UNIT-2
(Section-2)

RCS-603: COMPUTER GRAPHICS

Presented By :

Dr. Vinod Jain (Associate Professor, GLBITM)

UNIT-2 Syllabus – Section 2

C H A P T E R *6* Two-Dimensional Viewing

Donal D. Hearn and M. Pauline Baker

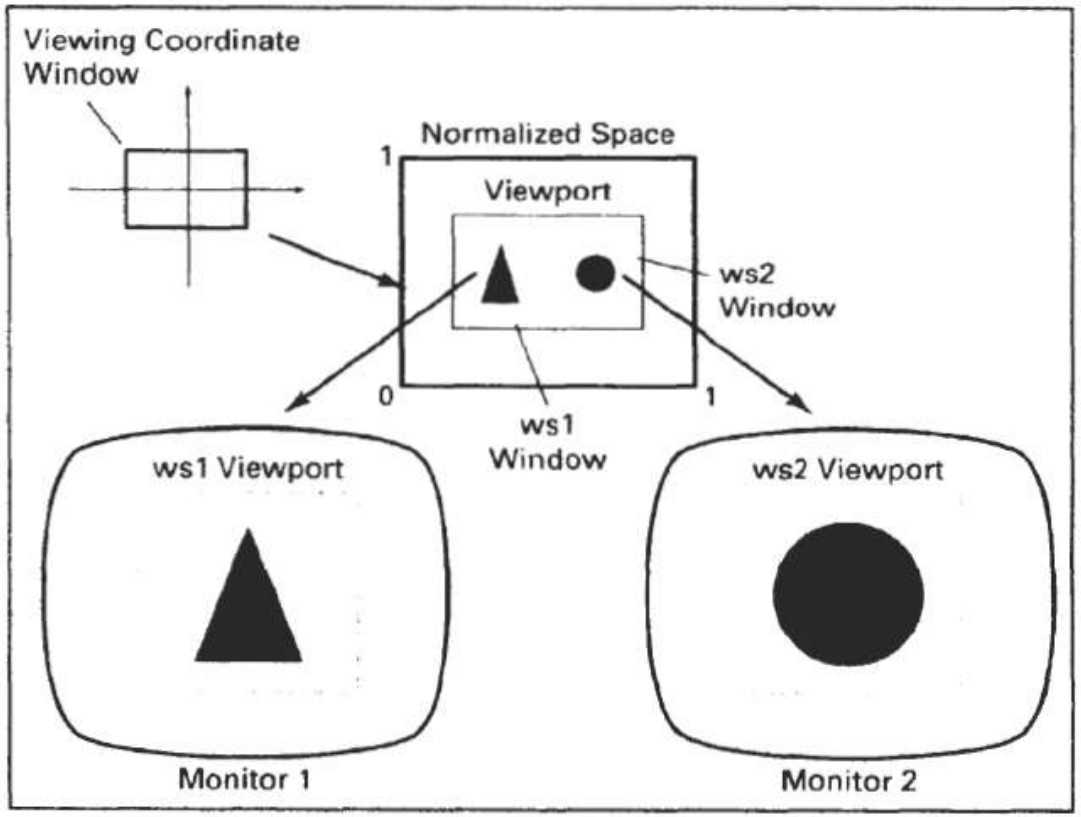
Windowing and Clipping: Viewing pipeline, Viewing transformations, 2-D Clipping algorithms- Line clipping algorithms such as Cohen Sutherland line clipping algorithm, Liang Barsky algorithm, Line clipping against non rectangular clip windows; Polygon clipping – Sutherland Hodgeman polygon clipping, Weiler and Atherton polygon clipping, Curve clipping, Text clipping

C H A P T E R *6* Two-Dimensional Viewing Donal D. Hearn and M. Pauline Baker

1. Windowing And Clipping: Viewing Pipeline
2. Viewing Transformations
3. 2-D Clipping Algorithms
    1. Line Clipping Algorithms Cohen Sutherland Line Clipping Algorithm
    2. Liang Barsky Algorithm
    3. Line Clipping Against Non Rectangular Clip Windows
4. Polygon Clipping
    1. Sutherland Hodgeman Polygon Clipping
    2. Weiler And Atherton Polygon Clipping
5. Curve Clipping
6. Text Clipping

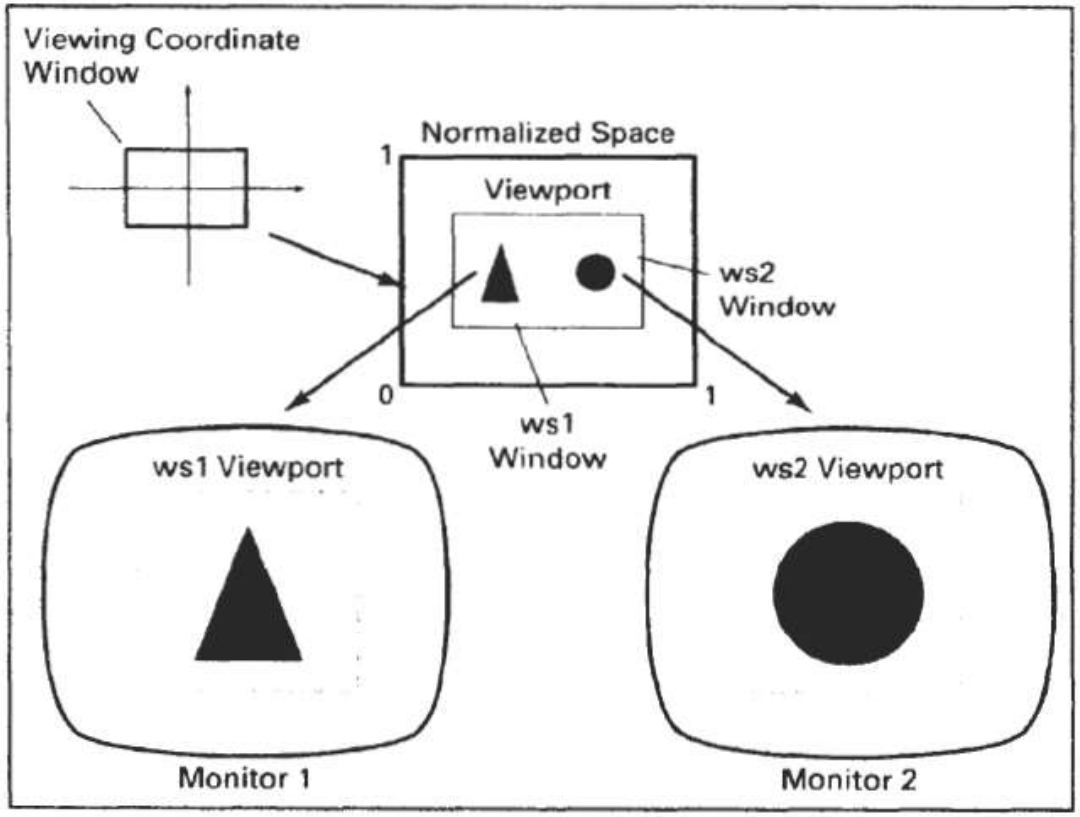
Windowing and Clipping:

- It consider the formal mechanism for displaying views of a picture on an output device.
- For a two-dimensional picture, a view is selected by specifying a subarea of the total picture area.



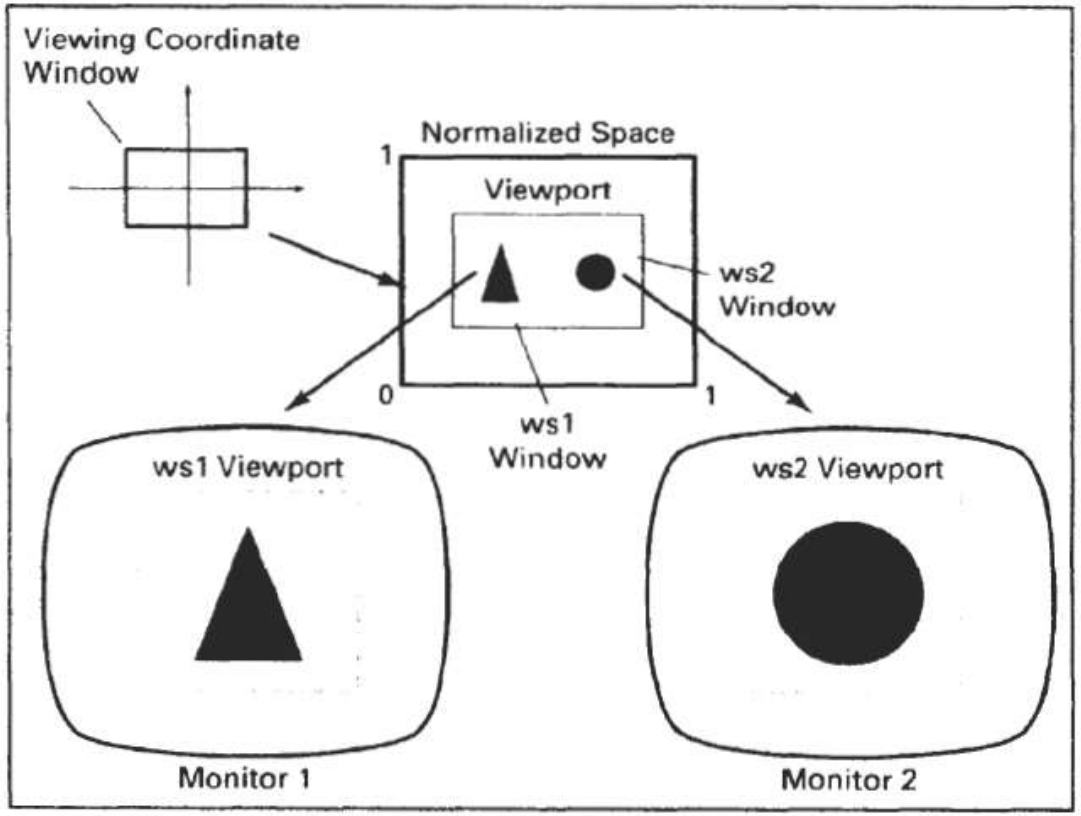
Windowing and Clipping:

- A user can select a single area for display, **or several areas could be selected for simultaneous display.**
- Transformations from world to device coordinates involve translation, rotation, and scaling operations, as well as procedures for deleting those parts of the picture that are outside the limits of a selected display area.(clipping)



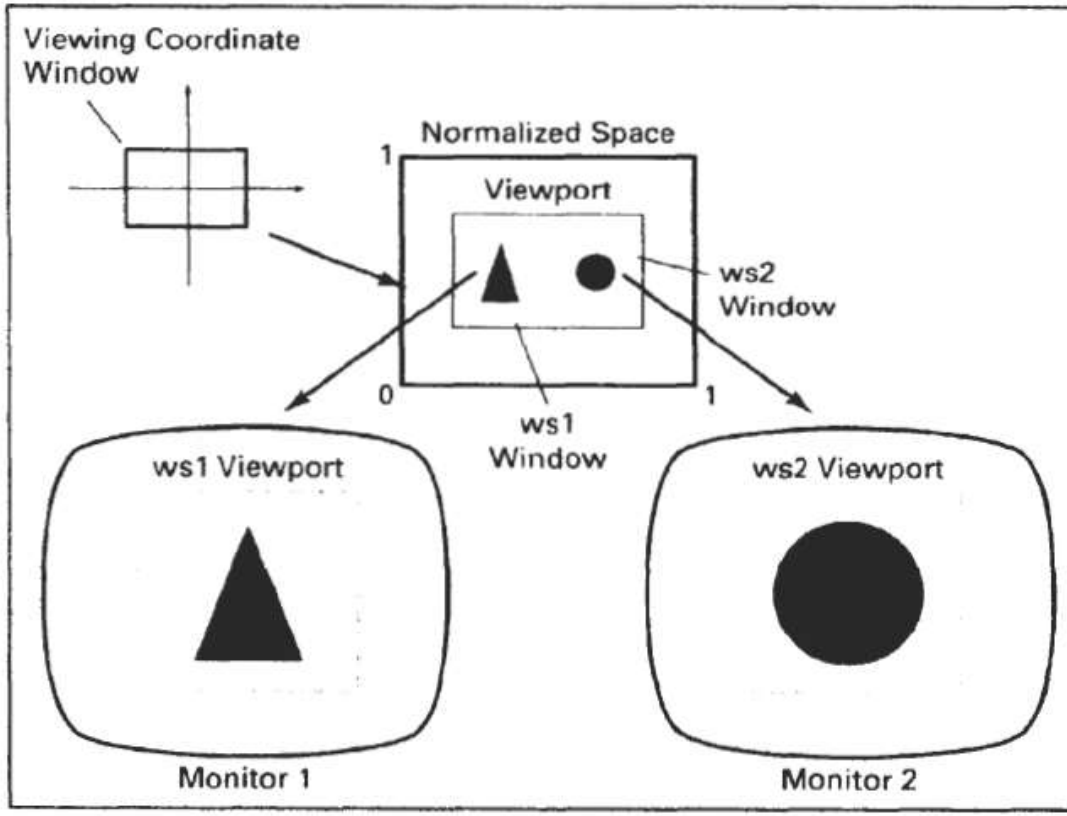
Windowing and Clipping:

- Transformations from world to device coordinates involve **translation, rotation, and scaling operations,** as well as procedures for deleting those parts of the picture that are outside the limits of a selected display area.(**clipping**)



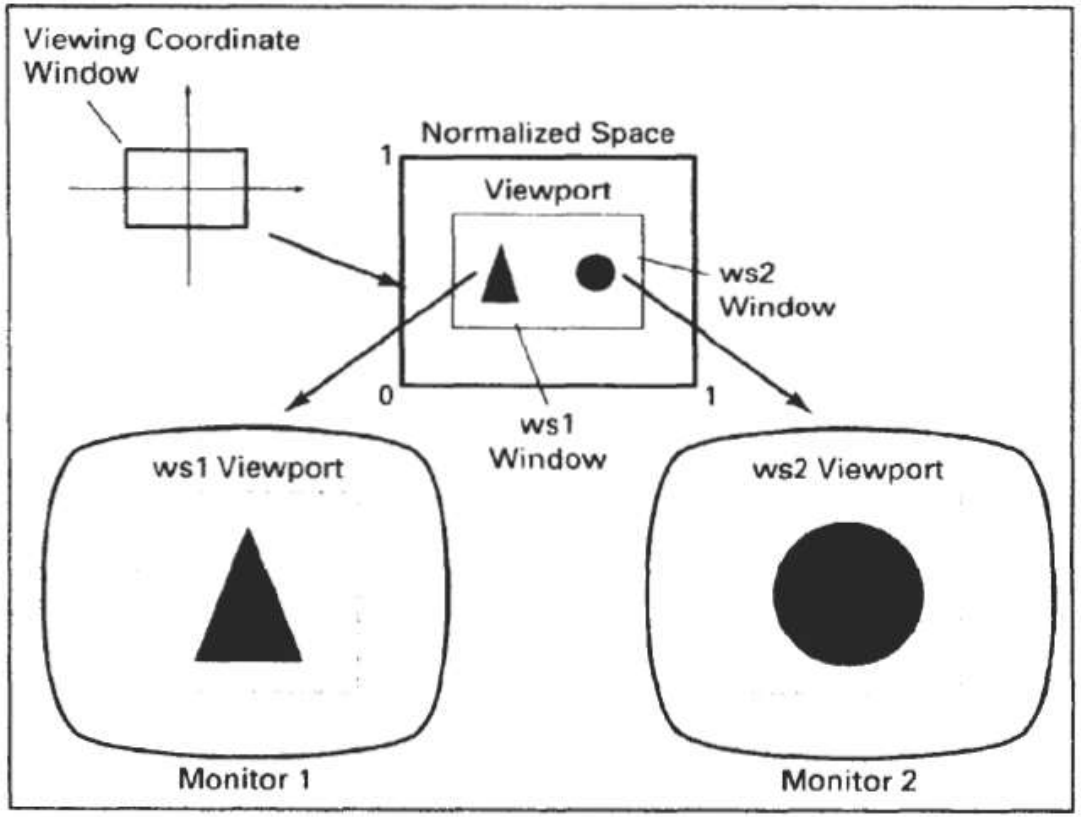
Viewing Pipeline

- **Window**
- A world-coordinate area selected for display is called a window.
- **Viewport**
- An area on a display device to which a window is mapped is called a viewport.
- The window defines ***what*** is to be viewed; the viewport defines ***where*** it is to be displayed.



Viewing Pipeline

- **Viewing Transformation.**
- The mapping of a part of a world-coordinate scene to device coordinates is referred to as a viewing transformation.



Viewing Pipeline

- Figure 6-1 illustrates the **mapping of a picture section that falls within a rectangular window** onto a designated & angular viewport.

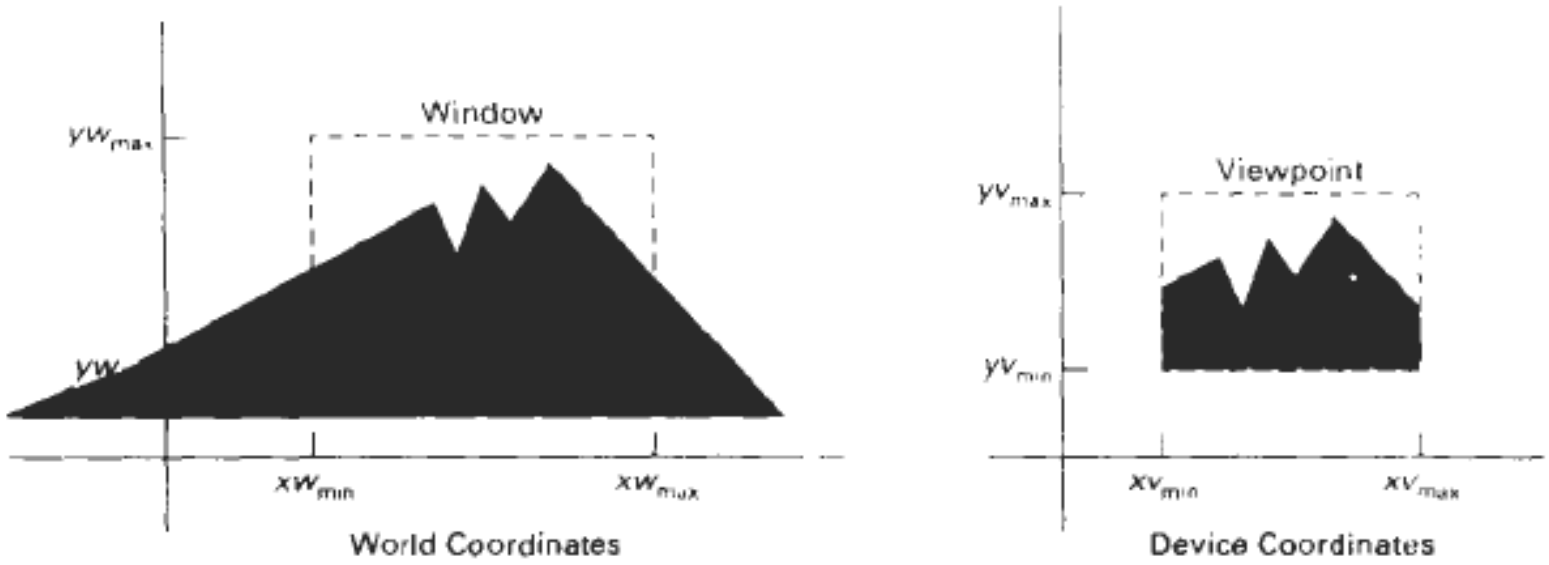


Figure 6-1
A viewing transformation using standard rectangles for the window and viewport.

