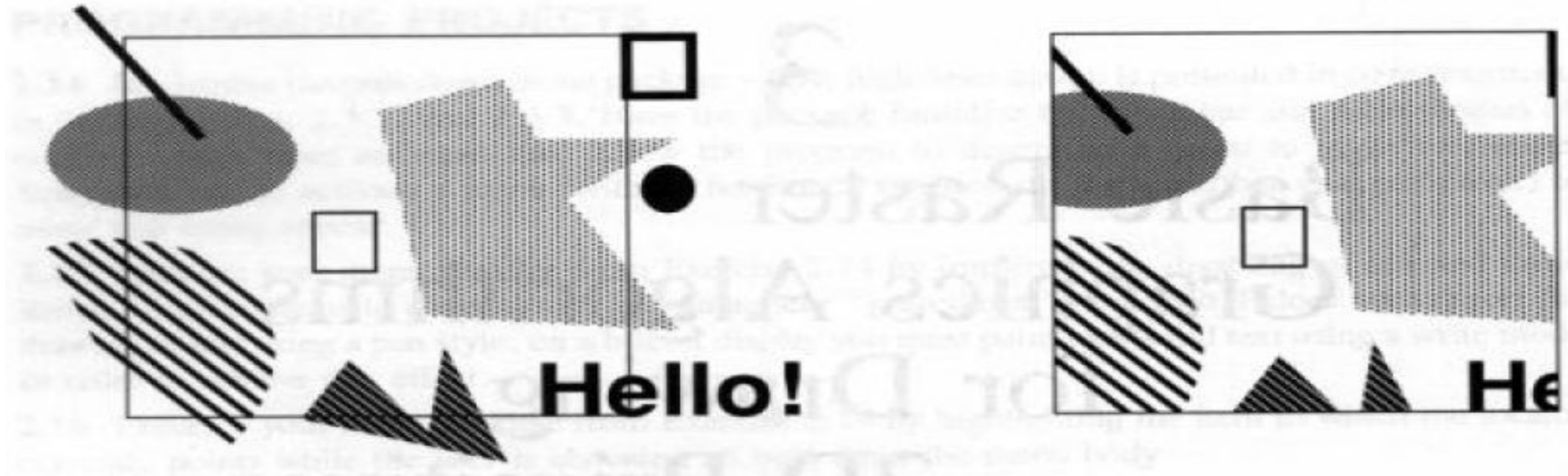


# Clipping

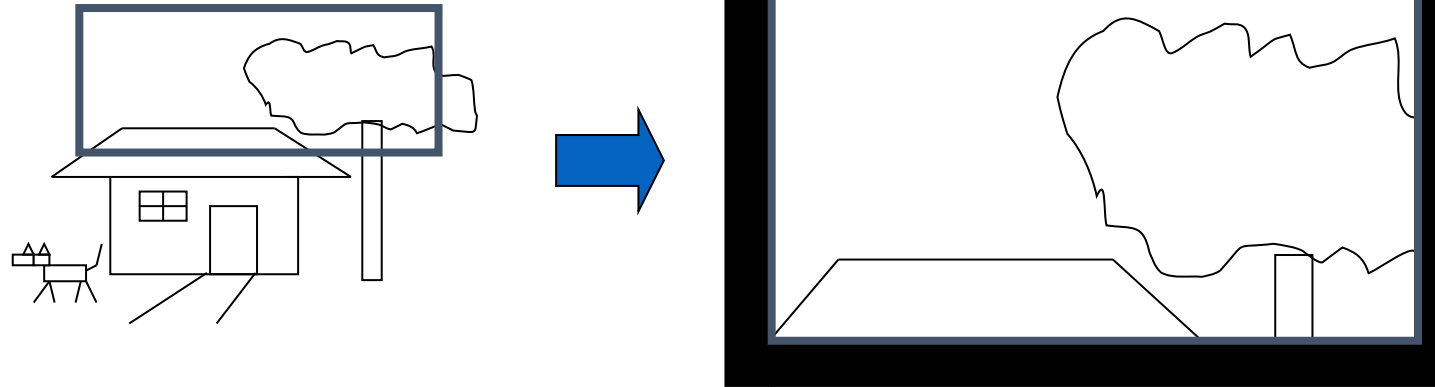
## Cohen-Sutherland Algo



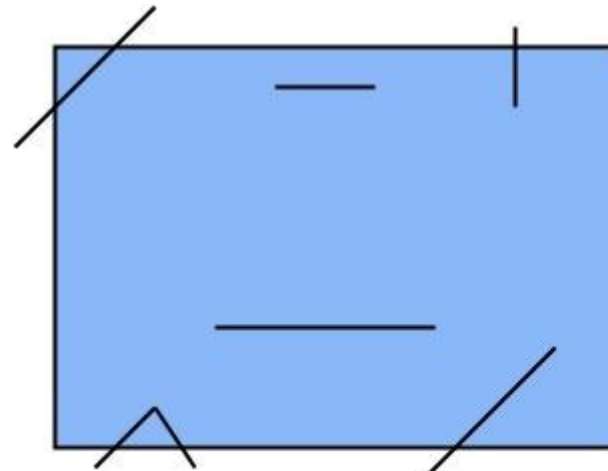
© 2015 Pearson Education, Inc. or its affiliate(s). All rights reserved.



