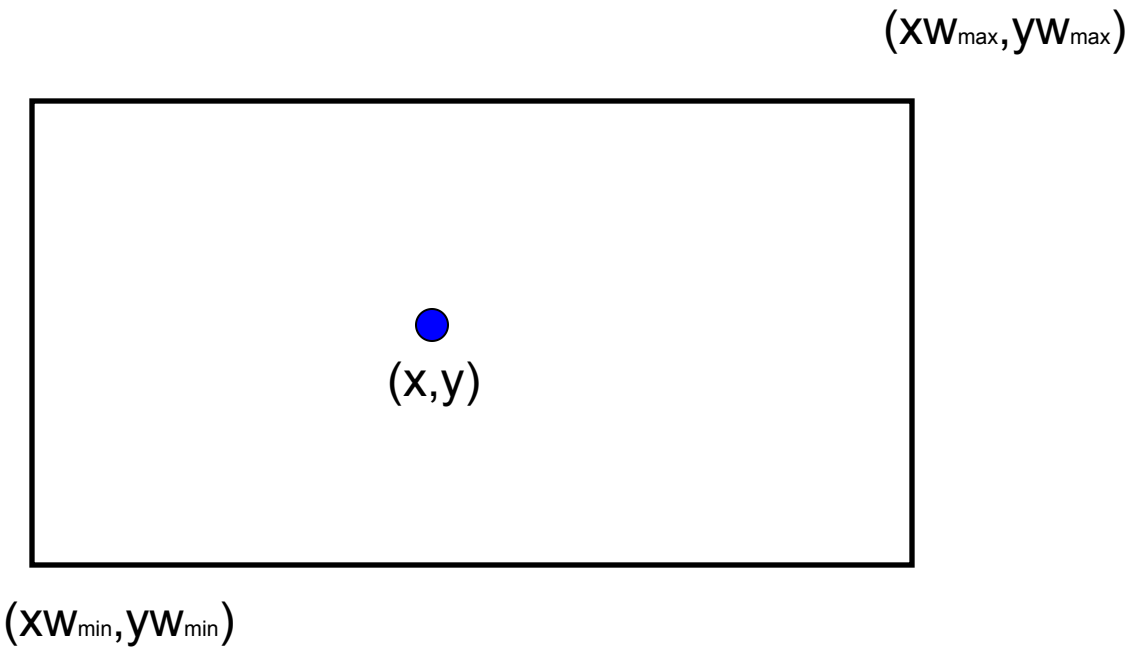
- Trivial: Save point **P** = (*x*,*y*) if it's in the box:

$$xw_{\min} \quad x \quad xw_{\max}$$

$$yw_{\min} \quad y \quad yw_{\max}$$

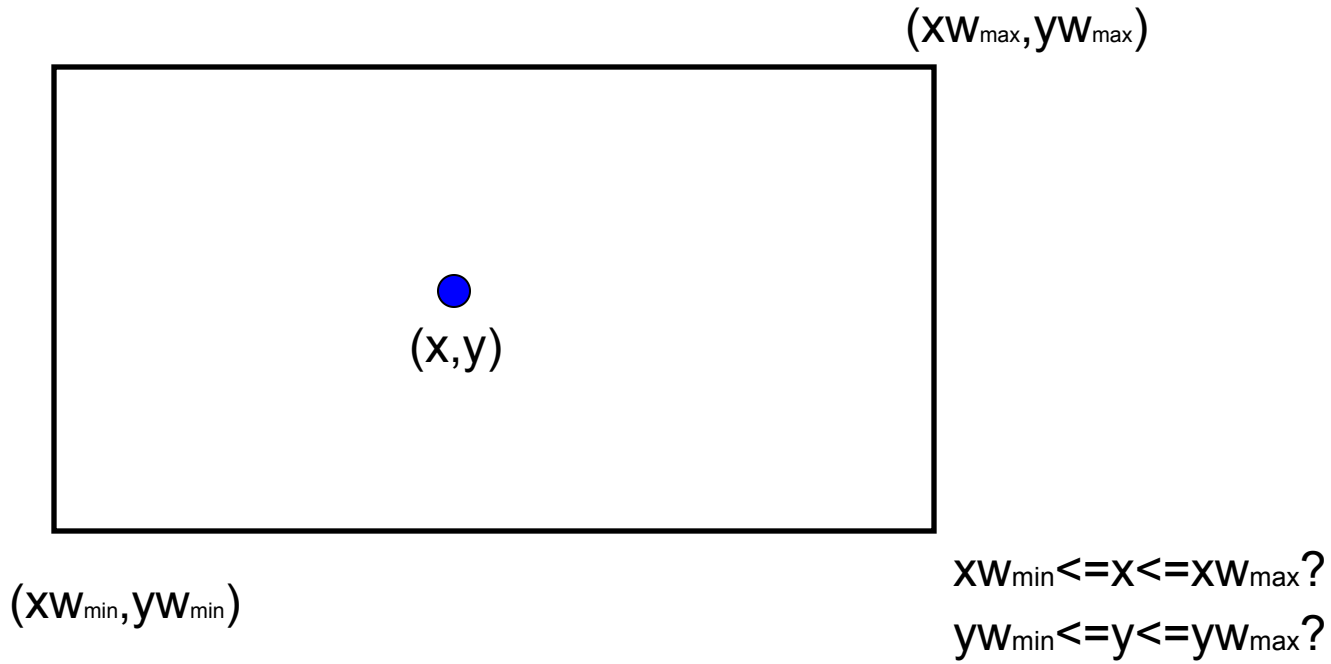
Clipping Points

- Given a point $(x, y)$and clipping window (xw_{min}, yw_{min}), (xw_{max}, yw_{max}), determine if the point should be drawn



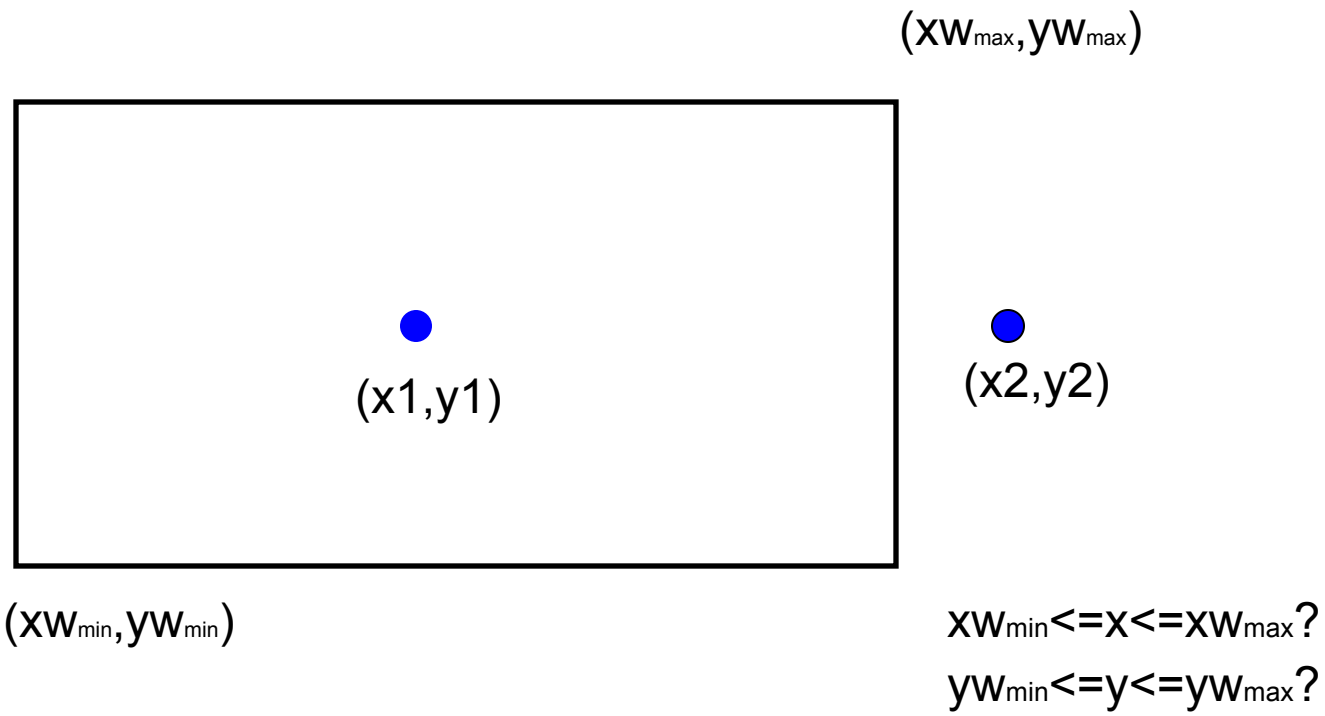
Clipping Points

- Given a point $(x, y)$and clipping window (xw_{min}, yw_{min}), (xw_{max}, yw_{max}), determine if the point should be drawn



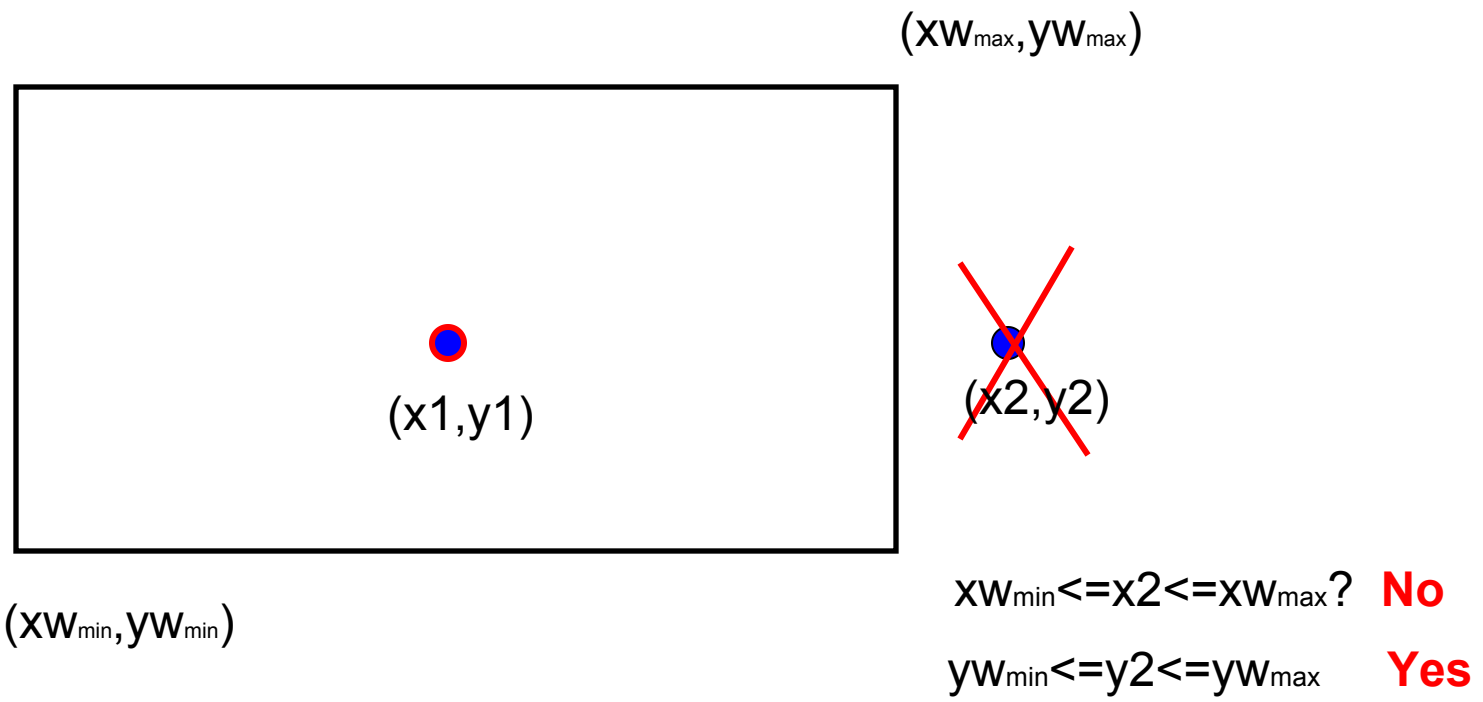
Clipping Points

- Given a point $(x, y)$and clipping window (xw_{min}, yw_{min}), (xw_{max}, yw_{max}), determine if the point should be drawn



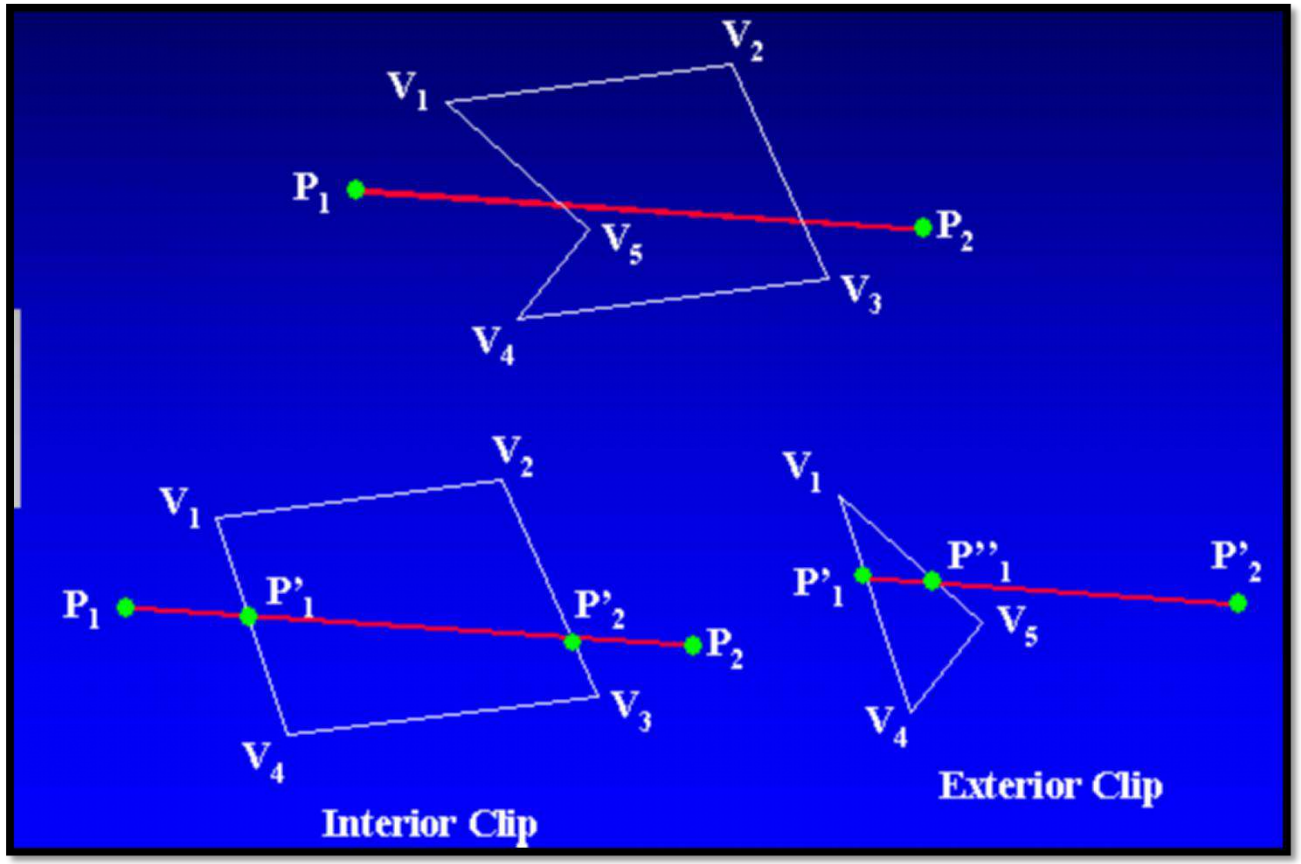
Clipping Points

- Given a point $(x, y)$and clipping window (xw_{min}, yw_{min}), (xw_{max}, yw_{max}), determine if the point should be drawn



Line Clipping

- Internal clipping removes parts of a picture outside a given region & External clipping removes parts inside a region.