Viewing Pipeline

- It consist a **sequence of steps used to perform two dimensional** viewing transformations
- The major steps are as follows

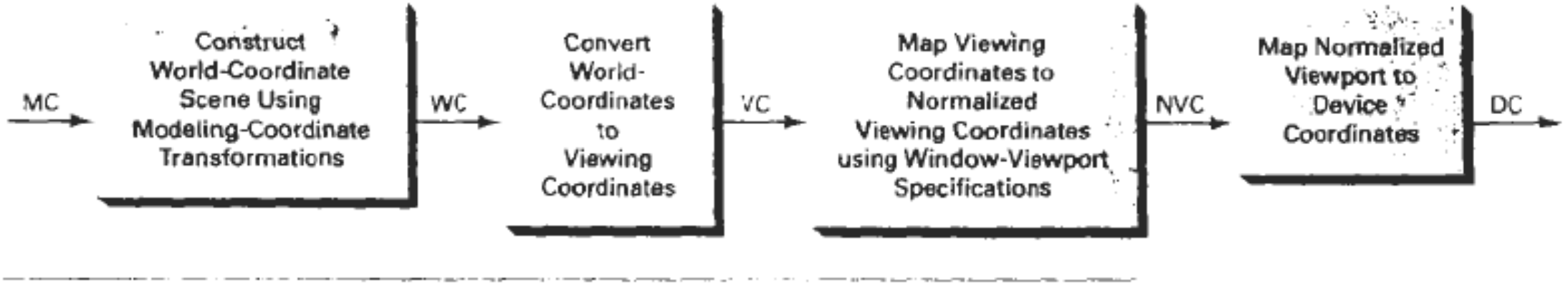
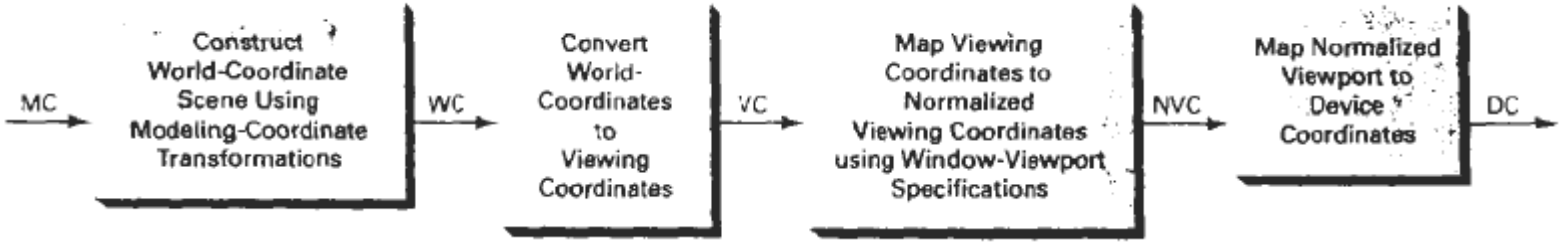


Figure 6-2
The two-dimensional viewing-transformation pipeline.

Viewing Pipeline

1. We construct the scene in world coordinates
2. We then transform descriptions in world coordinates to viewing coordinates.
3. We then define viewport in normalized coordinates (in the range from 0 to 1)
4. At the final step, all parts of the picture that he outside the viewport are clipped, and the contents of the viewport are transformed to device coordinates.



Viewing Pipeline

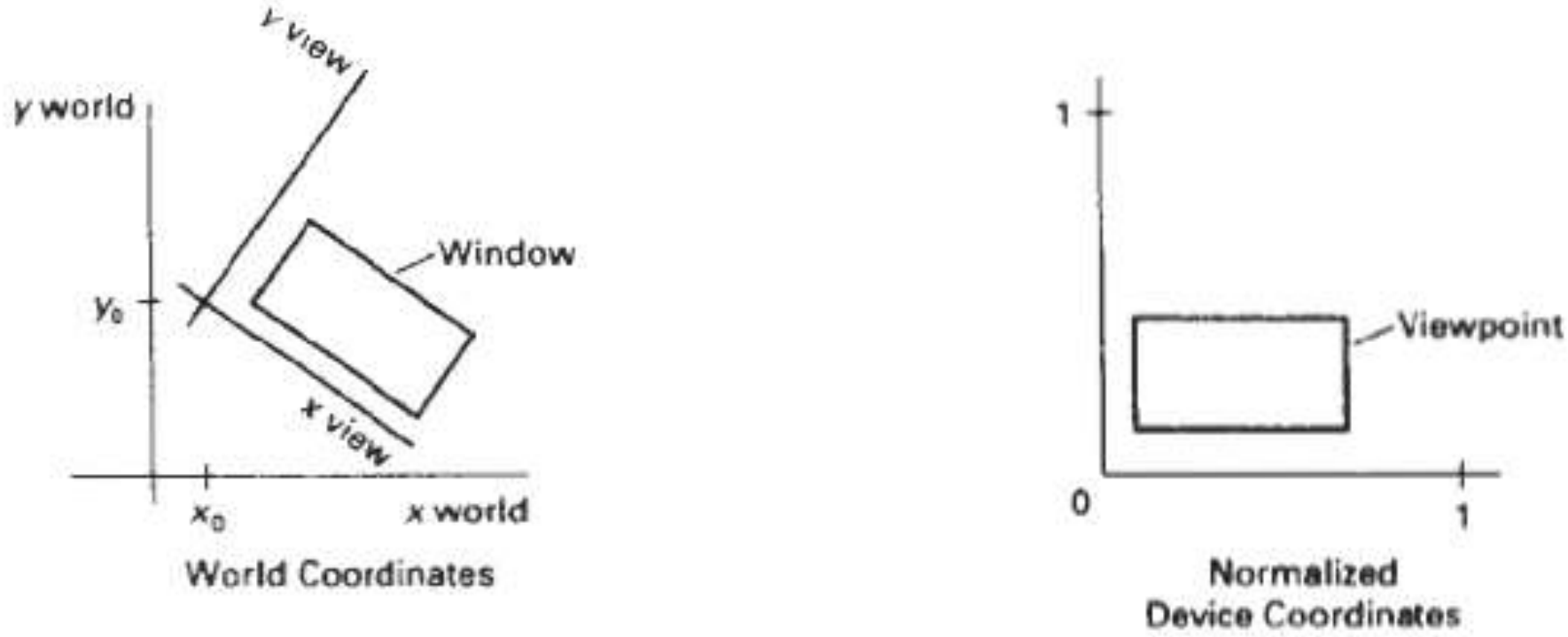
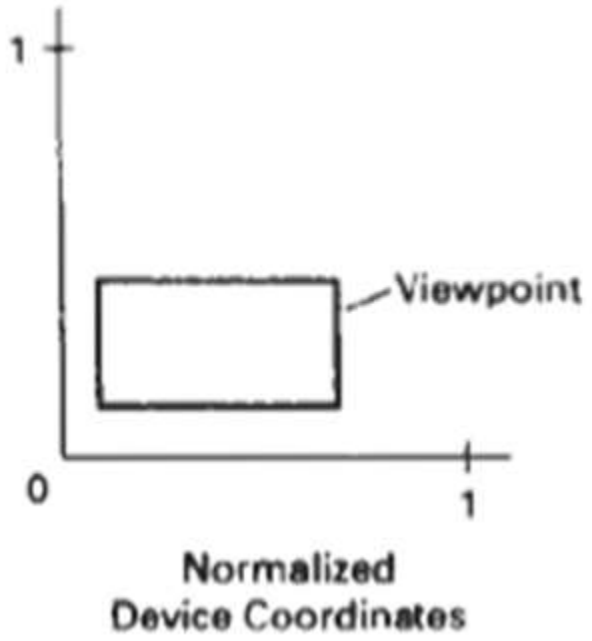


Figure 6-3
Setting up a rotated world window in viewing coordinates and the corresponding normalized-coordinate viewport.

Normalized Coordinates

- Viewports are typically defined within the unit square (normalized coordinates).
- This provides a **means for separating the viewing and other transformations from specific output-device requirements, so that the graphics package is largely device-independent.**



Viewing Coordinate Reference Frame

(a) translate the viewing origin to the world origin,

(b) rotate to align the axes of the two systems.

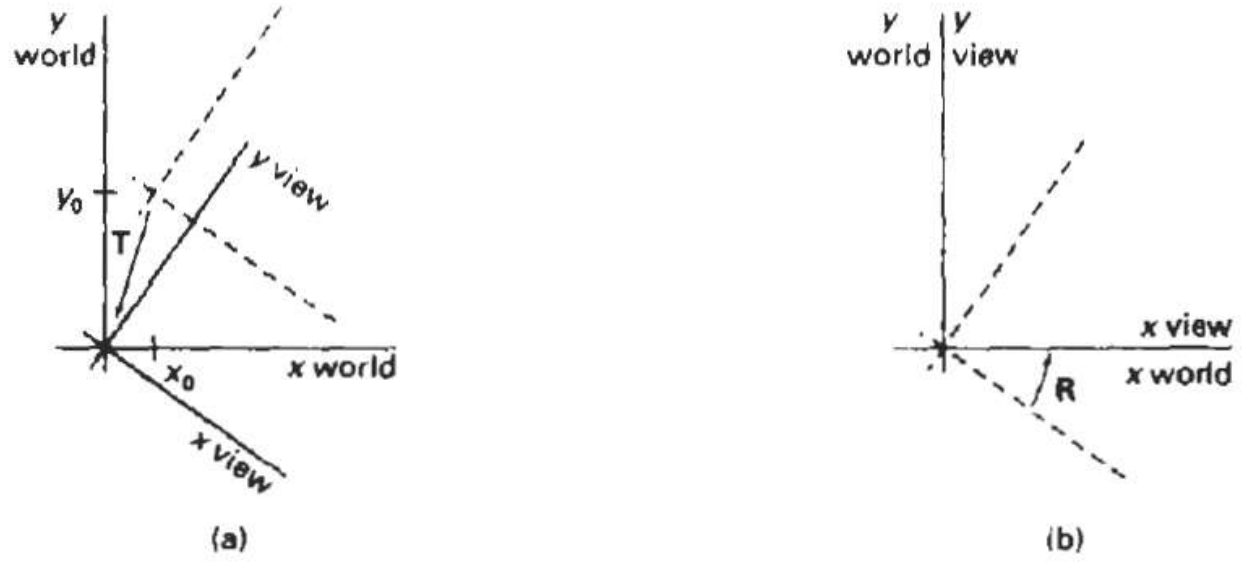


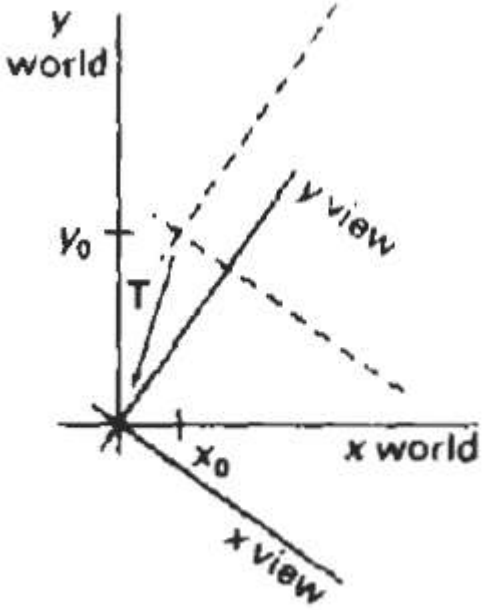
Figure 6-4
A viewing-coordinate frame is moved into coincidence with the world frame in two steps: (a) translate the viewing origin to the world origin, then (b) rotate to align the axes of the two systems.

Viewing Coordinate Reference Frame

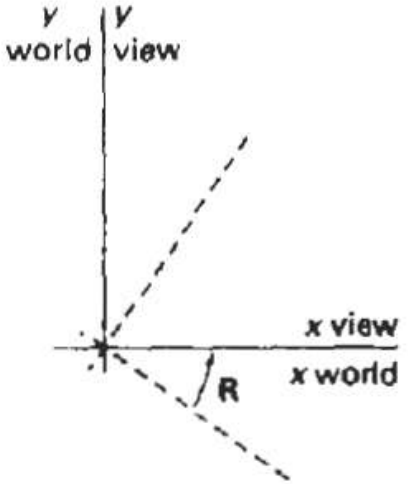
- The composite two-dimensional transformation to convert world coordinates to viewing coordinate is

$$\mathbf{M}_{WC,VC} = \mathbf{R} \cdot \mathbf{T} \qquad (6\text{-}1)$$

- where **T is the translation matrix** that takes the viewing origin point P to the world origin, and **R is the rotation matrix** that aligns the axes of the two reference frames.



(a)



(b)

6.3 WINDOW-TO-VIEWPORT COORDINATE TRANSFORMATION

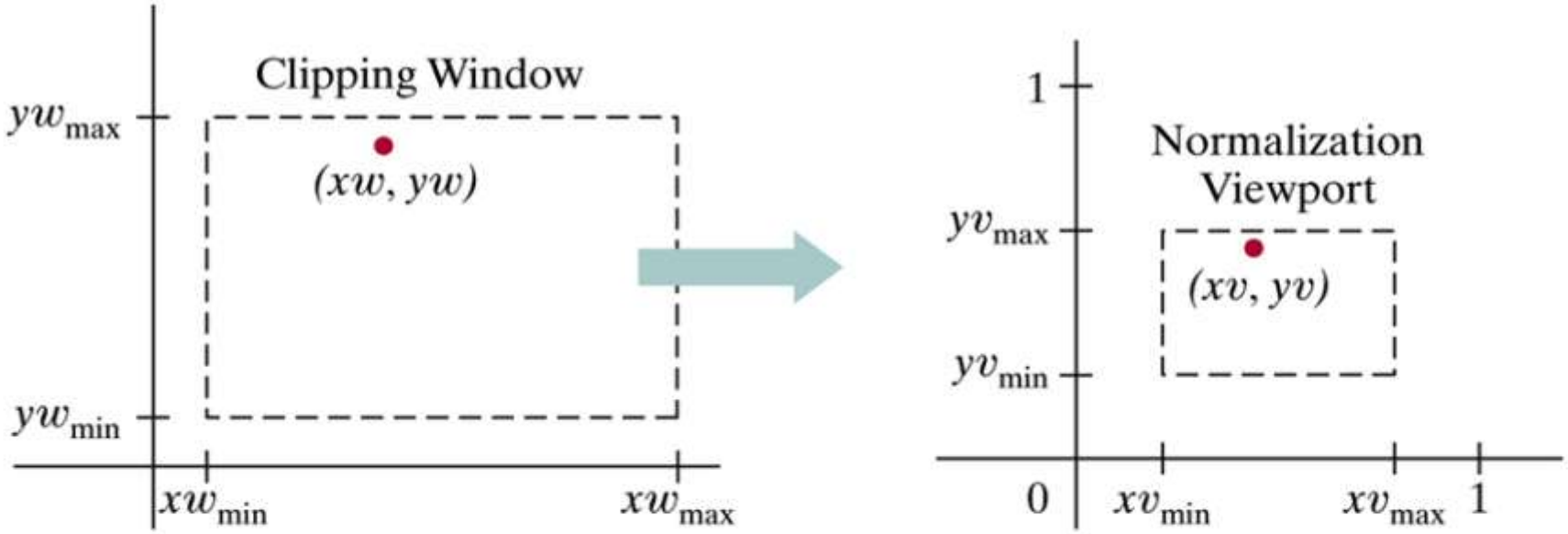
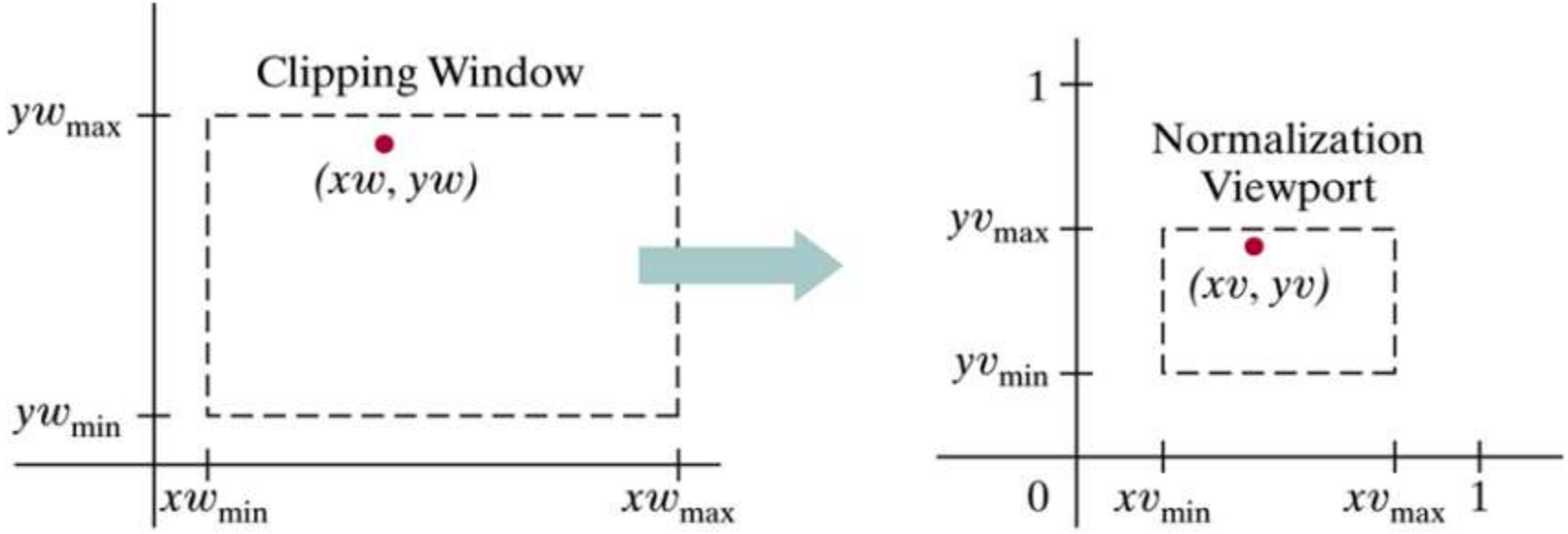


Figure 6-7

A point (xw, yw) in a world-coordinate clipping window is mapped to viewport coordinates (xv, yv), within a unit square, so that the relative positions of the two points in their respective rectangles are the same.