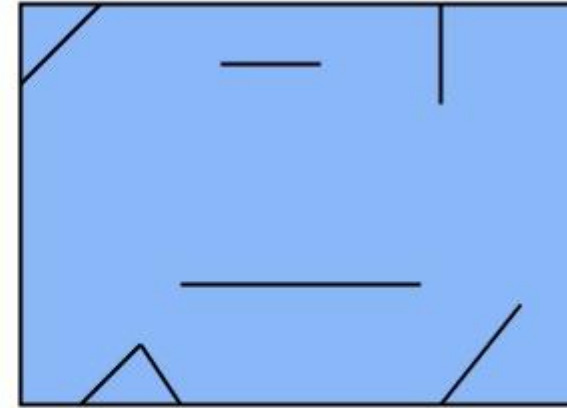
Clipped view in screen



## Line Clipping



**Original Picture  
or  
Before Clipping**



**After Clipping**

# Clipping in a Raster World

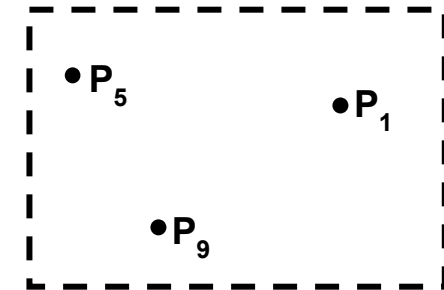
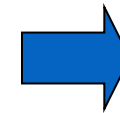
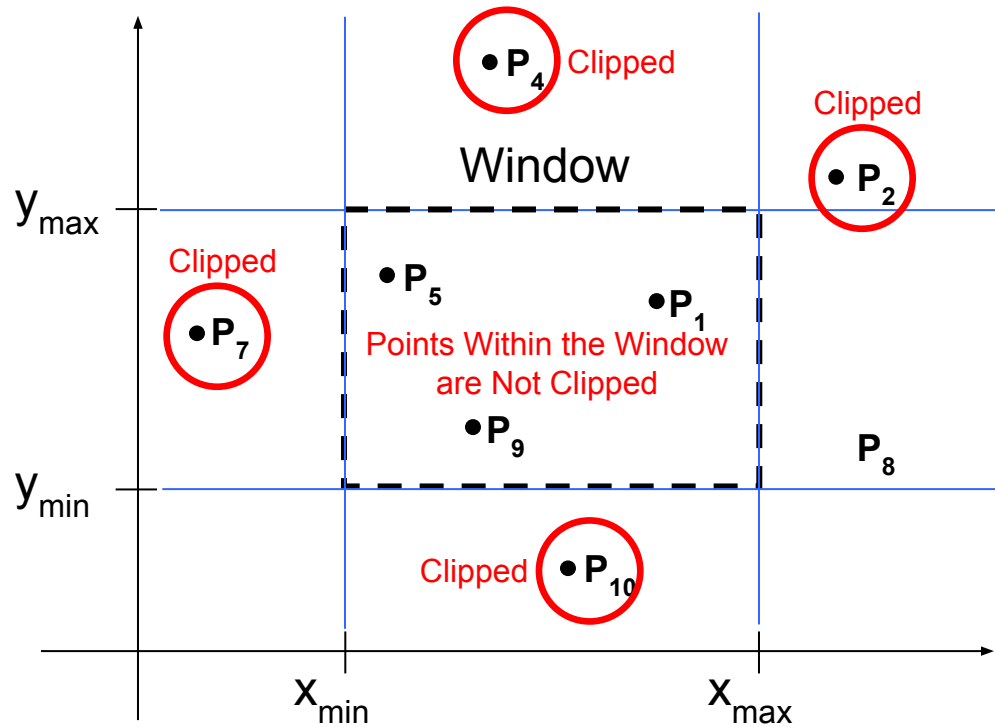
- Clipping techniques
  - **Analytical**
    - Lines, polygons etc.
    - Floating point graphics package
  - During scan conversion (scissoring)
    - Checking extrema suffices, internal points can be ignored
    - Circles, curves etc.
  - During writing a pixel
    - Outline primitive not much larger, few pixels are clipped

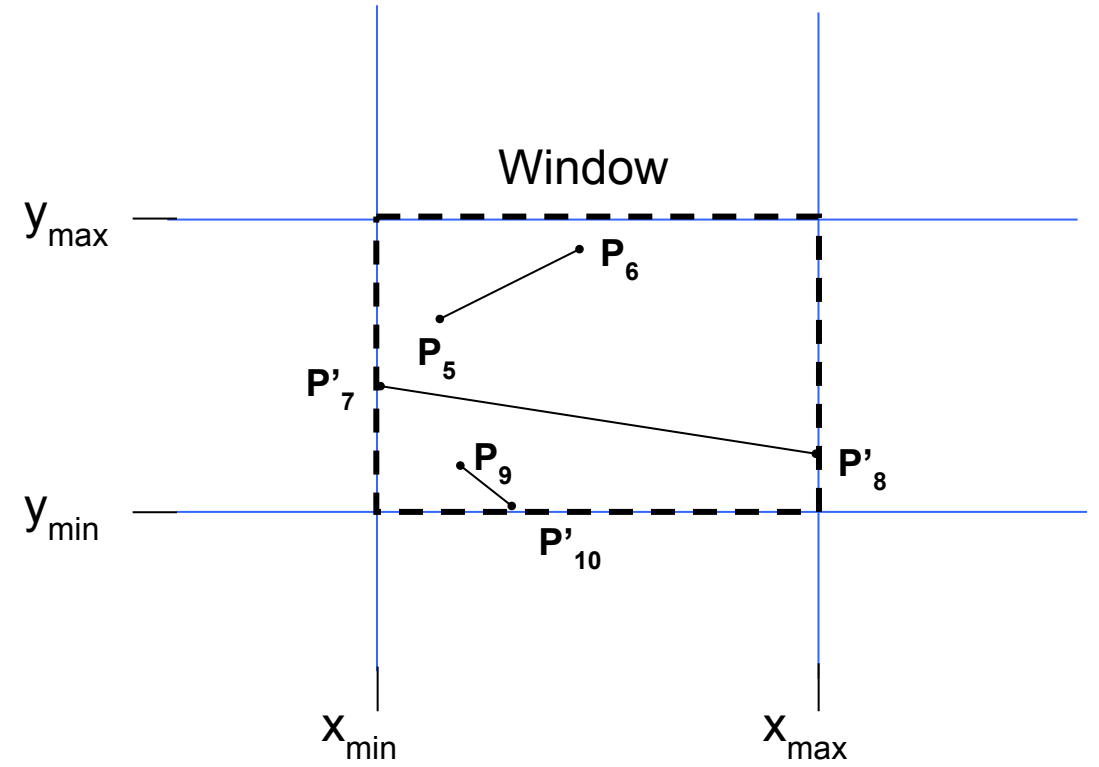
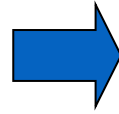
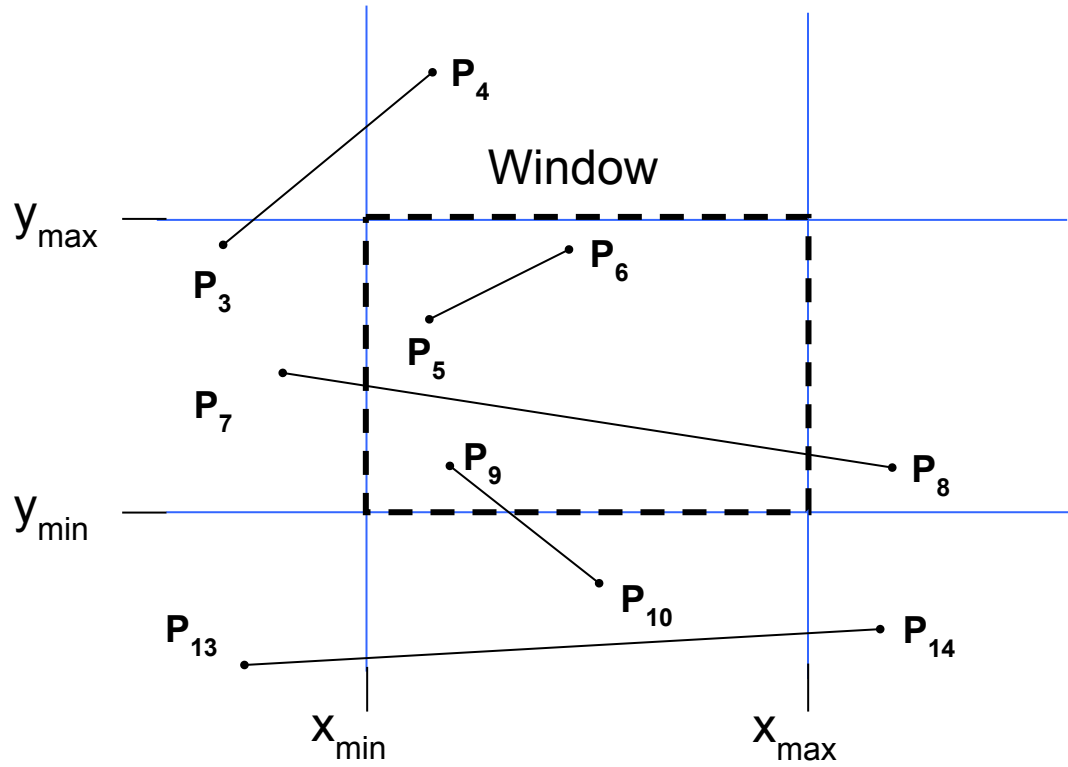