Window and clipping

- The method for selecting and enlarging portions of a drawing is called windowing and the technique for not showing the part of the drawing which one is not interested is called clipping
- **Window**
  - A world-coordinate area selected for display. defines *what* is to be viewed
- **Viewport**
  - An area on a display device to which a window is mapped.

    defines *where* it is to be displayed
- **Viewing transformation**
  - The mapping of a part of a world-coordinate scene to device coordinates.
- **A window could be a rectangle to have any orientation.**
- 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.

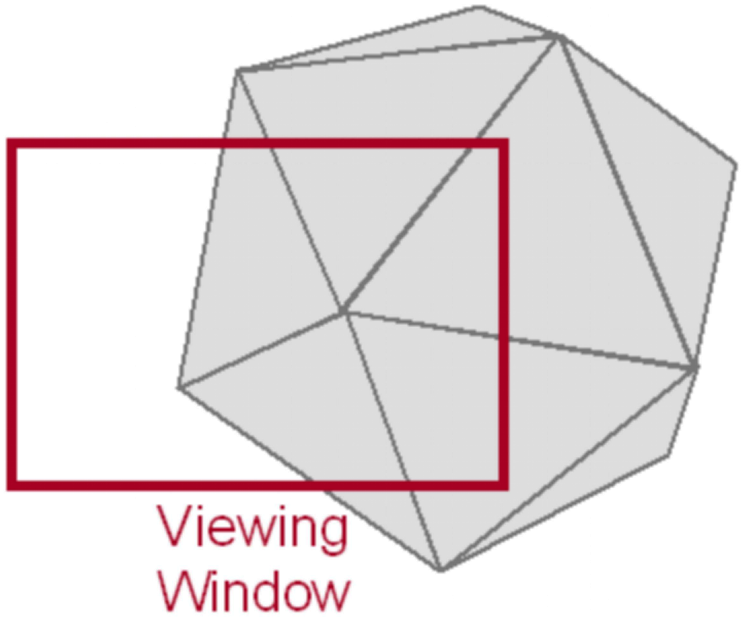
Clipping

- Avoid Drawing Parts of Primitives Outside Window
  - Window defines part of scene being viewed
  - Must draw geometric primitives only inside window

World Coordinates

Clipping

- Avoid Drawing Parts of Primitives Outside Window
  - Window defines part of scene being viewed
  - Must draw geometric primitives only inside window

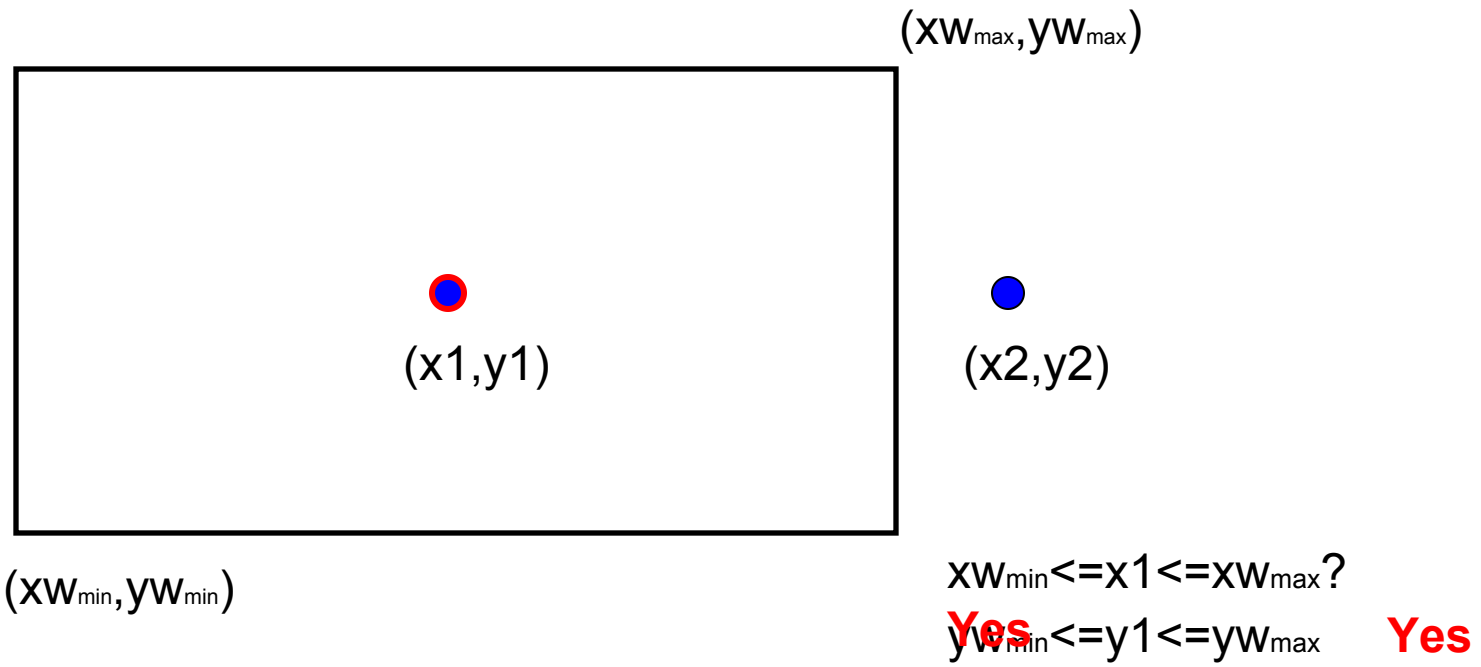


Viewing & Clipping

- When we define an image in some world coordinate system, to display that image we must map the image to the physical output device. This is a two stage process. For 3 dimensional images we must first determine the 3D camera viewpoint, called the View Reference Point (VRP) and orientation. Then we project from 3D to 2D, since our display device is 2 dimensional. Next, we must map the 2D representation to the physical device.
- **Example:** Graphic program which draw an entire building by an architect but we only interested on the ground floor. Map of sales for entire region but we only like to know from certain region of the country.

Clipping Points

- Given a point $(x, y)$and clipping window (xw_{min}, yw_{min}), (xw_{max}, yw_{max}), determine if the point should be drawn



Line Clipping

- 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:

1. Cohen Sutherland algorithm
2. Mid Point Subdivision Algorithm
3. Liang-Barsky Algorithm

To clip a line, we need to consider only its endpoints, not its infinitely many interior points.

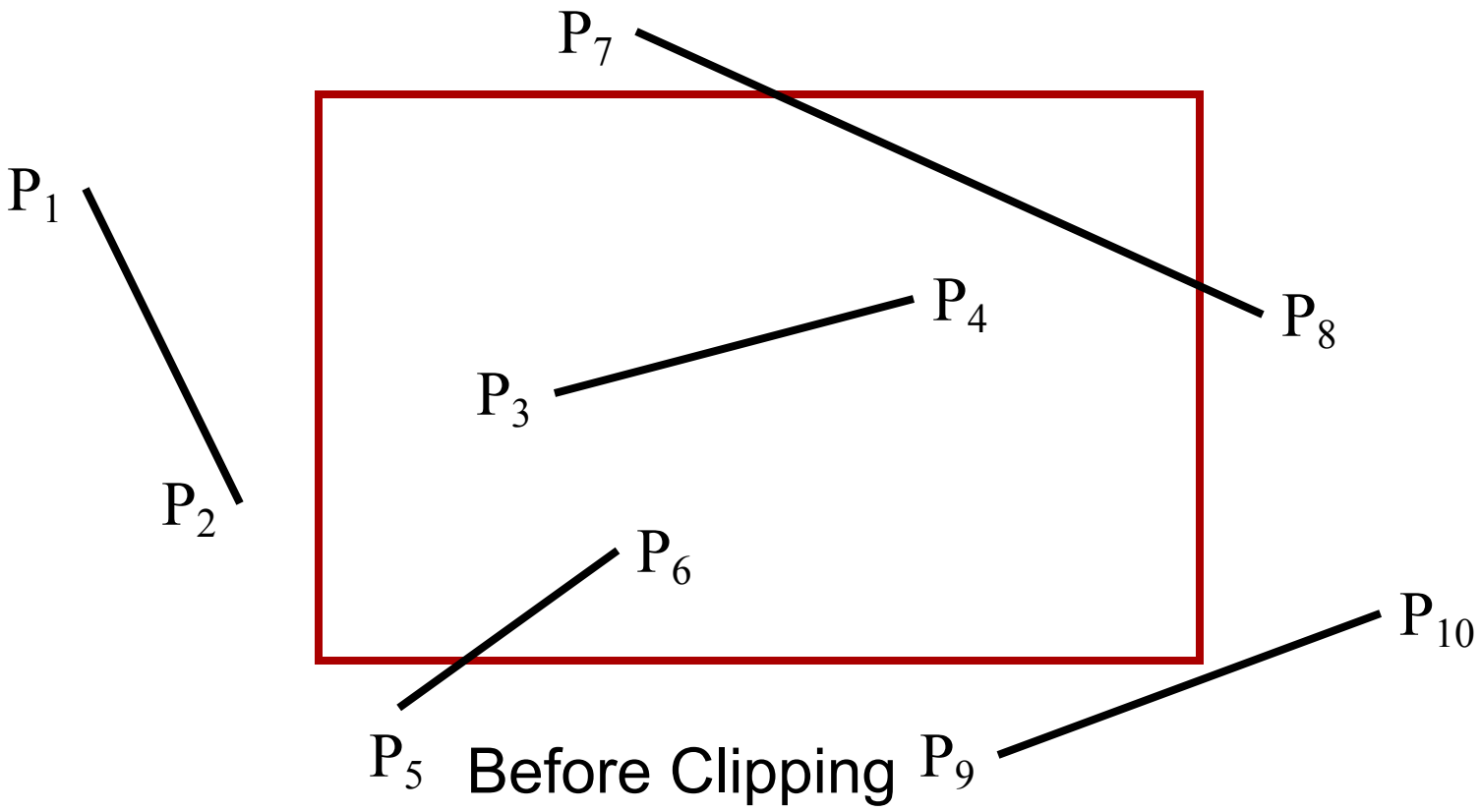
a) If both endpoints of a line lie inside the clip rectangle, the entire line lies inside the clip rectangle and can be trivially accepted.

b) If one endpoint lies inside and one outside, the line intersects the clip rectangle and we must compute the intersection point.

c) If both endpoints are outside the clip rectangle, the line may or may not intersect with the clip rectangle, and we need to perform further calculations to determine whether there are any intersections.