6.3 WINDOW-TO-VIEWPORT COORDINATE TRANSFORMATION

- We maintain the same relative placement in the viewport as in the window.
- If a coordinate position is at the center of the world window, for instance, it will be displayed at the center of the viewport.



6.3 WINDOW-TO-VIEWPORT COORDINATE TRANSFORMATION

- **A** point at position ***(xw,yx)*** in the window is mapped into position ***(xv,*yv)** in the associated viewport.

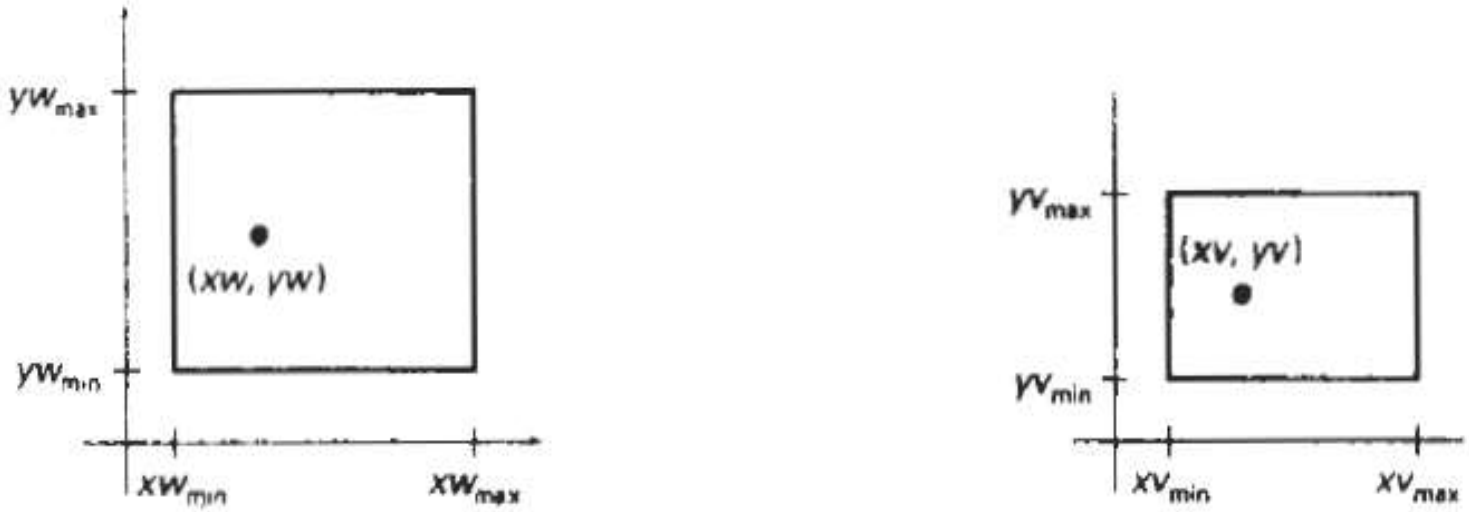


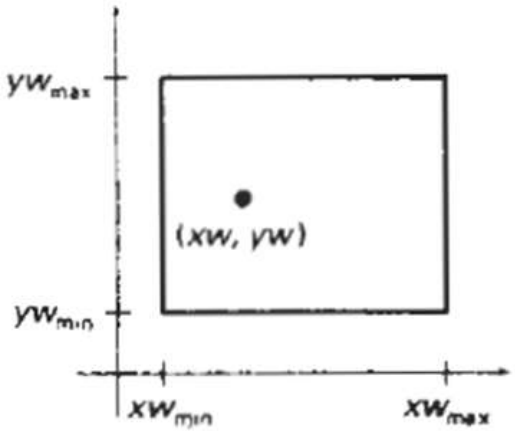
Figure 6-5
A point at position (xw, yw) in a designated window is mapped to viewport coordinates (xv, yv) so that relative positions in the two areas are the same.

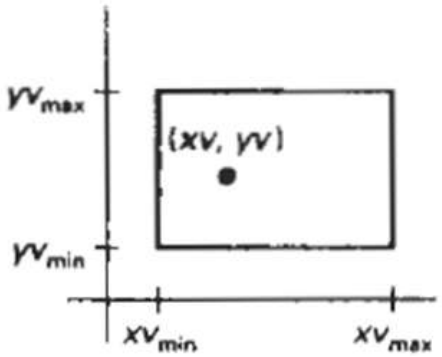
6.3 WINDOW-TO-VIEWPORT COORDINATE TRANSFORMATION

- We maintain the same relative placement in the viewport as in the window.

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

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





6.3 WINDOW-TO-VIEWPORT COORDINATE TRANSFORMATION

Solving these expressions for the viewport position (xv, yv), we have

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

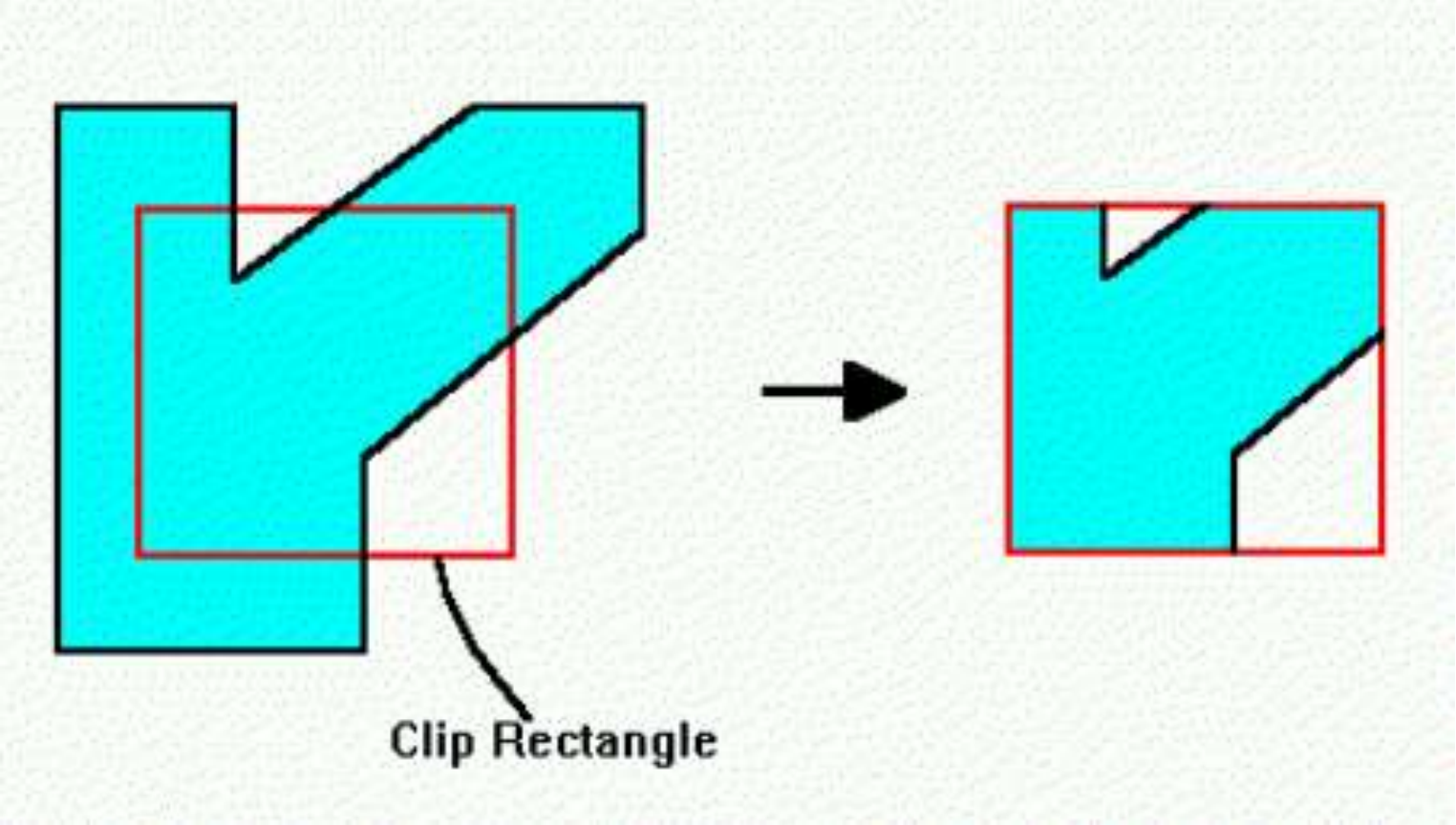
$$yv = yv_{min} + (yw - yw_{min})sy$$

where the scaling factors are

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

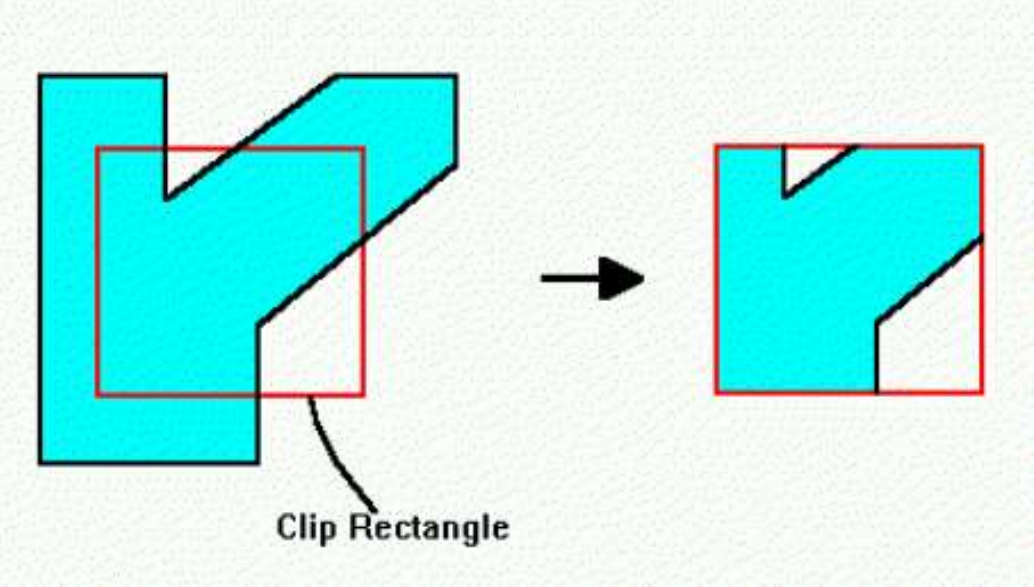
$$sy = \frac{yv_{max} - yv_{min}}{yw_{max} - yw_{min}}$$

2-D Clipping Algorithms



2-D Clipping Algorithms

- **Clipping - Any procedure that identifies those portions of a picture that are either inside or outside of a specified region of space is referred to as a clipping algorithm, or simply clipping.**
- The region against which an object is to clipped is called a clip window.



Applications Of Clipplng

Applications of clipping include

1. Drawing and painting operations that allow parts of a picture to be selected for copying, moving, erasing, or duplicating.
2. Extracting part of a defined scene for viewing;
3. Identifying visible surfaces in three-dimensional vlews;
4. Antialiasing line *seg*ments or object boundaries;
5. Creating objects using solid-modeling procedures;
6. Displaying a multi window environment;

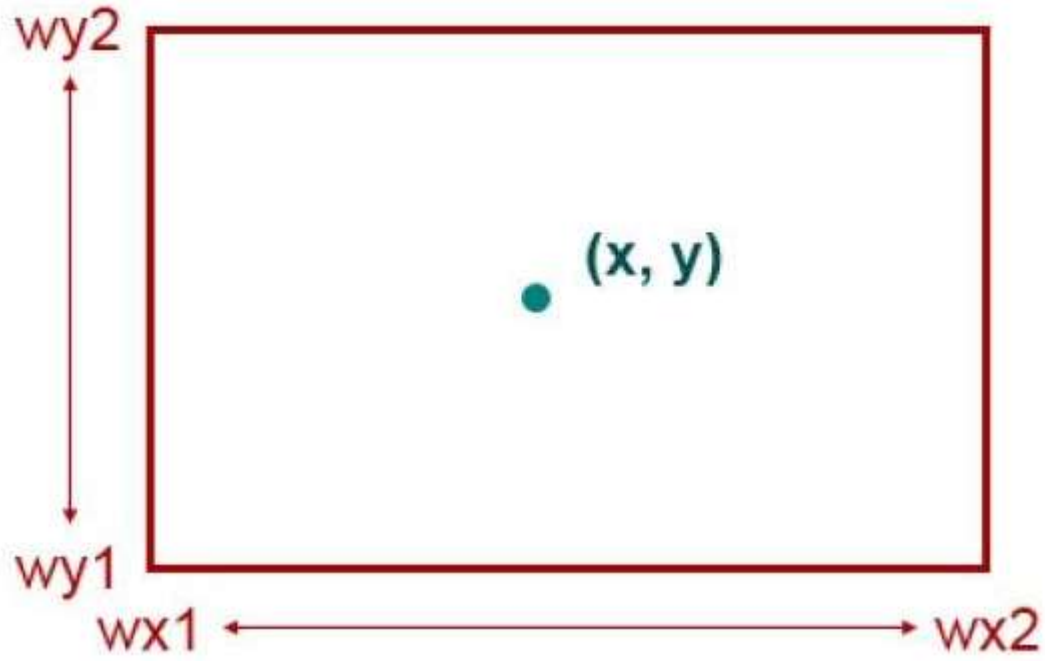
Types of clipping

- We consider algorithms for clipping the following primitive types

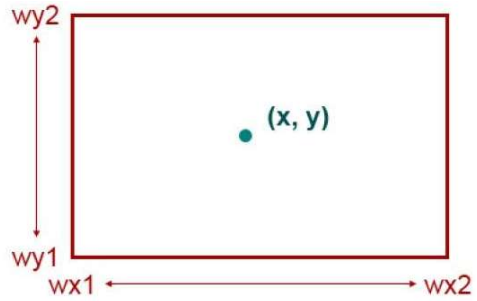
1. Point Clipping
2. Line Clipping (straight-line segments)
3. Area Clipping (polygons)
4. Curve Clipping
5. Text Clipping

Point Clipping

- we save a point P = ***(x,* y)** if it is inside the clipping window.



Point Clipping



- Assuming that the clip window is a rectangle in standard position, we save a point P = ***(x, y)*** for display if the following inequalities are satisfied:

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

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

- **where** the ***edges of*** the clip window $(xw_{min}, xw_{max}, yw_{min}, yw_{max})$ can be either the world-coordinate window boundaries or viewport boundaries. If any one of these four inequalities is not satisfied, the point is clipped (not saved for display).

Line Clipping Algorithms

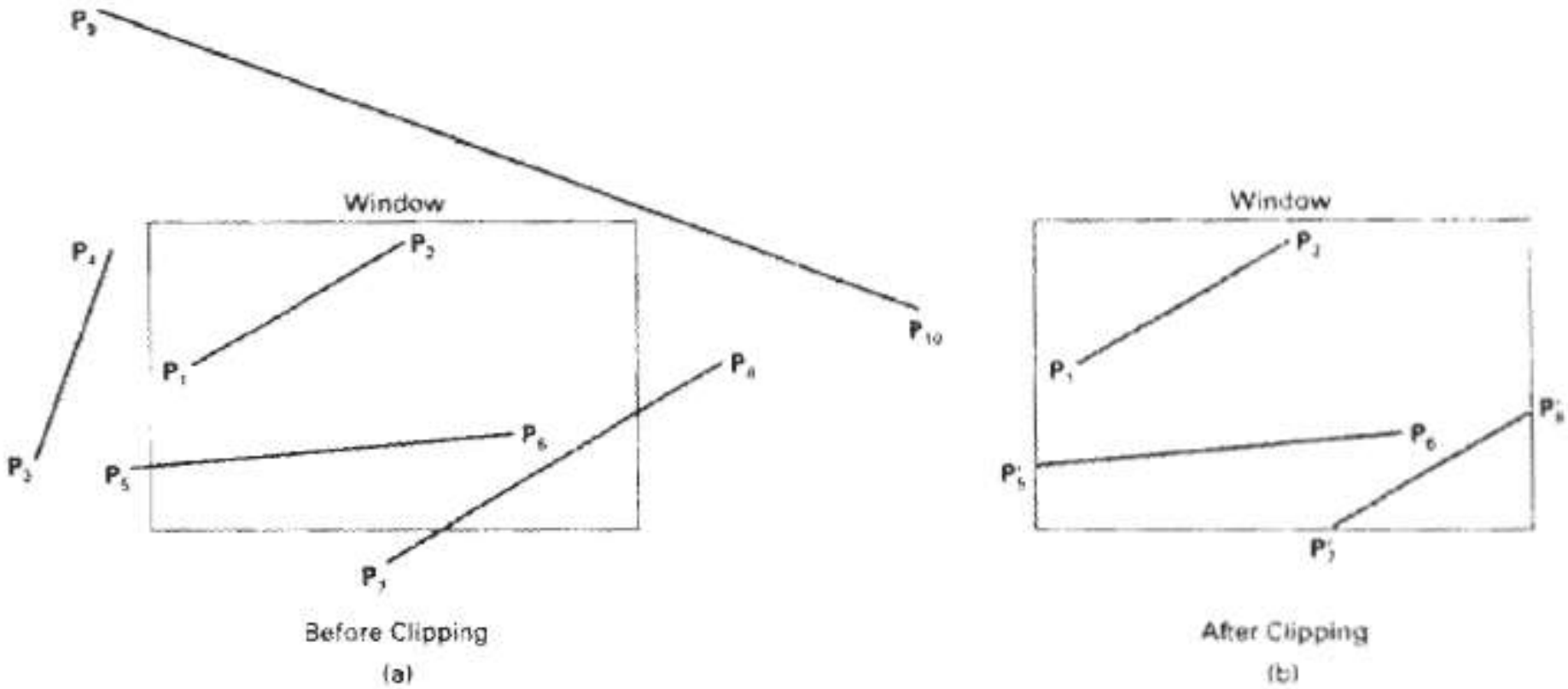
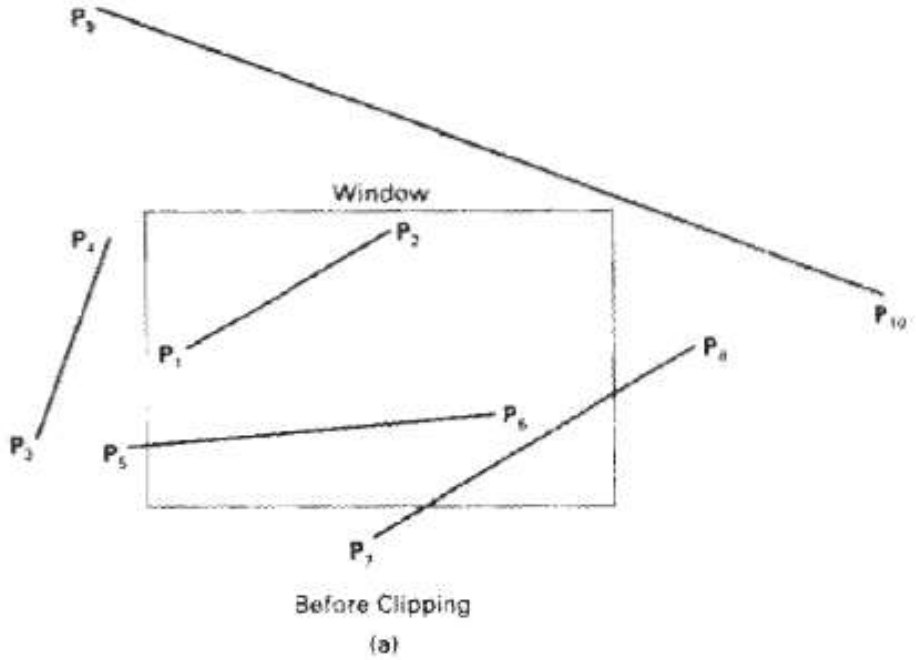


Figure 6-7
Line clipping against a rectangular clip window.