A point  $(x,y)$  is not clipped if:

$$x_{min} \leq x \leq x_{max} \text{ AND } y_{min} \leq y \leq y_{max}$$

otherwise it is clipped



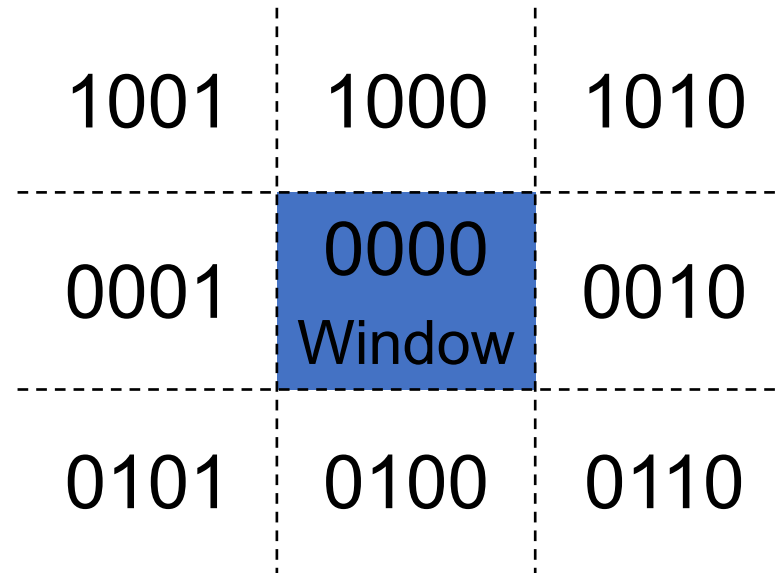


## World space is divided into regions based on the window boundaries

- Each region has a unique four bit outcode
- Outcodes indicate the position of the regions with respect to the window

3	2	1	0
above	below	right	left

Region Code



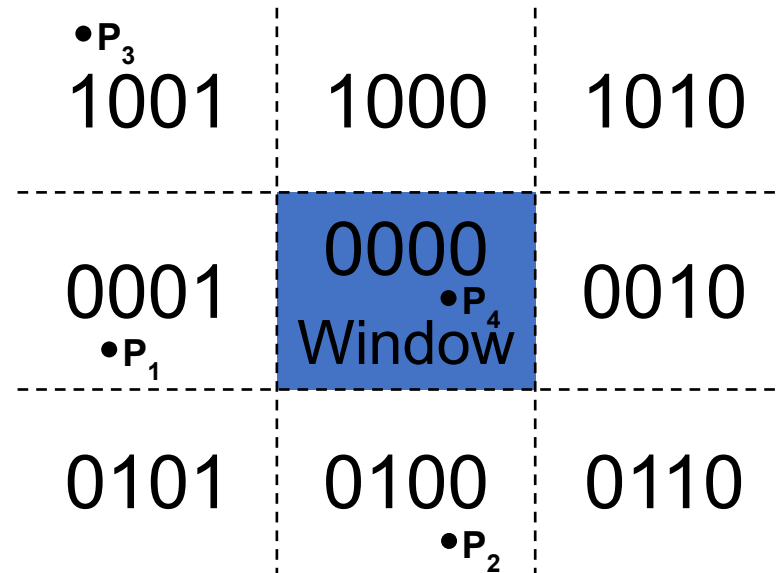


## World space is divided into regions based on the window boundaries

- Each region has a unique four bit outcode
- Outcodes indicate the position of the regions with respect to the window