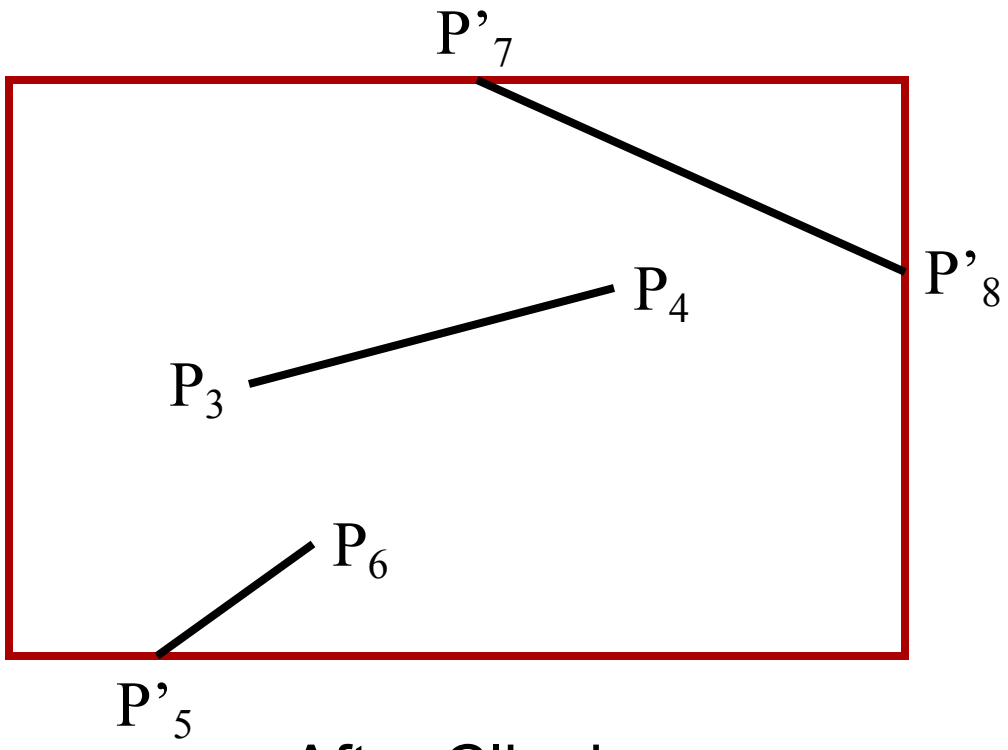
Line Clipping

- Find the Part of a Line Inside the Clip Window



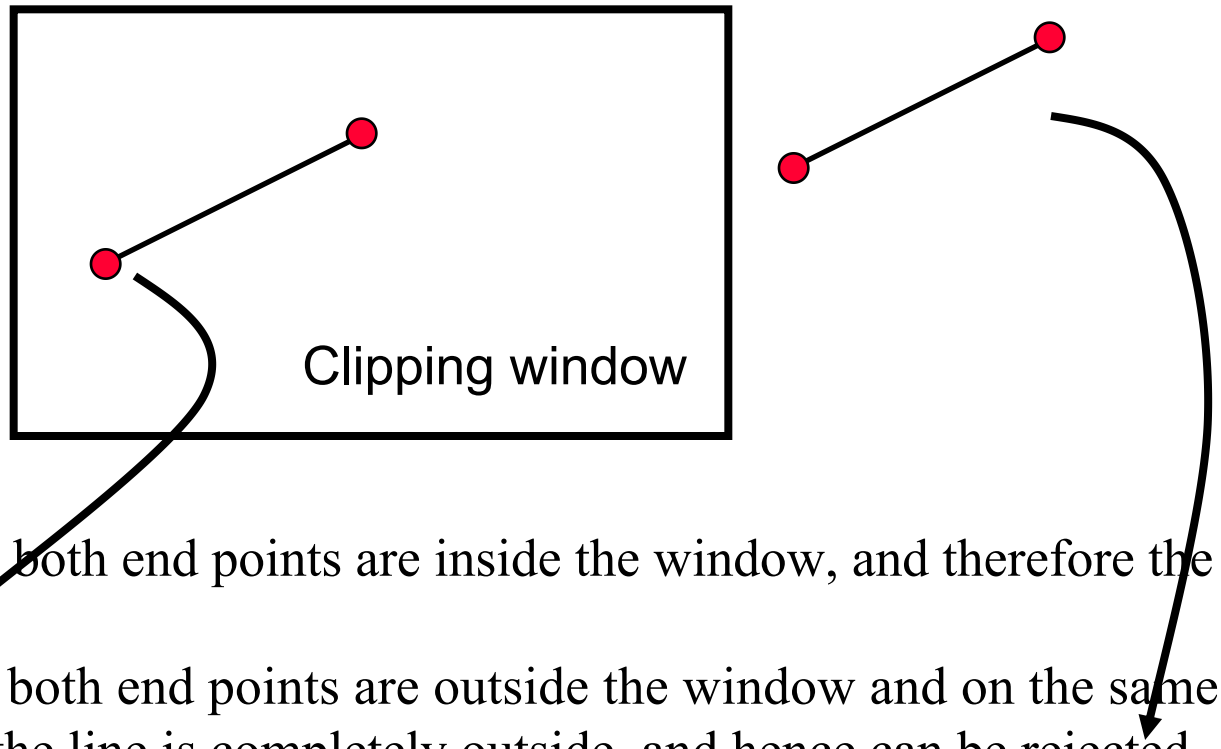
Line Clipping

- Find the Part of a Line Inside the Clip Window



Cohen-Sutherland Algorithm

A rapid divide-and-conquer approach to the line clipping

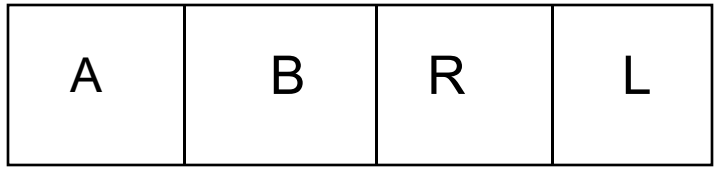


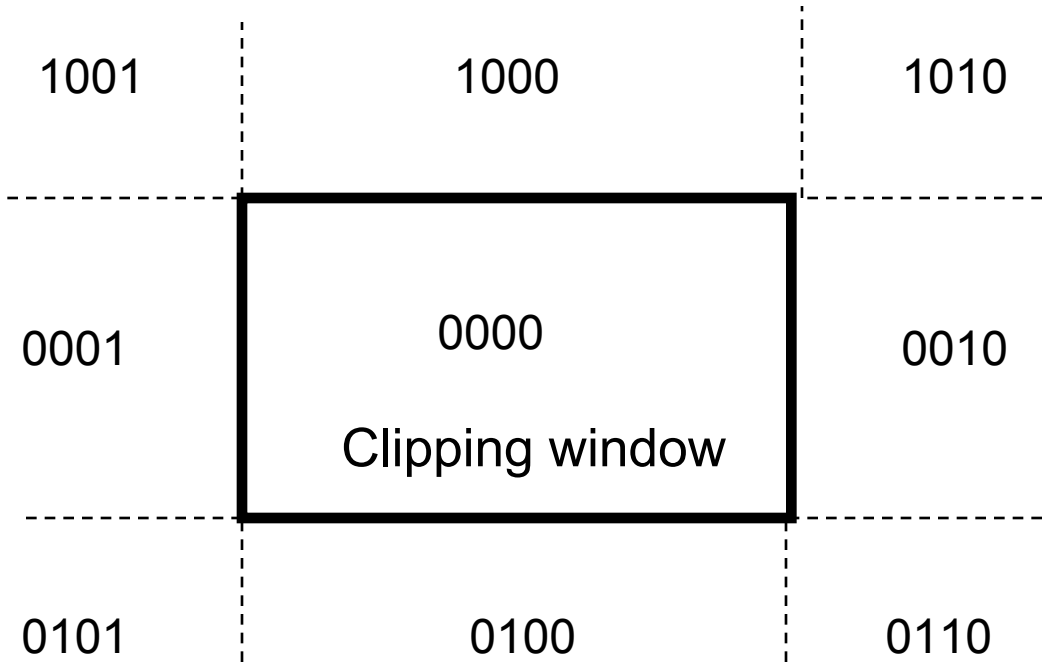
***Trivial Accept*:** When both end points are inside the window, and therefore the line is completely visible.

***Trivial Reject*:** When both end points are outside the window and on the same side of the window. Then the line is completely outside, and hence can be rejected.

Region Codes

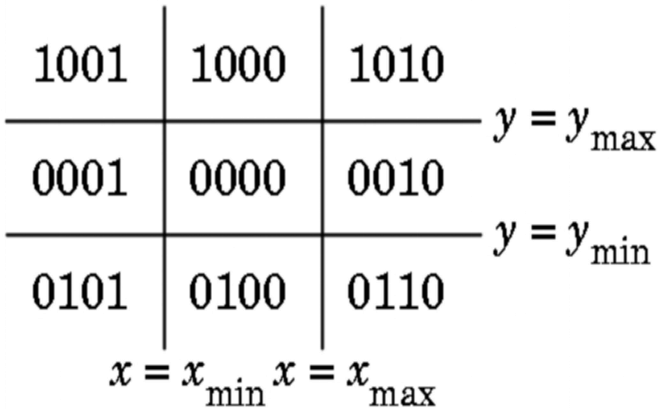
A point is assigned a unique region code depending on the location of the point with respect to the window.





Cohen-Sutherland Line Clipping

- Extend the window's sides to infinity and break up space into 9 regions.
- Each region is assigned a 4-bit binary out code, $b_1b_2b_3b_4$, as follows.



$$b_1 = \begin{cases} 1 & \text{if } y > y_{max} \\ 0 & \text{otherwise} \end{cases} \qquad b_2 = \begin{cases} 1 & \text{if } y < y_{min} \\ 0 & \text{otherwise} \end{cases}$$

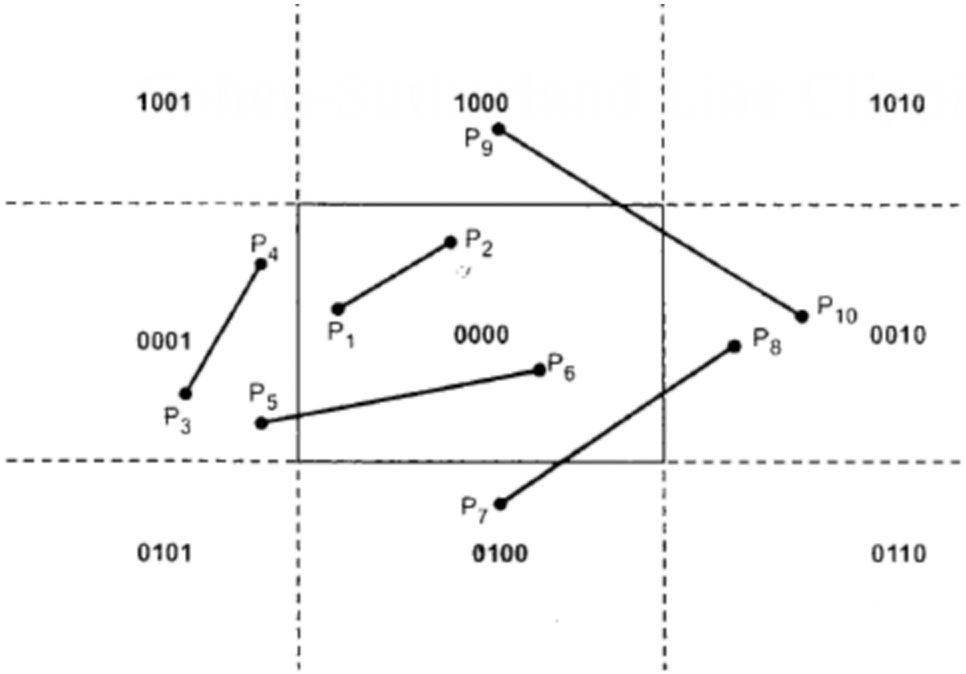
$$b_3 = \begin{cases} 1 & \text{if } x > x_{max} \\ 0 & \text{otherwise} \end{cases} \qquad b_4 = \begin{cases} 1 & \text{if } x < x_{min} \\ 0 & \text{otherwise} \end{cases}$$

- Bit 1: End point is above the window if $y > y_{max}$
- Bit 2: End point is below the window if $y < y_{min}$
- Bit 3: End point is right of the window if $x > x_{max}$
- Bit 4: End point is left to the window if $x < x_{min}$

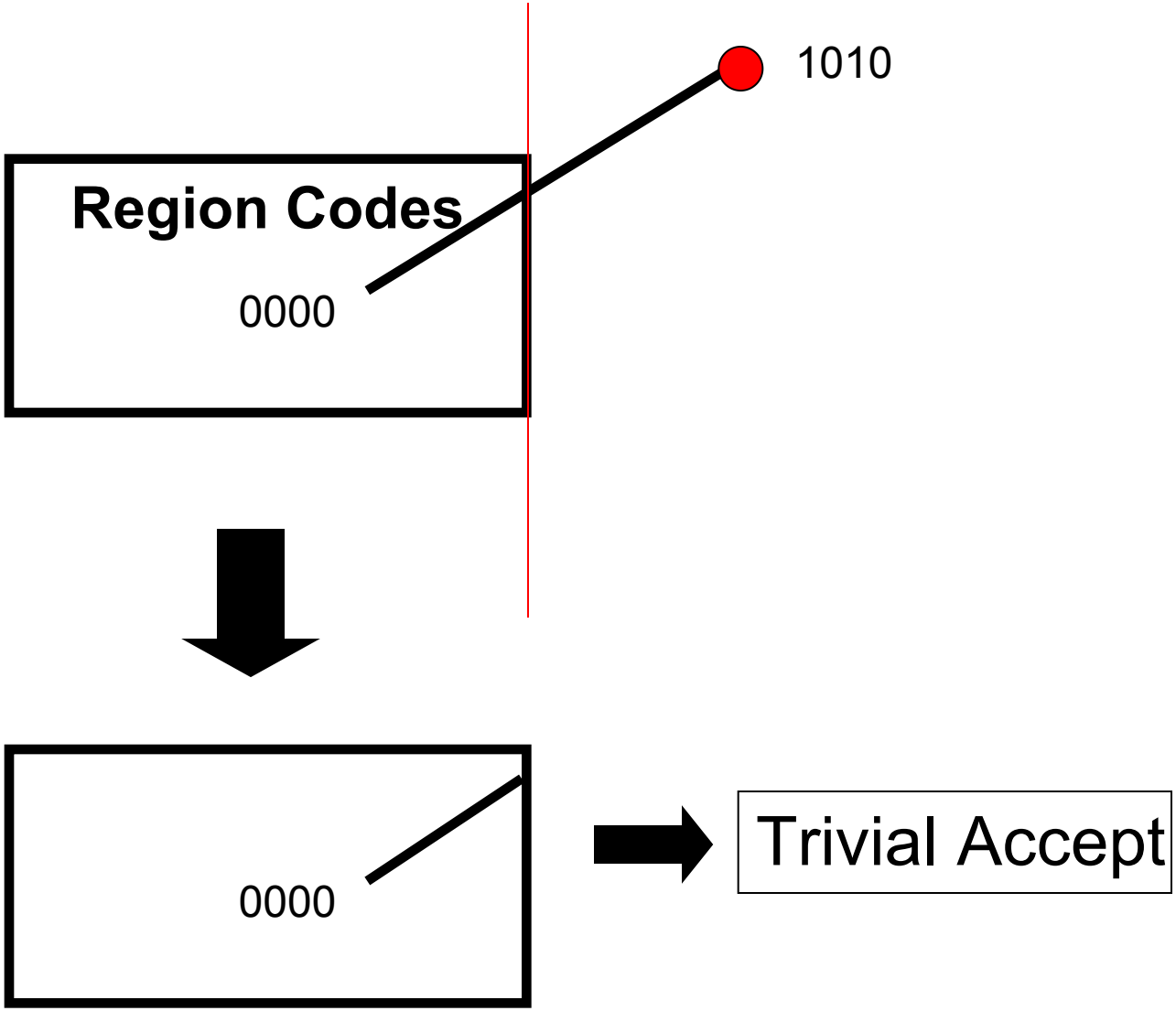
Cohen-Sutherland Line Clipping

- 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 window region; we use the tool of logical AND between the region codes of the points lying on the line.

1. If both the end point codes are 0000,the line is visible.
2. If the logical AND of the endpoint codes is not 0000,the line segment is not visible.
3. If the logical AND of the endpoint codes is 0000,the line segment is clipping candidate.