Line Clipping Algorithms

- *A* line clipping procedure involves several parts.

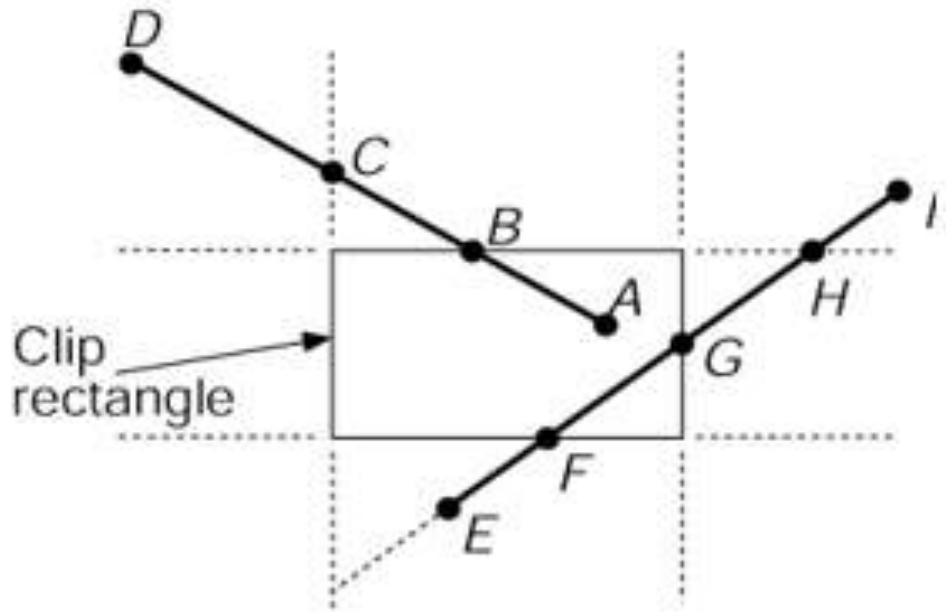
1. First, we can test a given line segment to determine **whether it lies completely inside** the clipping window.
2. If it does not, we try to determine whether it **lies completely outside** the window.
3. Finally, we must perform **intersection calculations with one or more clipping boundaries.**



Cohen Sutherland Line Clipping Algorithm

Basic Idea:

- **First**, do easy test
  - *completely* inside or outside the box?
- **If no**, we will need to figure out how line intersects the box



Cohen Sutherland Line Clipping Algorithm

- Every line end point in a picture is assigned a four-digit binary code, called a region code, that identifies the location of the point relative to the boundaries of the clipping rectangle.

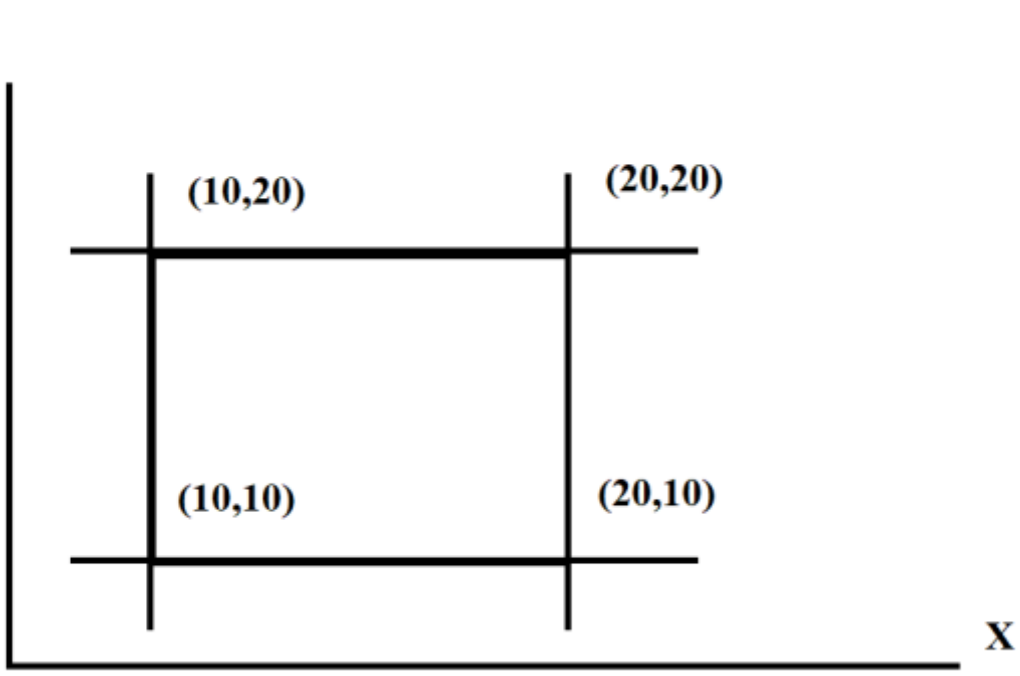
- First bit set **1** : Point lies to **left** of window $x < x_{min}$
- Second bit set **1** : Point lies to **right** of window $x > x_{max}$
- Third bit set **1** : Point lies below(**bottom**) window $y < y_{min}$
- fourth bit set **1** : Point lies above(**top**) window $y > y_{max}$

LRBT (Left, Right, Bottom, Top).

1001	0001	0101
1000	0000	0100
	Window	
1010	0010	0110

Cohen Sutherland Line Clipping Algorithm

- Find color codes of the points
- P1(5,6) P2 (25,15) P2 (25,15)
- P4(15,60) P5 (15,15) P6 (25,40)



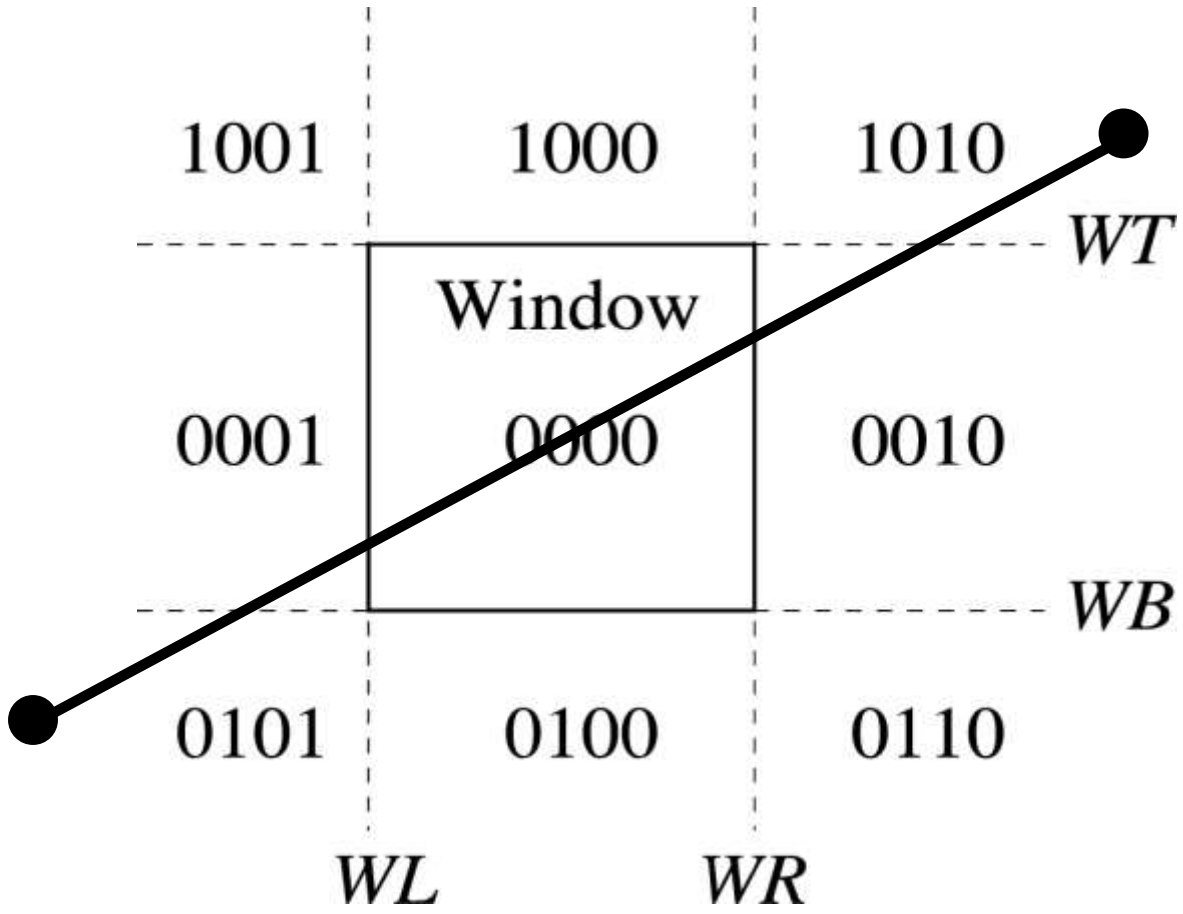
- First bit set **1** : Point lies to **left** of window $x < x_{min}$
- Second bit set **1** : Point lies to **right** of window $x > x_{max}$
- Third bit set **1** : Point lies below(**bottom**) window $y < y_{min}$
- fourth bit set **1** : Point lies above(**top**) window $y > y_{max}$

LRBT (Left, Right, Bottom, Top).