3	2	1	0
above	below	right	left

Region Code



Calculate\_outcode(x,y){

if ( $x < x_{\min}$ ) bit0 = 1

else bit0=0

if ( $x > x_{\max}$ ) bit1=1

else bit1=0

if( $y < y_{\min}$ ) bit2=1

else bit2=0

if ( $y > y_{\max}$ ) bit3=1

else bit3=0

}

3	2	1	0
above	below	right	left

Region Code

Calculate\_outcode(x,y){

if ( $x < x_{\min}$ ) bit0 = 1

else bit0=0

if ( $x > x_{\max}$ ) bit1=1

else bit1=0

if( $y < y_{\min}$ ) bit2=1

else bit2=0

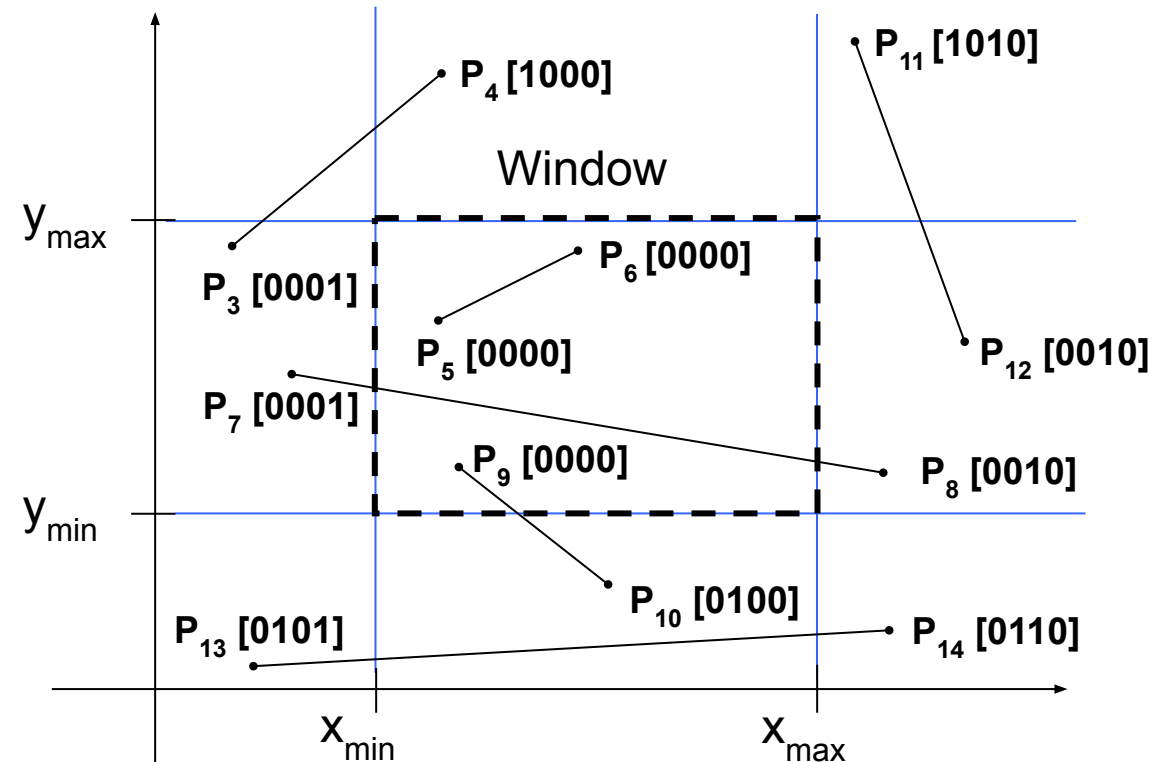
if ( $y > y_{\max}$ ) bit3=1

else bit3=0

}

3	2	1	0
above	below	right	left

Region Code



```

Calculate_Outcode_3D(x,y,z){
    if (x<xmin) bit0 =1
    else bit0=0
    if (x>xmax) bit1=1
    else bit1=0
    if(y<ymin) bit2=1
    else bit2=0
    if (y>ymax) bit3=1
    else bit3=0
    if(z<zmin) bit4=1
    else bit4=0
    if (z>zmax) bit5=1
    else bit5=0
}

```

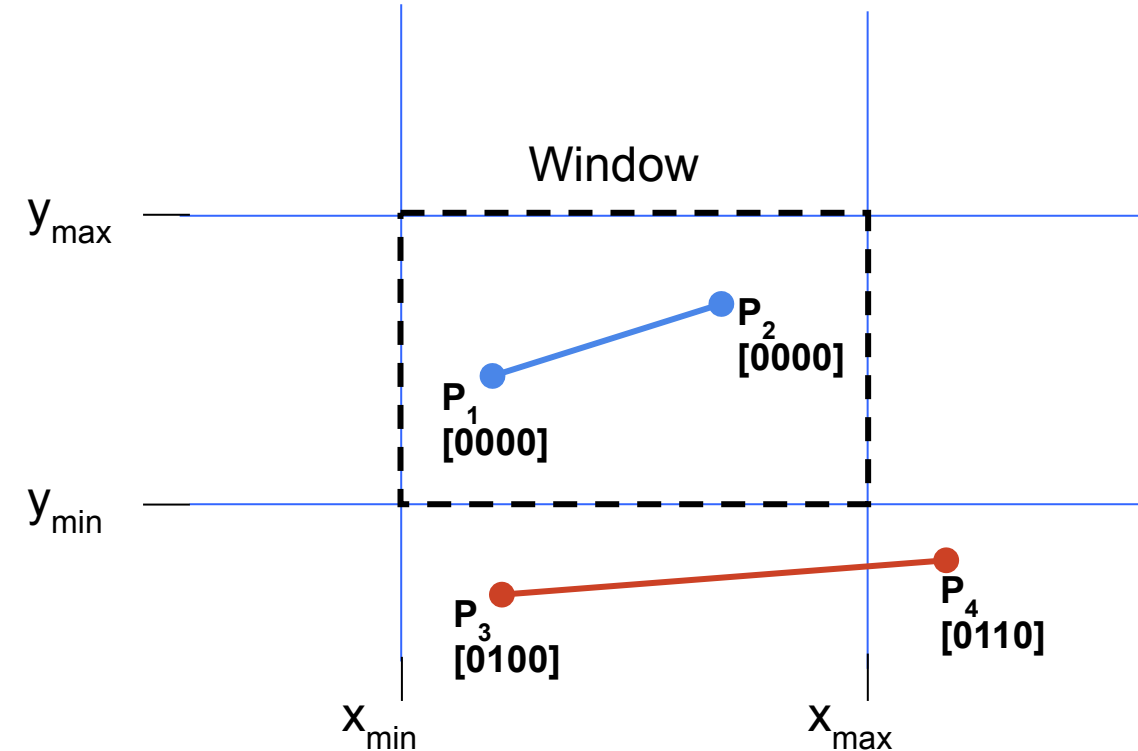
5	4	3	2	1	0
Near	Far	Above	Below	Right	Left
Region Code					