Line	End Point Codes		Logical ANDing	Result
$P_1 P_2$	0000	0000	0000	Completely visible
$P_3 P_4$	0001	0001	0001	Completely invisible
$P_5 P_6$	0001	0000	0000	Partially visible
$P_7 P_8$	0100	0010	0000	Partially visible
$P_9 P_{10}$	1000	0010	0000	Partially visible

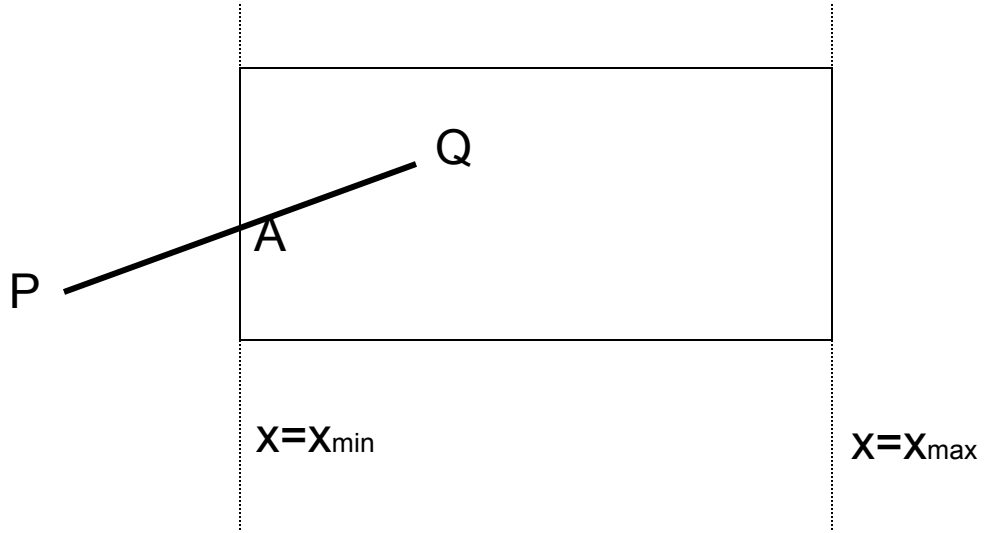

1010
Region Codes
0000
0000
Trivial Accept

Cohen-Sutherland Line Clipping

- Computing Intersection Point
- Intersection points with a clipping boundary can be calculated using the slope- intercept form of the line equation.
- The equation of any line with slope m is $y=mx+c$
- If it passes through (x_1,y_1) then $y_1=mx_1+c$
- Subtracting $y-y_1=m(x-x_1)$ where $m=\frac{y_2 \quad y_1}{x_2 \quad x_1}$ (Line is passing through (x_1,y_1) & (x_2,y_2)

- Determine the intersecting boundary
- If bit1 is 1,intersect with line $y=y_{max}$
- If bit2 is 1,intersect with line $y=y_{min}$
- If bit3 is 1,intersect with line $x=x_{max}$
- If bit4 is 1,intersect with line $x=x_{min}$

Cohen-Sutherland Line Intersection & Clipping



The equation of the line PQ is $\frac{y - y_1}{y_2 - y_1} = \frac{x - x_1}{x_2 - x_1}$

The intersection point A lies on the left edge and therefore its x-coordinate is $x = x_{min}$.

To get the y-coordinate of A, we substitute for x in the above equation of PQ:

$y = y_1 + m(x - x_1)$ where $m = \frac{y_2 - y_1}{x_2 - x_1}$

Similarly,

$x = x_1 + \frac{1}{m}(y - y_1)$ where $m = \frac{y_2 - y_1}{x_2 - x_1}$

Cohen-Sutherland Line Intersection & Clipping

- The coordinates of the intersection point are:
- If boundary line is vertical

$$y = y_1 \quad m(x \quad x_1)$$

- Where x value depends

a) The intersection point lies on the left edge and therefore its x-coordinate is $x = x_{min}$.

b) The intersection point lies on the Right edge and therefore its x-coordinate is $x = x_{max}$.

- If the boundary line is horizontal

$$x = x_1 \quad \frac{1}{m} \quad y \quad y_1$$

- Where y value depends

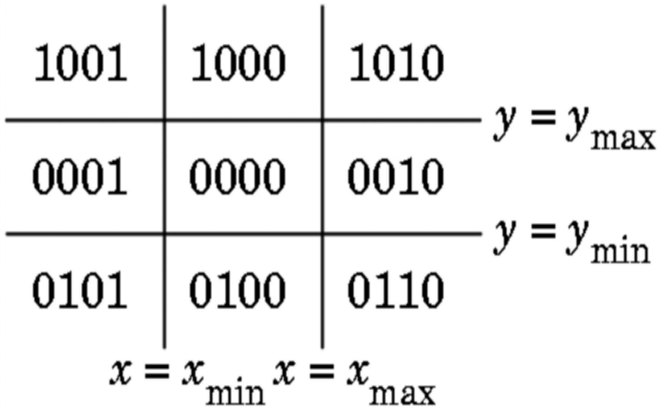
a) The intersection point lies on the below edge and therefore its y-coordinate is $y = y_{min}$.

b) The intersection point lies on the Above edge and therefore its y-coordinate is $y = y_{max}$.

Cohen-Sutherland Line Clipping

- Algorithm

1. Accept the end points of the line segment and window boundaries i.e.$x_1, y_1, x_2, y_2, x_{min}, x_{max}, y_{min}, y_{max}$.
2. Assign a 4 bit code to each end point of the line segment i.e. $b_1 b_2 b_3 b_4$, as follows



$$b_1 = \begin{cases} 1 & \text{if } y > y_{max} \\ 0 & \text{otherwise} \end{cases} \qquad b_2 = \begin{cases} 1 & \text{if } y < y_{min} \\ 0 & \text{otherwise} \end{cases}$$

$$b_3 = \begin{cases} 1 & \text{if } x > x_{max} \\ 0 & \text{otherwise} \end{cases} \qquad b_4 = \begin{cases} 1 & \text{if } x < x_{min} \\ 0 & \text{otherwise} \end{cases}$$

Cohen-Sutherland Line Clipping

3. Test using end point codes & Logical AND

a) If both the end point codes are 0000,the line is visible.

b) If the logical AND of the endpoint codes is not 0000,the line segment is not visible.

c) If the logical AND of the endpoint codes is 0000,the line segment is clipping candidate.

4. Determine the intersecting boundary

a) If bit1 is 1,intersect with line $y=y_{max}$

b) If bit2 is 1,intersect with line $y=y_{min}$

c) If bit3 is 1,intersect with line $x=x_{max}$

d) If bit4 is 1,intersect with line $x=x_{min}$

Cohen-Sutherland Line Clipping

5. Determine the intersecting point coordinates (x,y).The equation of a line passing through (x_1,y_1) and (x_2,y_2) and (x,y) will be

$$\frac{y - y_1}{y_2 - y_1} \quad \frac{x - x_1}{x_2 - x_1}$$

- If boundary line is vertical
- Where x value depends

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

a) The intersection point lies on the left edge and therefore its x-coordinate is $x = x_{min}$.

b) The intersection point lies on the Right edge and therefore its x-coordinate is $x = x_{max}$

- If the boundary line is horizontal
- Where y value depends

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

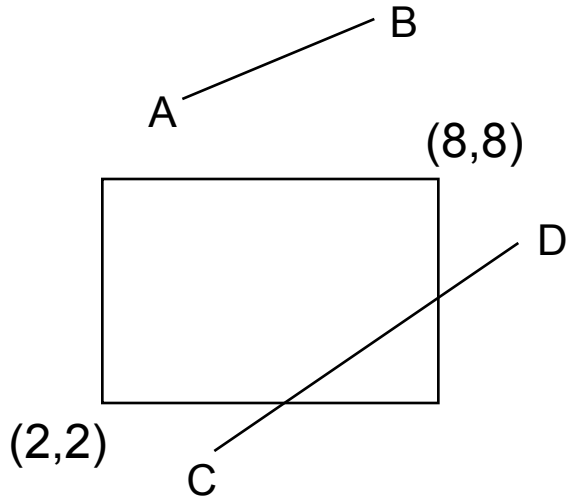
a) The intersection point lies on the below edge and therefore its y-coordinate is $y = y_{min}$.

b) The intersection point lies on the Above edge and therefore its y-coordinate is $y = y_{max}$.

6. Go to Step 2.
7. Draw the remaining line segments .
8. Stop.

Cohen-Sutherland Line Clipping

Example1



x_{min} 2, y_{min} 2

x_{max} 8, y_{max} 8

Line AB : A(3,10), B(6,12)
CD : C(4,1), D(10,6)

Cohen-Sutherland Line Clipping

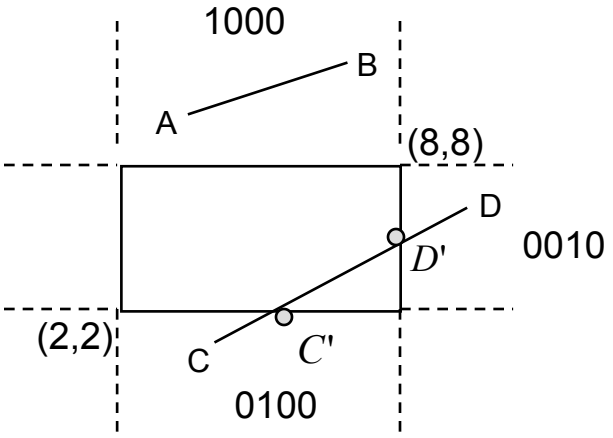
Encode end points

AB	A(3,10)	1000
	B(6,12)	1000
	Logical AND	1000

Invisible

CD	C(4, 1)	0100
	D(10,6)	0010
	Logical AND	0000

Indeterminate



Clip line CD

$y \quad y_{min} \quad 2 \qquad (\because C : 0100)$

$x_1 \quad 4$

C':

$$x \quad x_1 \quad \frac{1}{m} \; y_{min} \quad y_1$$

$$x \quad x_1 \quad \frac{x_2 \quad x_1}{y_2 \quad y_1} \; y_{min} \quad y_1$$

$$x \quad 4 \quad \frac{10 \quad 4}{6 \quad 1} \; 2 \quad 1$$

$$x \quad 5.2$$

$$C'(5.2, \quad 2)$$

$x \quad x_{max} \quad 8 \qquad (\because D : 0010)$

$y_1 \quad 1$

D':

$$y \quad y_1 \quad m(x_{max} \quad x_1)$$

$$y \quad y_1 \quad \frac{y_2 \quad y_1}{x_2 \quad x_1}(x_{max} \quad x_1)$$

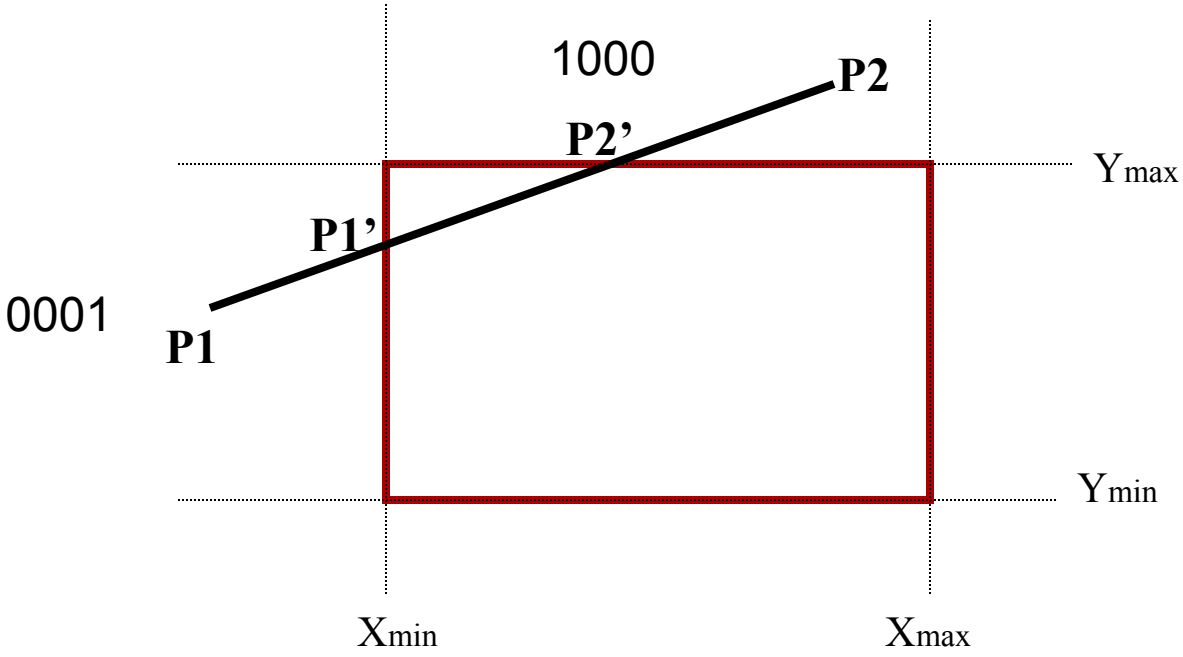
$$y \quad 1 \quad \frac{6 \quad 1}{10 \quad 4}(8 \quad 4)$$

$$y \quad 4.33$$

$$D'(8, \quad 4.33)$$

Cohen-Sutherland Line Clipping

- Example2
- A line from (2, 7) to (8, 12) in a window (Xmin = Ymin = 5 and Xmax = Ymax = 10)



Cohen-Sutherland Line Clipping

- The region code of point P1(2,7)=0001
- The region code of point P2(8,12)=1000
- The line is partially visible.(logical AND is 0000)
- The point of intersection of point P1 with the left boundary is =(x_{min},y)
- The point of intersection of the point P2 with the top boundary is (x,y_{max})

$$x = x_{min} = 5$$

$$y_1 = 7$$

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

$$y = y_1 + \frac{y_2 - y_1}{x_2 - x_1}(x_{min} - x_1)$$

$$y = 7 + \frac{12 - 7}{8 - 2}(5 - 2)$$

$$y = 9.5$$

$$P1'(5,9.5)$$

$$y = y_{max} = 10$$

$$x_1 = 2$$

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

$$x = x_1 + \frac{x_2 - x_1}{y_2 - y_1}(y_{max} - y_1)$$

$$x = 2 + \frac{8 - 2}{12 - 7}(10 - 7)$$

$$x = 5.6$$

$$P2'(5.6,10)$$

So the finally visible portion of the line would be between (5,9.5) and (5.6,10).

Cohen-Sutherland Line Clipping

- Intersection calculations are expensive. Find first lines completely inside or certainly outside clipping window. Apply intersection only to undecided lines.
- Perform cheaper tests before proceeding to expensive intersection calculations.
- Completely inside / certainly outside tests involve only logic operations of bits.
- Can be easily extended to 3D clipping using 6-bit out codes .
- Floating-point arithmetic required
  - Requires only floating-point subtractions and Boolean operations for decision test.
  - A single division is required for intersection computation.
- The algorithm is only applicable to rectangular windows and not to any other convex shaped window.

Liang-Barsky Line Clipping

- Basic Idea: Using the parametric representation of the line, we solve for the parameter to see if and where the line intersects the window boundaries.

- Parametric line from $(x_1,y_1) \rightarrow (x_2,y_2)$:

$$x = x_1 + u\,\Delta x \qquad \Delta x = x_2 - x_1$$
$$y = y_1 + u\,\Delta y \qquad \Delta y = y_2 - y_1$$
$$u \in [0,1]$$

Liang-Barsky Line Clipping

- Point Clipping in parameter form:

$$xw_{min} \le x_1 + u\,\Delta x \le xw_{max}$$
$$yw_{min} \le y_1 + u\,\Delta y \le yw_{max}$$

- Solve for u and obtain equivalently

$up_k \le q_k \qquad k = 1,2,3,4$

k = 1 (left): $p_1 = -\Delta x \quad q_1 = x_1 - xw_{min}$

k = 2 (right): $p_2 = \Delta x \quad q_2 = xw_{max} - x_1$

k = 3 (bottom): $p_3 = -\Delta y \quad q_3 = y_1 - yw_{min}$

k = 4 (top): $p_4 = \Delta y \quad q_4 = yw_{max} - y_1$

Liang-Barsky Line Clipping

Strategy:

1. If $p_k = 0$ for some k then the line is parallel to a clipping boundary. Now test q_k:

   If one $q_k < 0$ for these k then line is outside

   If all $q_k \geq 0$ then line is inside

2. For all $p_k < 0$ calculate $u_1 = \max(0, \{q_k/p_k\})$ to determine intersection point with the possibly extended clipping boundary k and obtain a new starting point for the line at u_1.