1001	0001	0101
1000	0000	0100
	Window	
1010	0010	0110

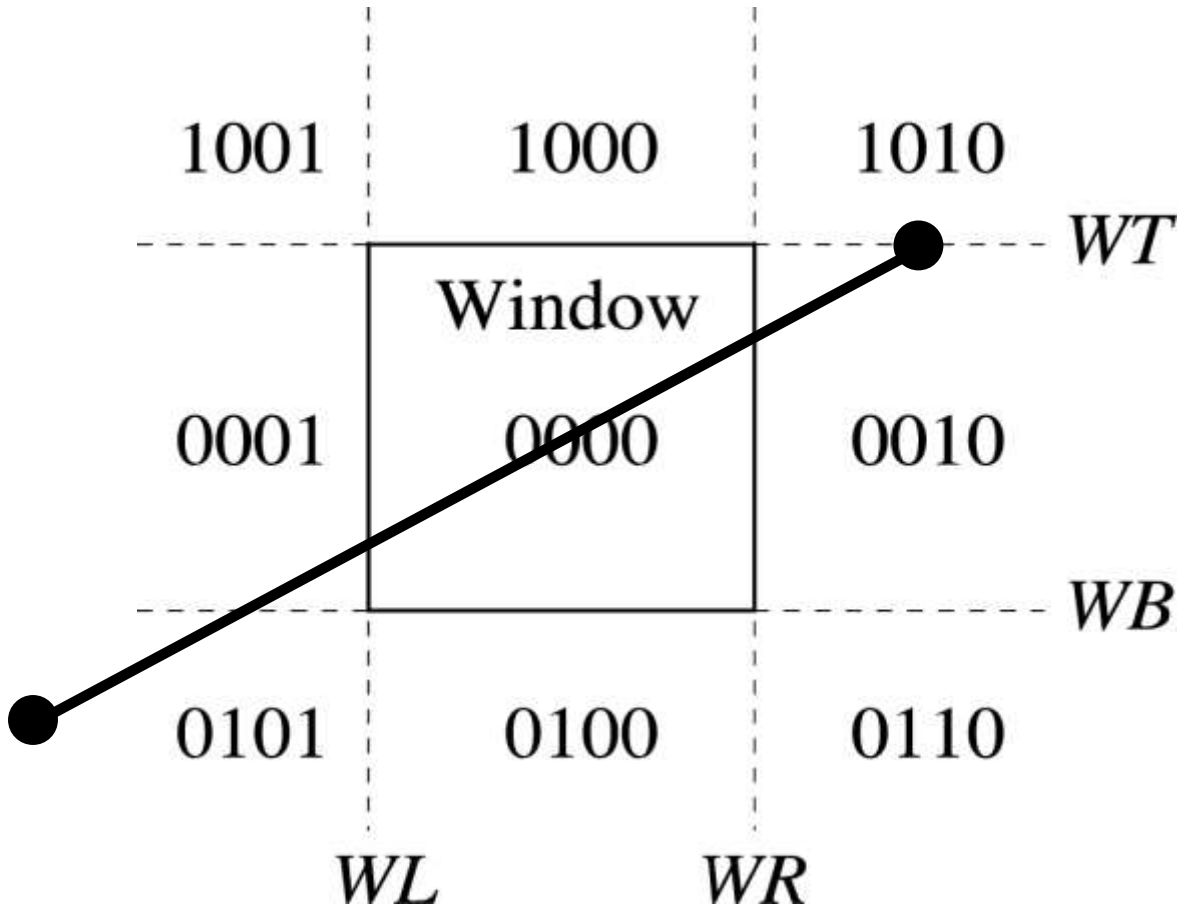
Cohen Sutherland

0101
1010



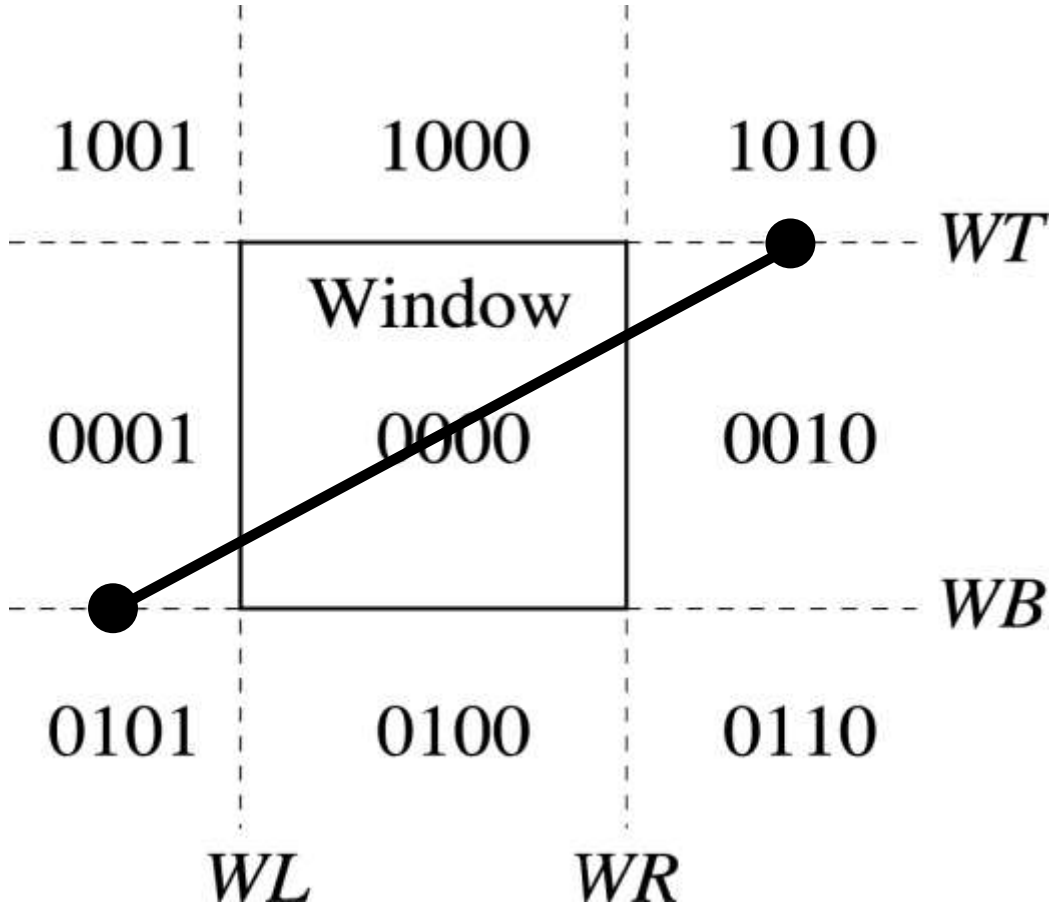
Cohen Sutherland

0101
0010



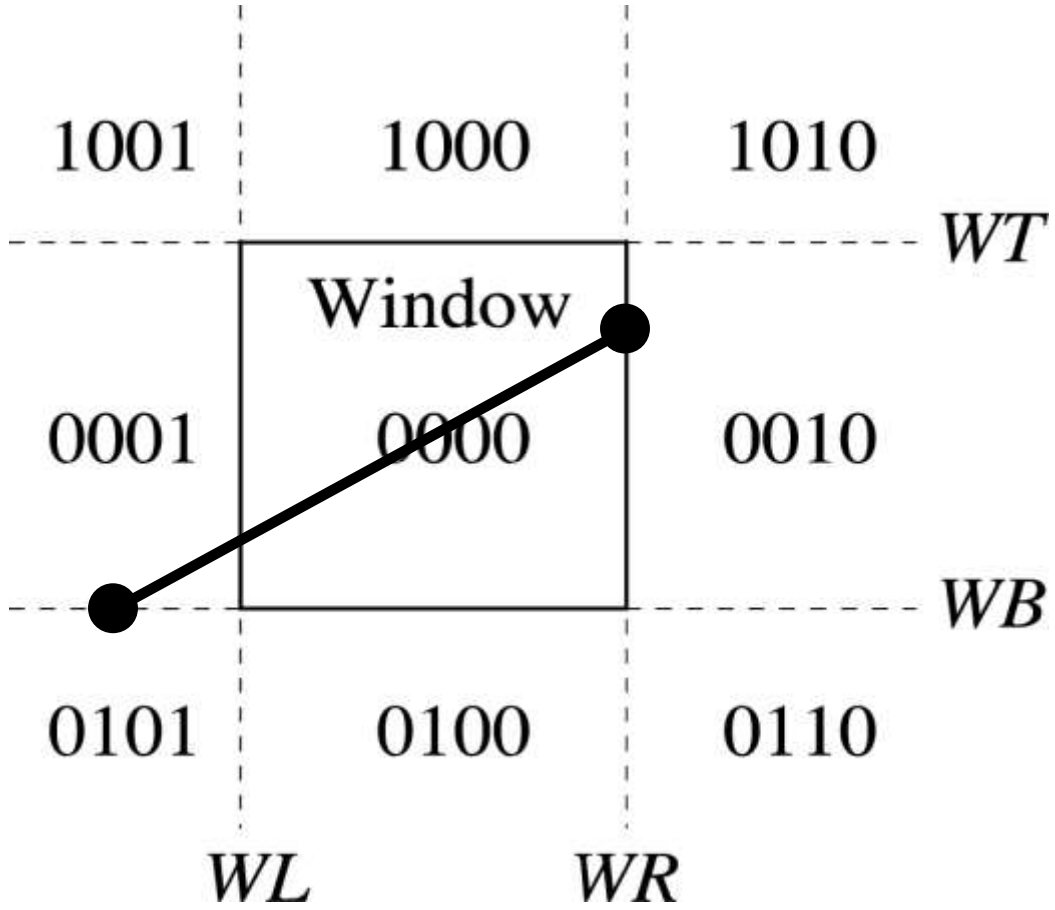
Cohen Sutherland

0001
0010



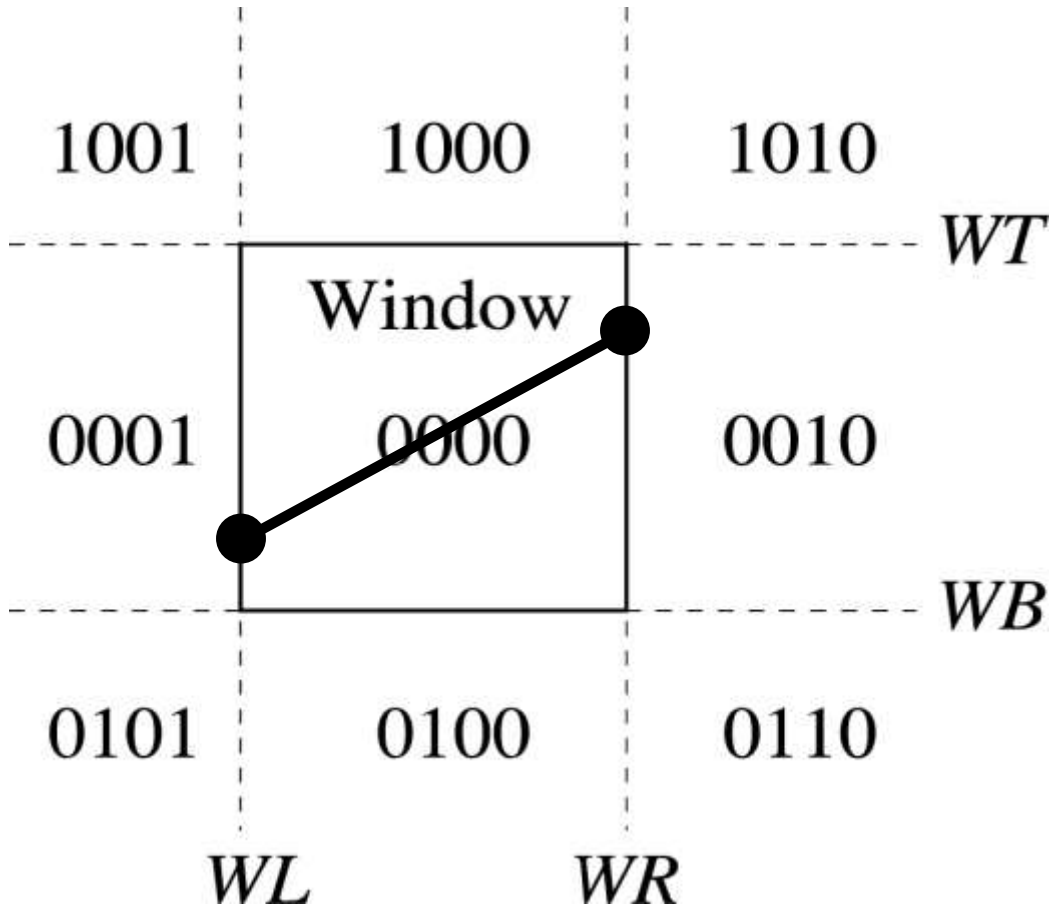
Cohen Sutherland

0001
0000



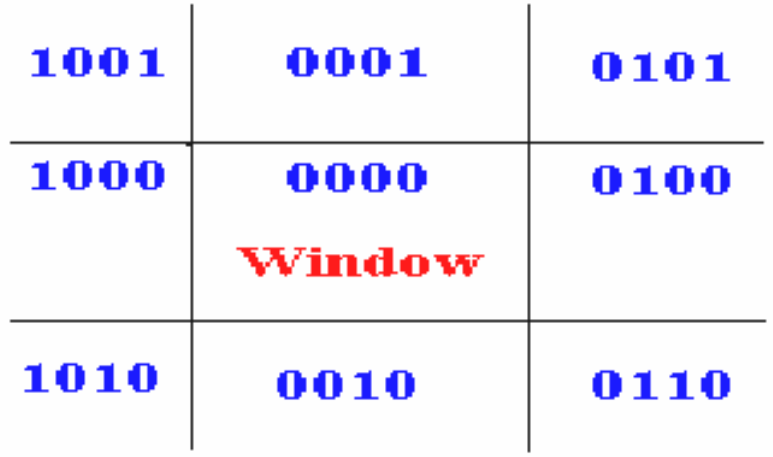
Cohen Sutherland

0000
0000



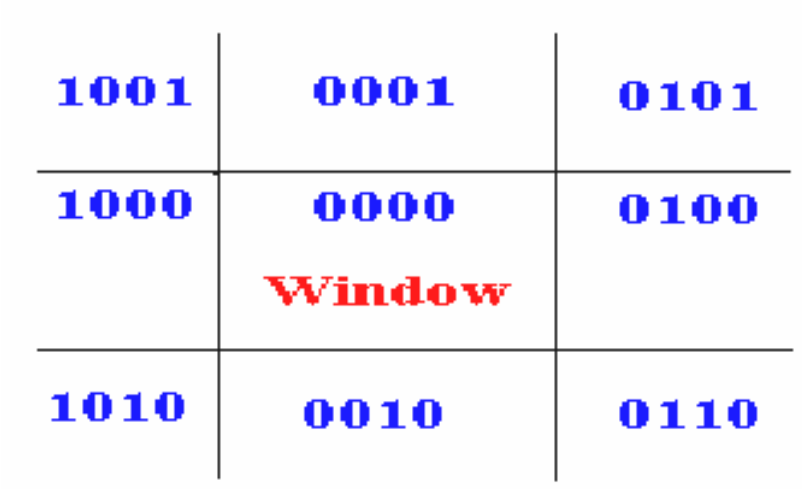
Cohen Sutherland Line Clipping Algorithm

- **Trivial Acceptance**
- If the logical OR is **zero**, the line can be trivially accepted.
- For example, if the endpoint codes are 0000 and 0000, the logical OR is 0000 - the line can be **trivially accepted.**
- If the endpoint codes are 0000 and 0110, the logical OR is 0110 and the **line can not be trivally accepted.**



Cohen Sutherland Line Clipping Algorithm

- **Trivial Rejection**
- If the logical AND of the endpoint codes is **not zero**, the line **can be trivally rejected.**
- For example, if an endpoint had a code of 1001 while the other endpoint had a code of 1010, the logical AND would be 1000 which indicates the line segment lies outside of the window.
- On the other hand, if the endpoints had codes of 1001 and 0110, the logical AND would be 0000, and the line **could not be trivally rejected.**



Cohen-Sutherland Algorithm (2/2)

- $y = y_0 + slope \times (x - x_0)$ and $x = x_0 + \left(\frac{1}{slope}\right) \times (y - y_0)$

- **Algorithm:**

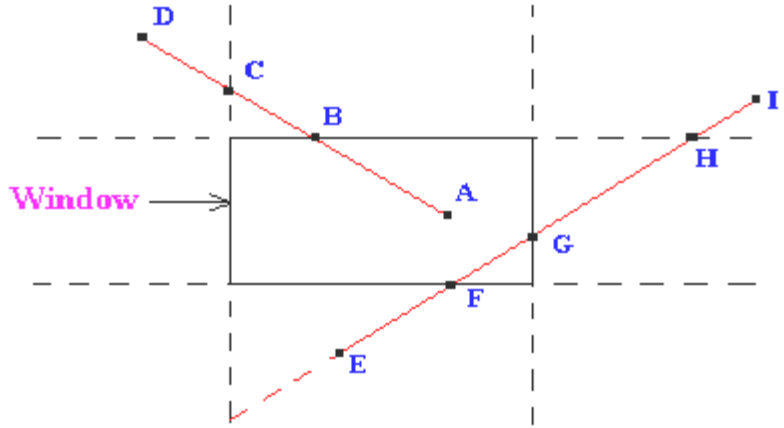
```
ComputeOutCode(x0, y0, outcode0);
ComputeOutCode(x1, y1, outcode1);
repeat
    check for trivial reject or trivial accept
    pick the point that is outside the clip rectangle

    if TOP then
        x = x0 + (x1 – x0) * (ymax – y0) / (y1 – y0);
        y = ymax;
    else if BOTTOM then
        x = x0 + (x1 – x0) * (ymin – y0) / (y1 – y0);
        y = ymin;
    else if RIGHT then
        y = y0 + (y1 – y0) * (xmax – x0) / (x1 – x0);
        x = xmax;
    else if LEFT then
        y = y0 + (y1 – y0) * (xmin – x0) / (x1 – x0);
        x = xmin;

    if (x0, y0 is the outer point) then
        x0 = x; y0 = y; ComputeOutCode(x0, y0,
    outcode0)
    else
        x1 = x; y1 = y; ComputeOutCode(x1, y1,
    outcode1)
until done
```

Cohen Sutherland Line Clipping Example

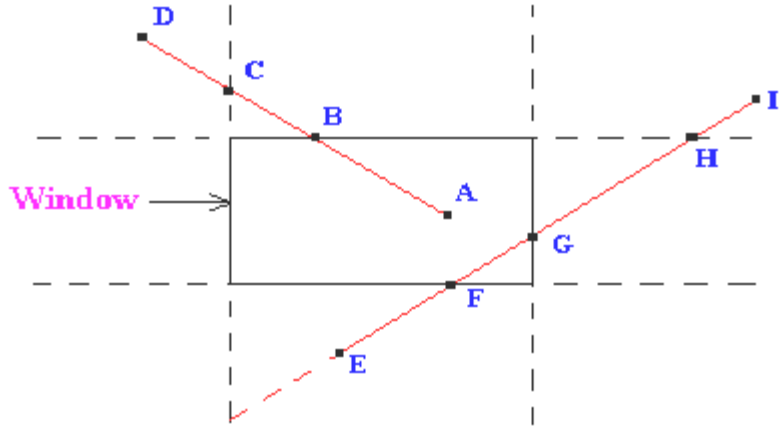
- Consider the line segment **AD**.
- Point **A** has an outcode of **0000** and point **D** has an outcode of **1001**.
- The logical AND of these outcodes is zero; therefore, the line **cannot be trivally rejected.**
- Also, the logical OR of the outcodes is not zero; therefore, the line **cannot be trivally accepted**.
- The algorithm then chooses **D** as the outside point (its outcode contains 1's).
- By our testing order, we first use the top edge to clip **AD** at **B**. The algorithm then recomputes **B**'s outcode as **0000**.
- With the next iteration of the algorithm, **AB** is tested and is trivially accepted and displayed.



1001	0001	0101
1000	0000 Window	0100
1010	0010	0110

Cohen Sutherland Line Clipping Example

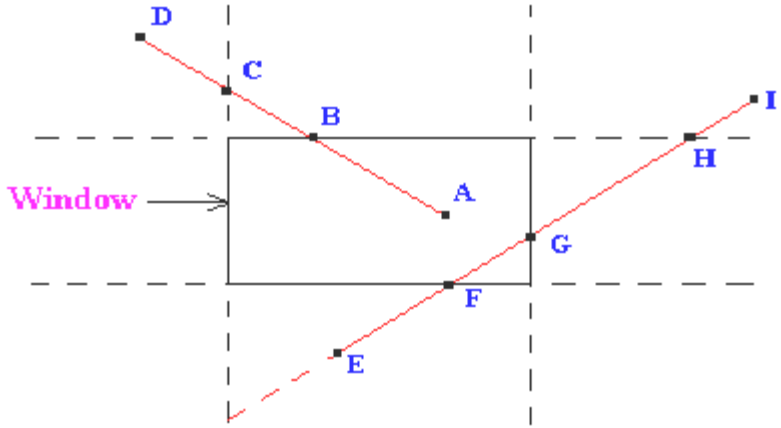
- Consider the line segment EI.
- Point E has an outcode of 0100, while point I's outcode is 1010.
- The results of the trivial tests show that the line can neither be trivally rejected or accepted.
- Point E is determined to be an outside point, so the algorithm clips the line against the bottom edge of the window.
- Now line EI has been clipped to be line FI.
- Line FI is tested and cannot be trivially accepted or rejected. Point F has an outcode of 0000, so the algorithm chooses point I as an outside point since its outcode is1010.



1001	0001	0101
1000	0000 Window	0100
1010	0010	0110

Cohen Sutherland Line Clipping Example

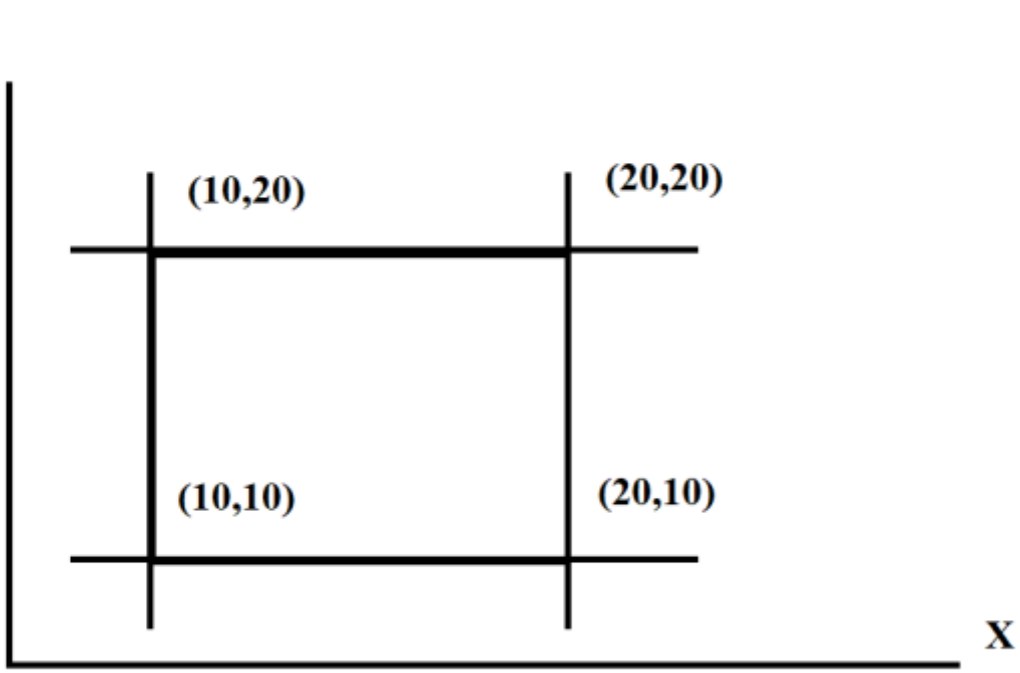
- Line FI is tested and cannot be trivially accepted or rejected.
- Point F has an outcode of 0000, so the algorithm chooses point I as an outside point since its outcode is1010.
- The line FI is clipped against the window's top edge, yielding a new line FH.
- Line FH cannot be trivally accepted or rejected. Since H's outcode is 0010, the next iteration of the algorthm clips against the window's right edge, yielding line FG.
- The next iteration of the algorithm tests FG, and it is trivially accepted and display.



