Window to Viewport Transformation & Clipping

Computer Graphics : Clipping

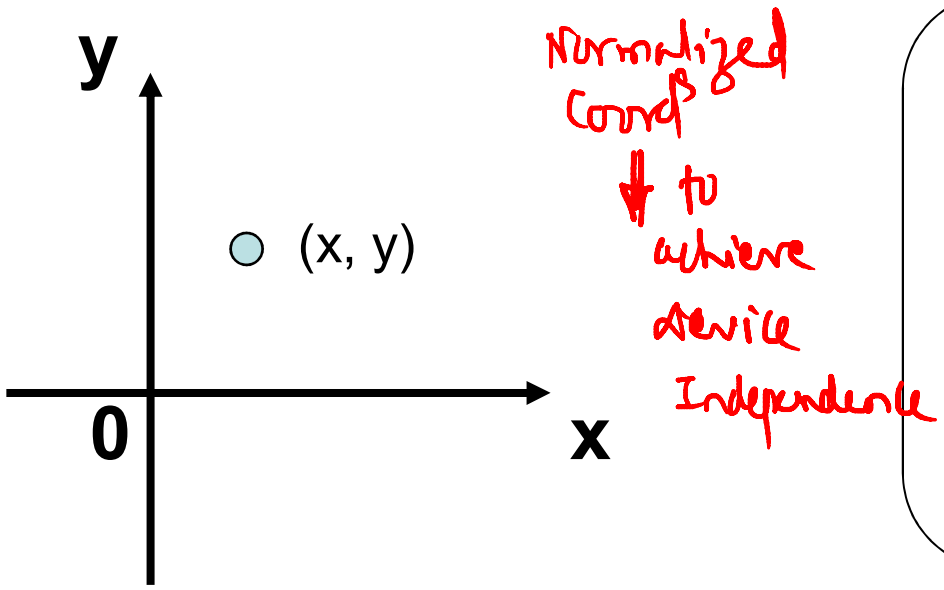
Modelling Coordinates (Local Coordinates): Are used to define individual objects in the scene

World Coordinates System (WCS): Are we to define the entire picture (scene)

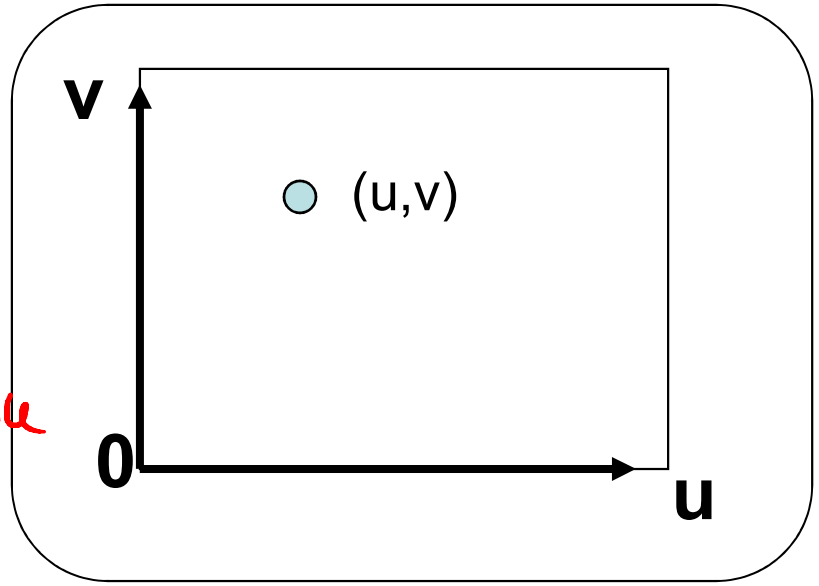
In theory is regarded as supporting an extent from $-\infty$ to $+\infty$

Device Coord System: pertains to the dimensions offered by the device

Coordinate Systems



Normalized Coords to achieve device Independence

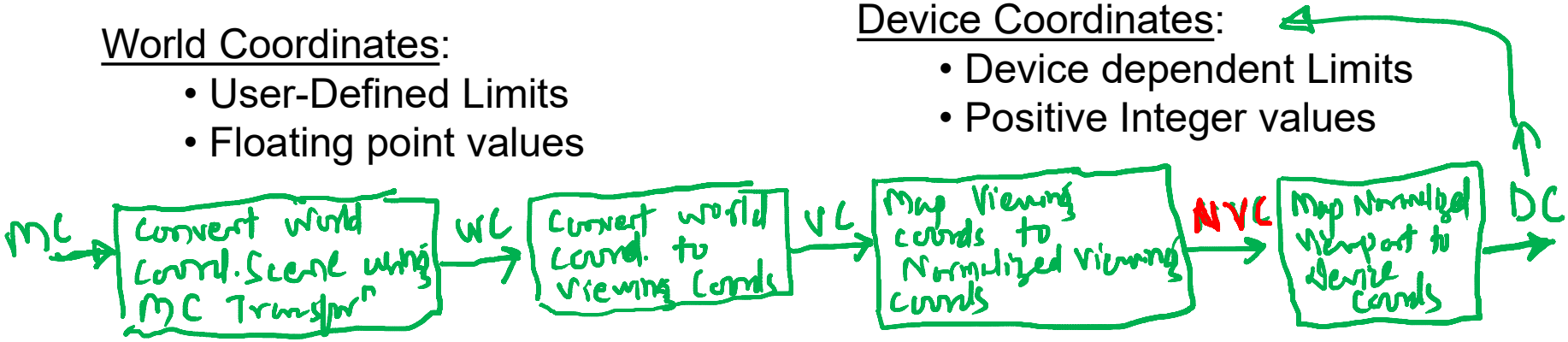


World Coordinates:

- User-Defined Limits
- Floating point values

Device Coordinates:

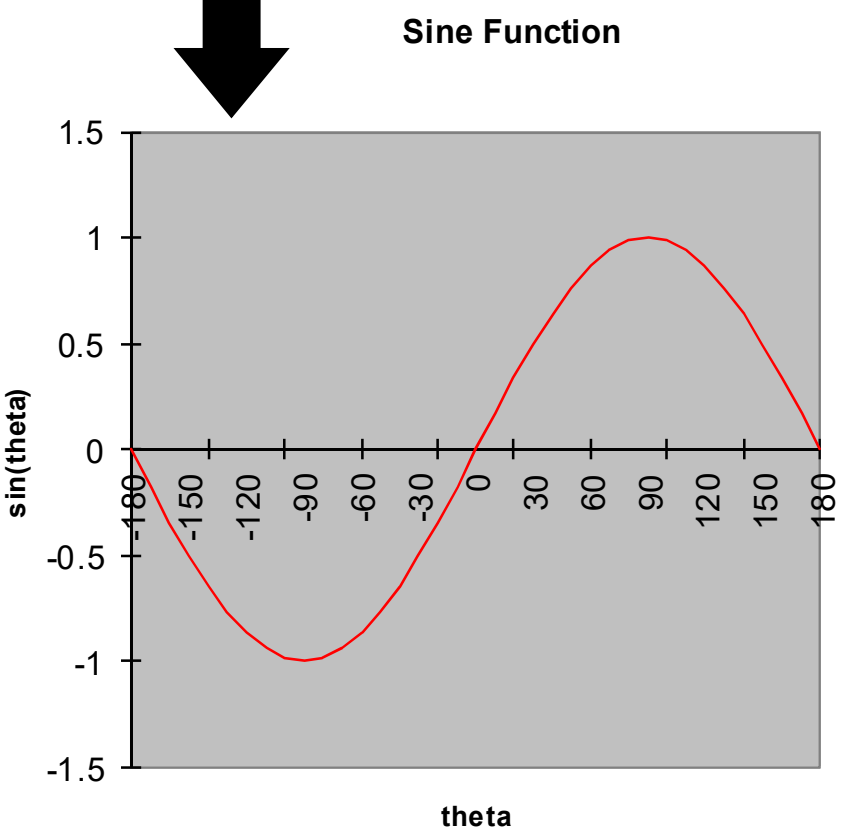
- Device dependent Limits
- Positive Integer values

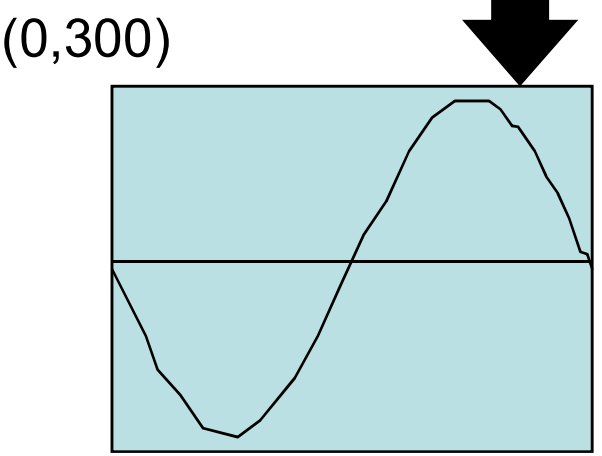


2D viewing Transformation Pipeline

Window: A rectangular region in World Coordinate System.

Viewport: A rectangular region in Device Coordinate System.





Window : Specifies what is to be displayed
It refers to a rectangular region selected from world coord picture defn for the purpose of display

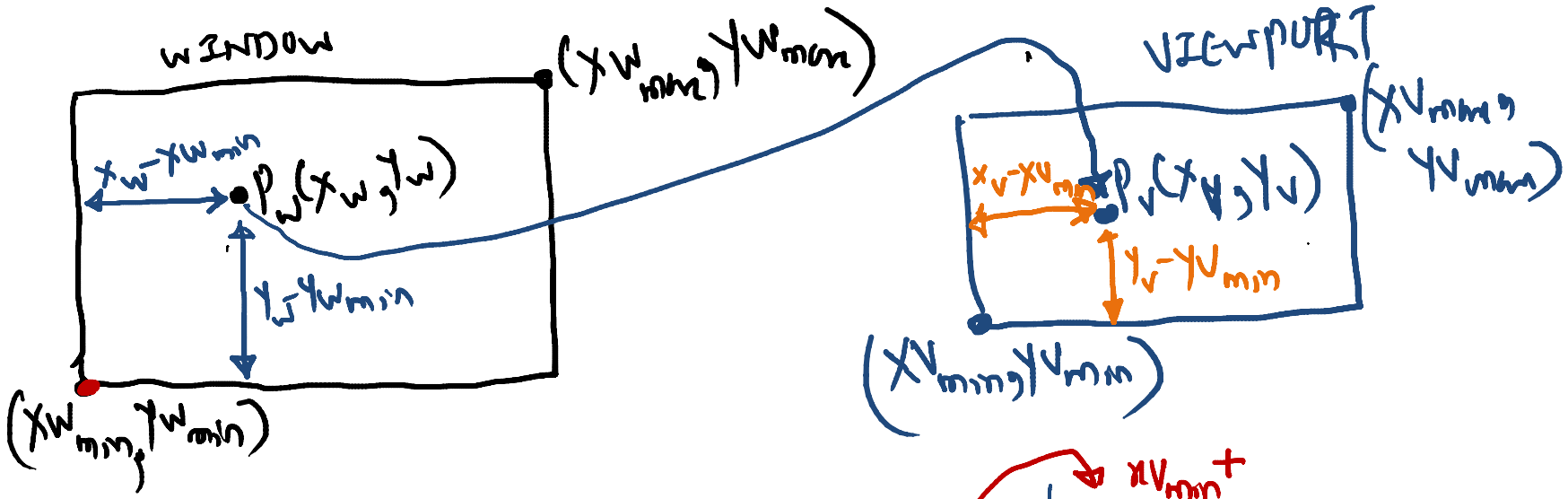
We denote the boundaries of the world window by four real values xmin, ymin, xmax, ymax denoting its left, bottom, right, top margins respectively. Similary the boundaries of the viewport on the screen are defined by four integer values umin, vmin, umax, vmax. When a graphics display is generated using arbitrary coordinates on the world window, the important problem encountered in viewing this display on the actual viewport on the screen is to have a function which maps every point (x,y) on the window to a corresponding point (u,v) on the viewport. The following **window to viewport transformation** achieves this relationship.

Viewport : Specifies where the selected window should be displayed on screen

Transformation used to map window to Viewport is called as window to Viewport transform

Window to Viewport Transformation

prerequisite: The relative posn of a point in window must be returned in viewport



$$\frac{x_w - xw_{min}}{xw_{max} - xw_{min}} = \frac{x_v - xv_{min}}{xv_{max} - xv_{min}}$$

$$x_v = (x_w - xw_{min}) * \frac{xv_{max} - xv_{min}}{xw_{max} - xw_{min}} + xv_{min}$$

as well as

$$\frac{y_w - yw_{min}}{yw_{max} - yw_{min}} = \frac{y_v - yv_{min}}{yv_{max} - yv_{min}}$$

$$x_v = (x_w - xw_{min}) * s_x + xv_{min}$$

where $s_x = \frac{xv_{max} - xv_{min}}{xw_{max} - xw_{min}}$

$\therefore$

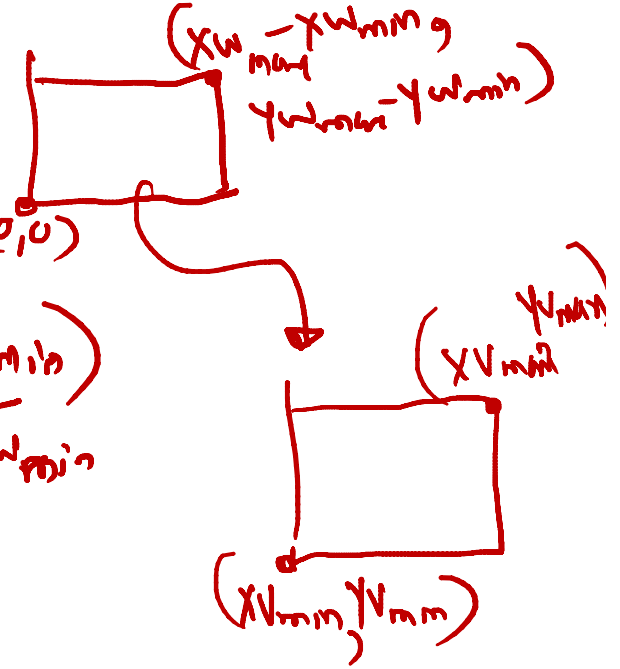
$$x_v = xv_{min} + (x_w - xw_{min}) * s_x$$

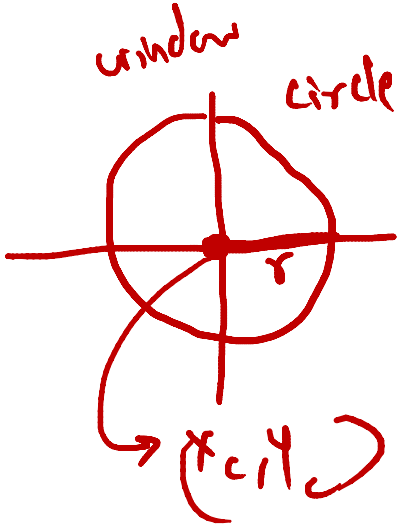
$$y_v = yv_{min} + (y_w - yw_{min}) * s_y$$

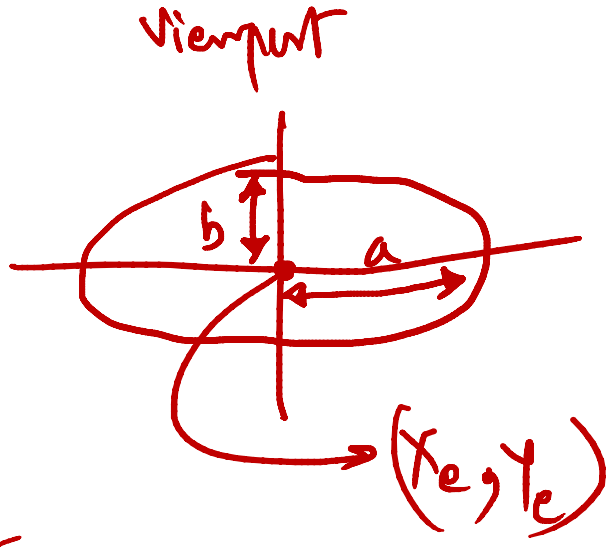
where $s_y = \dfrac{yv_{max} - yv_{min}}{yw_{max} - yw_{min}}$

Sequence of X_s^n

1. Transl^n ($T_x = -xw_{min}$, $T_y = -yw_{min}$)
2. Scaling ($S_x = \dfrac{xv_{max} - xv_{min}}{xw_{max} - xw_{min}}$, $S_y = \dfrac{yv_{max} - yv_{min}}{yw_{max} - yw_{min}}$)
3. Transl^n ($T_x = xv_{min}$, $T_y = yv_{min}$)







Seq^n to map window to viewport

1. Transl^n $(T_x = -x_c, T_y = -y_c)$
2. Scale $(S_x = \frac{a}{r}, S_y = b/r)$
3. Transl^n $(T_x = x_e, T_y = y_e)$

Find the sequence of transformations to map a window in fig (a) to a viewport in fig (b)

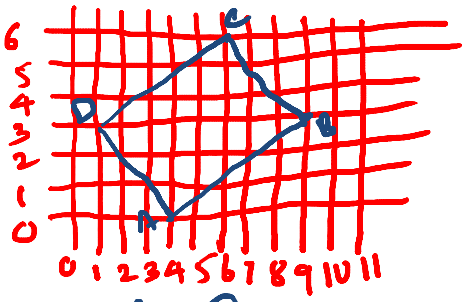


fig (a)

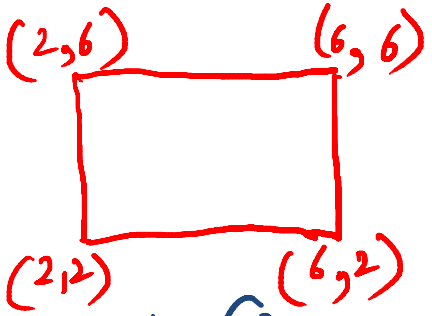


fig (b)

Clipping

- Introduction
- Brute Force
- Cohen-Sutherland Clipping Algorithm

purpose: To clip off the objects & part of objects which are outside the specified region (window) & retain the contents of window → Interior clipping

In case of exterior clipping we do exactly opposite.