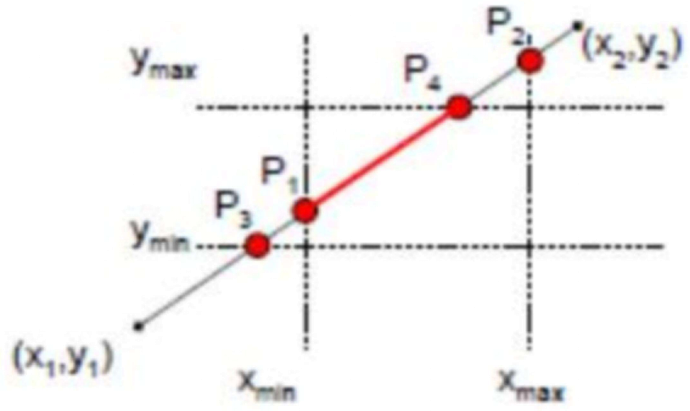
3. For all $p_k > 0$ calculate $u_2 = \min(1, \{q_k/p_k\})$ to determine intersection points with extended clipping boundary k and obtain a new end point at u_2.

4. If $u_1 > u_2$ then discard the line

5. The line is now between $[u_1,u_2]$

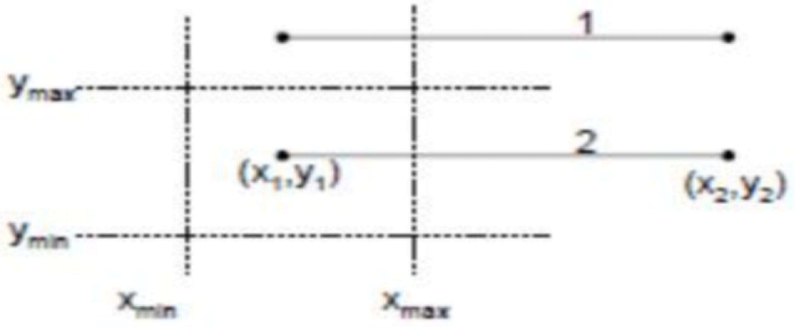
Line Clipping: Liang-Barsky

- Example 1: $p_1 < 0$, $p_2 > 0$, $p_3 < 0$, $p_4 > 0$. For k=1,3 calculate q_1/p_1 (P_1) and q_3/p_3 (P_3) and choose $u_1 = \max(0, q_1/p_1, q_3/p_3) = q_1/p_1$
- Likewise for k=2,4 calculate q_2/p_2 (P_2) and q_4/p_4 (P_4) and choose $u_2 = \min(1, q_2/p_2, q_4/p_4) = q_4/p_4$



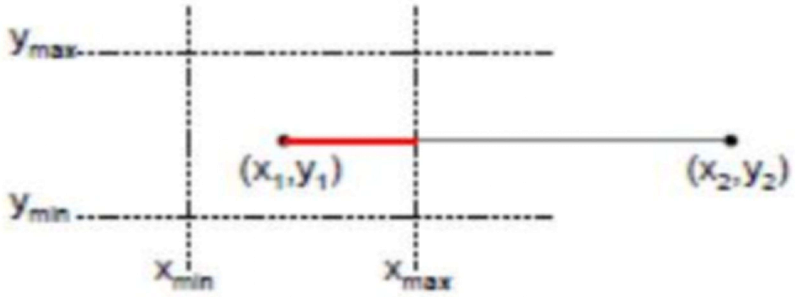
Line Clipping: Liang-Barsky

- Example 2: Consider horizontal lines with $\Delta y = 0$, $p_3 = p_4 = 0$.
- For line 1, $q_4 < 0$ and will be discarded.
- For line 2, p1 < 0, p2 > 0, $q_3 > 0$ and $q_4 > 0$. We proceed to calculate u_1 and u_2.



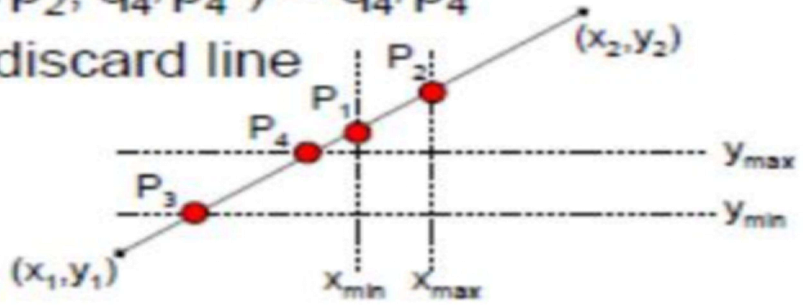
Line Clipping: Liang-Barsky

- Example 2: $p_1 < 0$, note that $q_1 > 0$ hence $q_1/p_1 < 0$, and $u_1 = \max\{0, q_1/p_1\} = 0$
- $p_2 > 0$, calculate q_2/p_2 and choose $u_2 = \min(q_2/p_2, 1) = q_2/p_2$.
- $u_1 < u_2$ and the line is between $[u_1, u_2]$



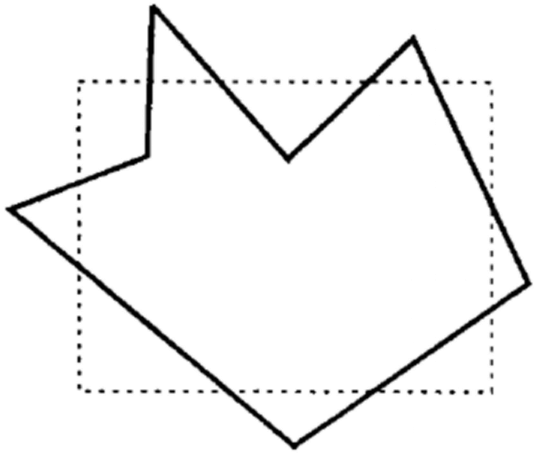
Line Clipping: Liang-Barsky

- Example 3:
  - $p_1 < 0$, $p_2 > 0$, $p_3 < 0$ and $p_4 > 0$.
  - calculate q_1/p_1 (P_1) and q_3/p_3 (P_3) and choose $u_1 = \max(0, q_1/p_1, q_3/p_3) = q_1/p_1$
  - calculate q_2/p_2 (P_2) and q_4/p_4 (P_4) and choose $u_2 = \min(1, q_2/p_2, q_4/p_4) = q_4/p_4$
  - $u_1 > u_2$ hence discard line

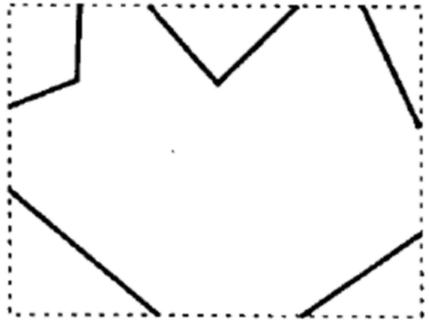


Polygon Clipping

- Polygon is a surface enclosed by several lines.
- Thus, Line clipping algorithm can be used directly for polygon clipping. But, 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.
- Polygon clipping done by line clipping algorithm



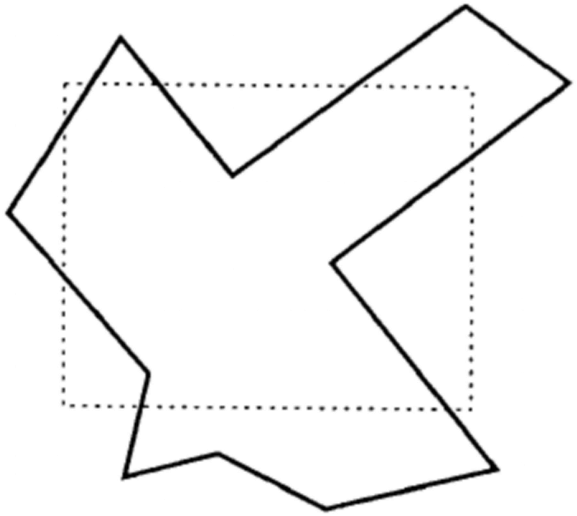
(a) Before clipping



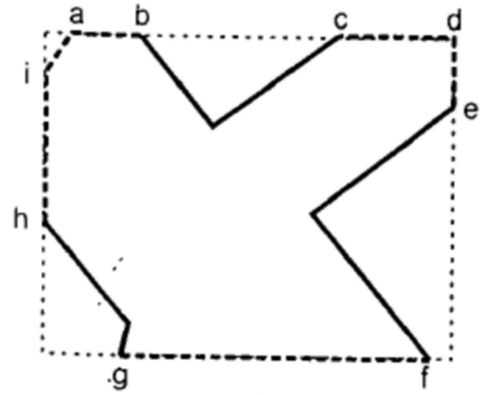
(b) After clipping

Polygon Clipping

- A polygon is considered as a closed solid area. Hence, after clipping it should remain closed. To achieve this, there is need an algorithm that will generate additional line segment which make the polygon as a closed area. e.g. the lines a----b, c----d, d----e, f-----g and h-----i are added to polygon description to make it closed.



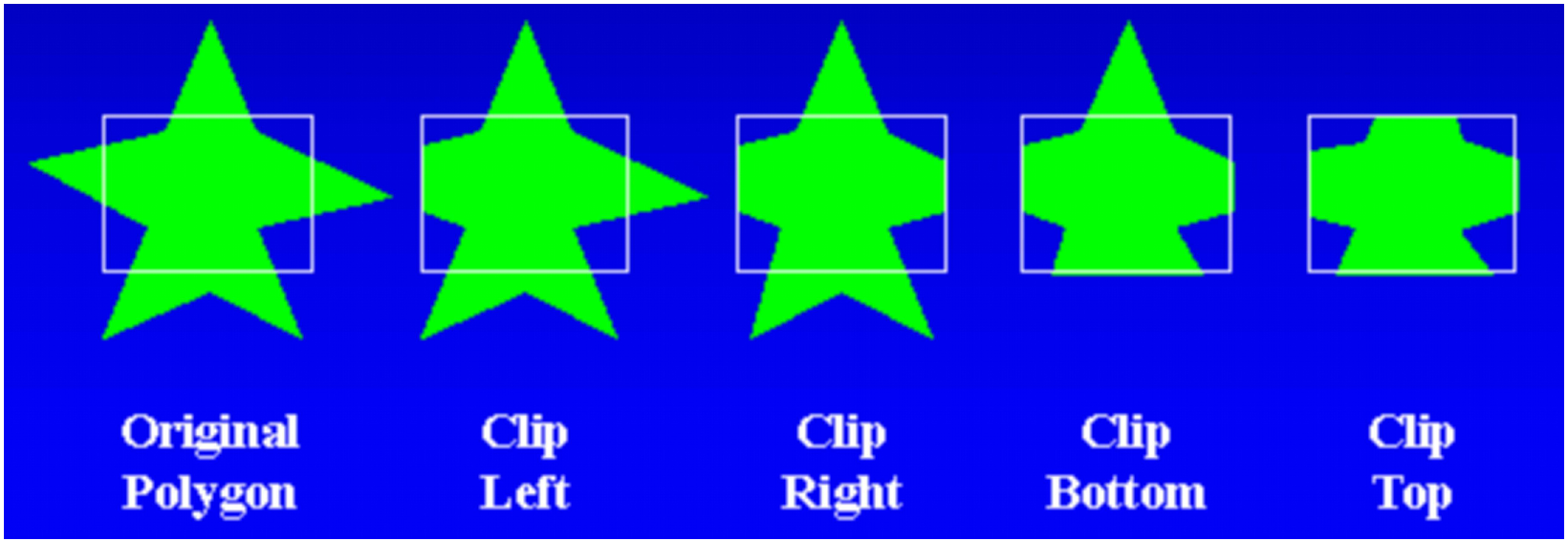
(a)



(b)

Polygon Clipping
Sutherland-Hodgman Algorithm

- 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, one window boundary at a time
- It Accepts a series of polygon vertices and output another series of vertices defining the clipped polygon.



Polygon Clipping
Sutherland-Hodgman Algorithm

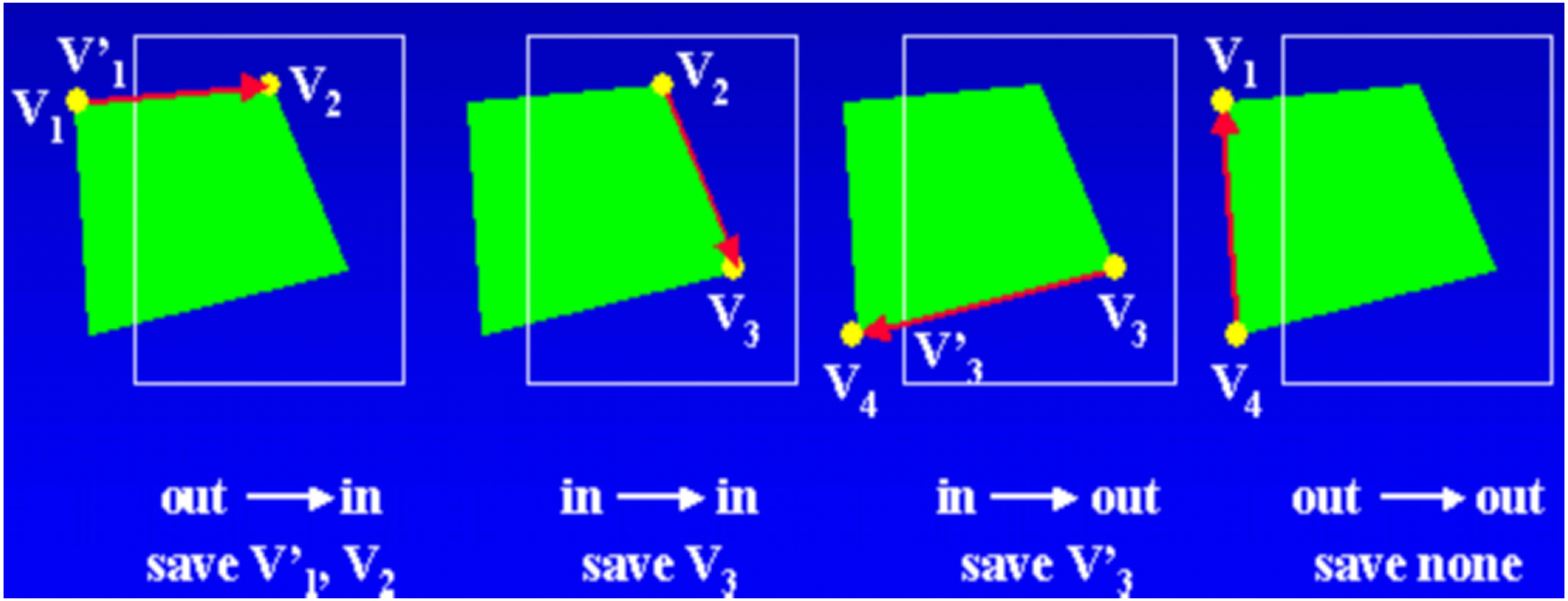
- The cases of the Sutherland-hodgman polygon clipping algorithm.
- In words, the 4 possible Tests are applied to clip any polygon states are as mentioned below:

1. If the 1st vertex is outside the window boundary 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.
2. If both Input vertices are inside the window boundary then only 2nd vertex is added to output vertex list.
3. 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.
4. If both Input vertices are outside the window boundary then nothing is added to the output list.