## Condition of Trivial Acceptance:

- Lines that are completely **INSIDE**
- IF  $OC1 == OC2 == 0000$

## Condition of Trivial Rejection:

- Lines that are completely **OUTSIDE**
- IF  $(OC1 \text{ AND } OC2) \neq 0000$



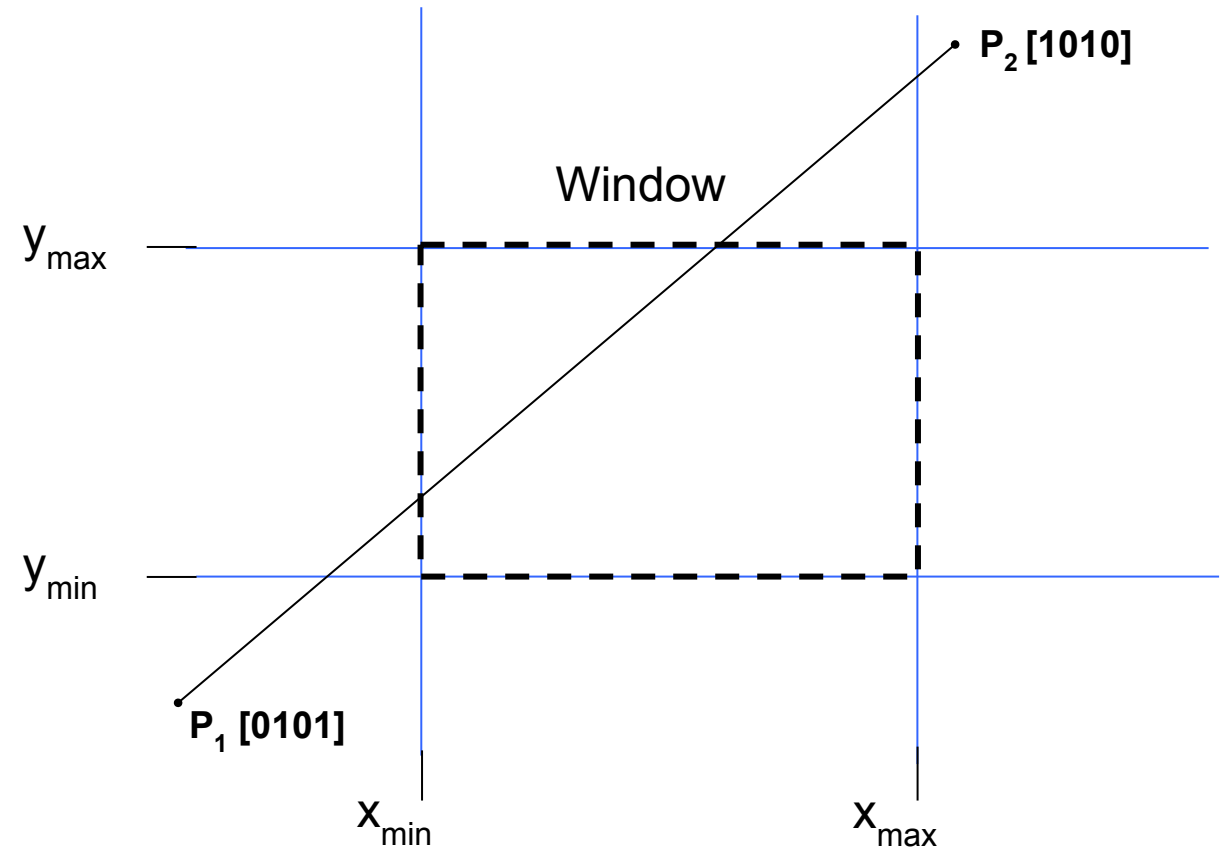
```
cohen-Sutherland(x1, y1, x2, y2):  
oc1 = calculate_outcode(x1, y1),  
oc2 = calculate_outcode(x2, y2);  
while(true) {  
    if(oc1 == oc2 == 0000) {  
        //declare completely inside  
        output (x1, y1), (x2, y2) as clipped line  
        break  
    }  
    else if((oc1 AND oc2) != 0000){ // condition to check matching bit  
        //declare completely outside and clip  
        break  
    }  
    else{  
        if(oc1 != 0000){  
            (x1, y1) = find intersection point of line  
                        and the boundary corresponding  
                        to non-zero bit of oc1  
            oc1 = calculate_outcode(x1, y1)  
        }  
        else{  
            (x2, y2) = find intersection point of line  
                        and the boundary corresponding  
                        to non-zero bit of oc2  
            oc2 = calculate_outcode(x2, y2)  
        }  
        continue  
    }  
}
```



```

cohen-Sutherland(x1, y1, x2, y2):
oc1 = calculate_outcode(x1, y1),
oc2 = calculate_outcode(x2, y2);
while(true) {
    if (oc1 == oc2 == 0000) {
        //declare completely inside
        output (x1, y1), (x2, y2) as clipped line
        break
    }
    else if ((oc1 AND oc2) != 0000) { // condition to check matching bit
        //declare completely outside and clip
        break
    }
    else{
        if(oc1 != 0000){
            (x1, y1) = find intersection point of line
                        and the boundary corresponding
                        to non-zero bit of oc1
            oc1 = calculate_outcode(x1, y1)
        }
        else{
            (x2, y2) = find intersection point of line
                        and the boundary corresponding
                        to non-zero bit of oc2
            oc2 = calculate_outcode(x2, y2)
        }
        continue
    }
}

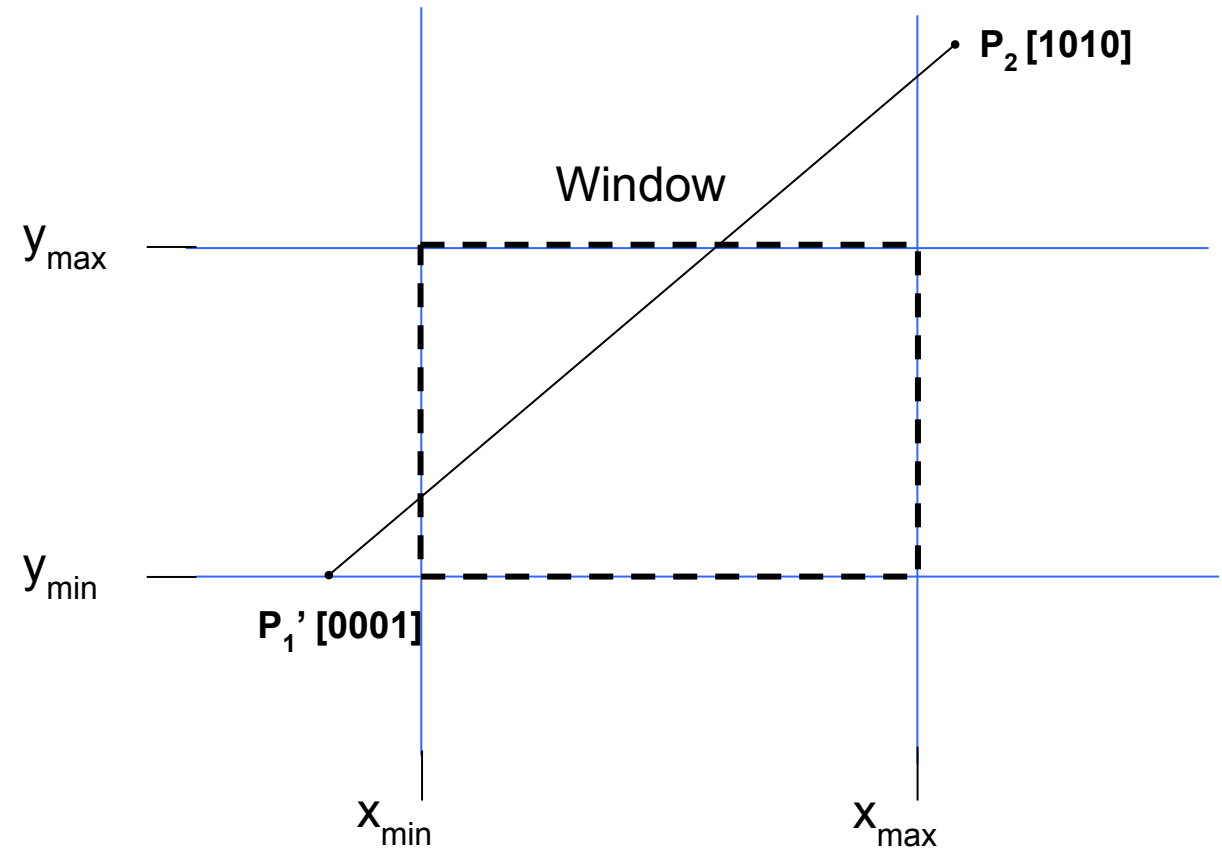
```



```

cohen-Sutherland(x1, y1, x2, y2):
oc1 = calculate_outcode(x1, y1),
oc2 = calculate_outcode(x2, y2);
while(true) {
    if (oc1 == oc2 == 0000) {
        //declare completely inside
        output (x1, y1), (x2, y2) as clipped line
        break
    }
    else if ((oc1 AND oc2) != 0000) { // condition to check matching bit
        //declare completely outside and clip
        break
    }
    else{
        if(oc1 != 0000){
            (x1, y1) = find intersection point of line
                        and the boundary corresponding
                        to non-zero bit of oc1
            oc1 = calculate_outcode(x1, y1)
        }
        else{
            (x2, y2) = find intersection point of line
                        and the boundary corresponding
                        to non-zero bit of oc2
            oc2 = calculate_outcode(x2, y2)
        }
        continue
    }
}