Point clipping

P(x, y)

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

& $YW_{min} \leq y \leq YW_{max}$

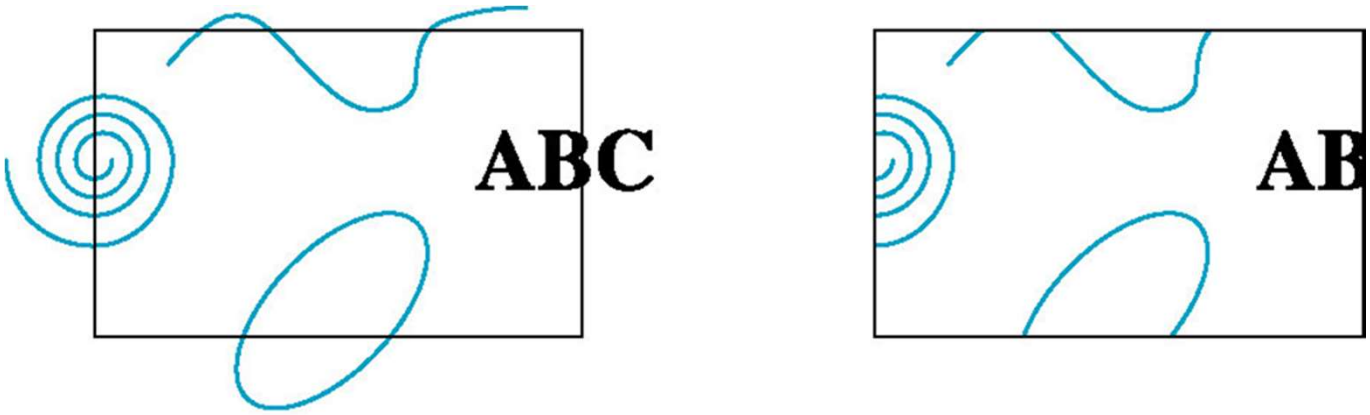
2D against clipping window

3D against clipping volume

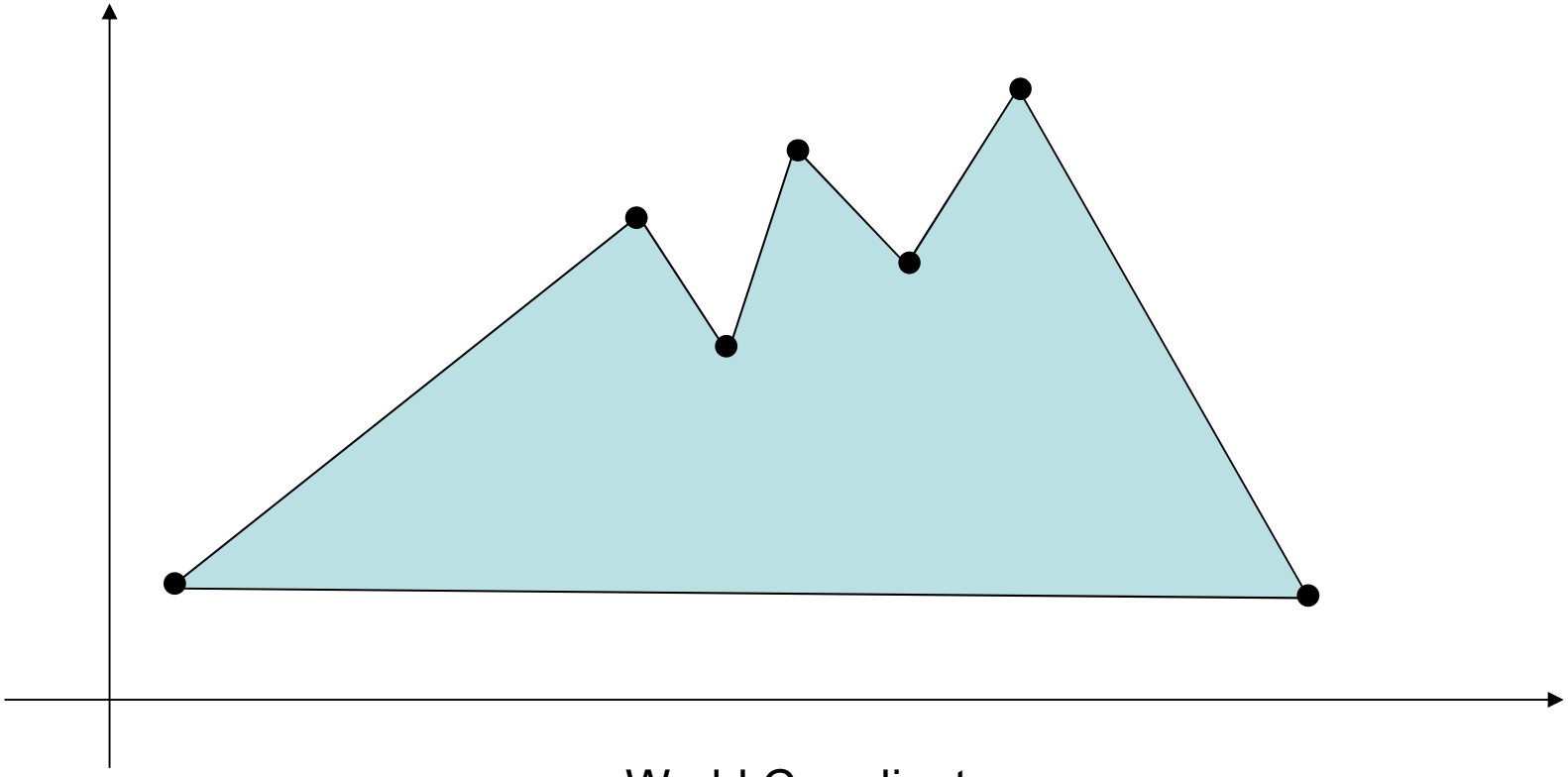
Easy for line segments polygons

Hard for curves and text

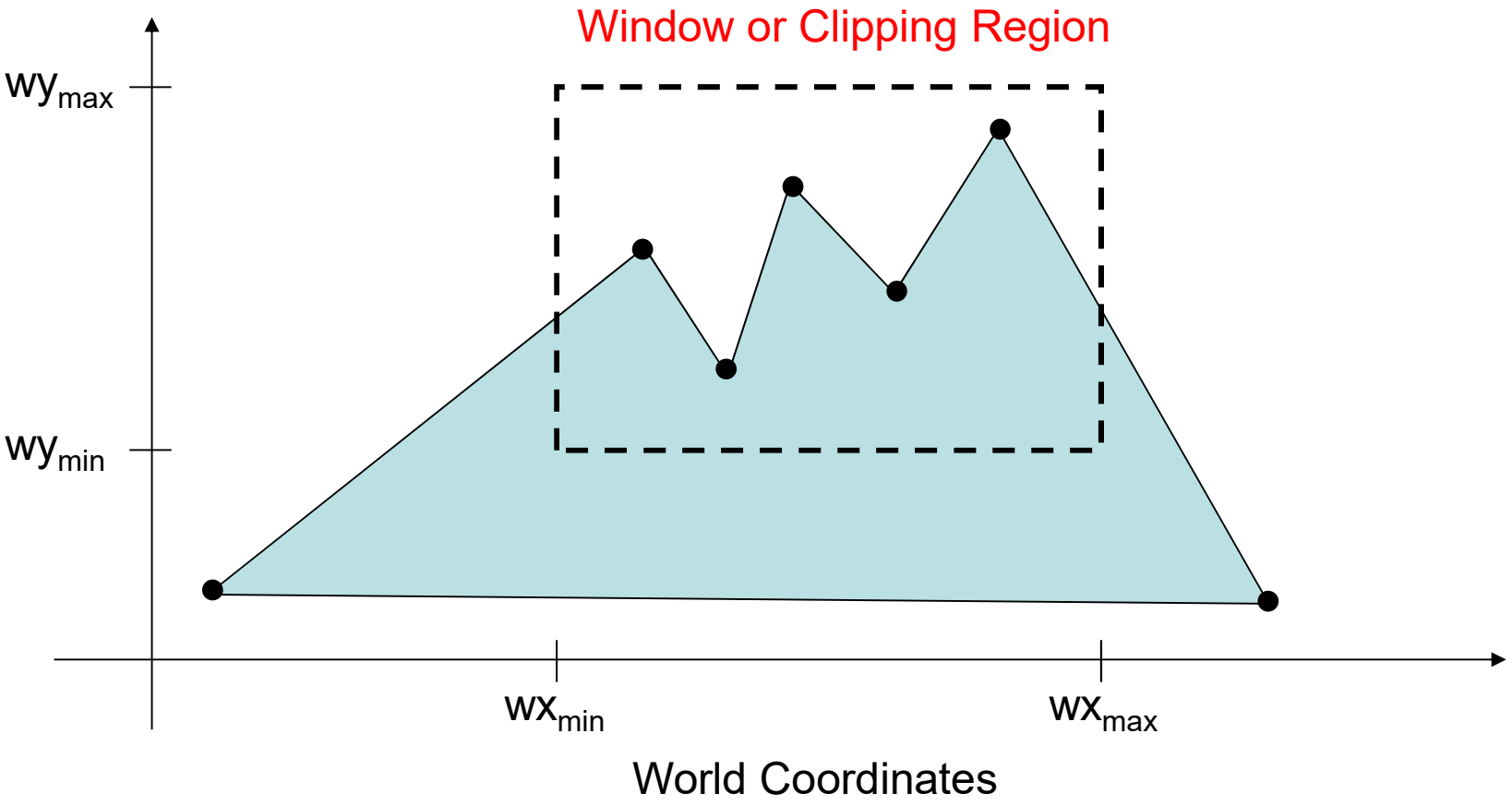
– Convert to lines and polygons first



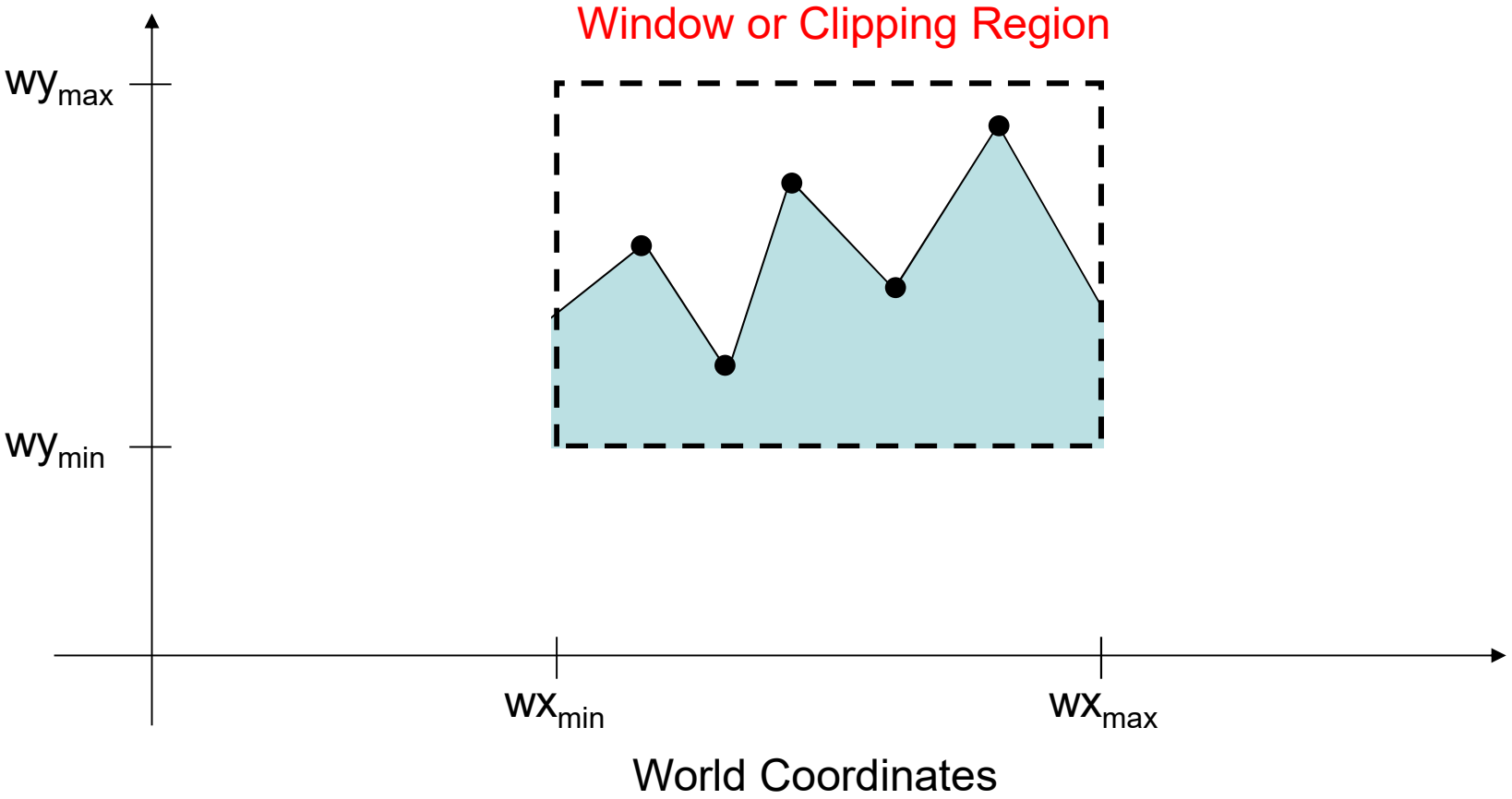
Any procedure that eliminates those portion of a picture that are either inside or outside a specified region is referred to as a clipping algorithm



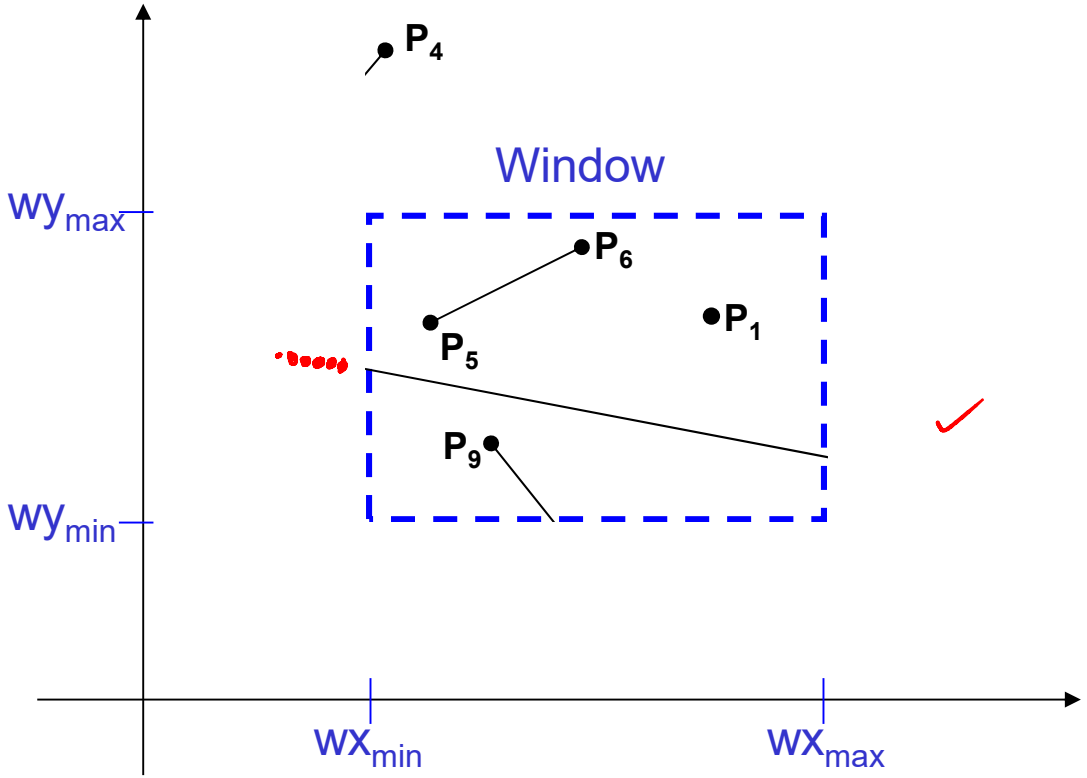
When we display a scene only those objects within a particular window are displayed

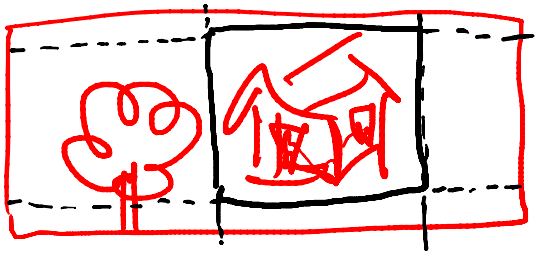


Because drawing things to a display takes time we *clip* everything outside the window