- Once all vertices have been processed for one clip window boundary, the output list of vertices is clipped against the next window boundary.

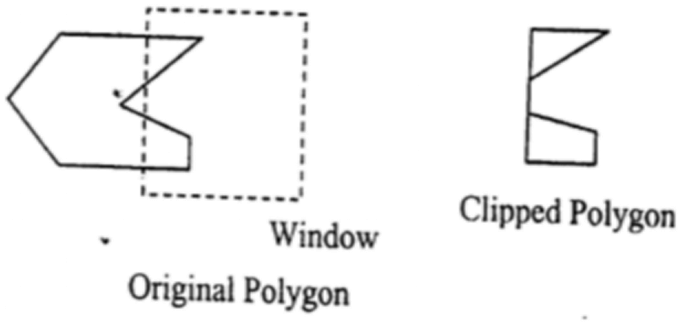
Polygon Clipping
Sutherland-Hodgman Algorithm

- Pass each pair of adjacent polygon vertices to a window boundary clipper
  - There are four cases:



Polygon Clipping
Sutherland-Hodgman Algorithm

- Convex polygons are correctly clipped by the Sutherland-Hodgeman algorithm, but concave polygons may be displayed with extraneous lines. This occurs when the clipped polygon should have two or more separate sections. But since there is only one output vertex list, the last vertex in the list is always joined to the first vertex.
- There are several things we could do to correctly display concave polygons. For one, we could split the concave polygon into two or more convex polygons and process each convex polygon separately.
- Another possibility is to modify the Sutherland-Hodgeman approach to check the final vertex list for multiple vertex points along any clip window boundary and correctly join pairs of vertices.



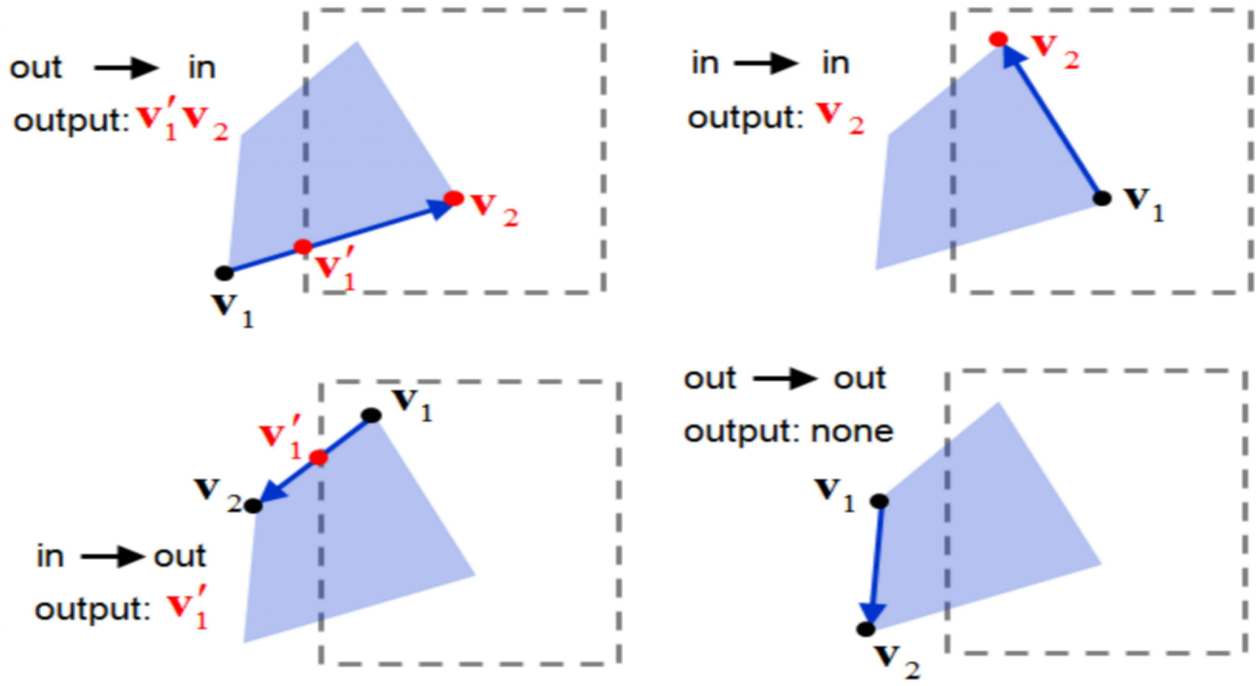
Polygon Clipping
Sutherland-Hodgman Algorithm

- Sutherland-Hodgman Algorithm

1. Read coordinates of all vertices of the polygon
2. Read coordinates of the clipping window.
3. Consider the left edge of the window.
4. Compare the vertices of each edge of the polygon, individually with the clipping plane.
5. Save the resulting intersections and vertices in the new list of vertices according to four possible relationships between the edge and the clipping boundary.

Polygon Clipping

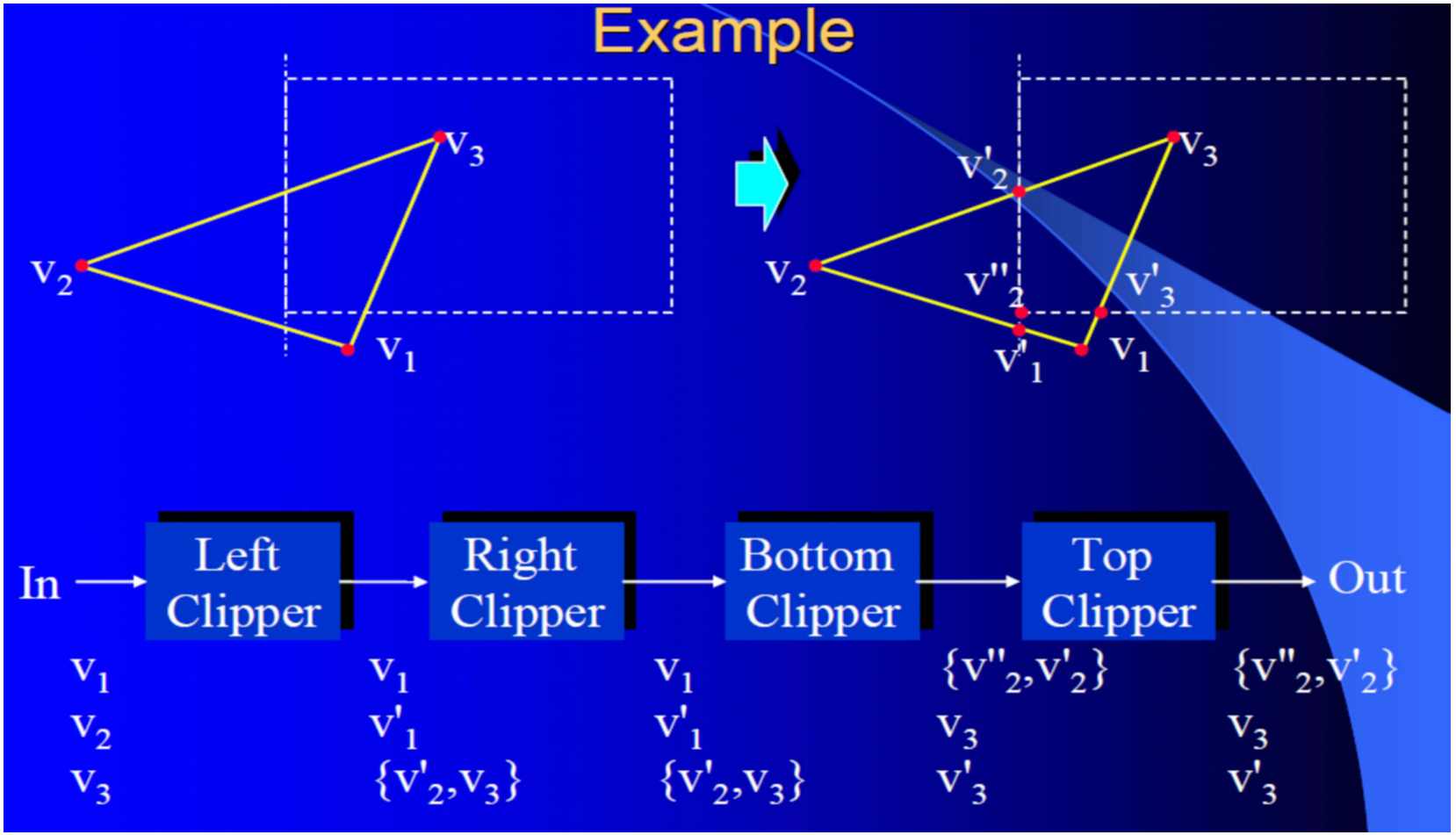
Sutherland-Hodgman Algorithm



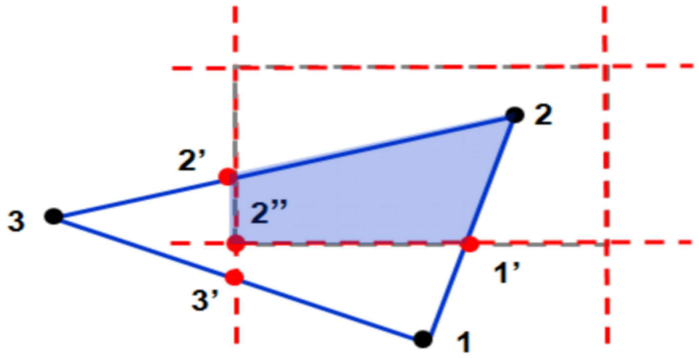
6. Repeat the steps 4 and 5 for remaining edges of the clipping window. Each time the resultant list of vertices is successively passed to process the next edge of the clipping window.
7. Stop.

Polygon Clipping

Sutherland-Hodgman

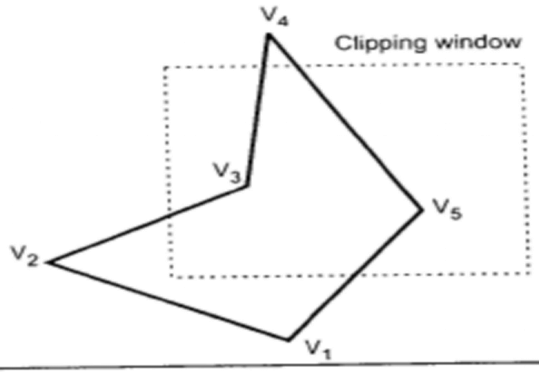


Polygon Clipping: Sutherland-Hodgman

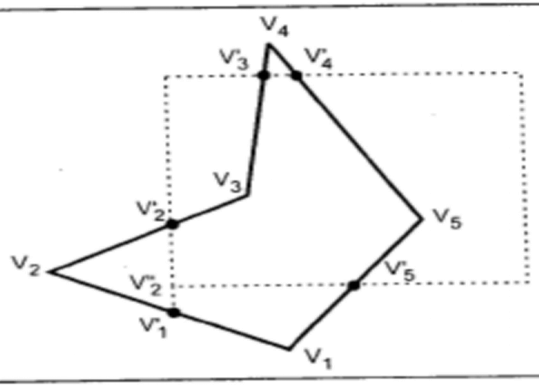


Input	Left Clipper	Right Clipper	Bottom Clipper	Top Clipper
[1,2]:	(in-in)>{2}			
[2,3]:	(in-out)>{2'}	[2,2']:(in-in)>{2'}		
[3,1]:	(out-in)>{3',1}	[2',3']:(in-in)>{3'}	[2',3']:(in-out)>{2"}	
		[3',1]:(in-in)>{1}	[3',1]:(out-out)>{}	
		[1,2]:(in-in)>{2}	[1,2]:(out-in)>{1',2}	[2",1']:(in-in)>1'}
			[2,2']:(in-in)>{2'}	[1',2]:(in-in)>{2}
				[2,2']:(in-in)>{2'}
				[2',2"]:(in-in)>{2"}

Polygon Clipping : Sutherland-Hodgman



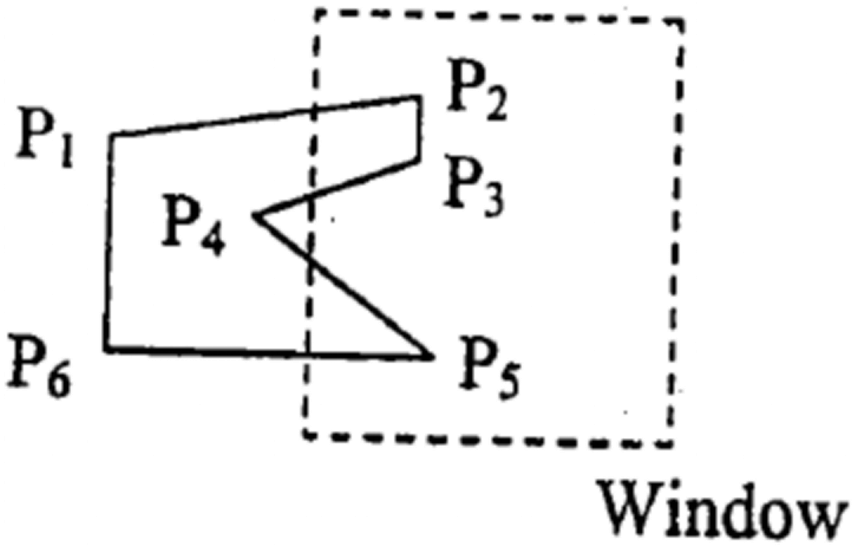
Sol. : Original polygon vertices are V_1, V_2, V_3, V_4, V_5. After clipping each boundary the new vertices are given in Fig. 5.26