1001	0001	0101
1000	0000 Window	0100
1010	0010	0110

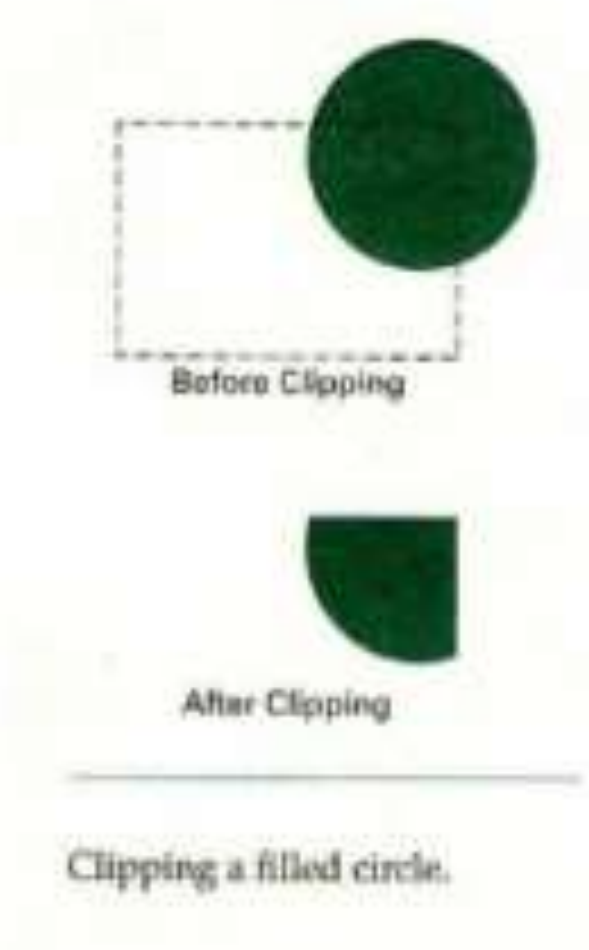
Cohen Sutherland Line Clipping Algorithm

- Numerical
- Find visible portion of the line between points
- P1(x0,y0) = (12,8)
- P2(x1,y1) = (15,15)



Curve Clipping

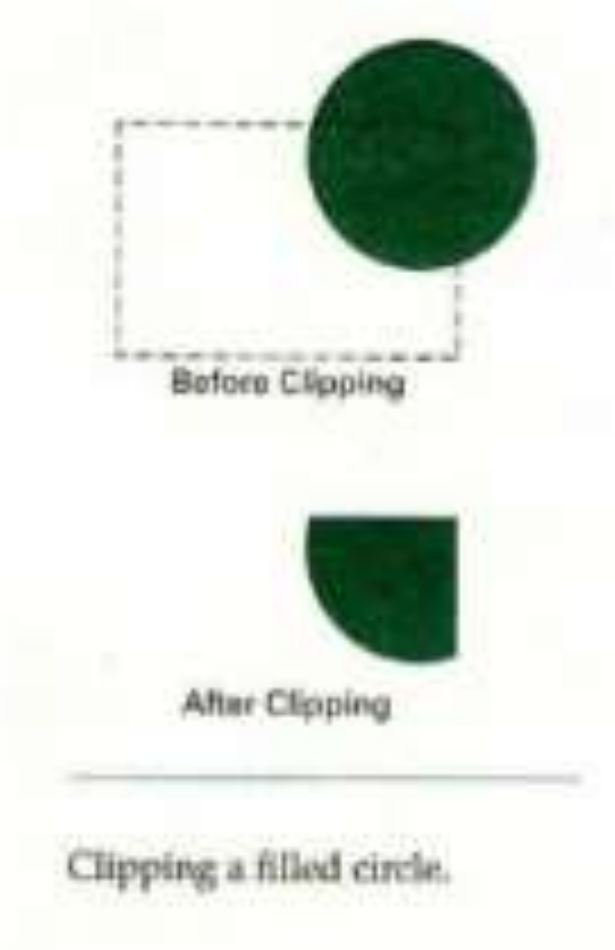
- Areas with curved boundaries can be clipped with methods similar to those discussed in the previous sections.
- Curve-clipping procedures will involve nonlinear equations, however, and this requires more processing than for objects with linear boundaries.



Clipping a filled circle.

Curve Clipping : Complete accept or reject

- If the bounding rectangle for the object is **completely inside** the window, we save the object.
- If the rectangle is determined to be **completely outside** the window, we discard the object.
- But if the bounding rectangle test fails, we can look for other computation-saving approaches.



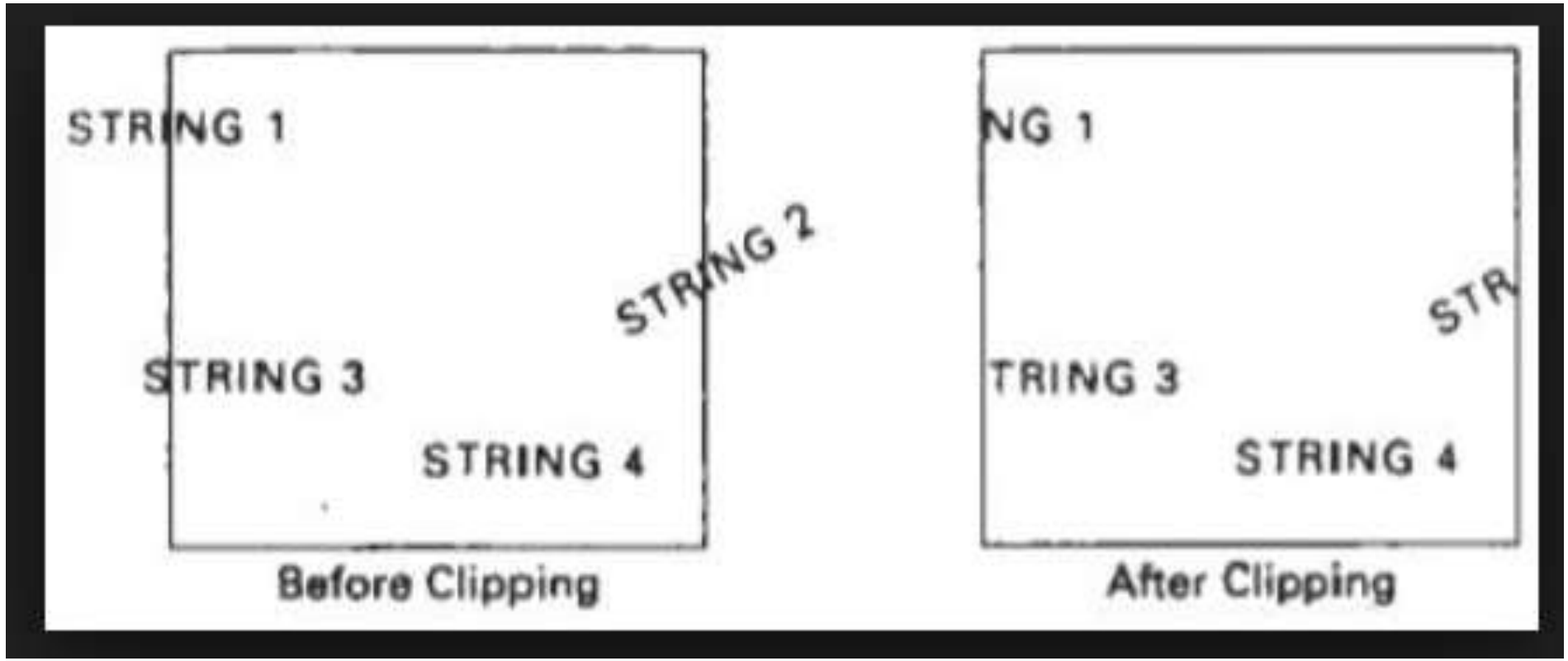
Clipping a filled circle.

Curve Clipping : Complete accept or reject

- For a circle,
- we can use the coordinate extents of individual quadrants and then octants for preliminary testing (outside or inside) and then calculate curve-window intersections.

Clipping a filled circle.

Text Clipping



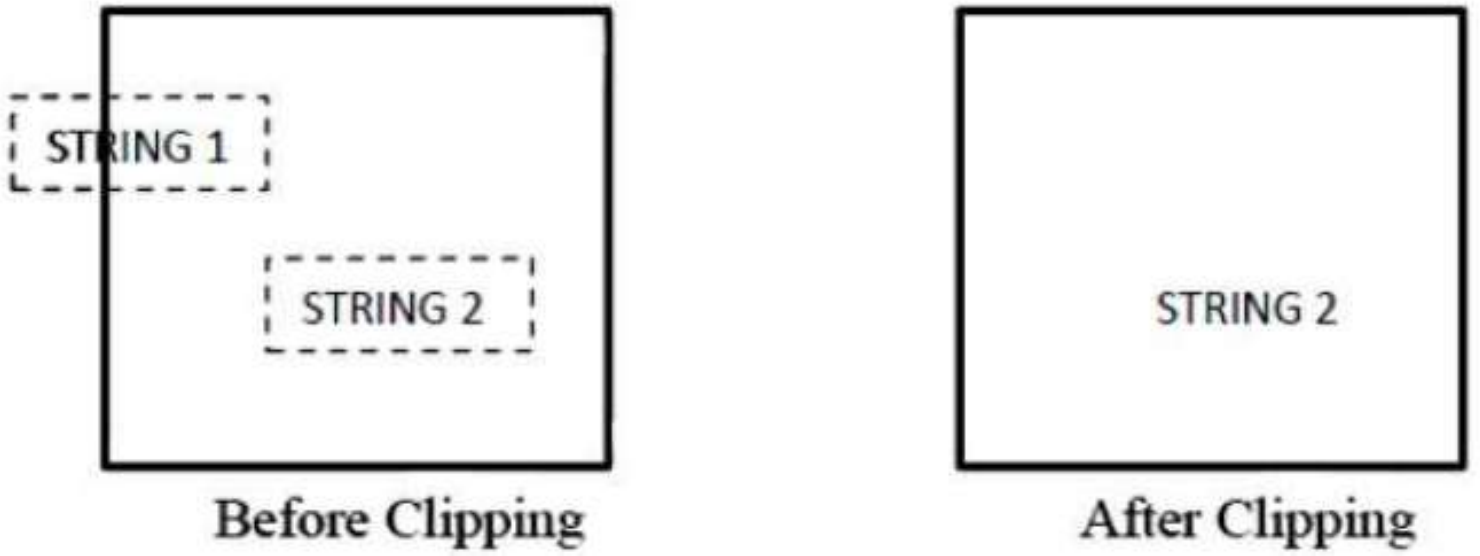
Text Clipping

- Various techniques are used to provide text clipping in a computer graphics.
- It depends on the methods used to generate characters and the requirements of a particular application.
- There are **three methods** for text clipping which are listed below –

1. **All or none string clipping**
2. **All or none character clipping**
3. **Text clipping**

All or none string clipping

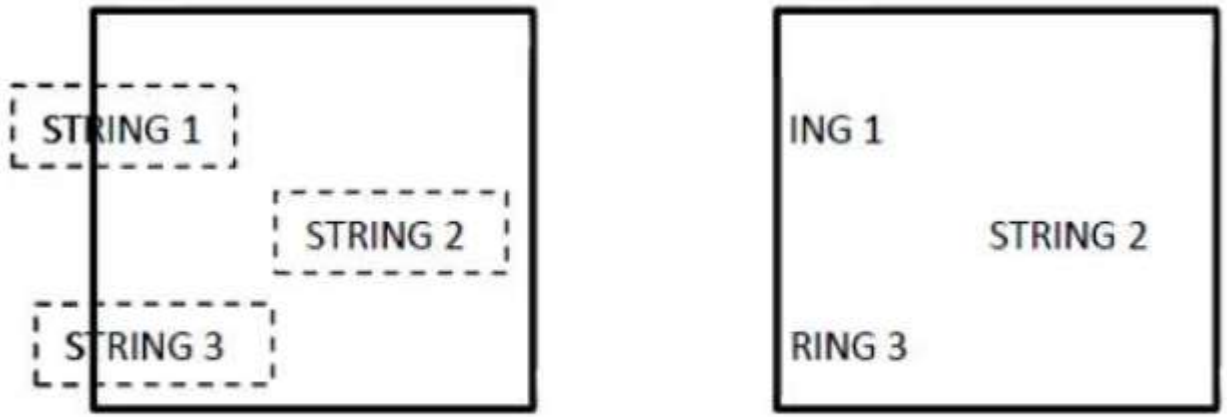
- In all or none string clipping method**, either we keep the entire string or we reject entire string** based on the clipping window.



All or none character clipping

In this method if the string is partially outside the window, then −

1. You reject only the portion of the string being outside
2. If the character is on the boundary of the clipping window, then we discard that entire character and keep the rest string.



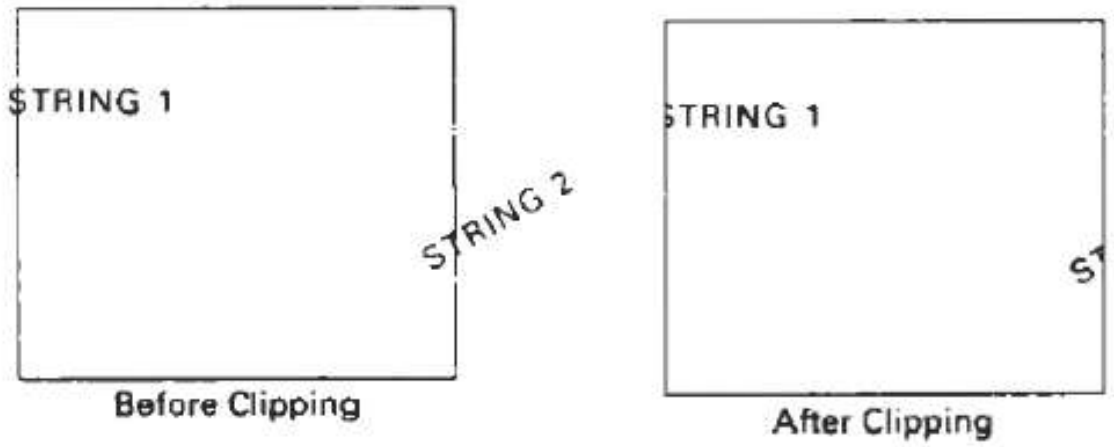
Before Clipping

After Clipping

Text clipping

- If it is partially outside the window, then

1. You reject only the portion of string being outside.
2. If the character is on the boundary of the clipping window, then we discard only that portion of character that is outside of the clipping window.



C H A P T E R *6* Two-Dimensional Viewing Donal D. Hearn and M. Pauline Baker

1. Windowing And Clipping: Viewing Pipeline
2. Viewing Transformations
3. 2-D Clipping Algorithms
    1. Line Clipping Algorithms Cohen Sutherland Line Clipping Algorithm
    2. Liang Barsky Algorithm
    3. Line Clipping Against Non Rectangular Clip Windows
4. Polygon Clipping
    1. Sutherland Hodgeman Polygon Clipping
    2. Weiler And Atherton Polygon Clipping
5. Curve Clipping
6. Text Clipping

C H A P T E R *6* Two-Dimensional Viewing Donal D. Hearn and M. Pauline Baker

1. 2-D Clipping Algorithms
    1. Liang Barsky Algorithm
    2. Line Clipping Against Non Rectangular Clip Windows
2. Polygon Clipping
    1. Sutherland Hodgeman Polygon Clipping
    2. Weiler And Atherton Polygon Clipping

Liang – Barsky clipping

- In computer graphics, the **Liang–Barsky algorithm** (named after You-Dong Liang and Brian A. Barsky) is a line clipping algorithm.
- The Liang–Barsky algorithm uses the parametric equation of a line and inequalities describing the range of the clipping window to determine the intersections between the line and the clipping window. With these intersections it knows which portion of the line should be drawn.
- This algorithm is significantly more efficient than Cohen–Sutherland.

Liang-Barsky Algorithm

- The **Liang-Barsky algorithm** is a line clipping algorithm.
- This algorithm is more efficient than Cohen–Sutherland line clipping algorithm and can be extended to 3-Dimensional clipping.
- This algorithm is considered to be the faster parametric line-clipping algorithm.

Liang-Barsky Algorithm

- The following concepts are used in this clipping:
- The parametric equation of the line.
- X = x0 + u(x1-x0)
- Y = y0 + u(y1-y0)
- The inequalities describing the range of the clipping window which is used to determine the intersections between the line and the clip window.
- xwmax <= x0 + u(x1-x0) <= xwmin
- ywmax <= y0 + u(y1-y0) <= ywmin

LIANG - BARSKY ALGORITHM

- THIS ALGORITHM USES PARAMETRIC EQUATIONS OF LINE TO CALCULATE AND FIND OUT INTERSECTION POINTS AND CLIP THE LINE LYING OUTSIDE IT.

NOTE: ① $U_{min} = 0$; $U_{max} = 1$ ... (BY DEFAULT)

② $U_K = \frac{q_K}{p_K}$ .... ('K' is constant)

We consider values for 'k' and find multiple values of U_K.

③ $U_{min} \leq U_K \leq U_{max}$ i.e $0 \leq U_K \leq 1$

USE FOLLOWING INEQUALITIES TO CALCULATE q_K and p_K.

$p_0 = -\Delta x$; $q_0 = x_0 - x_{min}$

$p_1 = \Delta x$; $q_1 = x_{max} - x_0$

$p_2 = -\Delta y$; $q_2 = y_0 - y_{min}$

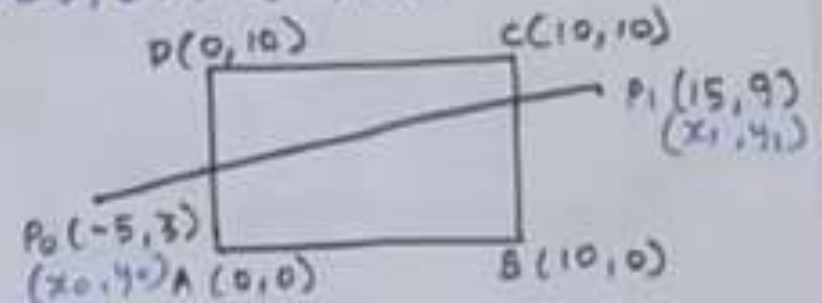
$p_3 = \Delta y$; $q_3 = y_{max} - y_0$

④ TO CALCAUTE INTERSECTION PTS.

$x = x_0 + U\Delta x$; $x_{min} \leq x_0 + U\Delta x \leq x_{max}$

$y = y_0 + U\Delta y$; $y_{min} \leq y_0 + U\Delta y \leq y_{max}$

Q.) LET ABCD BE THE RECTANGULAR WINDOW WITH A(0,0), B(10,0); C(10,10); D(0,10). USE LIANG BARSKY ALGORITHM TO CLIP THE LINE P_0P_1 WITH $P_0(-5,3)$; $P_1(15,9)$.



⇒ **STEP 1**: PLOT THE POINTS.

$x_{min} = 0$; $x_{max} = 10$

$y_{min} = 0$; $y_{max} = 10$

STEP 2: $\Delta x = x_1 - x_0 = 15 - (-5) = 20$

$\Delta y = y_1 - y_0 = 9 - 3 = 6$

STEP 3: $u_k = q_k / p_k$

consider $k = 0, 1, 2, 3$.

$\therefore \quad u_0 = \frac{q_0}{p_0} = \frac{x_0 - x_{min}}{-\Delta x} = \frac{-5-0}{-20} = \frac{1}{4} = 0.25$

$u_1 = \frac{q_1}{p_1} = \frac{x_{max} - x_0}{\Delta x} = \frac{10+5}{20} = \frac{15}{20} = \frac{3}{4} = 0.75$

$u_2 = \frac{q_2}{p_2} = \frac{y_0 - y_{min}}{-\Delta y} = \frac{3-0}{-6} = -\frac{3}{6} = -0.5$

$u_3 = \frac{q_3}{p_3} = \frac{y_{max} - y_0}{\Delta y} = \frac{10-3}{6} = \frac{7}{6} = 1.16$

CONSIDER VALUES OF U_K IF IT SATISFIES

$u_{min} \leq U_k \leq u_{max}$; i.e) $0 \leq U_K \leq 1$.