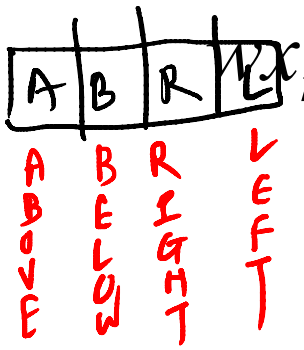


For the image below consider which lines and points should be kept and which ones should be clipped

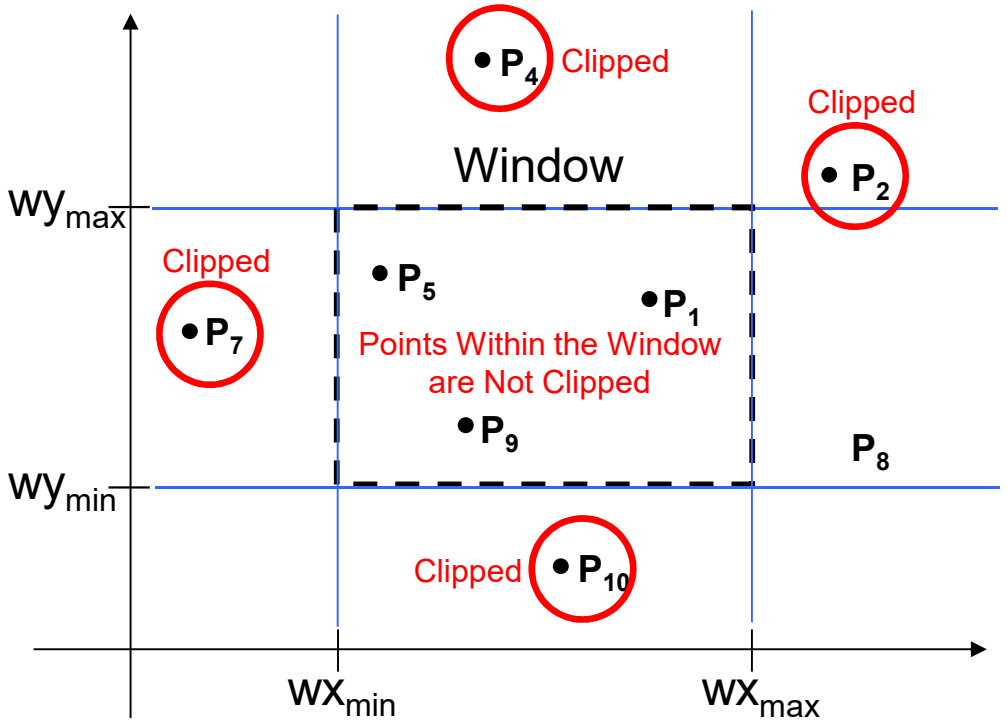




Easy - a point (x,y) is not clipped if:

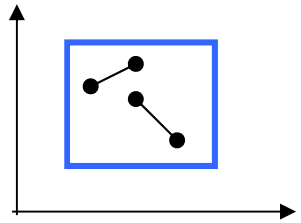
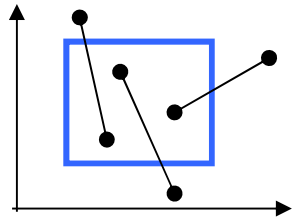
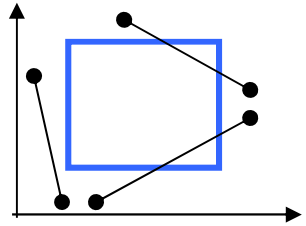


$$wx_{min} \le x \le wx_{max} \text{ AND } wy_{min} \le y \le wy_{max}$$



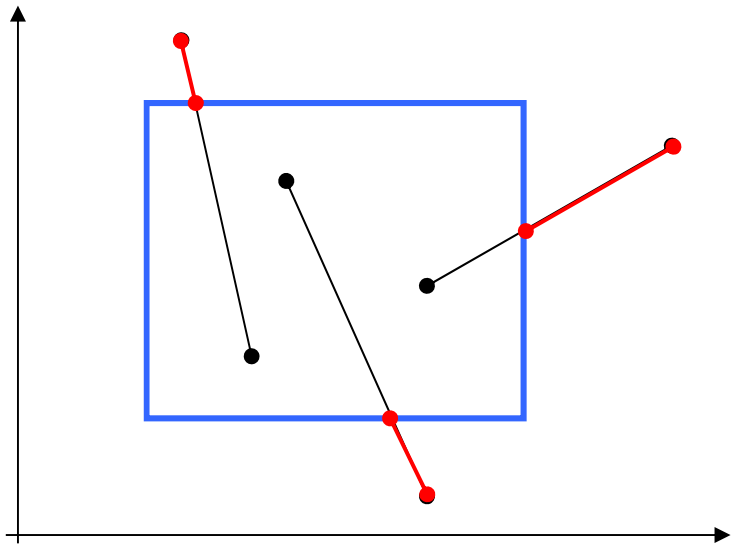
Even a point on window edge is not clipped (It is retained)

Harder - examine the end-points of each line to see if they are in the window or not

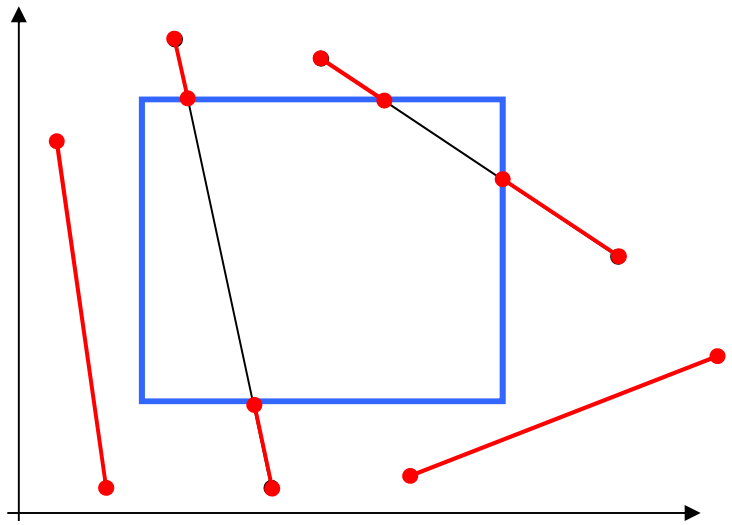
Situation	**Solution**	**Example**
Both end-points inside the window	Don’t clip	
One end-point inside the window, one outside	Must clip	
Both end-points outside the window	Don’t know!	

Brute force line clipping can be performed as follows:

- Don’t clip lines with both end-points within the window
- For lines with one end-point inside the window and one end-point outside, calculate the intersection point (using the equation of the line) and clip from this point out



– For lines with both end-points outside the window test the line for intersection with all of the window boundaries, and clip appropriately



However, calculating line intersections is computationally expensive

Because a scene can contain so many lines, the brute force approach to clipping is much too slow

An efficient line clipping algorithm

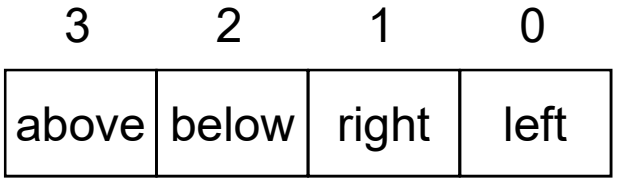
The key advantage of the algorithm is that it vastly reduces the number of line intersections that must be calculated

Dr. Ivan E. Sutherland co-developed the Cohen-Sutherland clipping algorithm. Sutherland is a graphics giant and includes amongst his achievements the invention of the head mounted display.

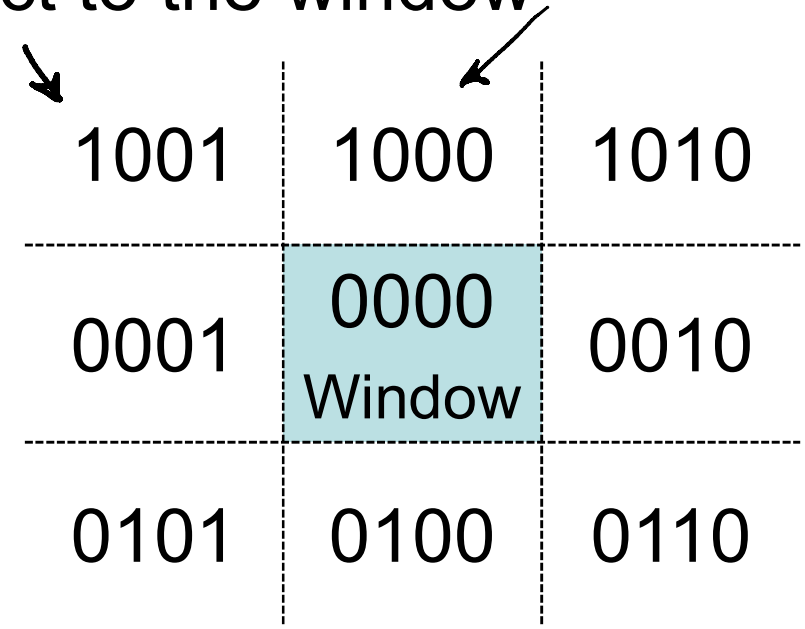
Cohen Sutherland line clipping

World space is divided into regions based on the window boundaries

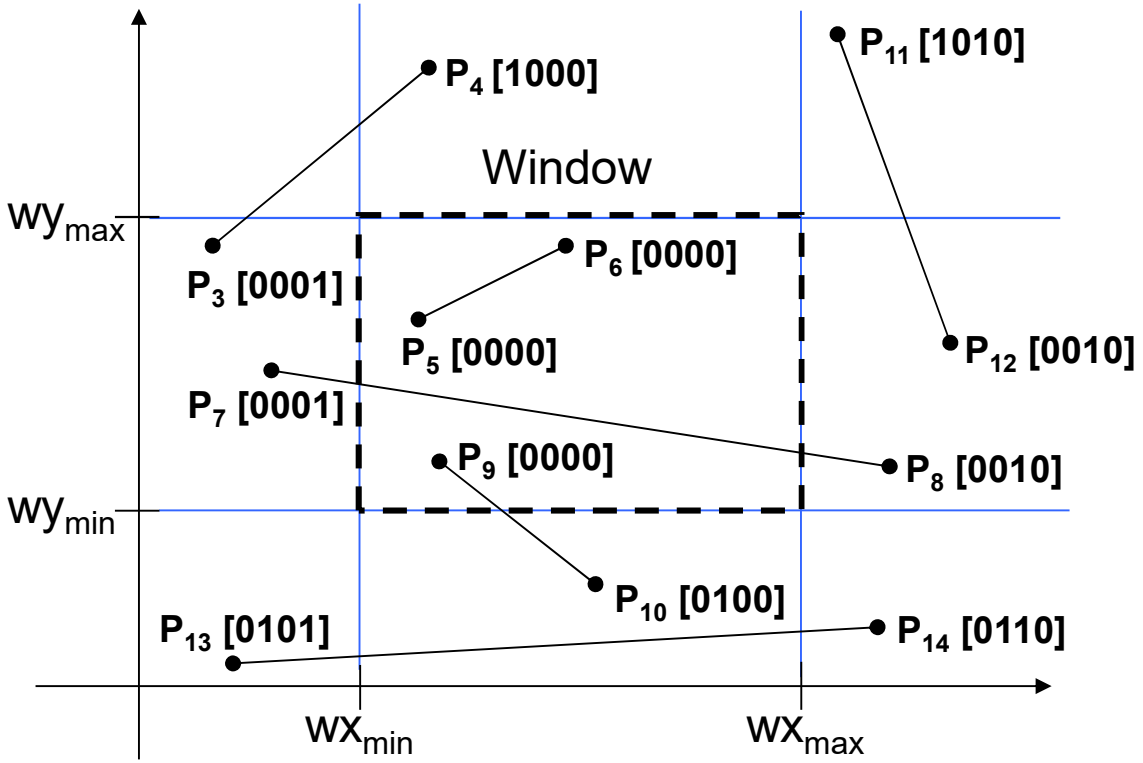
- Each region has a unique four bit region code
- Region codes indicate the position of the regions with respect to the window



Region Code Legend

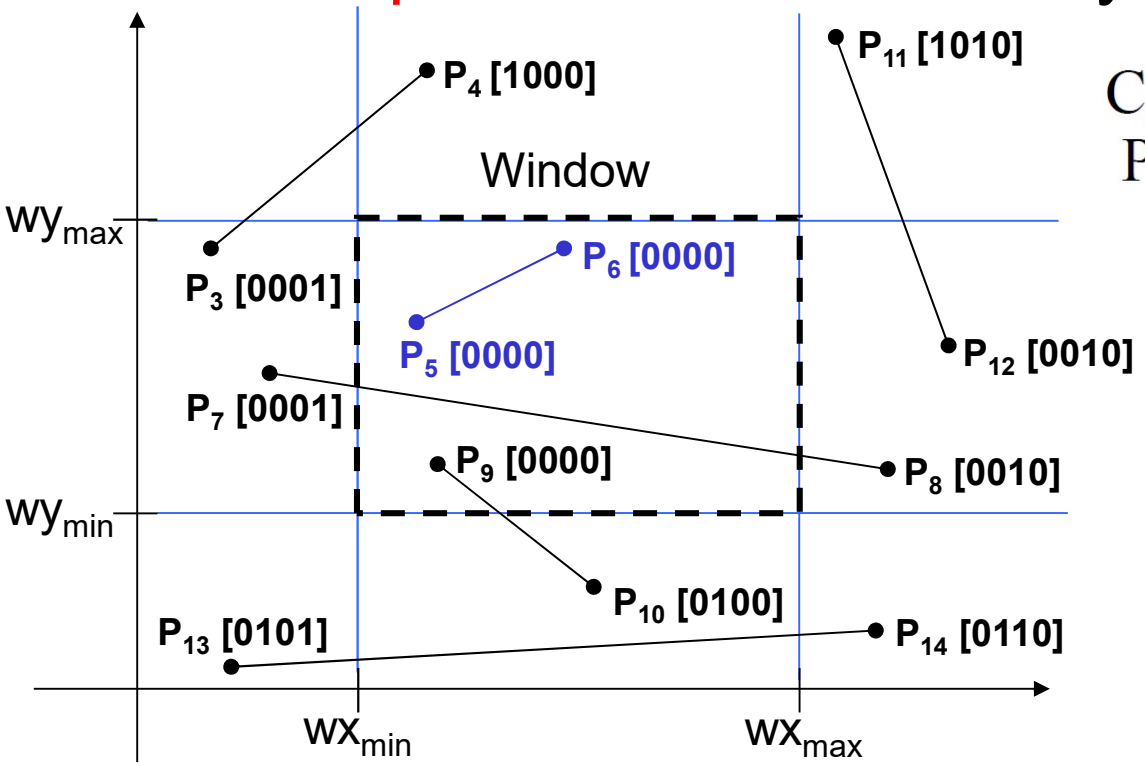


Every end-point is labelled with the appropriate region code



Lines completely contained within the window boundaries have region code [0000] for both end-points so are not clipped

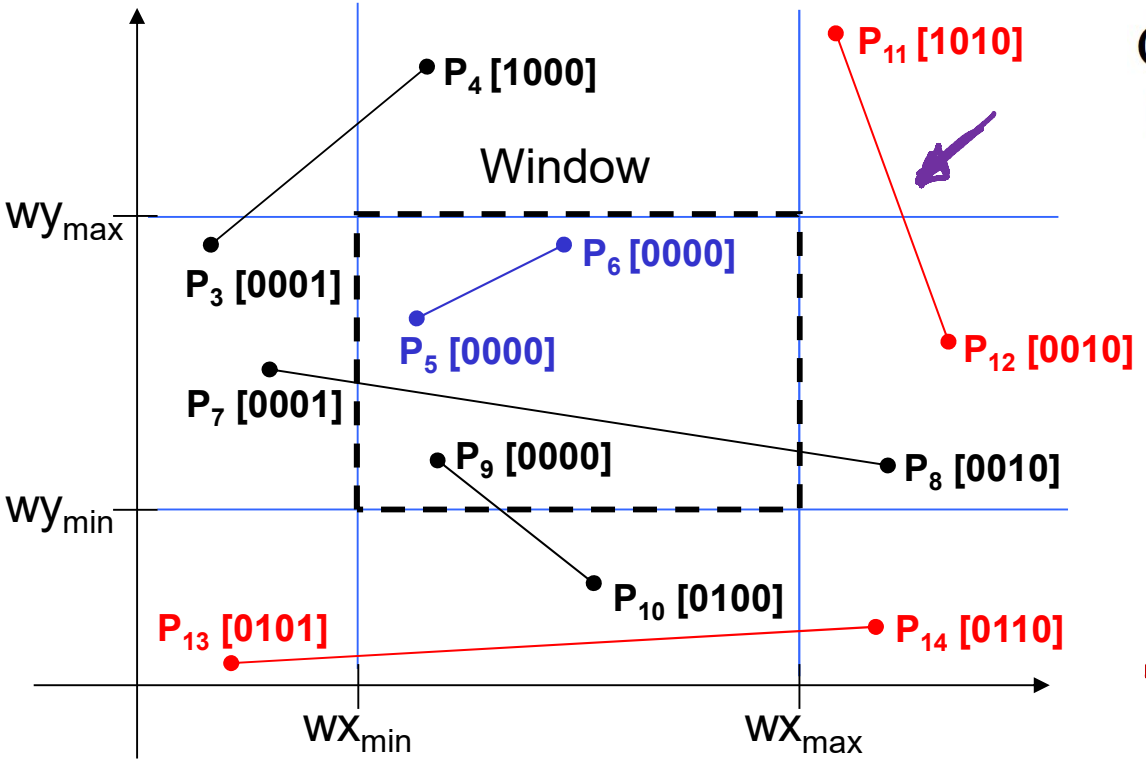
The OR operation can efficiently check this