After left clipping : $V_1, V'_1, V'_2, V_3, V_4, V_5$
After right clipping : $V_1, V'_1, V'_2, V_3, V_4, V_5$
After top clipping : $V_1, V'_1, V'_2, V_3, V'_3, V'_4, V_5$
After bottom clipping : $V''_2, V'_2, V_3, V'_3, V'_4, V_5, V'_5$

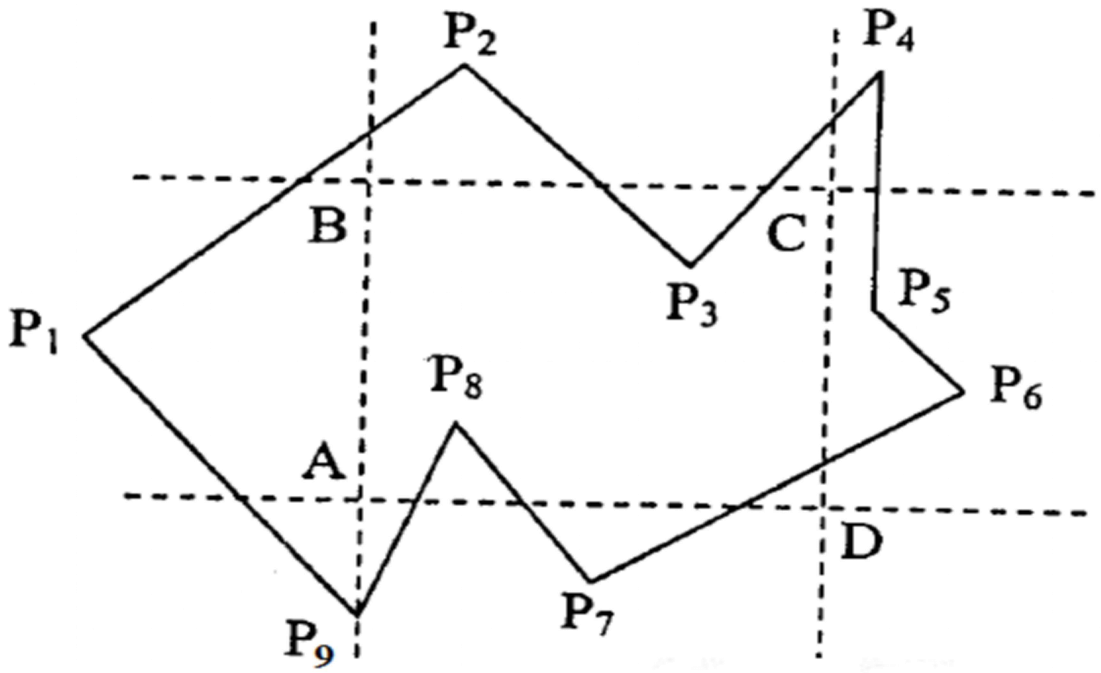
Polygon Clipping: Sutherland-Hodgman

- Example
- Illustrate the Sutherland-Hodgman algorithm for clipping the polygon P_1 P_2 P_3 P_4 P_5 P_6 against a rectangular window ABCD.



Polygon Clipping: Sutherland-Hodgman

- Example
- Illustrate the Sutherland-Hodgman algorithm for clipping the polygon P_1 P_2 P_3 P_4 P_5 P_6 P_7 P_8 P_9 against a rectangular window ABCD.



Viewing Transformation

Viewing transformation is the mapping of a part of a world-coordinate scene to device coordinates.

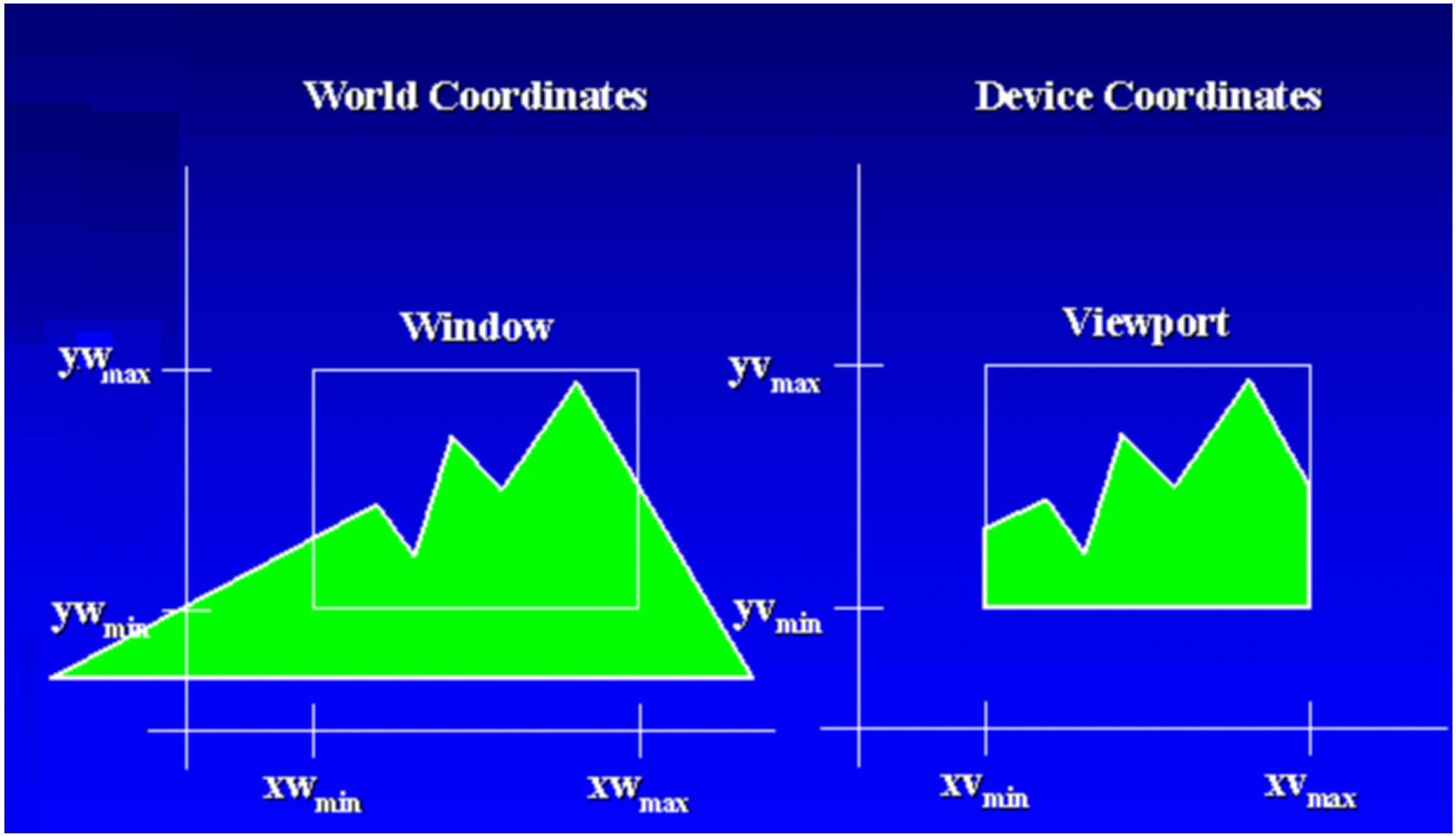
In 2D (two dimensional) viewing transformation is simply referred as the *window-to-viewport transformation* or the *windowing transformation*.

Mapping a window onto a viewport involves converting from one coordinate system to another.

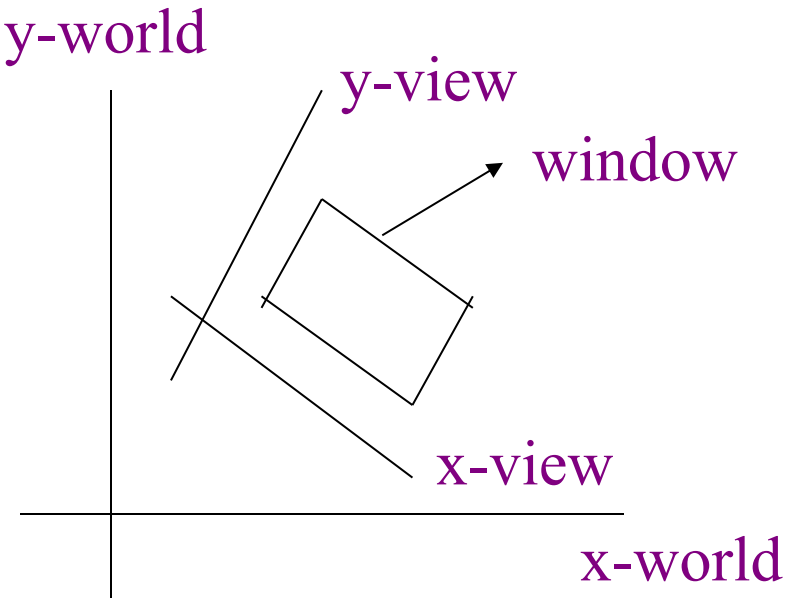
If the window and viewport are in standard position, this just

- involves translation and scaling.
- if the window and/or viewport are not in standard, then extra transformation which is rotation is required.

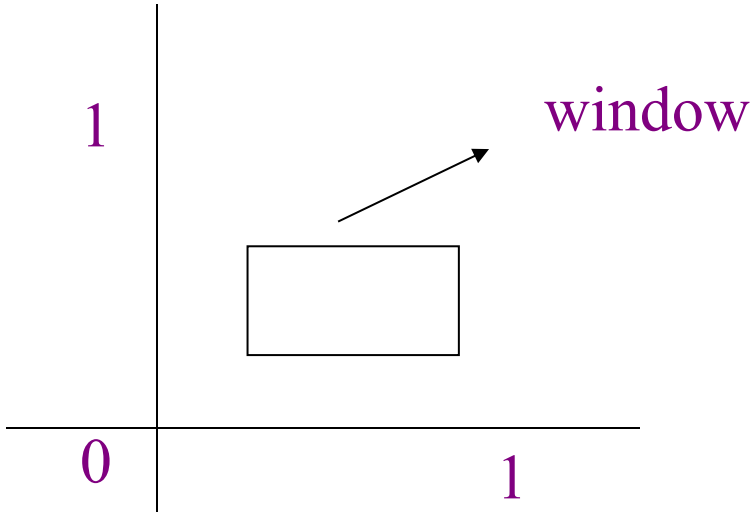
Viewing Transformation



Viewing Transformation



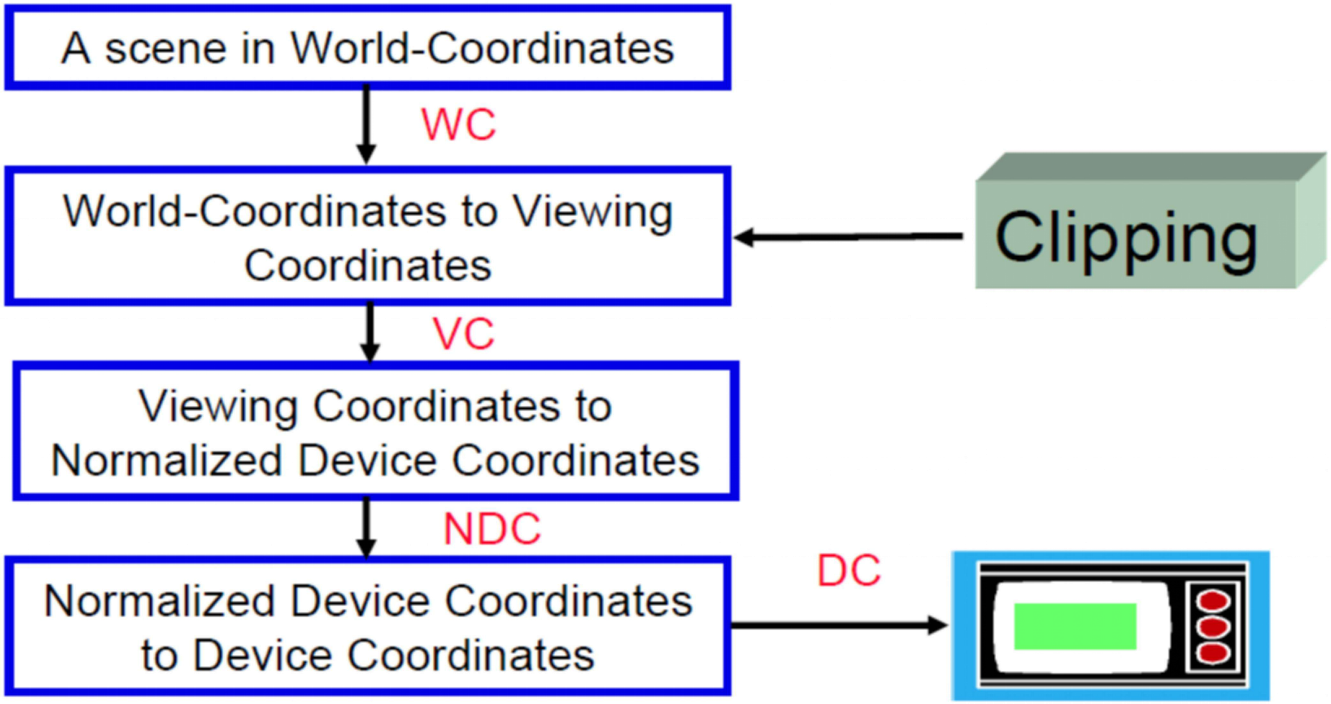
world



Normalised device

Viewing Transformation in 2D

- Objects are given in *world coordinates*
- The world is viewed through a *window*
- The window is mapped onto a *device viewport*



Viewing Transformation

- The viewing transformation which maps picture coordinates in the world coordinate system to display coordinates in the physical device coordinate system is performed by the following transformations:
- Normalization transformation(N)
- Workstation Transformation(W)
- **Normalization transformation(N)**
- Different display devices may have different screen sizes as measured in pixels. Size of the screen in pixels increases as resolution of the screen increases. When a picture is defined in pixel values then it is displayed large in size on the low resolution screen while small in size on the high resolution .To avoid it and make the programs to be device independent, we define the picture coordinates in some other units than pixels and use the interpreter to convert these coordinates to pixel values for the particular display device. The device independent units are called the normalized device coordinates.

Viewing Transformation

- **Normalization transformation(N)**
- The interpreter uses a linear formula to convert the normalized device coordinates to the actual device coordinates.
  - x= $x_n \quad X_W$
  - y= $y_n \quad Y_H$

Where x,y represent actual device x & y coordinate,

x_n , y_n represent normalized x & y coordinate,

X_W represent width of actual screen in pixels,

Y_H represent height of actual screen in pixels.

The transformation which maps the world coordinate to normalized device coordinate is called normalization transformation. It involves scaling of x & y,thus it is also refereed to as scaling transformation.