$\therefore$ WE CONSIDER $U_0 = 0.25$ AND $U_1 = 0.75$

STEP 4: CALCULATE INTERSECTION POINTS

① When $U = 0.25$

$x = x_0 + U\Delta x$

$\therefore x = -5 + 0.25(20) = 0$

$y = y_0 + U\Delta y$

$= 3 + 0.25(6) = 4.5$

$\therefore (x, y) = (0, 4.5)$

② When $U = 0.75$

$x = x_0 + U\Delta x$

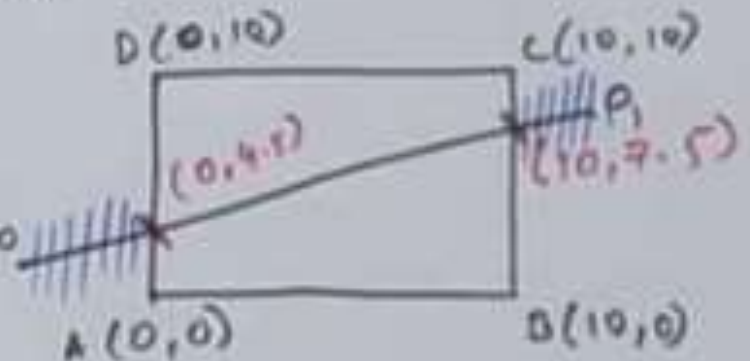
$x = -5 + 0.75(20) = 10$

$y = y_0 + U\Delta y$

$= 3 + 0.75(6) = 7.5$

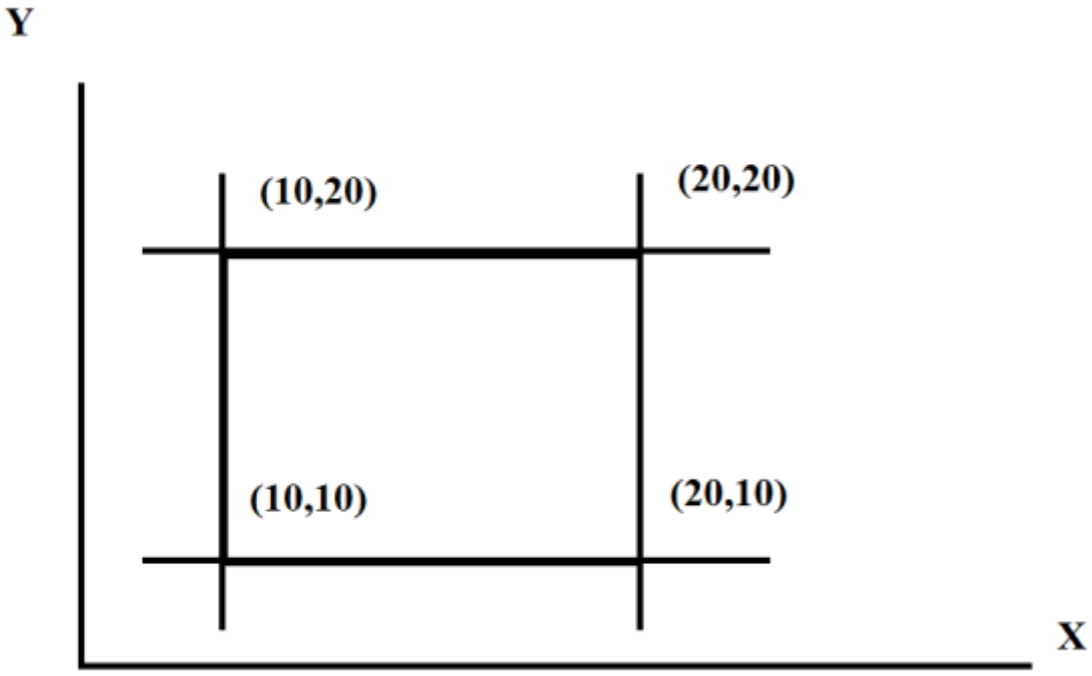
$\therefore (x, y) = (10, 7.5)$.

STEP 5: Plot intersection points and clip the line.



Liang-Barsky Algorithm

- Numerical
- Find visible portion of the line between points
- P1(x0,y0) = (6,12)
- P2(x1,y1) = (24,18)



- P1(x0,y0) = (12,30)
- P2(x1,y1) = (16,4)

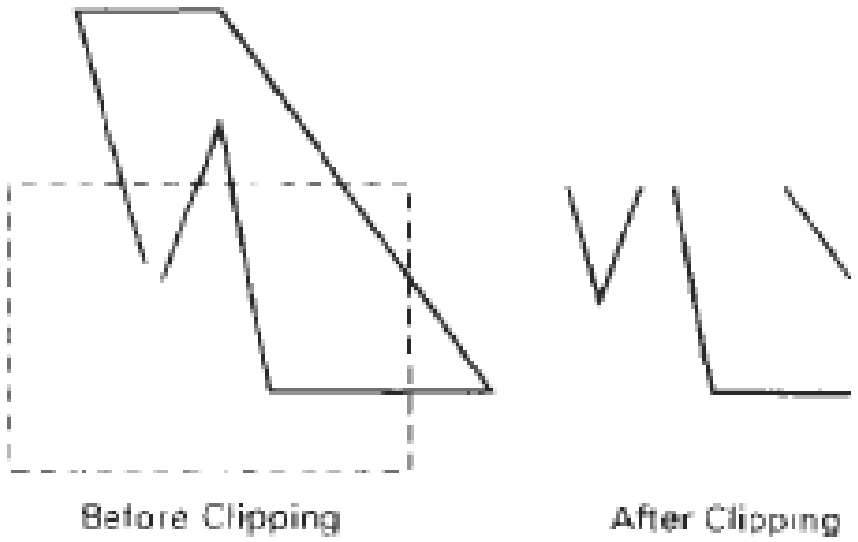
Polygon Clipping

Polygon Clipping

Polygon Clipping

Polygon Clipping

- For polygon clipping, we require an algorithm that will generate one or more closed areas that are then scan converted for the appropriate area fill.
- What we really want to display is a bounded area after clipping.



Sutherland Hodgeman Polygon Clipping

- Beginning with the initial set of polygon vertices, we could first clip the polygon against the left rectangle boundary to produce a new sequence of vertices.
- The new set of vertices could then k successively passed to a right boundary clipper, a bottom boundary clipper, and a top boundary clipper.

Four Cases Out-In, In-In, In-out, Out-out

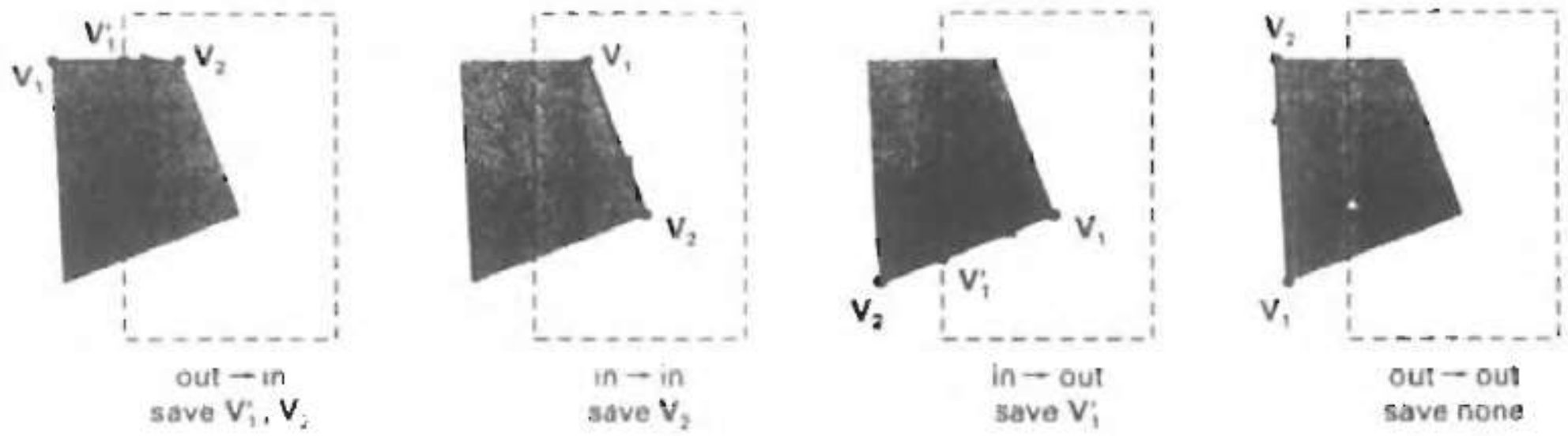


Figure 6-20
Successive processing of pairs of polygon vertices against the left window boundary.

Sutherland Hodgeman Polygon Clipping

As each pair of adjacent polygon vertices is passed to a window boundary clipper, we make the following tests:

(1) Out-In - If the first vertex is outside the window boundary and the second vertex is inside, both the **intersection point of the polygon edge with the window boundary and the second vertex** are added to the output vertex list.

(2) In-In If both input vertices are inside the window boundary, **only the second vertex** is added to the output vertex list.

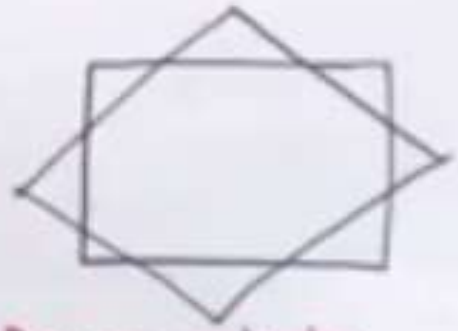
(3) In-Out- If the first vertex is inside the window boundary and second vertex is outside the boundary, **only the edge intersection with the window boundary is added** to the output vertex list.

(4) Out-Out- If both input vertices are outside the window boundary, **nothing** is added to the output list.

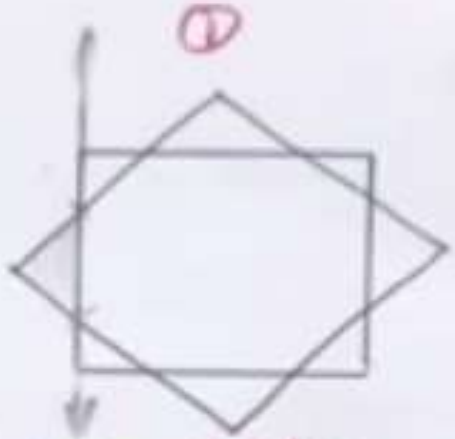
Example

o ORDER OF EDGES IS DECIDED AS LRBT OR ANY ORDER AS RBLT

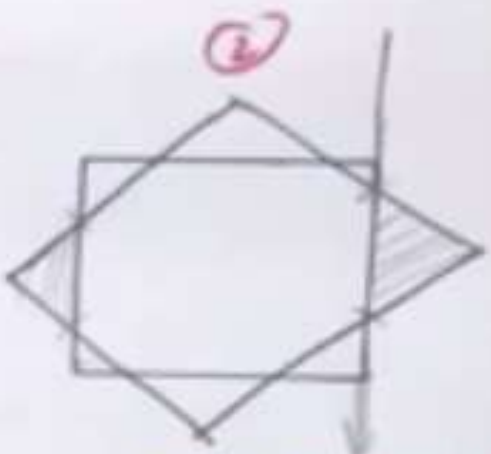
STEPS



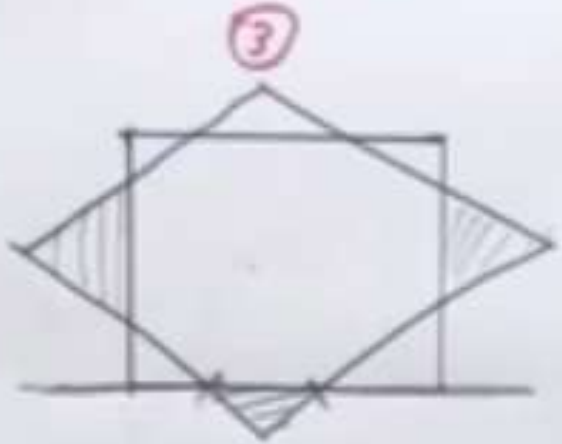
Before clipping



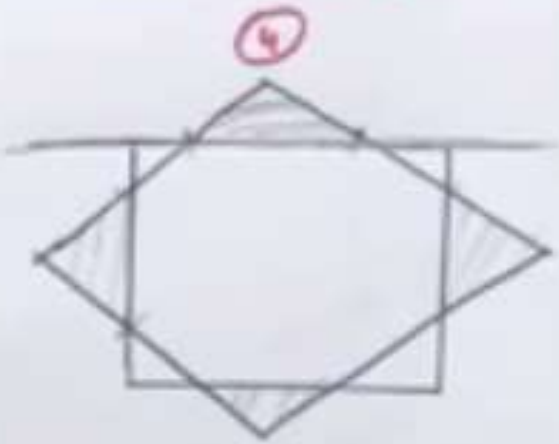
LEFT clipping



RIGHT clipping



BOTTOM clipping

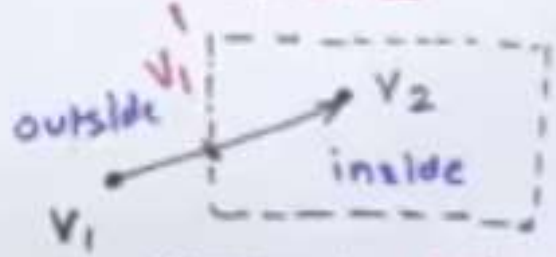


TOP clipping

After clipping

- TO FIND NEW SEQUENCE OF VERTICES, THERE ARE FOUR CASES TO BE CONSIDERED.

CASE I

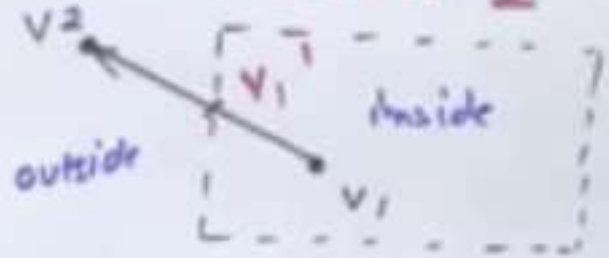


movement → out → in

O/P → Intersect Pt + Destination Vertex

→ $V_1' V_2$

CASE 2

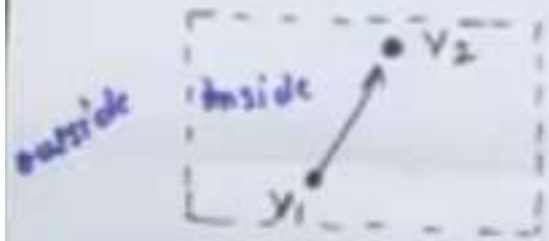


movement → in → out

O/P → Intersection Pt

V_1'