$C_0 \vee C_{end} = 0$ then
P_0P_{end} is trivially accepted

Any lines with a common set bit in the region codes of both end-points can be clipped

– The AND operation can efficiently check this



$C_0 \wedge C_{end} \neq 0$ then
P_0P_{end} is trivially rejected

$P_{11} = 1010$ Bitwise
$P_{12} = 0010$ Anding
0010 Non Zero

& $P_3 = 0001$
$P_4 = 1000$
0000 →

If the line is not totally acceptable nor totally rejectable the find point of intersection $P(x', y')$

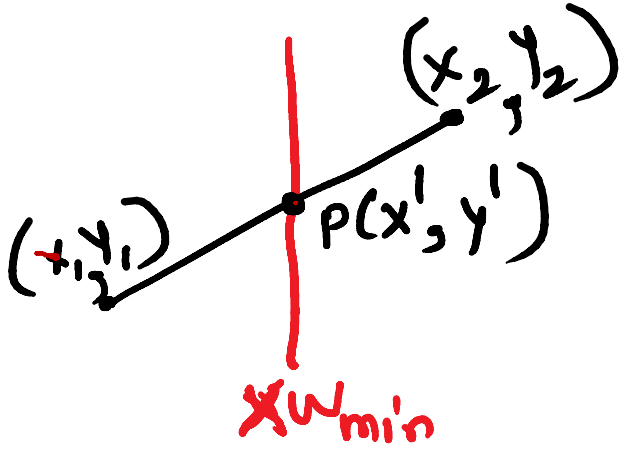
How ? :

Case 1: w.r.t. left Boundary (Edge)

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

$$\Rightarrow \boxed{y' = y_1 + m(x' - x_1)}$$

where $x' = XW_{min}$ (∵ any point on left edge will have $x = XW_{min}$)

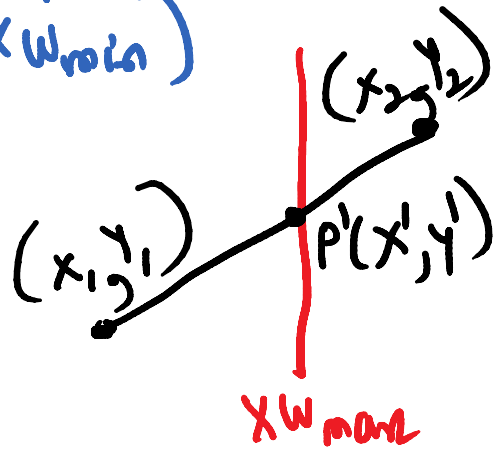


Case 2: w.r.t. Right Boundary (Edge)

In similar lines of case 1

$$\Rightarrow y' = y_1 + m(x' - x_1)$$

where $x' = XW_{max}$

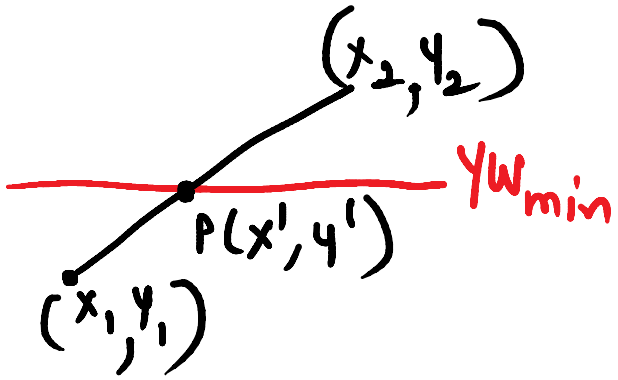


Case 3 : w.r.t. bottom Edge

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

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

where $y' = yw_{min}$

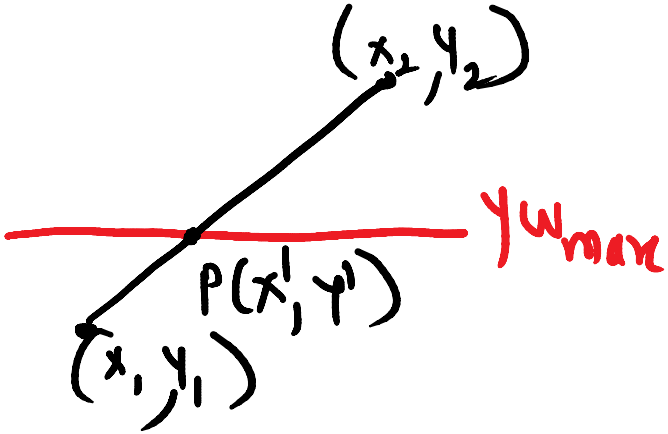


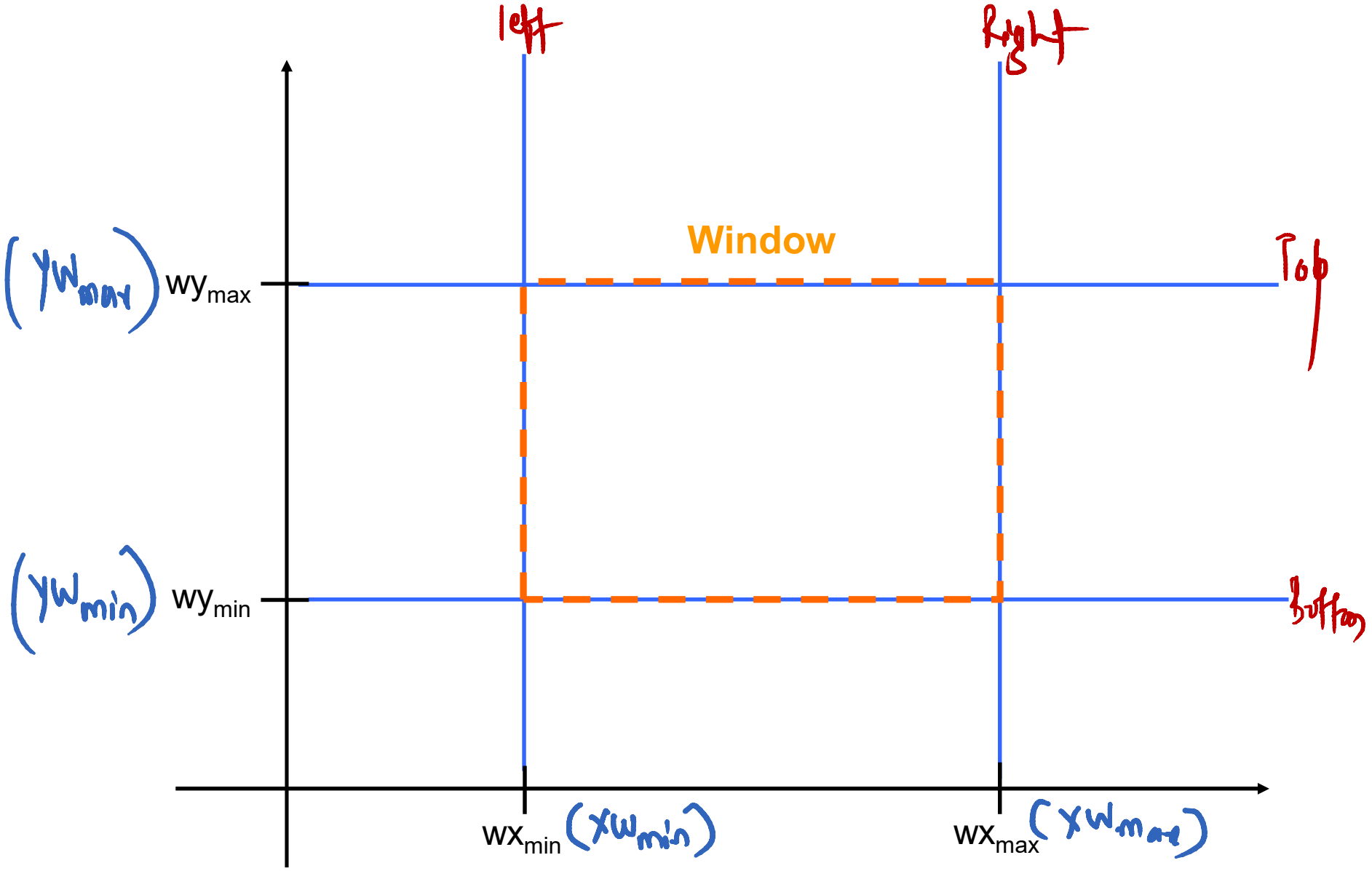
Case 4: w.r.t. Top Edge

In similar lines of case 3

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

where $y' = yw_{max}$




left
Right
Window
Top
Bottom
(yw_{max})
wy_{max}
(yw_{min})
wy_{min}
wx_{min} (xw_{min})
wx_{max} (xw_{max})

Algorithm

Step 1: Accept window extents (xw_{min}, yw_{min}) & (xw_{max}, yw_{max})

Step 2: Accept end point coordinates of a line segment (x_1, y_1) & (x_2, y_2)

Step 3: Assign region code for both the end points

To assign a region code for any point P(x,y): Initialize l=r=a=b=0

```
if (x < xwmin)
    l=1
else
 if (x > xwmax)
    r=1
if (y < ywmin)
    b=1
else
 if (y > ywmax)
    a=1
```

Step 4: If the region code for both the end points are consisting of all zeros

→ line is totally acceptable

∴ display the line and stop.

Step 5: If the Bitwise Anding bet^n the region codes of 2 end points results into a nonzero value

→ line is totally outside

∴ do not display the line and stop.

Step 6: Find a point of intersection $I(x', y')$ of a line w.r.t. appropriate window edge as :

if (l==1) → $y' = y_1 + m(x' - x_1)$ where $x' = xw_{min}$

else if (r==1) → $y' = y_1 + m(x' - x_1)$ where $x' = xw_{max}$

else if (b==1) → $x' = x_1 + \frac{1}{m}(y' - y_1)$ where $y' = yw_{min}$

else if (a==1) → $x' = x_1 + \frac{1}{m}(y' - y_1)$ where $y' = yw_{max}$

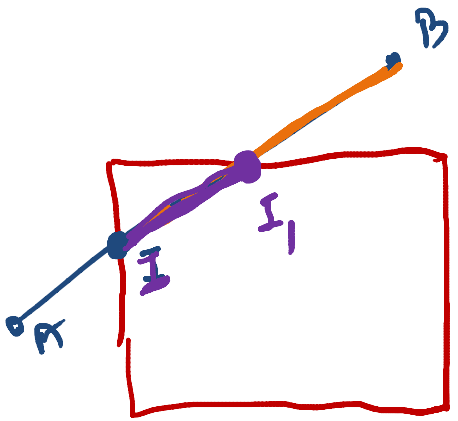
i.e if left bit of end point which is outside window is 1

i.e if right bit of end point which is outside window is 1

step 7: replace the coordinates (x,y) of an appropriate end point which is outside window with (x',y') (i.e calculated point of intersection)

step 8: Goto step 3

step 9: Stop

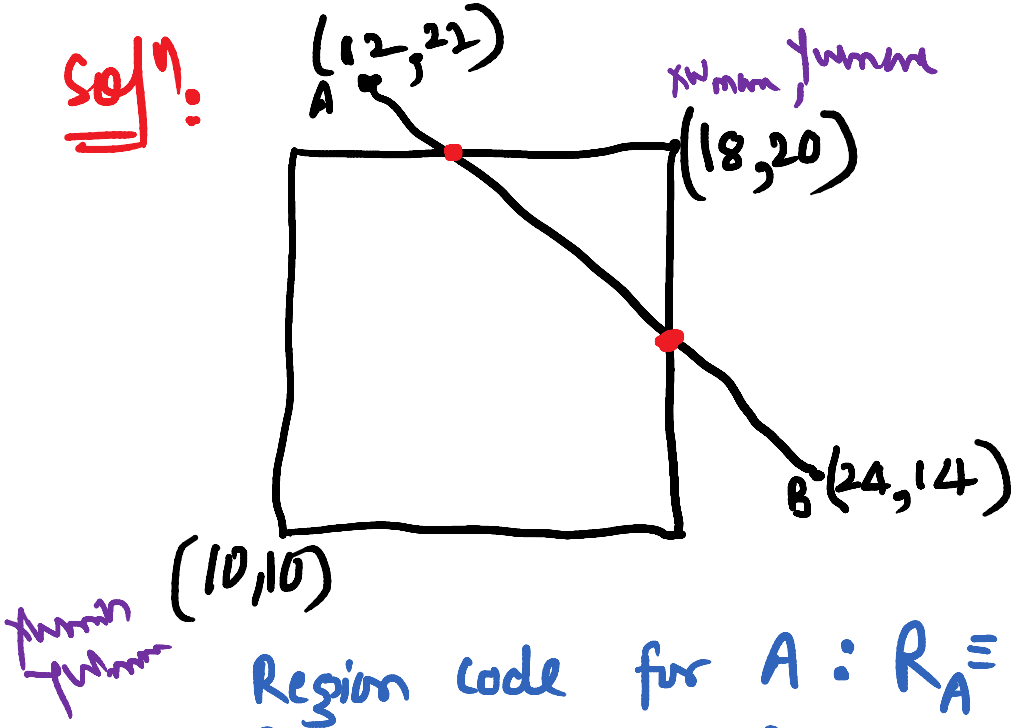


Find the coordinates of a line segment A(12,22) B(24,14) after it is clipped using cohen sutherland Algo. against a window

$(xw_{min}, yw_{min}) \equiv (10,10)$

$(xw_{max}, yw_{max}) \equiv (18,20)$

Sol":



abrl

Region code for A : $R_A \equiv 1000$

Region code for B : $R_B \equiv 0010$

→ line is not totally acceptable

→ line is not totally rejectable

∵

$$\begin{array}{r} 1000 \\ \& \;\; 0010 \\ \hline 0000 \end{array}$$

→ If result is non zero then it becomes totally rejectable

A $(x_1, y_1) \equiv (12, 22)$

B $(x_2, y_2) \equiv (24, 14)$

→ finding point of intersection $I(x', y')$ w.r.t. top edge (∵ a bit of R_A is 1)

$$\therefore \quad x' = x_1 + \frac{1}{m}(y' - y_1) \qquad \left(m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{-8}{12} = -\frac{2}{3} \right)$$

$y' = y_{W_{max}} = 20$

$$\therefore \quad x' = 10 + \left(-\frac{3}{2}\right)(20 - 22) = 10 + \frac{-3}{2} \times -2$$

$$\therefore \quad x' = 13 \Rightarrow (x', y') \equiv (13, 20)$$

Now replacing coord's of A with calculated pt. of intersection ⇒ new coord's of A(13,20)

⇒ The line now reduces to A(13,20) B(24,14)

∴ Now $R_A \equiv 0000$

$R_B \equiv 0010$

⇒ line is not totally acceptable nor totally rejectable

∴ Finding point of intersection $I(x', y')$ w.r.t. right edge (∵ r bit in R_B is 1)

$\Rightarrow \quad y' = y_1 + m(x' - x_1)$

where $x' = xw_{max} = 18$

$\therefore \quad y' = 22 + \left(-\frac{2}{3}\right)(18 - 12)$

$= 22 + \left(-\frac{2}{3}\right) * 6$

$= 22 - 4$

$y' = 18 \quad \Rightarrow \quad (x', y') \equiv (18, 18)$

Now replacing coord^{n}s of B with calculated pt. of intersec^n

$\Rightarrow$ new coord^{n}s of B (18, 18)