Viewing Transformation

- **Window To Viewport Coordinate Transformation**
- Consider a point (xw,yw) in world coordinate system mapped to p'(xv,yv) in the device coordinate system.
- In order to maintain the same relative placement in the viewport as in the window, the relationship is

$$\frac{xv - xv_{min}}{xv_{max} - xv_{min}} = \frac{xw - xw_{min}}{xw_{max} - xw_{min}}$$

$$\frac{yv - yv_{min}}{yv_{max} - yv_{min}} = \frac{yw - yw_{min}}{yw_{max} - yw_{min}}$$

Viewing Transformation

- Window To Viewport Coordinate Transformation

  Solving the expressions,

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

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

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

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

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

$$xv - xv_{min} = (xw - xw_{min})sx$$

$$yv - yv_{min} = (yw - yw_{min})sy$$

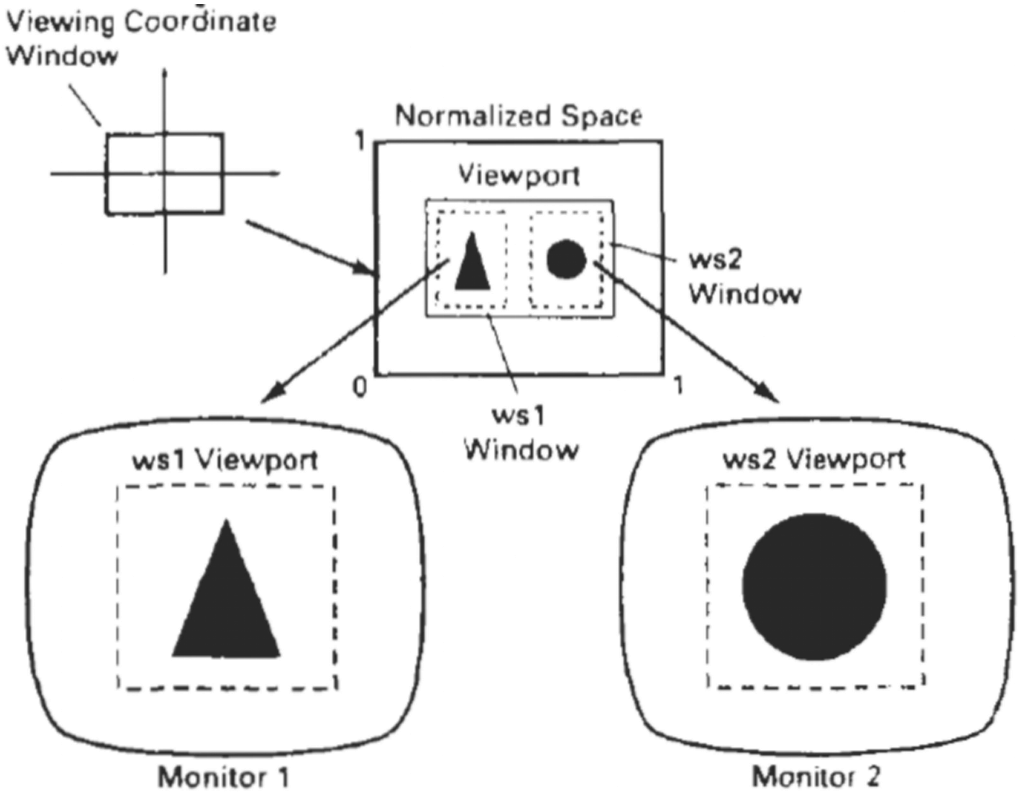
Where sx,sy are the scaling factor

$$sx = \frac{xv_{max} - xv_{min}}{xw_{max} - xw_{min}} = \frac{\text{Viewport x extent}}{\text{window x extent}}$$

$$sy = \frac{yv_{max} - yv_{min}}{yw_{max} - yw_{min}} = \frac{\text{Viewport y extent}}{\text{window y extent}}$$

Window To Viewport Coordinate Transformation

When mapping window-to-viewport transformation is done to different devices from one normalized space, it is called workstation transformation.



Viewing Transformation

- Window To Viewport Coordinate Transformation

The sequence of transformations are:

1. Perform a scaling transformation using a fixed-point position of (xw_{min}, yw_{min}) that scales the window area to the size of the viewport.
2. Translate the scaled window area to the position of the viewport.

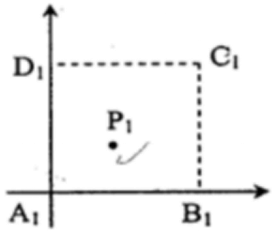
- Relative proportions of objects are maintained if the scaling factors are the same (sx = sy). Otherwise, world objects will be stretched or contracted in either x or y direction when displayed on output device.

Viewing Transformation

- The steps in the sequence of transformation is as follows:

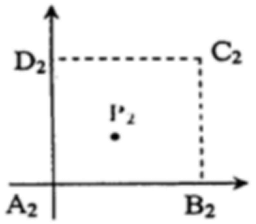
Step I:
Translate (xw_{min}, yw_{min}) to origin
the required translation matrix is

$$T_1 = \begin{bmatrix} 1 & 0 & -xw_{min} \\ 0 & 1 & -yw_{min} \\ 0 & 0 & 1 \end{bmatrix}$$



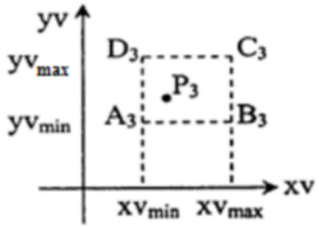
Step II:
Scale wrt origin. The required scaling matrix is

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

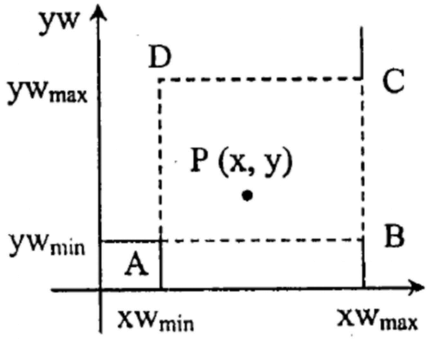


Step III:
Translate to (xv_{min}, yv_{min})
The required translation matrix is,

$$T_2 = \begin{bmatrix} 1 & 0 & xv_{min} \\ 0 & 1 & yv_{min} \\ 0 & 0 & 1 \end{bmatrix}$$



The composite transformation matrix is
$V = ((T_2 \cdot s) \cdot T_1)$
Any point p (x, y, 1) on the object will be transformed to p′ (x′, y′ 1) such that
$P' = V \cdot P$



Initial position of Window

Viewing Transformation

- Find the normalization transformation window to viewport with window lower left corner at(1,1) and upper right corner at (3,5) onto a viewpoint with lower left corner at (0,0) and upper right corner at $\frac{1}{2}$ $\frac{1}{2}$

Given Coordinates for window

$xw_{min}=1$, $yw_{min}=1$

$xw_{max}=3$, $yw_{max}=5$

Given Coordinates for viewport

$xv_{min}=0$, $yv_{min}=0$

$xv_{max}=\frac{1}{2}$ 0.5 $yv_{max}=\frac{1}{2}$ 0.5

Now calculate sx,sy.

sx=0.25

sy=0.125

$$sx = \frac{xv_{max} - xv_{min}}{xw_{max} - xw_{min}} = \frac{\text{Viewport x extent}}{\text{window x extent}}$$

$$sy = \frac{yv_{max} - yv_{min}}{yw_{max} - yw_{min}} = \frac{\text{Viewport y extent}}{\text{window y extent}}$$

Viewing Transformation

- Transformation matrix is given by
  - Translate-Scale-Translate

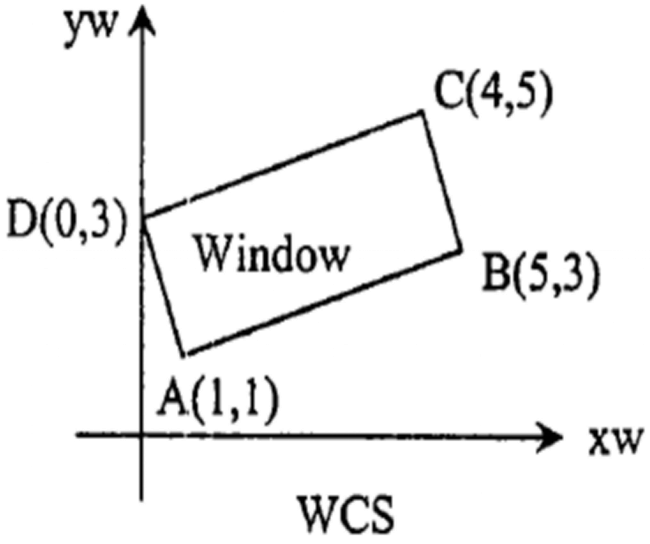
$$\begin{matrix} 1 & 0 & xv_{\min} \\ 0 & 1 & yv_{\min} \\ 0 & 0 & 1 \end{matrix} \begin{matrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{matrix} \begin{matrix} 1 & 0 & -xw_{\min} \\ 0 & 1 & -yw_{\min} \\ 0 & 0 & 1 \end{matrix}$$

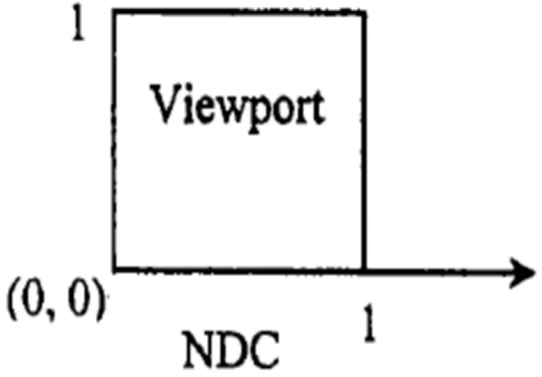
$$\begin{matrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{matrix} \begin{matrix} 0.25 & 0 & 0 \\ 0 & 0.125 & 0 \\ 0 & 0 & 1 \end{matrix} \begin{matrix} 1 & 0 & -1 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{matrix}$$

$$\begin{matrix} 0.25 & 0 & 0.25 \\ 0 & 0.125 & 0.125 \\ 0 & 0 & 1 \end{matrix}$$ Ans.

Viewing Transformation

- Find a normalization transformation which uses the rectangle A(1,1),B(5,3), C(4,5) & D(0,3) as window and NDC as a viewport.





Viewing Transformation

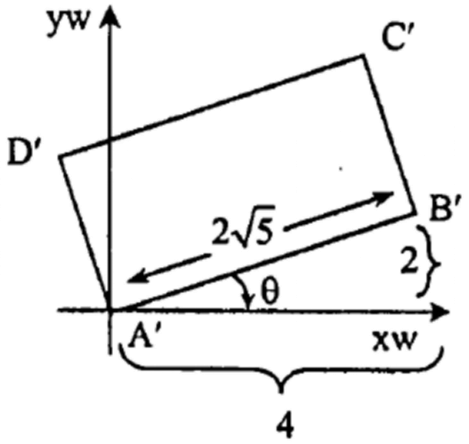
- Step-1. Translate point A(1,1) to origin. The required translation matrix is
- $T_1 = \begin{matrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{matrix}$

- Step-2.Rotate the rectangle about origin by an angle in clockwise direction so that it is aligned with the coordinate axis.
- The required rotation matrix is
- $R = \begin{matrix} \cos & \sin & 0 \\ \sin & \cos & 0 \\ 0 & 0 & 1 \end{matrix}$

where,

$$\cos\theta = \frac{4}{2\sqrt{5}} = \frac{2}{\sqrt{5}}$$

$$\sin\theta = \frac{2}{2\sqrt{5}} = \frac{1}{\sqrt{5}}$$

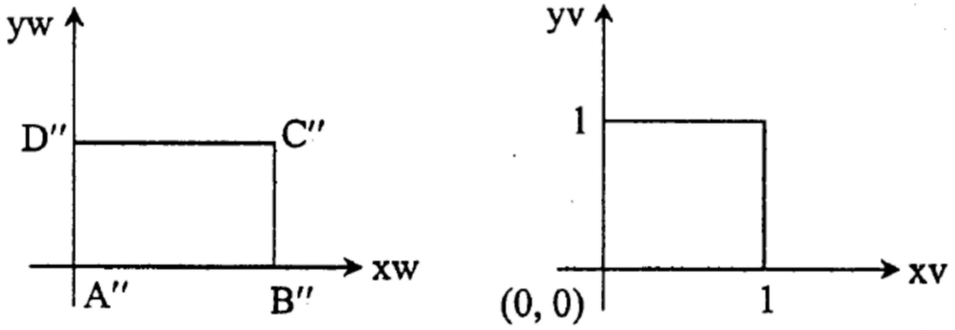


Viewing Transformation

- Thus R= $\begin{matrix} \frac{2}{\sqrt{5}} & \frac{1}{\sqrt{5}} & 0 \\ \frac{1}{\sqrt{5}} & \frac{2}{\sqrt{5}} & 0 \\ 0 & 0 & 1 \end{matrix}$

- Step-3.Perform scaling transformation V, that transforms window to viewport. The scaling matrix is

- S= $\begin{matrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{matrix}$



Viewing Transformation

where

$$sx = \frac{\text{Viewport x extent}}{\text{window x extent}} = \frac{1}{d(AB)}$$

$$sy = \frac{\text{viewport y extent}}{\text{window y extent}} = \frac{1}{d(AD)}$$

$$d(AB) = \sqrt{(3-1)^2+(5-1)^2} = \sqrt{20} = 2\sqrt{5}$$

$$d(AD) = \sqrt{(0-1)^2+(3-1)^2} = \sqrt{5}$$

$$sx = \frac{1}{2\sqrt{5}}$$

$$sy = \frac{1}{\sqrt{5}} \qquad S = \begin{bmatrix} \frac{1}{2\sqrt{5}} & 0 & 0 \\ 0 & \frac{1}{\sqrt{5}} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Viewing Transformation

The composite normalization transformation matrix is,

$N_T = S.R.T$

$$= \begin{bmatrix} \frac{1}{2\sqrt{5}} & 0 & 0 \\ 0 & \frac{1}{\sqrt{5}} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{2}{\sqrt{5}} & \frac{1}{\sqrt{5}} & 0 \\ -\frac{1}{\sqrt{5}} & \frac{2}{\sqrt{5}} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} \frac{1}{5} & \frac{1}{10} & \frac{3}{10} \\ \frac{1}{5} & \frac{2}{5} & \frac{1}{5} \\ 0 & 0 & 1 \end{bmatrix}$$

Reference

1. *Schaum's Outline of Computer Graphics*
2. *Computer Graphics by Donald Hearn and M. Pauline Baker*
3. *Principles of Interactive Computer Graphics by William Newman and Robert Sproull*