```

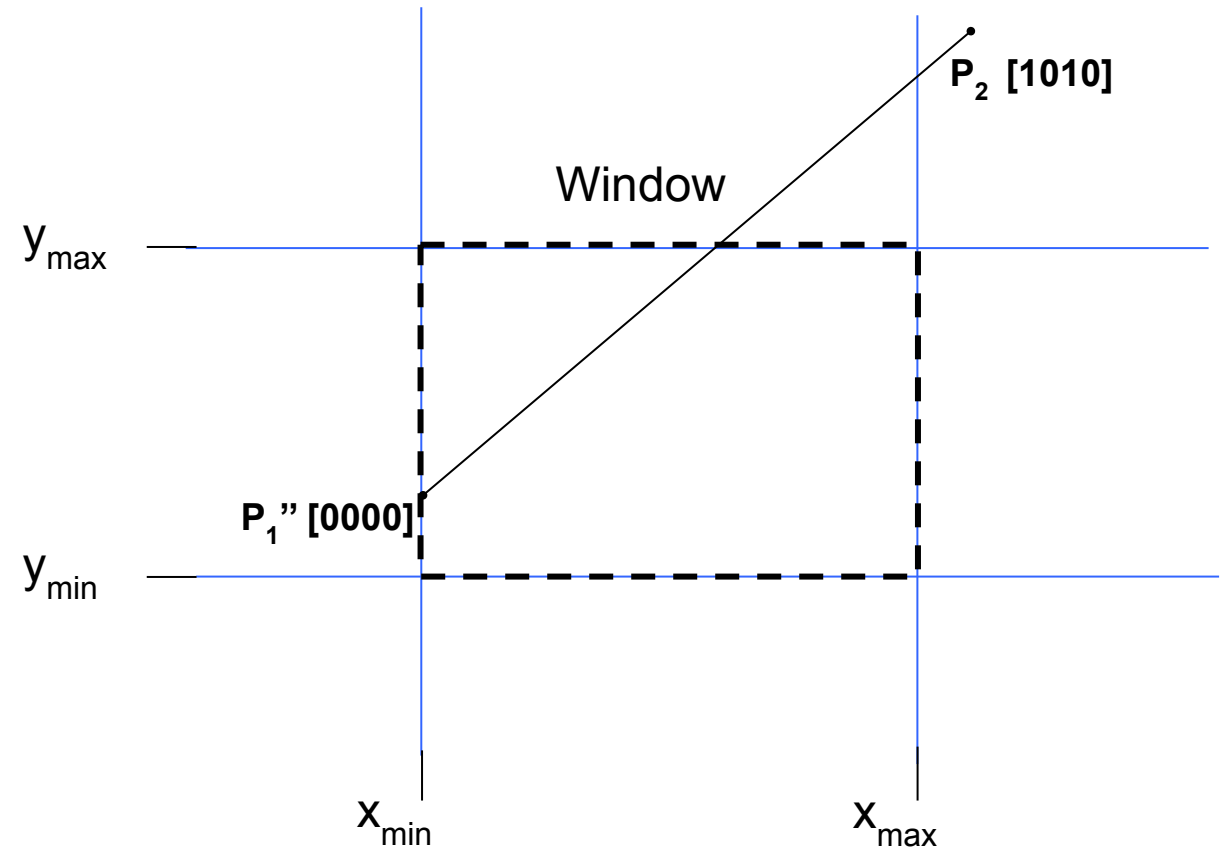




```

cohen-Sutherland(x1, y1, x2, y2):
oc1 = calculate_outcode(x1, y1),
oc2 = calculate_outcode(x2, y2);
while(true) {
    if (oc1 == oc2 == 0000) {
        //declare completely inside
        output (x1, y1), (x2, y2) as clipped line
        break
    }
    else if ((oc1 AND oc2) != 0000) { // condition to check matching bit
        //declare completely outside and clip
        break
    }
    else{
        if(oc1 != 0000){
            (x1, y1) = find intersection point of line
                        and the boundary corresponding
                        to non-zero bit of oc1
            oc1 = calculate_outcode(x1, y1)
        }
        else{
            (x2, y2) = find intersection point of line
                        and the boundary corresponding
                        to non-zero bit of oc2
            oc2 = calculate_outcode(x2, y2)
        }
        continue
    }
}

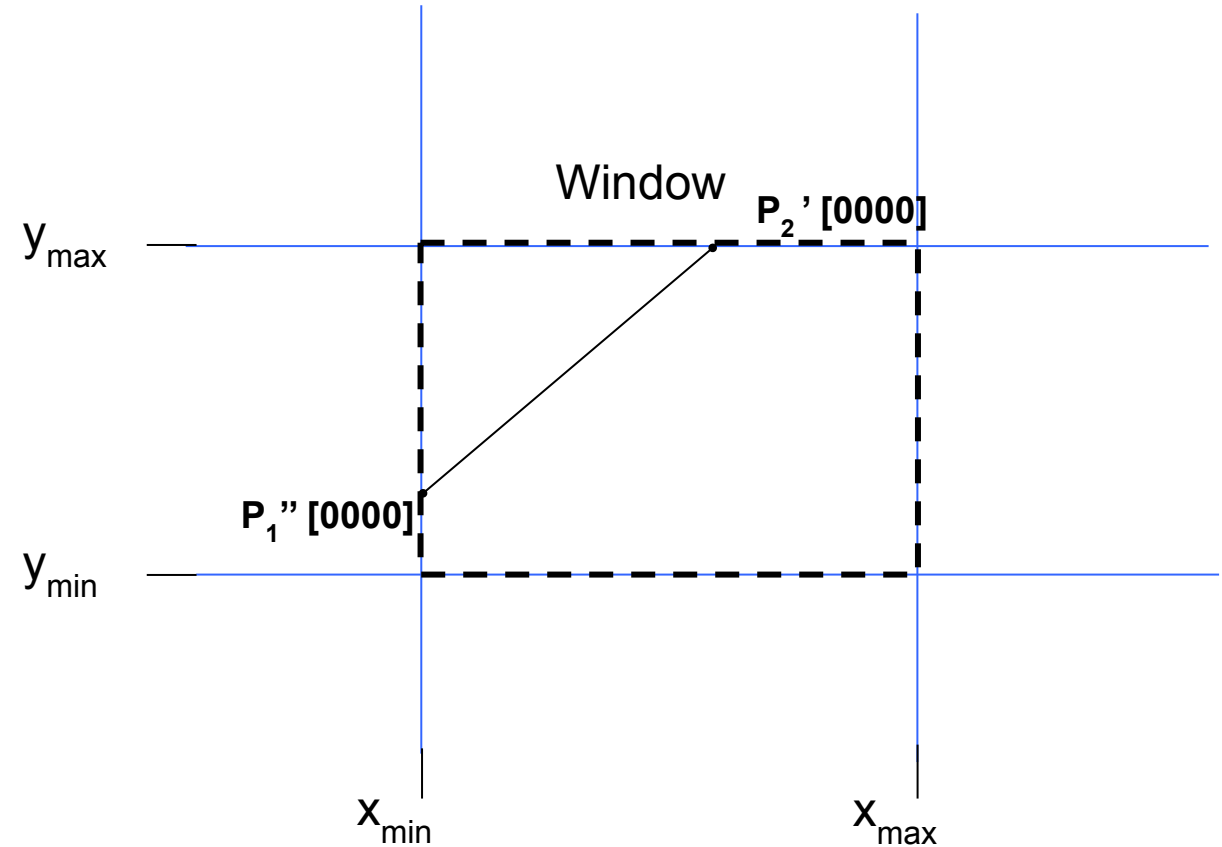
```



```

cohen-Sutherland(x1, y1, x2, y2):
oc1 = calculate_outcode(x1, y1),
oc2 = calculate_outcode(x2, y2);
while(true) {
    if (oc1 == oc2 == 0000) {
        //declare completely inside
        output (x1, y1), (x2, y2) as clipped line
        break
    }
    else if ((oc1 AND oc2) != 0000) { // condition to check matching bit
        //declare completely outside and clip
        break
    }
    else{
        if(oc1 != 0000){
            (x1, y1) = find intersection point of line
                        and the boundary corresponding
                        to non-zero bit of oc1
            oc1 = calculate_outcode(x1, y1)
        }
        else{
            (x2, y2) = find intersection point of line
                        and the boundary corresponding
                        to non-zero bit of oc2
            oc2 = calculate_outcode(x2, y2)
        }
        continue
    }
}

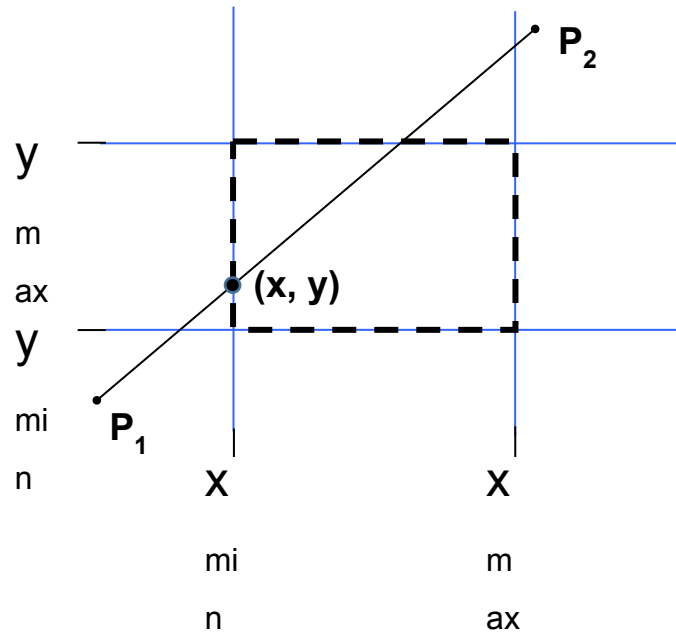
```