∴ Now $R_A \equiv 0000$

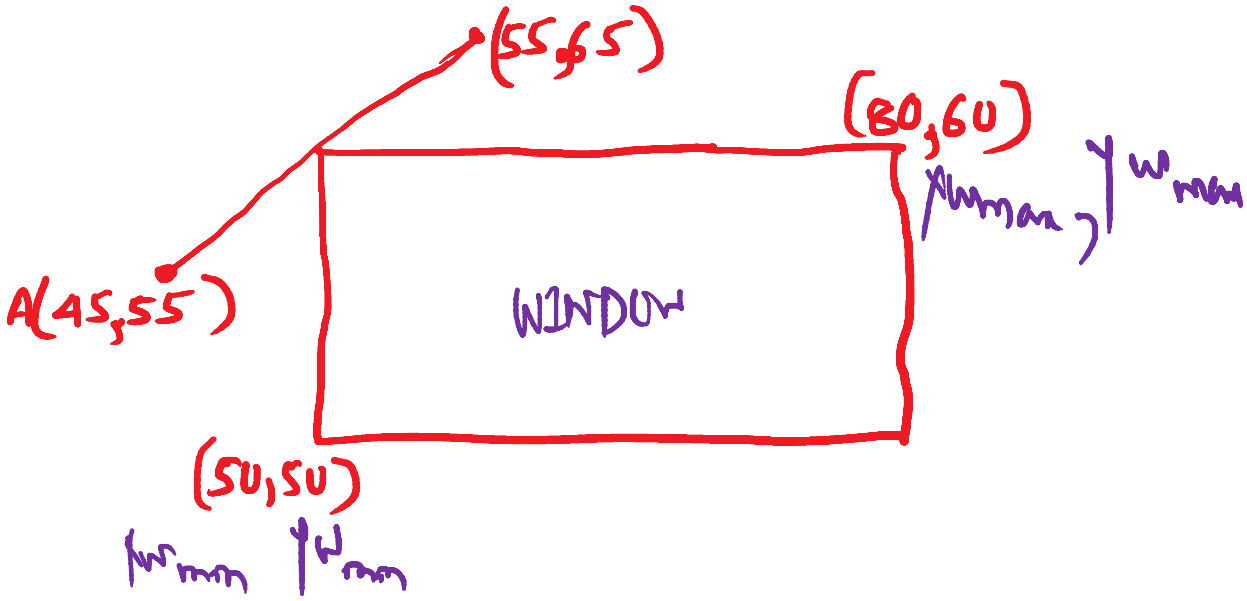
$R_B \equiv 0000$

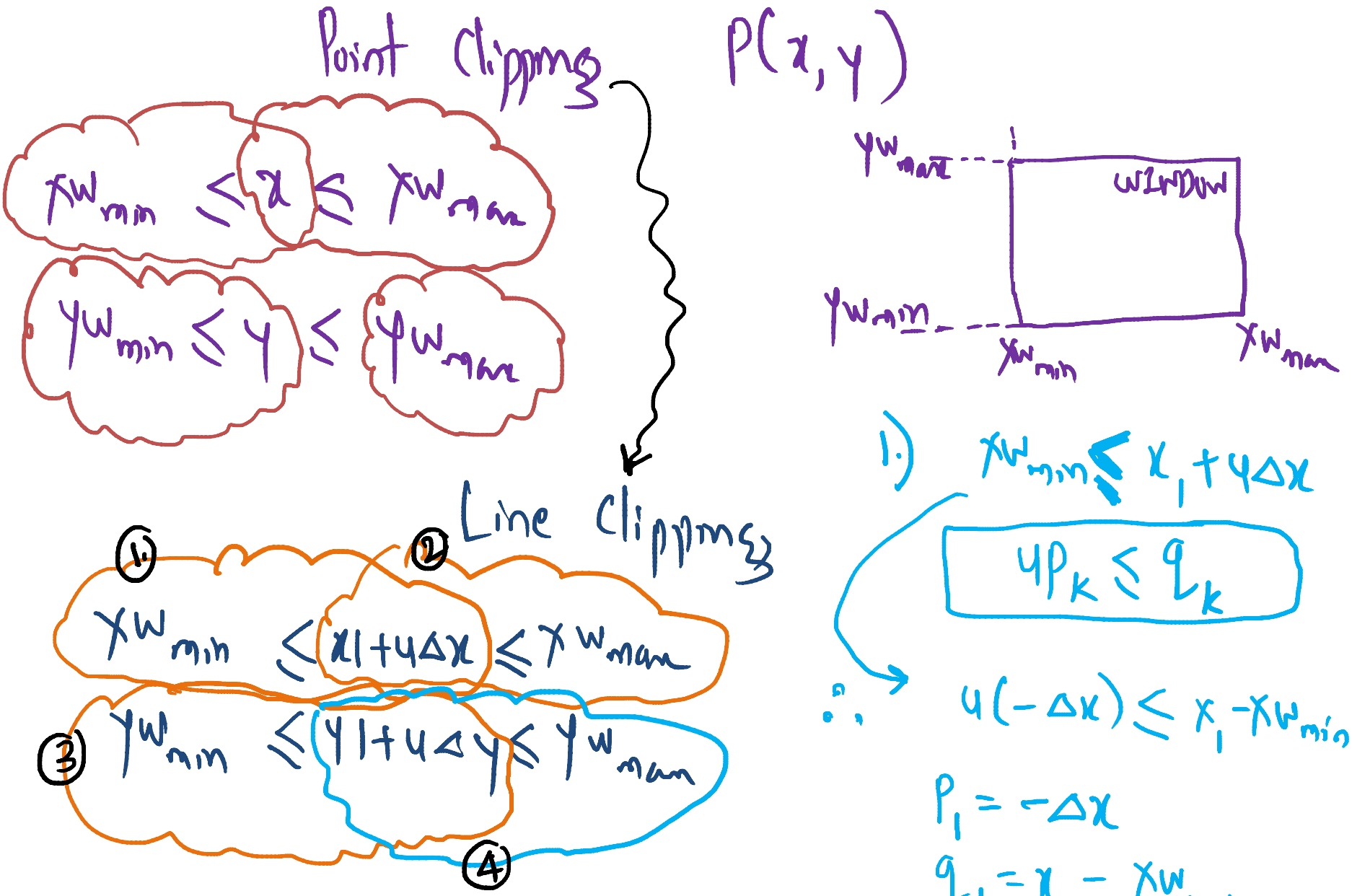
→ line is totally acceptable

∴ display the line betⁿ (13, 20) & (18, 18)

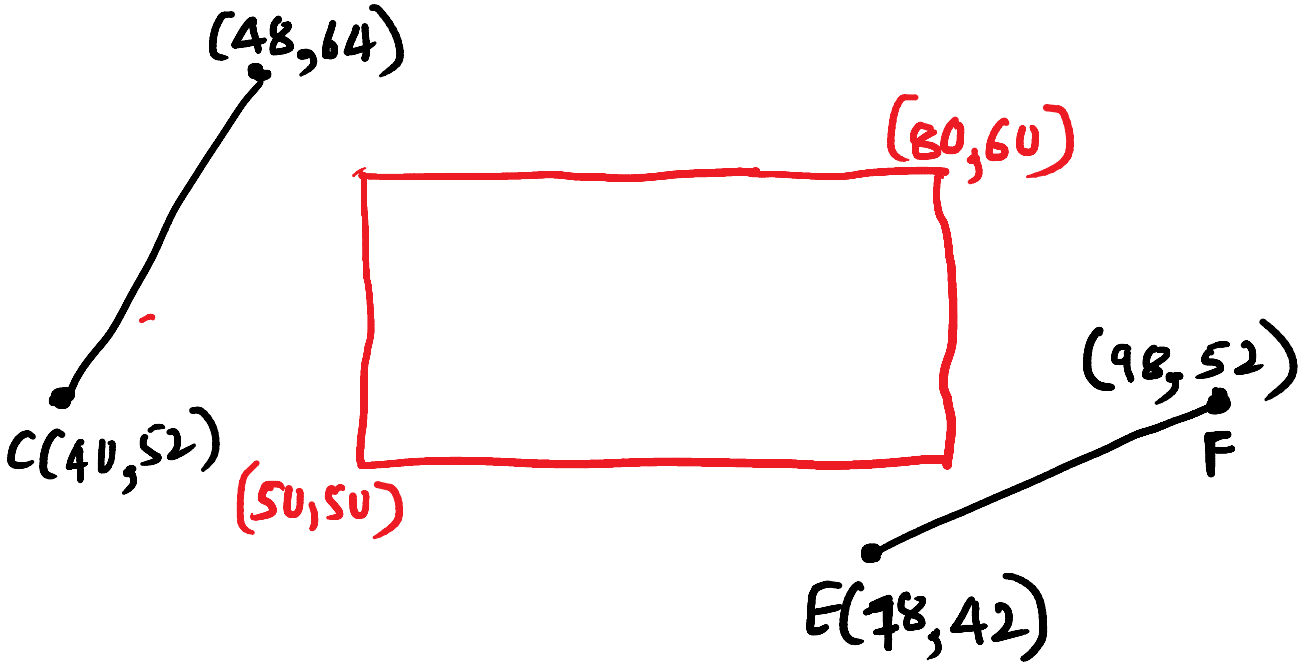
clipped line (13, 20) to (18, 18)

Ex:




Point clipping
$P(x, y)$
$xw_{min} \le x \le xw_{max}$
$yw_{min} \le y \le yw_{max}$
yw_{max}
window
yw_{min}
xw_{min}
xw_{max}
Line clipping
1.) $xw_{min} \le x_1 + u\Delta x$
$up_k \le q_k$
$u(-\Delta x) \le x_1 - xw_{min}$
$p_1 = -\Delta x$
$q_1 = x_1 - xw_{min}$
1
2
3
4
$xw_{min} \le x1 + u\Delta x \le xw_{max}$
$yw_{min} \le y1 + u\Delta y \le yw_{max}$

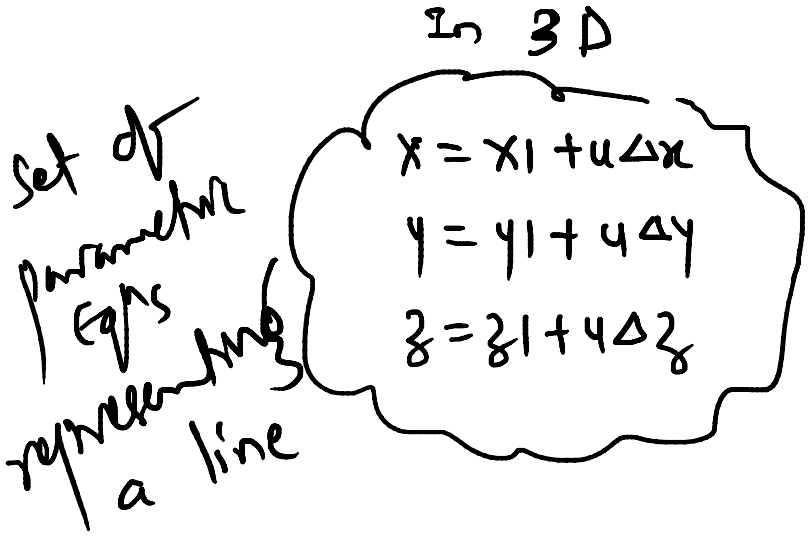
Ex1:



Cohen Sutherland line Clipping

↑ Better than Brute Force approach (As it avoids processing the line on pixel by pixel basis)

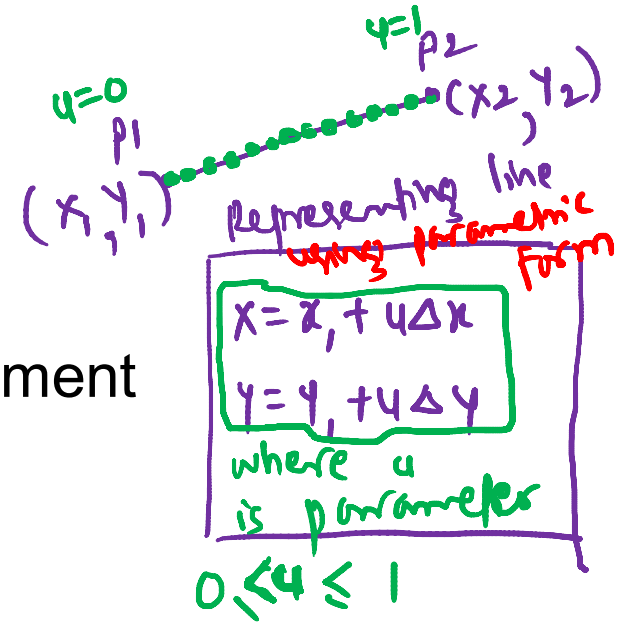
↓ May require multiple iterations



Liang-Barsky Line Clipping Algorithm

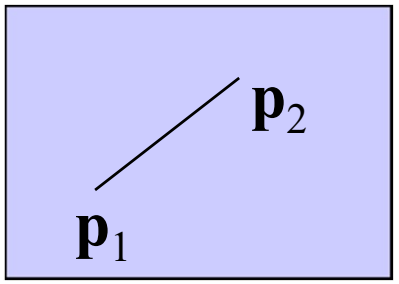
It uses Parametric equation to represent a line.

→ Clipping is done in single iteration.



Consider the parametric form of a line segment

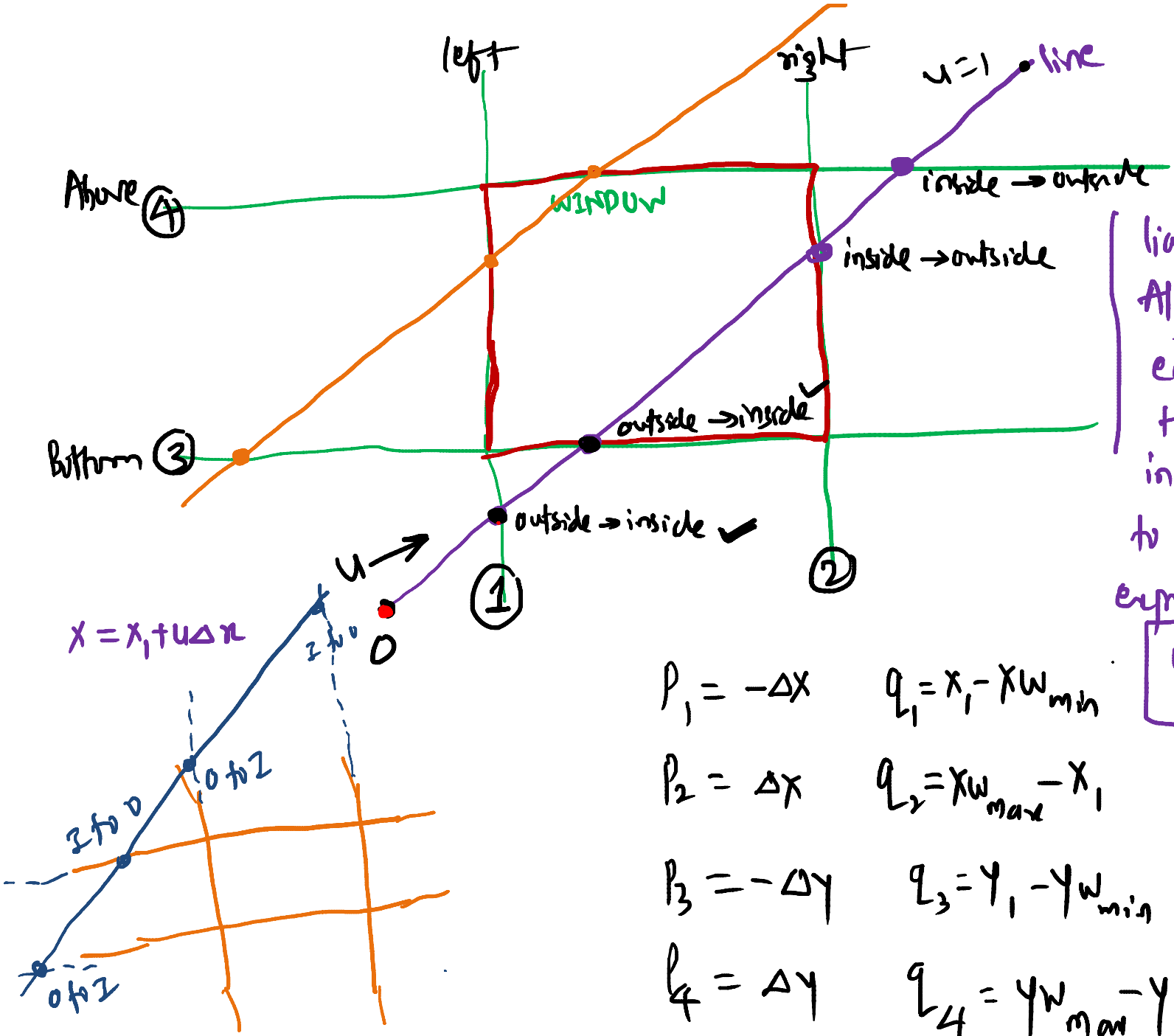
$$\mathbf{p}(\alpha) = (1-\alpha)\mathbf{p}_1 + \alpha\mathbf{p}_2 \quad 1 \geq \alpha \geq 0$$



$$P(u) = P_1 + u\Delta P$$
$$= P_1 + u(P_2 - P_1)$$

We can distinguish between the cases by looking at the ordering of the values of α where the line determined by the line segment crosses the lines that determine the window