CASE 3

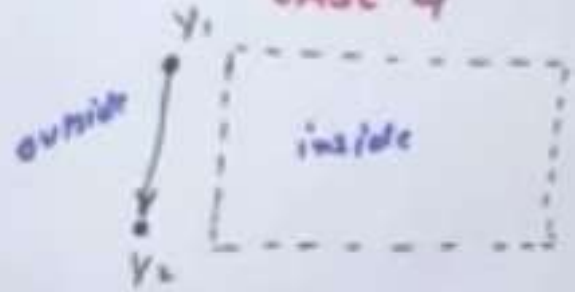


movement → in → in

O/P → Destination Vertex

V_2

CASE 4

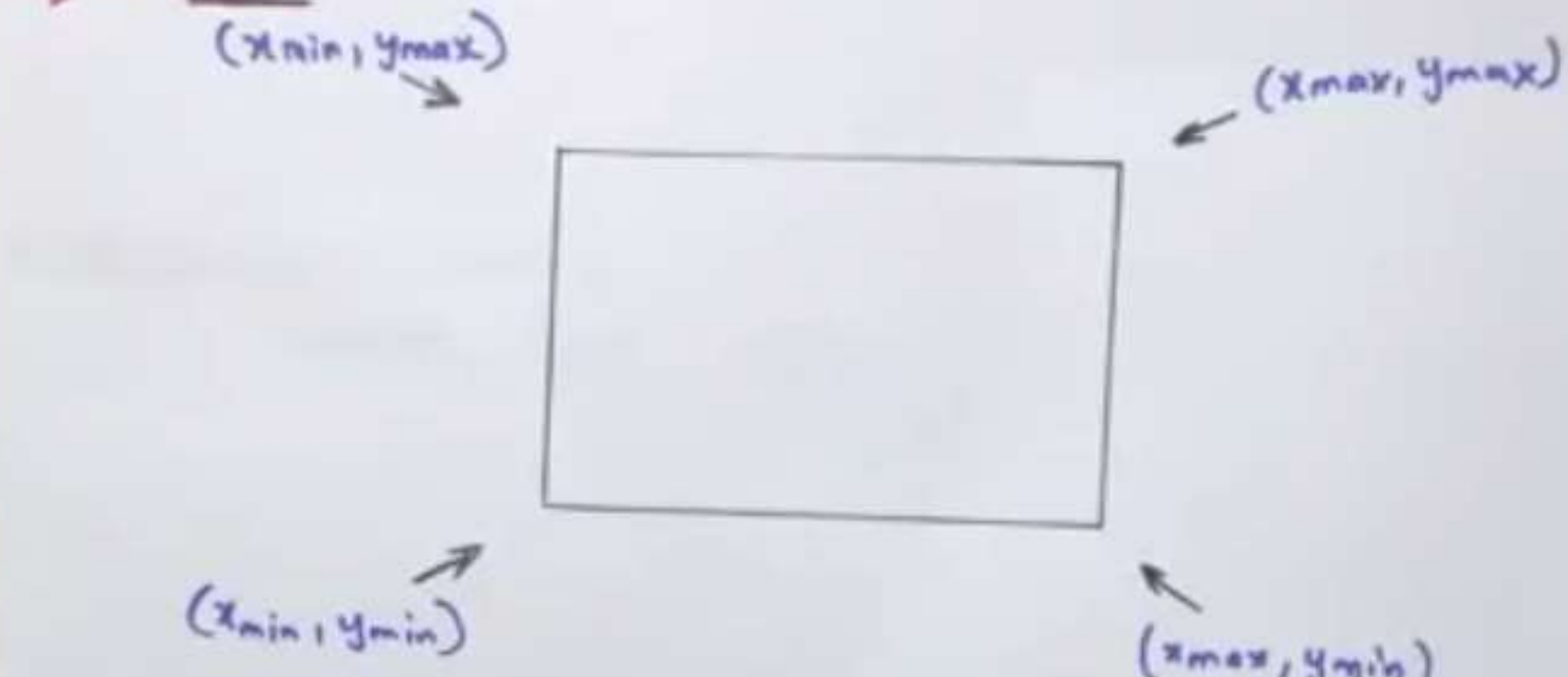


movment → out → out

O/P → None

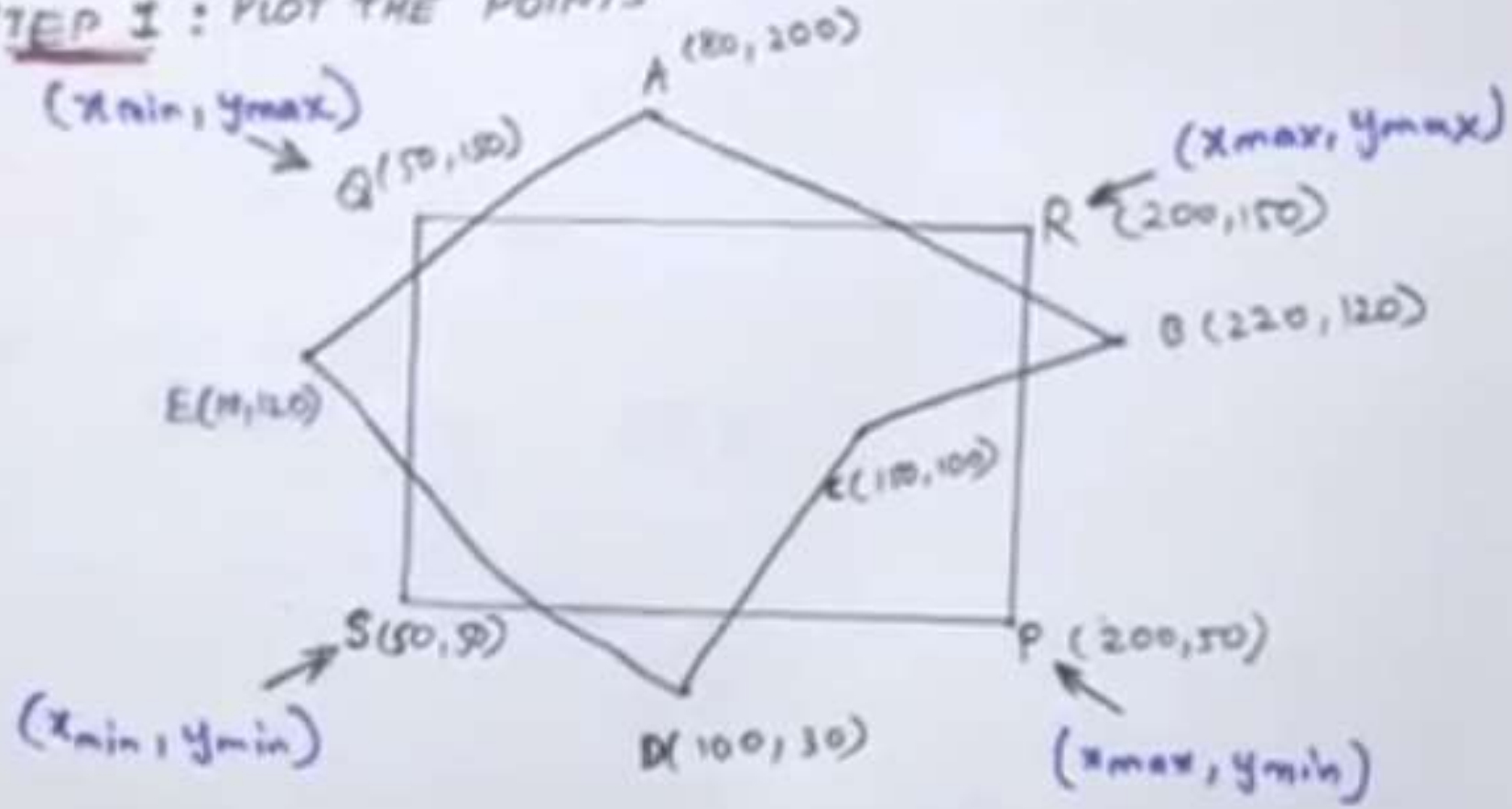
Q.) CLIP POLYGON ABCDE AGAINST WINDOW PQRS. THE CO-ORDINATES OF THE POLYGON ARE A(80,200); B(220,120); C(150,100); D(100,30); E(10,120). CO-ORDINATES OF THE WINDOW ARE P(200,50); Q(50,150); R(200,150); S(50,50)

⇒ STEP I : PLOT THE POINTS



Q.) CLIP POLYGON ABCDE AGAINST WINDOW PQRS. THE CO-ORDINATES OF THE POLYGON ARE A(80, 200), B(220, 120), C(150, 100), D(100, 30), E(10, 120). CO-ORDINATES OF THE WINDOW ARE P(200, 50), Q(50, 150), R(200, 150), S(50, 50)

⇒ STEP I : PLOT THE POINTS



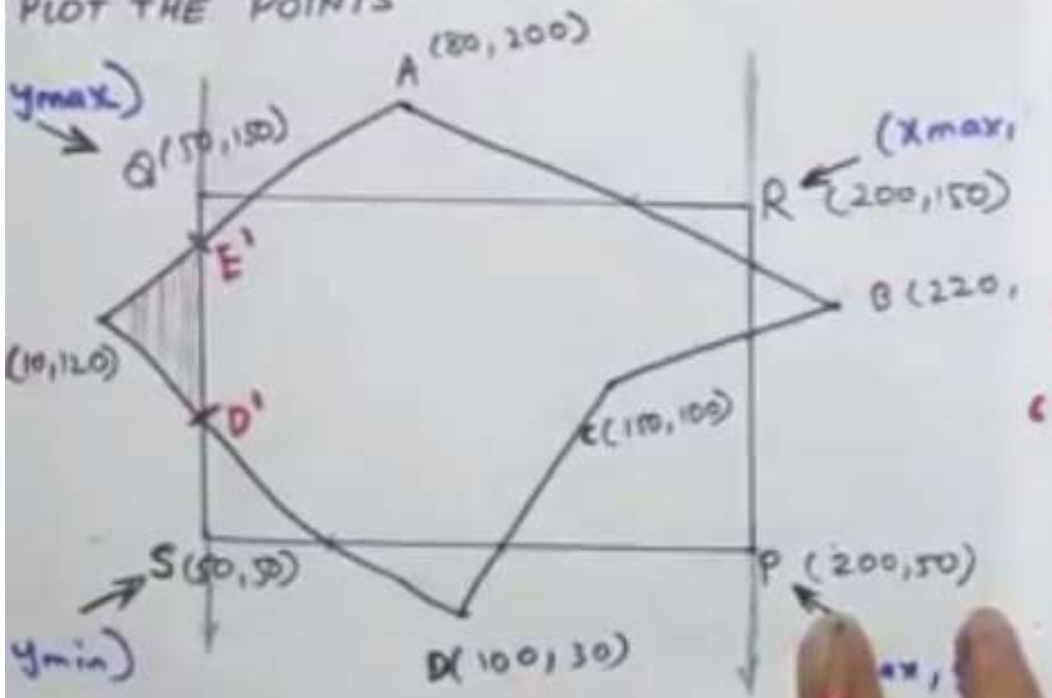
- STEP 2: CLIPPING AGAINST LEFT EDGE OF THE WINDOW USING LEFT CLIPPER

VERTEX	CASE	O/P

- STEP 3: RIGHT CLIPPING

VERTEX	CASE	O/P

ON ABCDE AGAINST WINDOW PQRS. TH
THE POLYGON ARE A(80,200); B(220,120)
100,30); E(10,120). CO-ORDINATES OF THE W
(50,150); R(200,150); S(50,50)
PLOT THE POINTS



- STEP 2: CLIPPING AGAINST LEFT EDGE OF THE WINDOW
LEFT CLIPPER

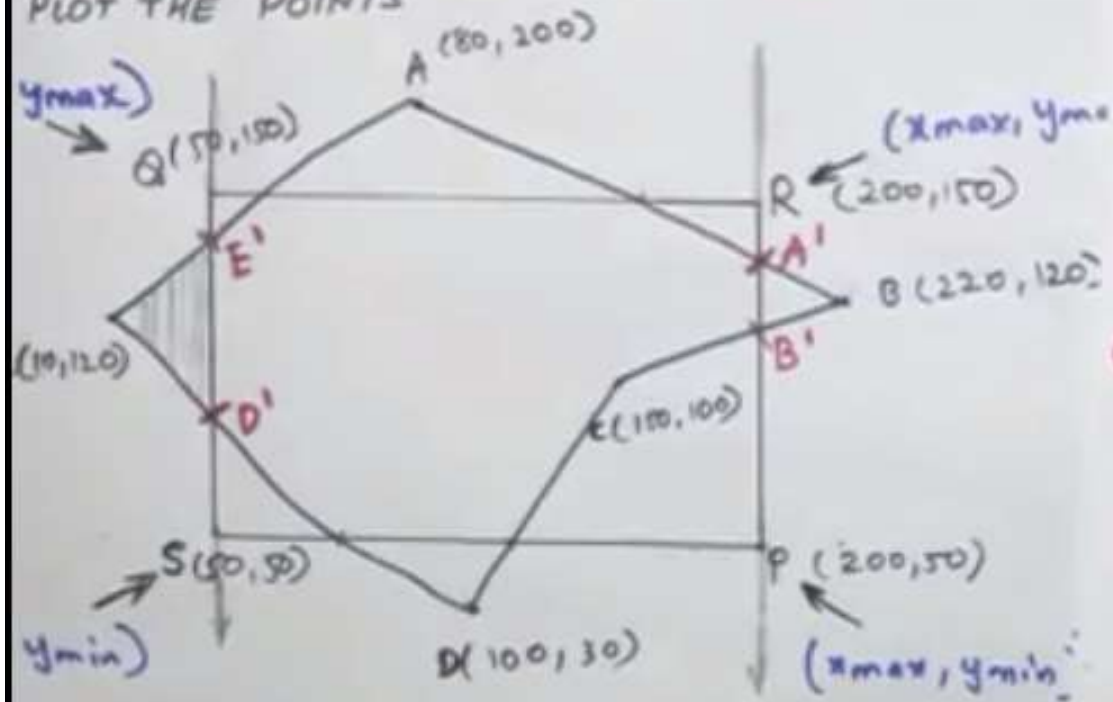
VERTEX	CASE	O/P
AB	in → in	B
BC	in → in	C
CD	in → in	D
DE	in → out	D'
EA	out → in	E' A

} New vertices

- STEP 3: RIGHT CLIPPING

VERTEX	CASE	O/P

ON ABCDE AGAINST WINDOW PQRS. THE
THE POLYGON ARE A(80,200); B(220,120);
100,30); E(10,120). CO-ORDINATES OF THE WIND
(50,150); R(200,150); S(50,50)
PLOT THE POINTS



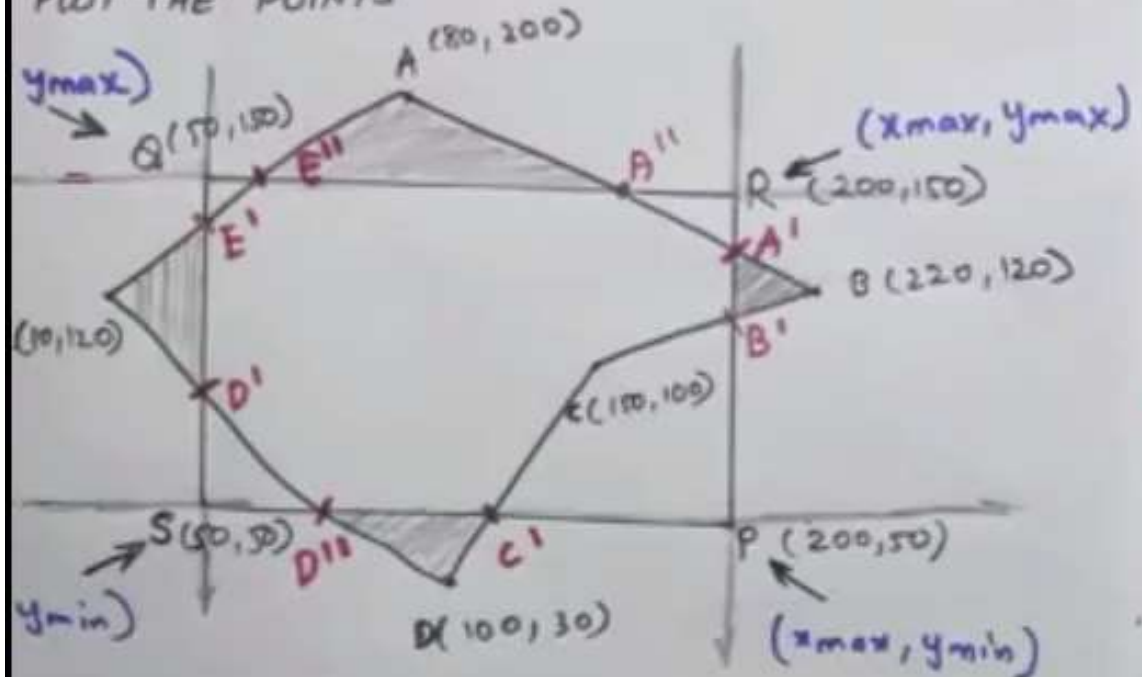
- STEP 2: CLIPPING AGAINST LEFT EDGE OF THE WINDO
LEFT CLIPPER

VERTEX	CASE	O/P
AB	in → in	B
BC	in → in	C
CD	in → in	D
DE	in → out	D' } New vertices
EA	out → in	E' A } New vertices

- STEP 3: RIGHT CLIPPING

VERTEX	CASE	O/P
AB	in → out	A' } New vertices
BC	out → in	B' C } New vertices
CD	in → in	D
DD'	in → in	D'
D'E'	in → in	E'
E'A	in → in	A

GON ABCDE AGAINST WINDOW PQRS. THE CO-
THE POLYGON ARE A(80,200); B(220,120);
100,30); E(10,120). CO-ORDINATES OF THE WINDOW
(50,150); R(200,150); S(50,50)
PLOT THE POINTS



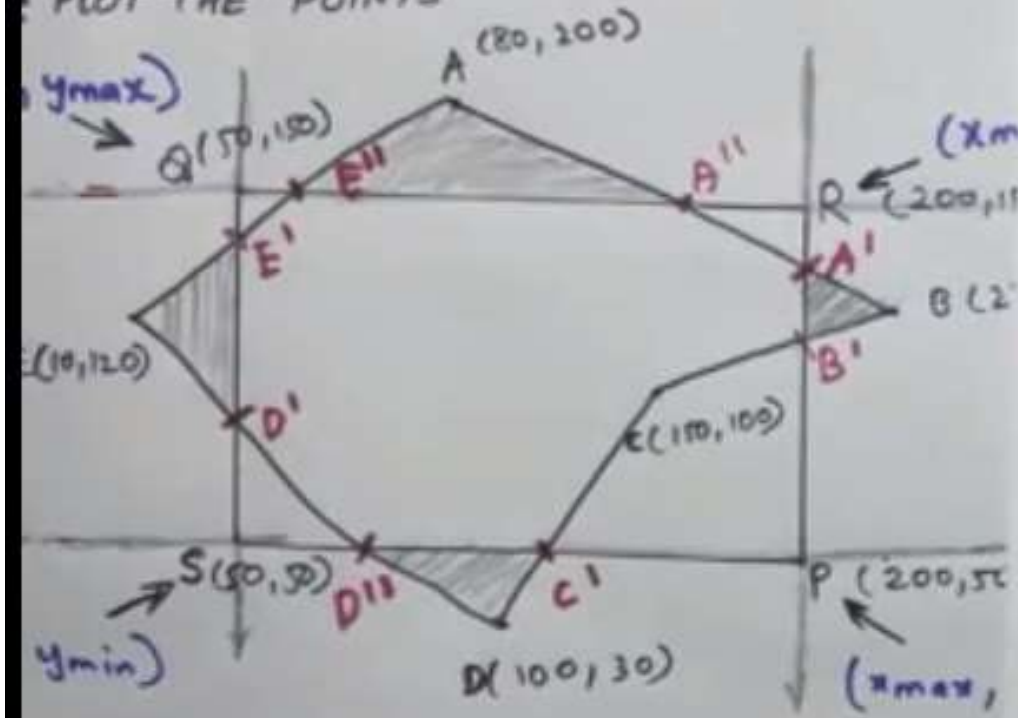
- **STEP 4: BOTTOM CLIPPING**

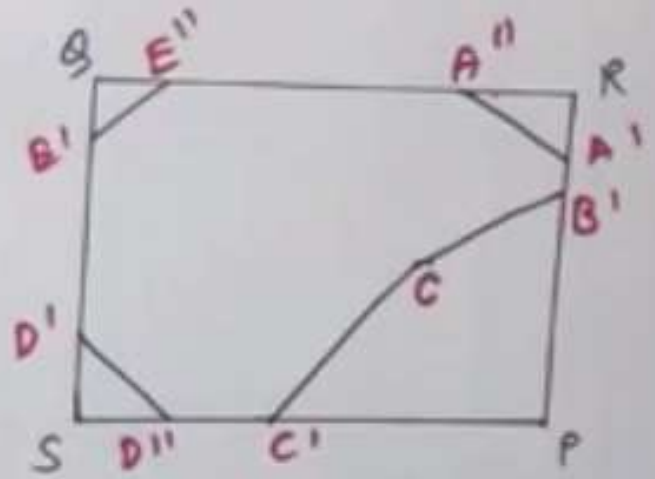
VERTEX	CASE	O/P
AA'	in → in	A'
A'B'	in → in	B'
B'C	in → in	C
CD	in → out	C'
DD'	out → in	D'' D' (New vertices)
D'E'	in → in	E'
E'A	in → in	A

- **STEP 5: TOP CLIPPING**

VERTEX	CASE	O/P
AA'	out → in	A'' A'
A'B'	in → in	B'
B'C	in → in	C
CC'	in → in	C'
C'D''	in → in	D''
D''D'	in → in	D'
D'E'	in → in	E'
E'A	in → out	E'' (New vertices.)

GON ABCDE AGAINST WINDOW PQRS.
THE POLYGON ARE A(80,200); B(220,1
(100,30); E(10,120). CO-ORDINATES OF THE
3(50,150); R(200,150); S(50,50)
PLOT THE POINTS





Weiler And Atherton Polygon Clipping

References

- https://www.youtube.com/watch?v=RGSnlK4-BhI
- https://www.cs.helsinki.fi/group/goa/viewing/leikkaus/lineClip.html

  For Liang Barsky Algorithm
- https://www.youtube.com/watch?v=N5Sxq3kuc_0 Weiler And Atherton Polygon Clipping
- https://www.youtube.com/watch?v=LCMyWFxeuro&list=PLsyTslBOa4NwekHdOyE1Dg8eV4lJsyyd7