→ $P(u) = (1-u)P_1 + uP_2$



Liang Barsky Algo requires each of the 4 (four) inequalities to be expressed in

$$uP_k \leq q_k$$

form

$$p_1 = -\Delta x \qquad q_1 = x_1 - xw_{min}$$

$$p_2 = \Delta x \qquad q_2 = xw_{max} - x_1$$

$$p_3 = -\Delta y \qquad q_3 = y_1 - yw_{min}$$

$$p_4 = \Delta y \qquad q_4 = yw_{max} - y_1$$

for any k (i.e $0, 1, 2, 3, 4$), if

eg: if for $k=1 \Rightarrow p_1 < 0$

$\Rightarrow -\Delta x < 0$

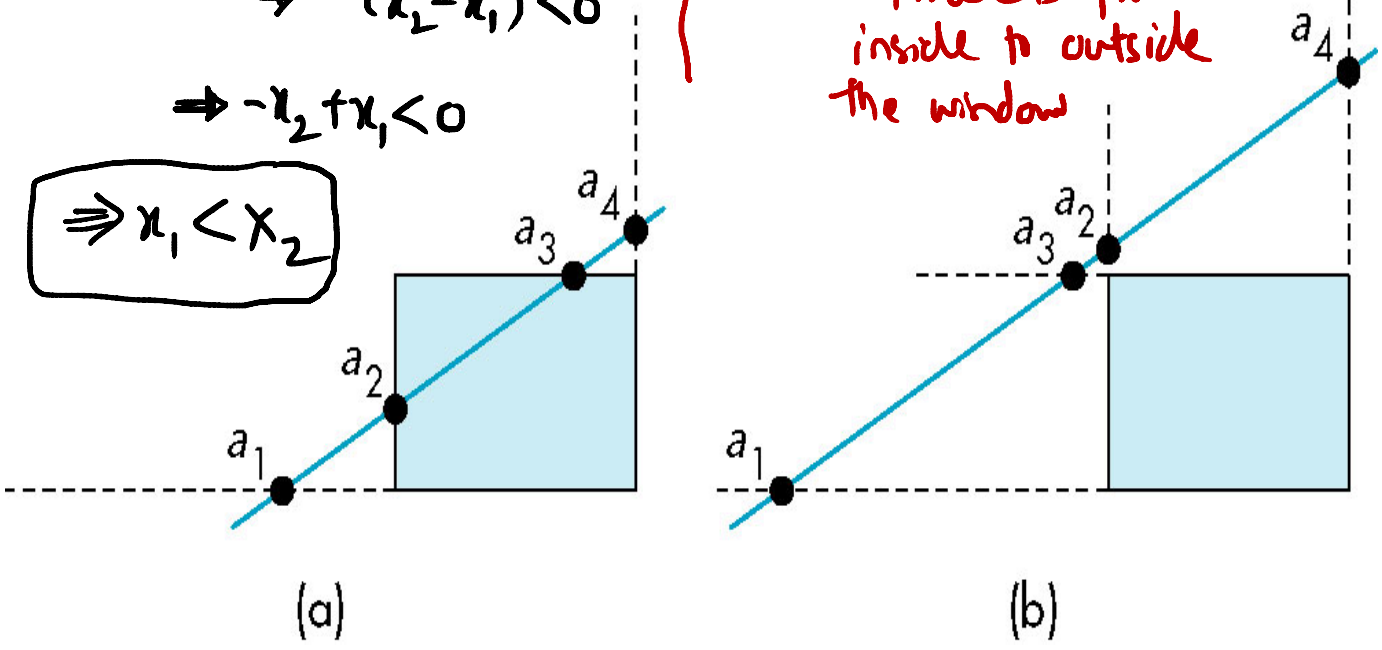
$\Rightarrow -(x_2 - x_1) < 0$

$\Rightarrow -x_2 + x_1 < 0$

$\Rightarrow x_1 < x_2$

$p_k < 0 \Rightarrow$ infinite extension of line proceeds from outside to inside the window

Similarly if $p_k > 0 \Rightarrow$ infinite extension of line proceeds from inside to outside the window



if $p_k == 0 \Rightarrow$ line is either horizontal or vertical for that k if $q_k < 0 \rightarrow$ line is totally outside

Algo:

step 1: Accept window extents (xw_{min}, yw_{min}) & (xw_{max}, yw_{max})

step 2: Accept end point coords of line seg
(x_1, y_1) & (x_2, y_2)

step 3: Calculate

K	p_K	q_K	$r_K = \frac{q_K}{p_K}$	
1	$-\Delta x$	$x_1 - xw_{min}$		
2	Δx	$xw_{max} - x_1$		
3	$-\Delta y$	$y_1 - yw_{min}$		
4	Δy	$yw_{max} - y_1$		

for any K, if we find $p_K == 0$ & $q_K < 0$

⇒ line is totally outside
⇒ do not display the line & STOP

Step 4: Calculate values for parameter u corresponding to two point of intersections

i.e calculate u_1 & u_2

$u_1 = \{r_k, 0\}_{max}$ for all r_k values for $p_k < 0$ (i.e outside to inside)

$u_2 = \{r_k, 1\}_{min}$ for all r_k values for $p_k > 0$

If $up_k \leq q_k$

$u = q_k / p_k$

Step 5: if $u_1 > u_2$

$\Rightarrow$ line is totally outside

$\therefore$ stop

else

find point of intersection $I_1(x', y')$ & $I_2(x'', y'')$ as:

$$x' = x_1 + t_1 \Delta x$$
$$y' = y_1 + t_1 \Delta y$$

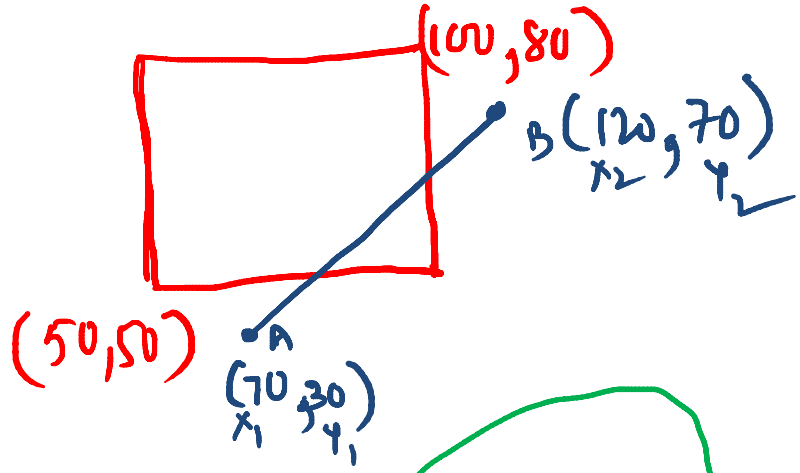
&

$$x'' = x_1 + t_2 \Delta x$$
$$y'' = y_1 + t_2 \Delta y$$

step 6: Display line betⁿ I_1 & I_2

step 7: Stop

Ex:



$xw_{min} = 50$

$yw_{min} = 50$

$xw_{max} = 100$

$yw_{max} = 80$

soln : $\Delta x = 50$

$\Delta y = 40$

k	p_k	q_k	$r_k = \frac{q_k}{p_k}$
1	-50	20	-2/5
2	50	30	3/5 #
3	-40	-20	1/2
4	40	50	5/4 #

$u_1 \equiv \{0, -2/5, 1/2\}_{max} = 1/2$

$u_2 \equiv \{1, 3/5, 5/4\}_{min} = 3/5$

is $u_1 > u_2$? → No

$x' = x_1 + u_1 \Delta x = 70 + \frac{1}{2} * 50 = 95$

$y' = y_1 + u_1 \Delta y = 30 + \frac{1}{2} * 40 = 50$

Similarly $x'' = x_1 + u_2 \Delta x = 70 + \frac{3}{5} * 50 = 100$

$y'' = y_1 + y_2 \Delta y$

$= 30 + \frac{3}{5} * 40$

$y'' = 54$

$\therefore$ line will be displayed betn

$(95, 50)$ & $(100, 54)$

Cl 2

$xw_{min} = 50$ $\quad$ $xw_{max} = 100$

$yw_{min} = 50$ $\quad$ $yw_{max} = 80$

$(x_1, y_1) \equiv (40, 30)$

$(x_2, y_2) \equiv (40, 90)$

Cl 2

$(x_1, y_1) = (40, 30)$

$(x_2, y_2) = (48, 60)$

$\Delta x = 8$ $\quad$ $\Delta y = 30$

k	p_k	q_k	r_k
1	-8	-16	5/4
2	8	60	15/2
3	-30	-20	2/3
4	30	40	4/3

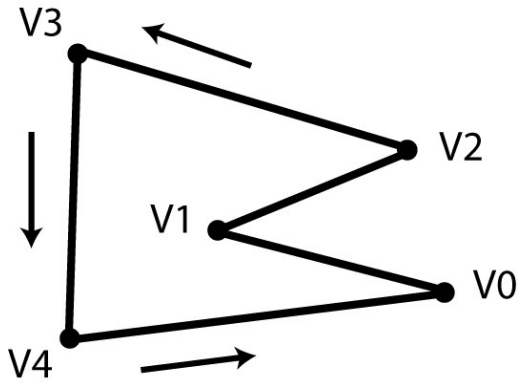
$u_1 = \{0, 5/4, 2/3\}_{max} = 5/4$

$u_2 = \{1, 15/2, 4/3\}_{min} = 1$

$\because u_1 > u_2$

$\Rightarrow$ line is totally outside

Polygon Clipping



Ordered set of vertices (points)

– Usually counter-clockwise

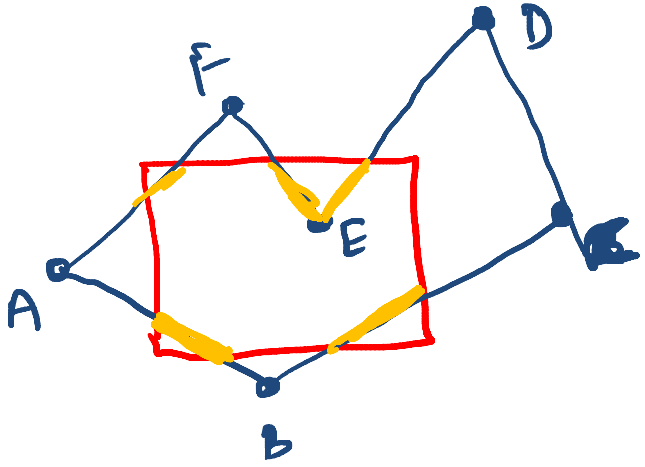
Two consecutive vertices define an edge

Left side of edge is inside

Right side is outside

Last vertex implicitly connected to first

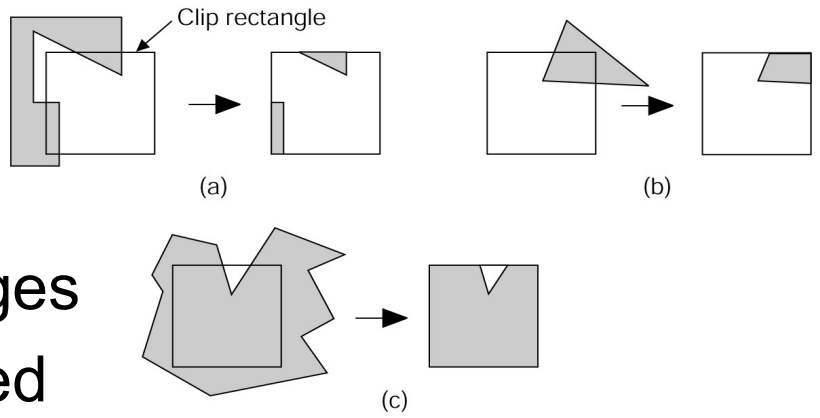
In 3D vertices are co-planar



Lots of different cases

Issues

- Edges of polygon need to be tested against clipping rectangle
- May need to add new edges
- Edges discarded or divided
- Multiple polygons can result from a single polygon



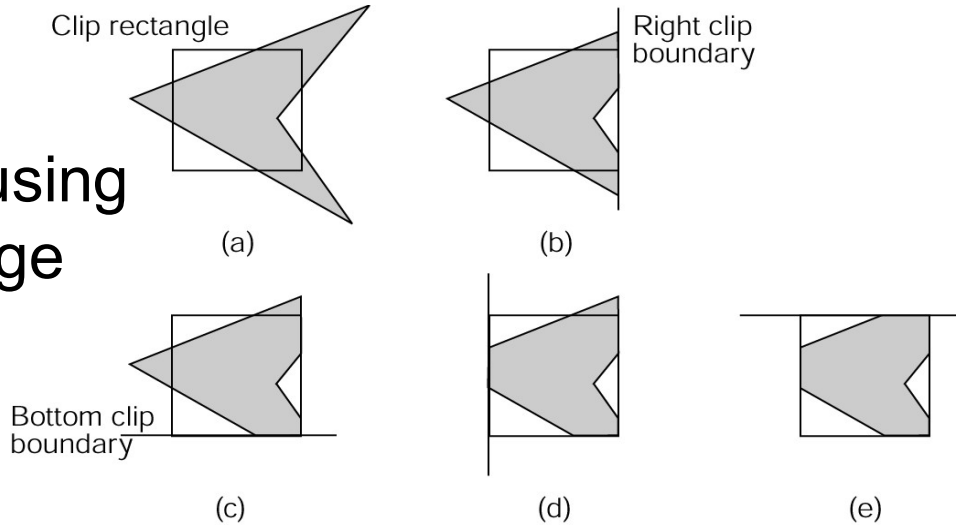
Divide and Conquer

Idea:

- Clip single polygon using single infinite clip edge
- Repeat 4 times

Note the generality:

- 2D convex n-gons can clip arbitrary n-gons
- 3D convex polyhedra can clip arbitrary polyhedra



Sutherland Hodgman Polygon Clipping Algorithm:

Input:

- $v_1, v_2, \ldots v_n$ the vertices defining the polygon
- Single infinite clip edge w/ inside/outside info

Output:

- $v'_1, v'_2, \ldots v'_m$, vertices of the clipped polygon

Do this 4 (or n_e) times

Traverse vertices (edges)

Add vertices one-at-a-time to output polygon

- Use inside/outside info
- Edge intersections

Can be done incrementally

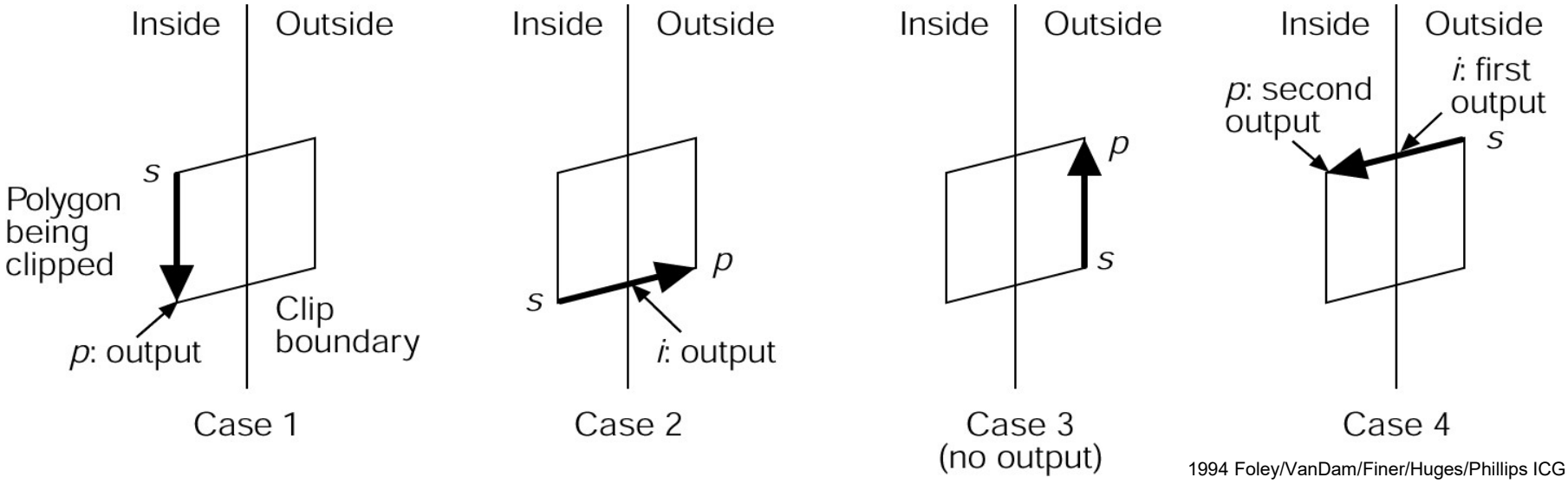
If first point inside add. If outside, don't add

Move around polygon from v_1 to v_n and back to v_1

Check v_i, v_{i+1} wrt the clip edge

Need v_i, v_{i+1}'s inside/outside status

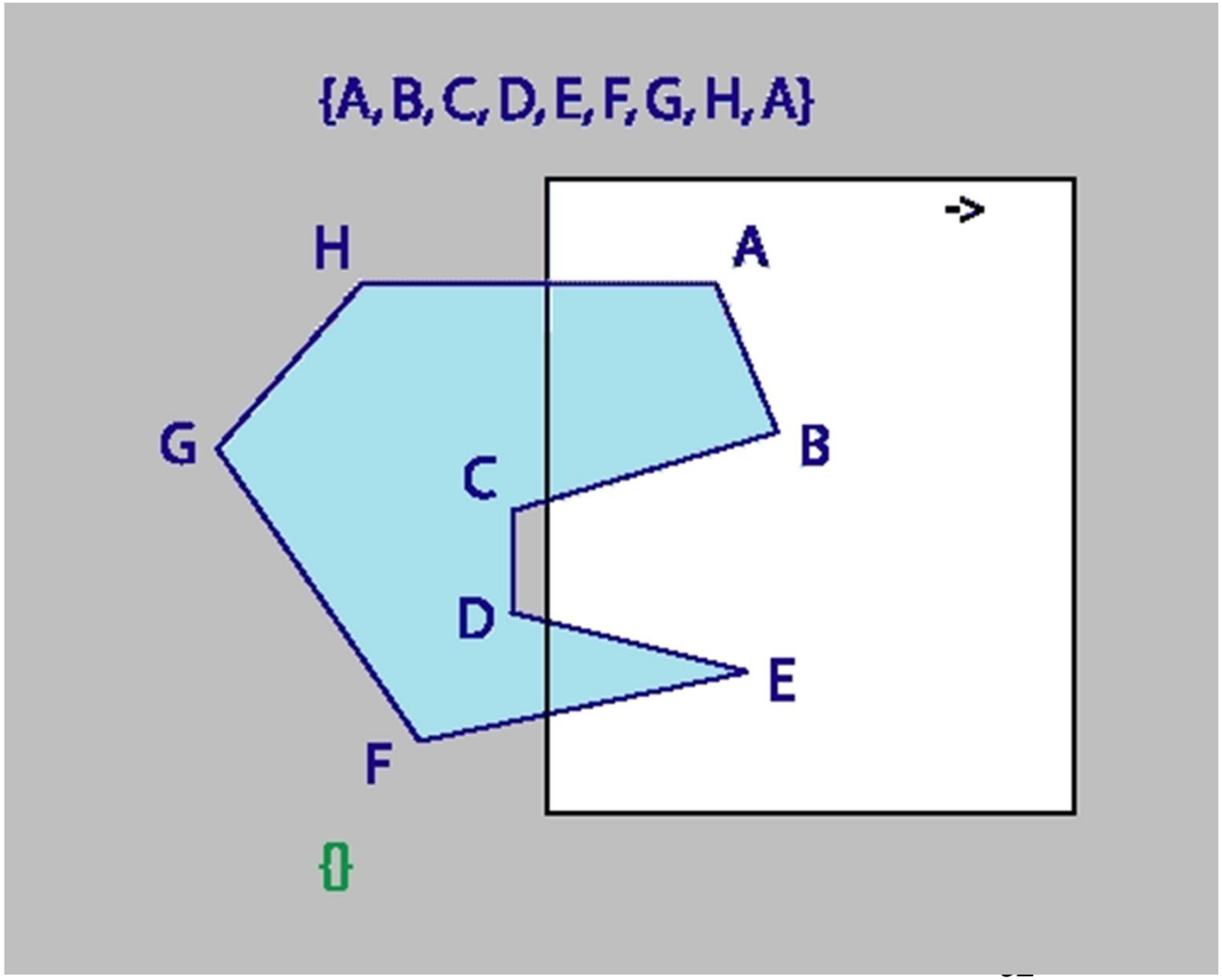
Add vertex one at a time. There are 4 cases:



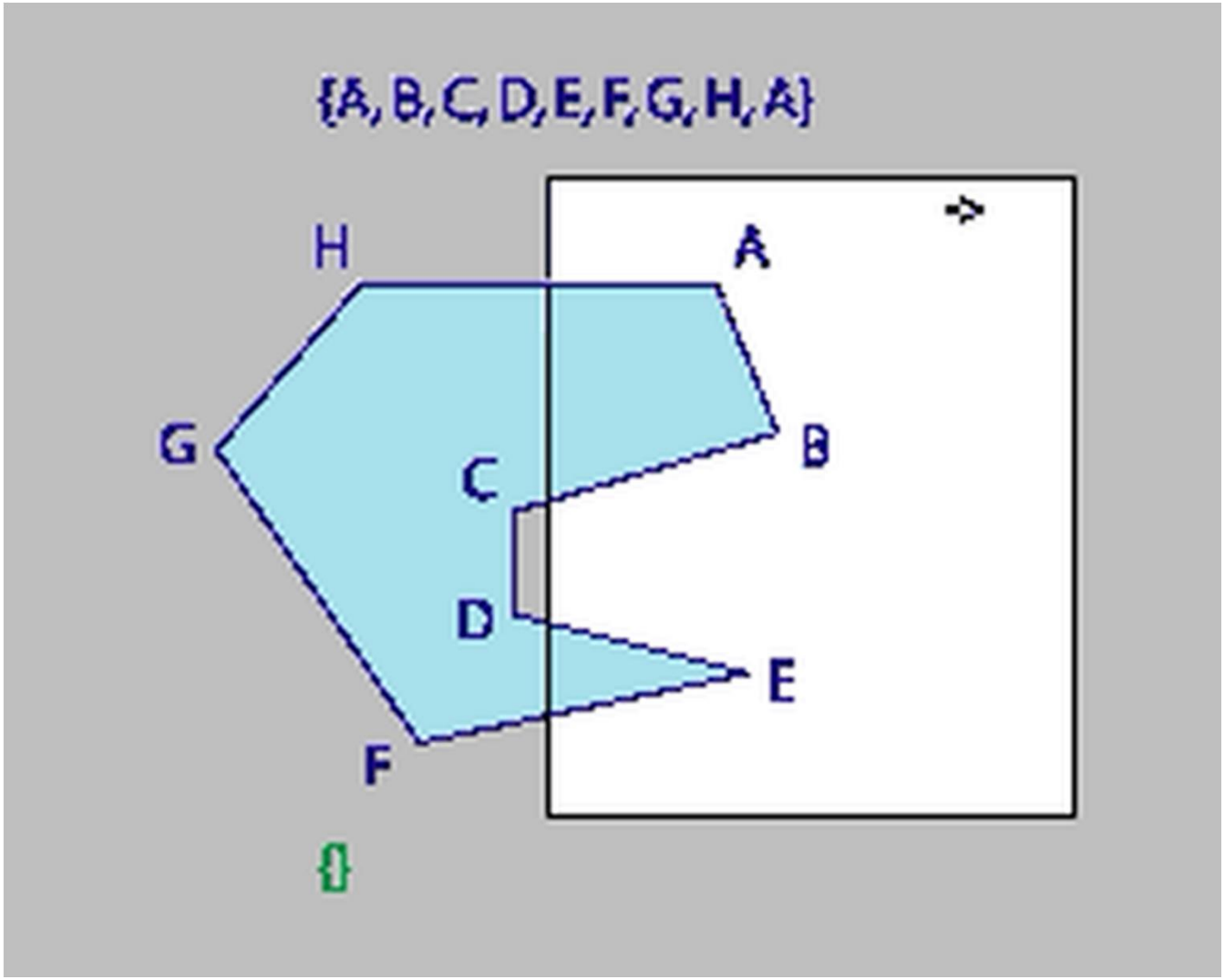
Sutherland-Hodgman Algorithm

foreach polygon ***P*** ***P' = P***

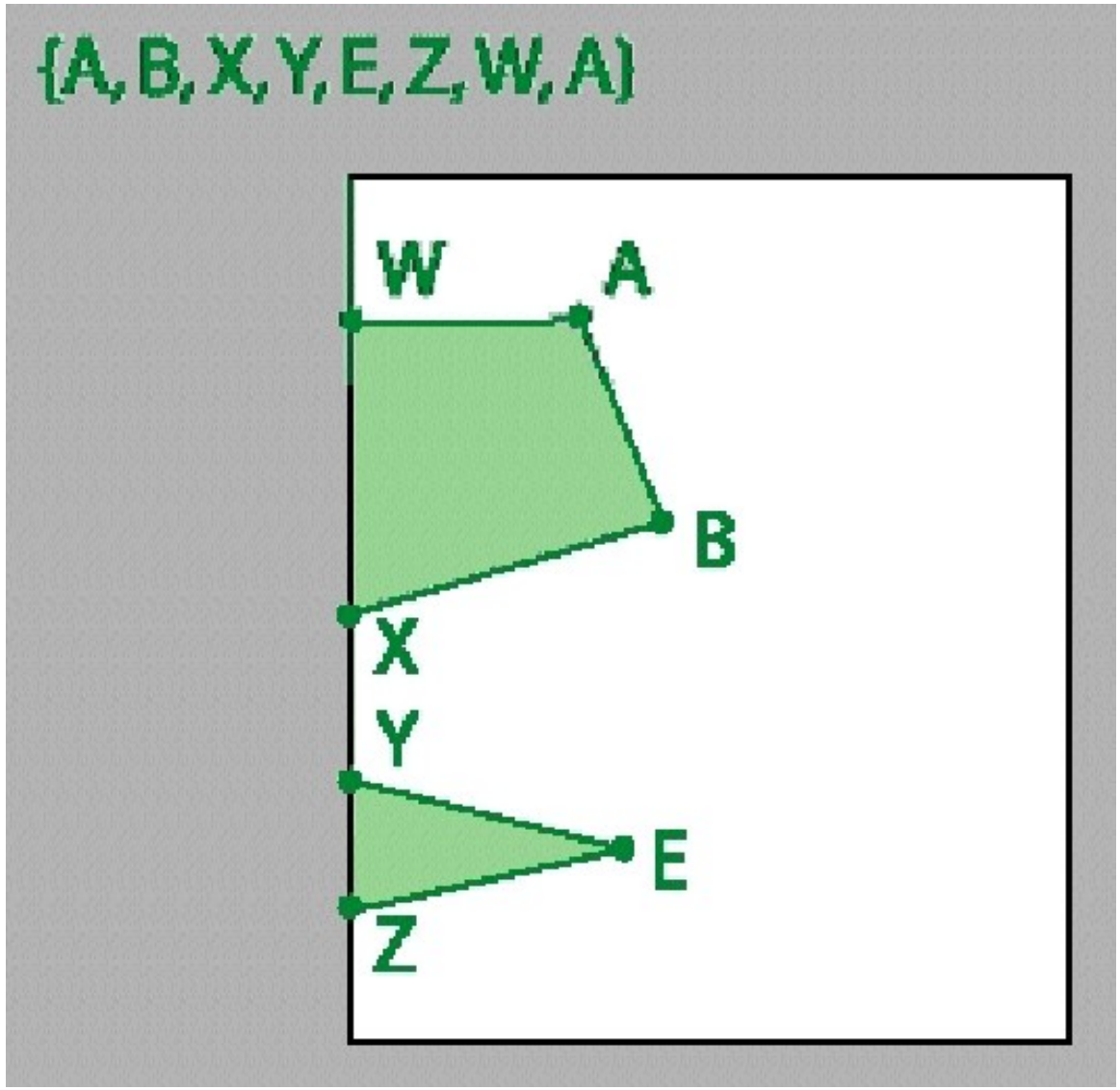
- foreach clipping edge (there are 4) {
  - Clip polygon ***P'*** to clipping edge
    - foreach edge in polygon ***P'***
      - » Check clipping cases (there are 4)
        - » Case 1 : Output v_{i+1}
        - » Case 2 : Output intersection point
        - » Case 3 : No output
        - » Case 4 : Output intersection point & v_{i+1} }



Animated by Max Peysakhov @ Drexel University



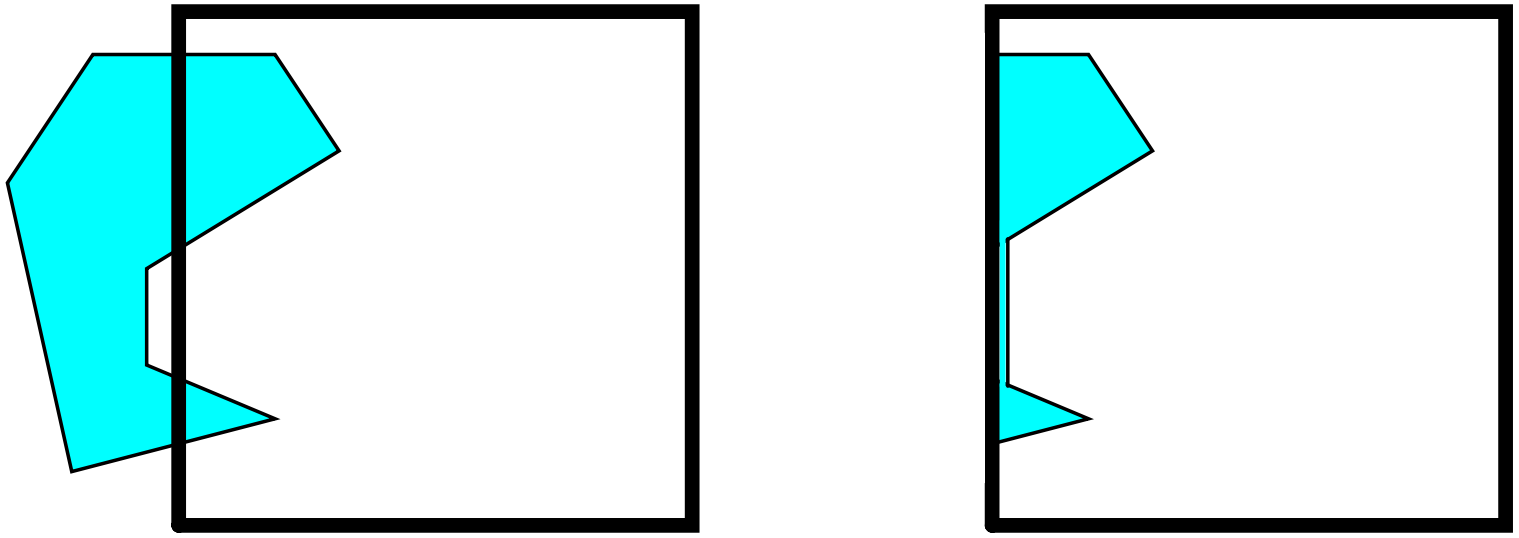
Animated by Max Peysakhov @ Drexel University



Note: Edges XY and ZW!

Clipping a concave polygon

Can produce two CONNECTED areas

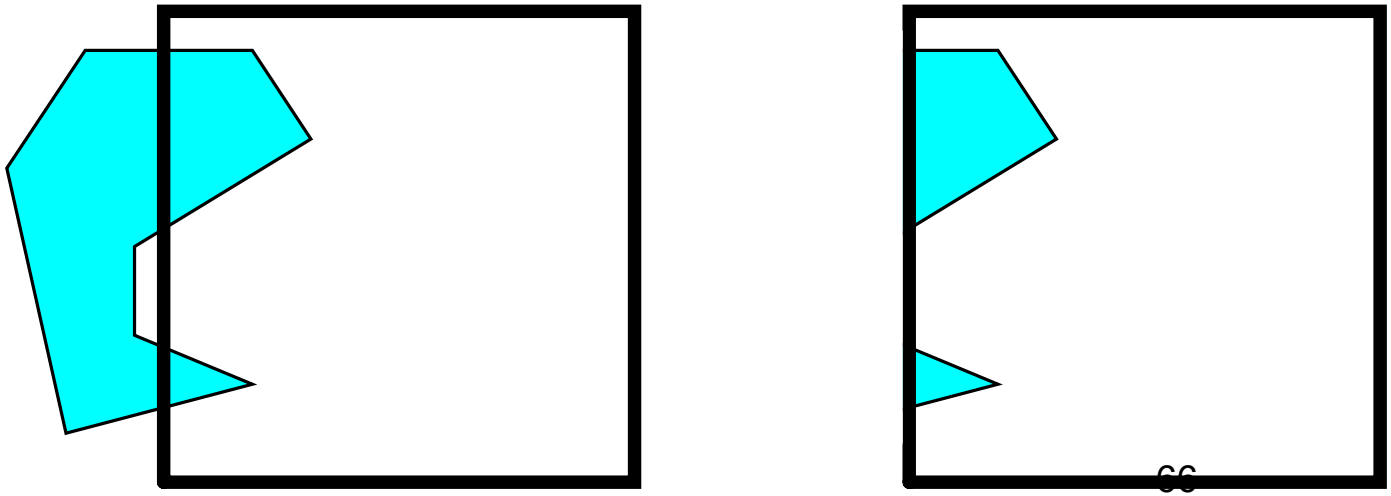


General clipping algorithm for concave polygons with holes

Produces multiple polygons (with holes)

Make linked list data structure

Traverse to make new polygon(s)



Given polygons A and B as linked list of vertices (counter-clockwise order)

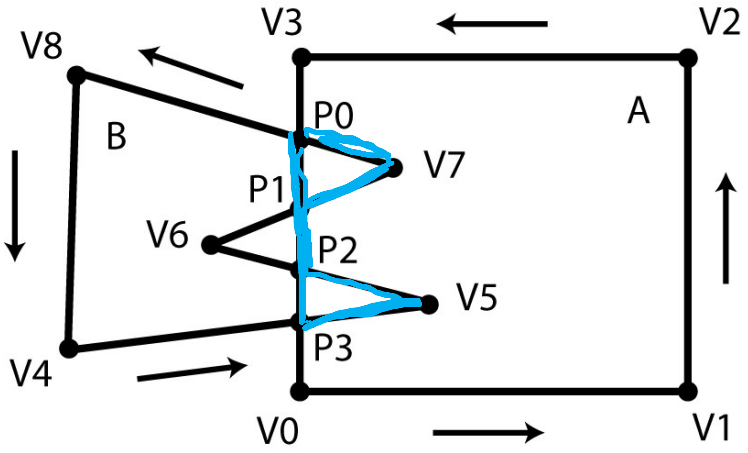
Find all edge intersections & place in list

Insert as “intersection” nodes

Nodes point to A & B

Determine in/out

status of vertices



If “intersecting” edges are parallel, ignore

Intersection point is a vertex

- Vertex of A lies on a vertex or edge of B
- Edge of A runs through a vertex of B
- Replace vertex with an intersection node

Find a vertex of A outside of B

Traverse linked list

At each intersection point switch to other polygon

Do until return to starting vertex

All visited vertices and nodes define union'ed polygon

Start at intersection point

- If connected to an “inside” vertex, go there
- Else step to an intersection point
- If neither, stop

Traverse linked list

At each intersection point switch to other polygon and remove intersection point from list

Do until return to starting intersection point

If intersection list not empty, pick another one

All visited vertices and nodes define and’ed polygon

If polygons don’t intersect

- Union
  - If one inside the other, return polygon that surrounds the other
  - Else, return both polygons
- Intersection
  - If one inside the other, return polygon inside the other
  - Else, return no polygons

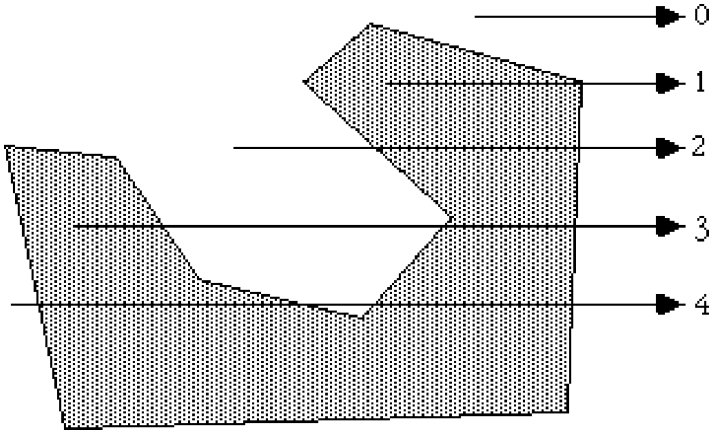
Connect P with another point P` that you know is outside polygon

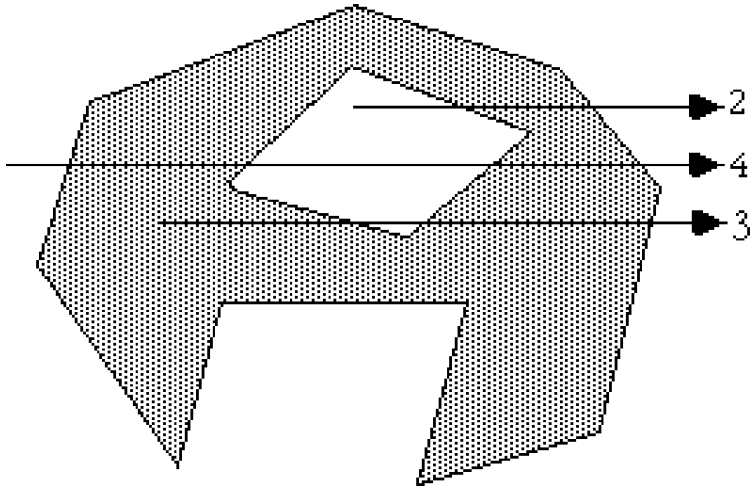
Intersect segment PP` with polygon edges

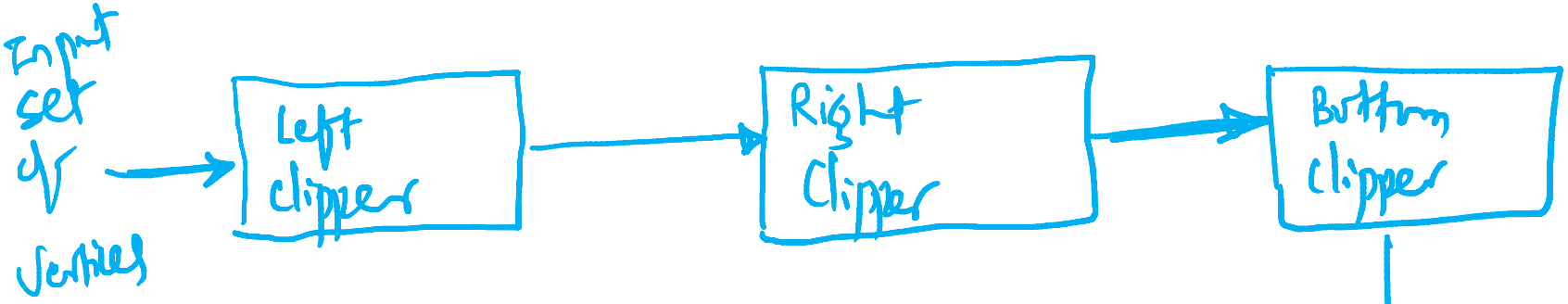
Watch out for vertices!

If # intersections is even (or 0) ⌫ Outside

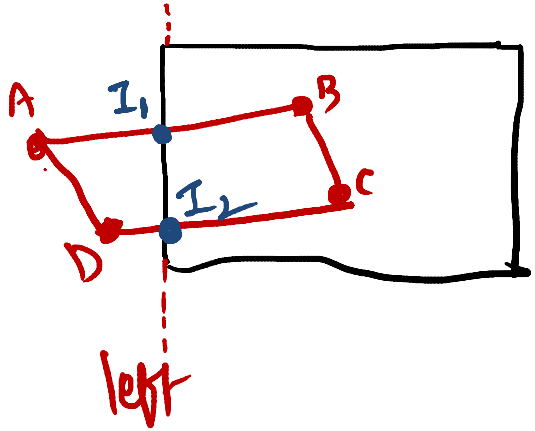
If odd ⌫ Inside







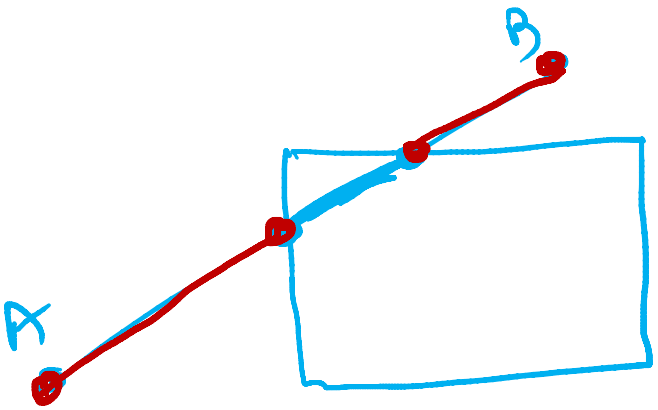
Every clipper is required to deal with 4 cases:



CASE I: Moving from Outside to Inside (i.e A to B)

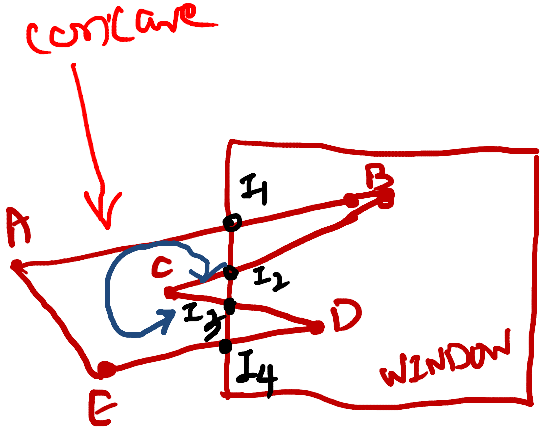
Action: record pt. of intersect (i.e I_1)

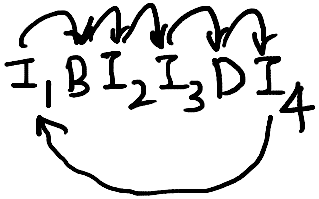
& next vertex (i.e B)

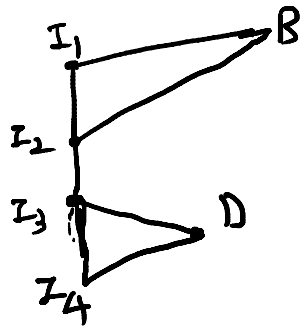


Interior clipping

↳ think of applying exterior clipping

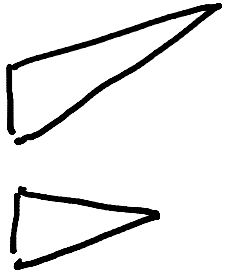






for a concave polygon, Sutherland Hodgman Algo may not ensure successful clipping

Actual Result should be



case II : moving from Inside to Inside B to C

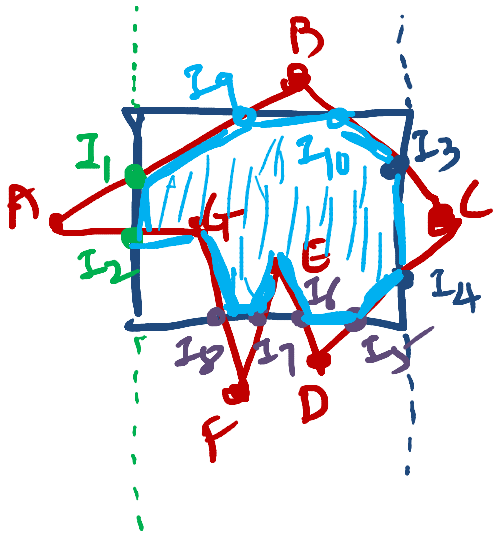
Action: Record next vertex (i.e C)

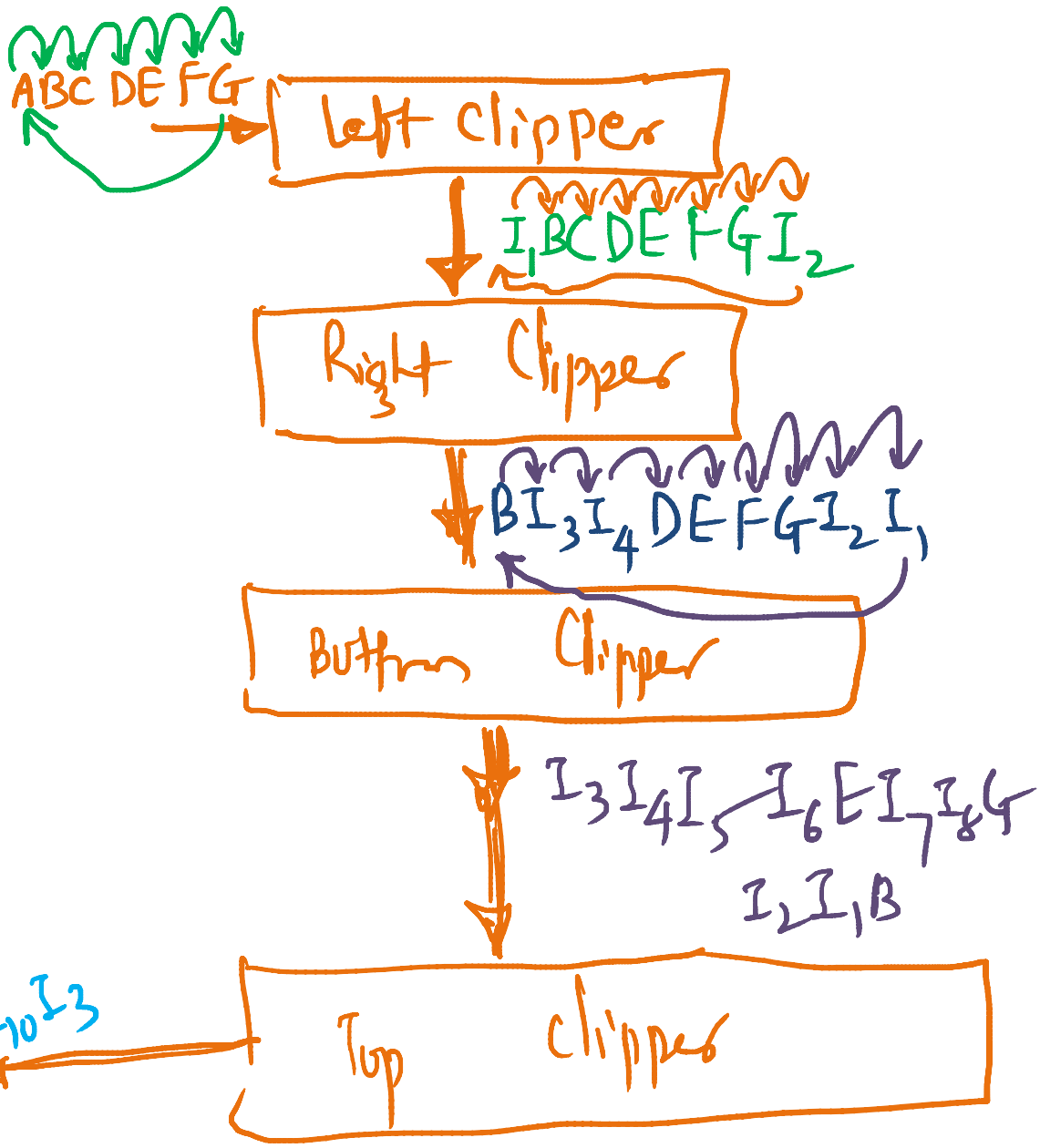
Case III : moving from Inside to Outside
(i.e C to D)

Action : Record pt. of intersect^n
(i.e I_2)

Case IV : moving from Outside to Outside
(D to A)

Action : nothing needs to be recorded


B
I9
I1
I10
I3
A
G
C
E
I2
I6
I4
I8
I7
I5
D
F

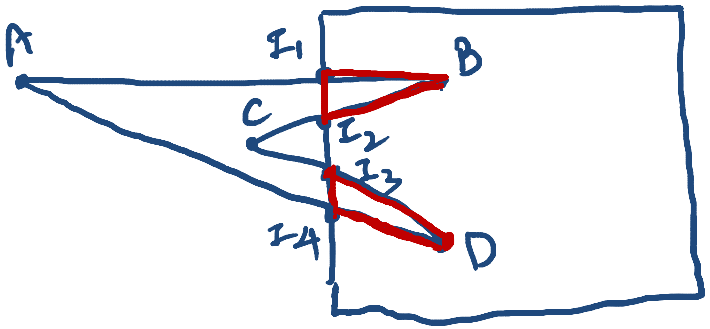

ABC DE FG
Left clipper
I1BCDE FGI2
Right Clipper
BI3I4DEFGI2I1
Bottom Clipper
I3I4I5I6EI7I8G
I2I1B
Top clipper
Final
I4I5I6EI7I8GI2I1I9I10I3

Weiler Atherton Polygon Clipping Algo:

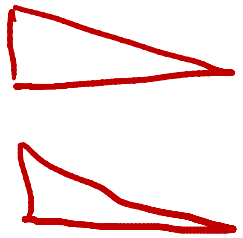
→ Extension of Sutherland Hodgman Algo.

1.) While moving from outside to inside the window, follow the polygon edge.

2.) While moving from inside to outside the window, follow the polygon edge till point of intersection & then start following window edge



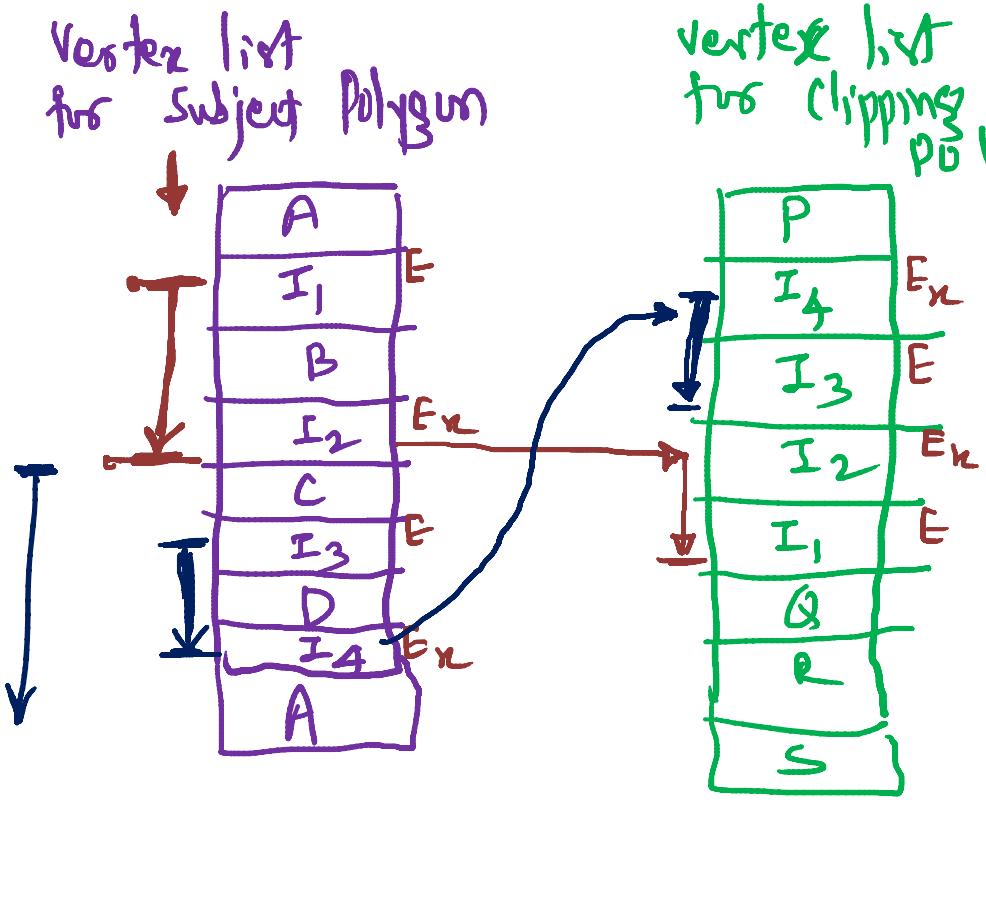
∴ Results into

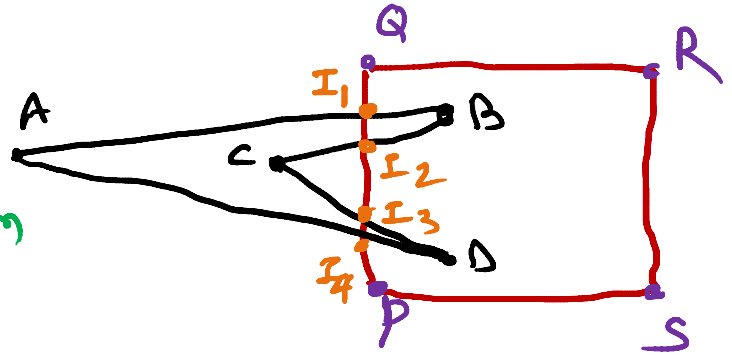


→ The algo. maintains 2 vertex lists

1) Vertex list for subject polygon (i.e polygon to be clipped)

2) Clipping polygon (i.e Window)





E → Entering one

Ex → Exiting one

Result : $I_1 B I_2 I_1$

: $I_3 D I_4 I_3$