LEFT boundary intersection:

$$x = x_{\min}$$

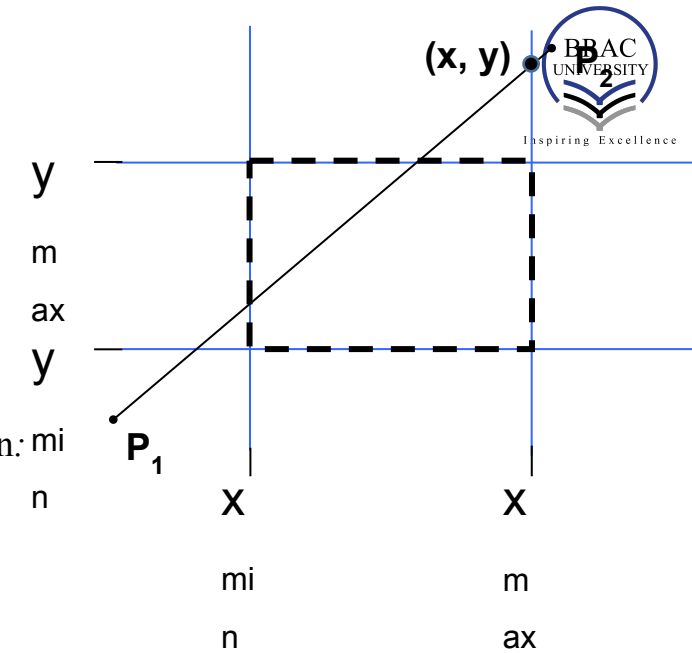
$$y = y_l + m (x_{\min} - x_l)$$



RIGHT boundary intersection:

$$x = x_{\max}$$

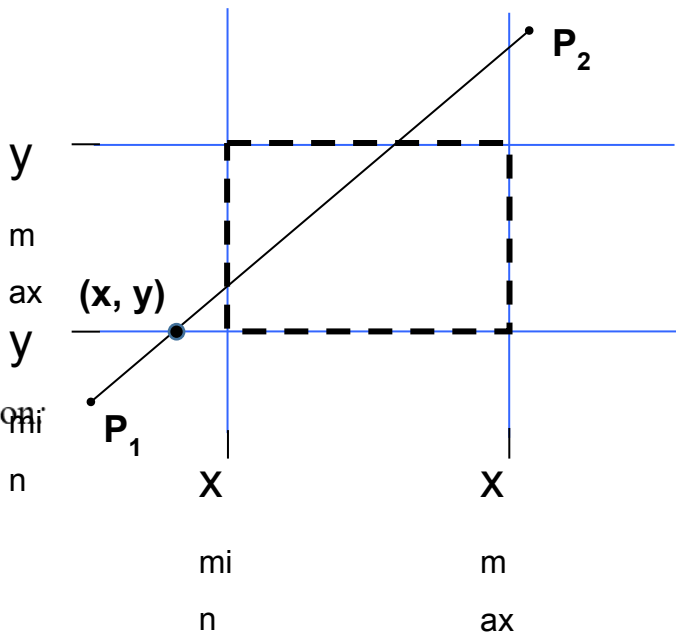
$$y = y_l + m (x_{\max} - x_l)$$



BOTTOM boundary intersection:

$$y = y_{\min}$$

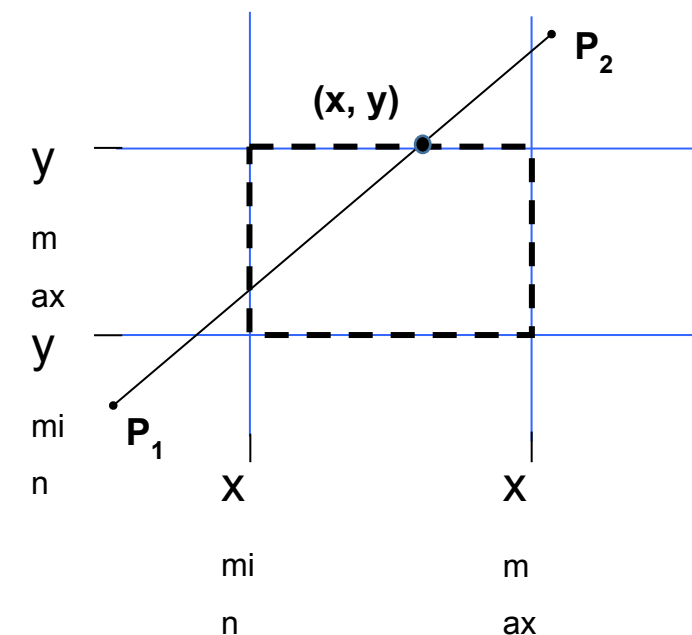
$$x = x_l + \frac{1}{m} \cdot (y_{\min} - y_l)$$



TOP boundary intersection:

$$y = y_{\max}$$

$$x = x_l + \frac{1}{m} \cdot (y_{\max} - y_l)$$



Determine whether the following line are accepted/rejected/partial using Cohen Sutherland line clipping algorithm.

a) Given  $(-250, -200)$  to  $(250, 200)$  be the clip region.

- (i)  $(-100, -220)$  to  $(300, -210)$ .
- (ii)  $(-250, 200)$  to  $(250, -200)$ .

b) Given  $(0, 0)$  to  $(300, 200)$  be the clip region.

- (i)  $(50, -125)$  to  $(-100, 225)$ .
- (ii)  $(-250, 200)$  to  $(250, -200)$ .

If they are partially accepted/rejected find the line segment within the clipping window.



a)(i) boundary:

$$x_{min} = -250, x_{max} = 250, y_{min} = -200, y_{max} = 200$$

points:

$$x_1 = -100, y_1 = -220, x_2 = 300, y_2 = -210$$

Outcode calculation:

$$x_{min} < x_1 < x_{max}$$

so, no left or right bit

$$y_1 < y_{min},$$

so, bottom bit is 1

$$\text{so, outcode1} = 0100$$

$$x_1 > x_{max}$$

so, right bit

$$y_2 < y_{min}$$

so, bottom bit

$$\text{so, outcode2} = 0110$$

$$\text{outcode1 AND outcode2} = 0100 \neq 0000$$

so the line is completely outside.

cohen-Sutherland( $x_1, y_1, x_2, y_2$ ):

oc1 = calculate\_outcode( $x_1, y_1$ ),

oc2 = calculate\_outcode( $x_2, y_2$ );

while(true) {

if (oc1 == oc2 == 0000) {

//declare completely inside

output ( $x_1, y_1$ ), ( $x_2, y_2$ ) as clipped line

break

}

else if ((oc1 AND oc2) != 0000) { // condition to check matching bit

//declare completely outside and clip

break

}

else{

if(oc1 != 0000){

( $x_1, y_1$ ) = find intersection point of line

and the boundary corresponding

to non-zero bit of oc1

oc1 = calculate\_outcode( $x_1, y_1$ )

}

else{

( $x_2, y_2$ ) = find intersection point of line

and the boundary corresponding

to non-zero bit of oc2

oc2 = calculate\_outcode( $x_2, y_2$ )

}

continue

}

a)(ii) boundary:

$$x_{min} = -250, \quad x_{max} = 250, \quad y_{min} = -200, \quad y_{max} = 200$$

points:

$$x_1 = -250, y_1 = 200, x_2 = 250, y_2 = -200$$

Outcode calculation:

$$x_{min} \leq x_1 < x_{max}$$

so, no left or right bit

$$y_{min} < y_1 \leq y_{max},$$

so, no top or bottom bit

so, outcode1 = 0000

similarly, outcode2 = 0000

since outcode1 & outcode 2 both are 0000

so the line is completely inside.

cohen-Sutherland( $x_1, y_1, x_2, y_2$ ):

oc1 = calculate\_outcode( $x_1, y_1$ ),

oc2 = calculate\_outcode( $x_2, y_2$ );

```
while(true) {
    if (oc1 == oc2 == 0000) {
        //declare completely inside
        output (x1, y1), (x2, y2) as clipped line
        break
    }
    else if ((oc1 AND oc2) != 0000) { // condition to check matching bit
        //declare completely outside and clip
        break
    }
    else{
        if(oc1 != 0000){
            (x1, y1) = find intersection point of line
                        and the boundary corresponding
                        to non-zero bit of oc1
            oc1 = calculate_outcode(x1, y1)
        }
        else{
            (x2, y2) = find intersection point of line
                        and the boundary corresponding
                        to non-zero bit of oc2
            oc2 = calculate_outcode(x2, y2)
        }
        continue
    }
}
```

● b)(i) boundary:

$$x_{min} = 0, x_{max} = 300, y_{min} = 0, y_{max} = 200$$

points:

$$x_1 = 50, y_1 = -125, x_2 = -100, y_2 = 225$$

Outcode calculation:

outcode1 = 0100

outcode2 = 1001

outcode1 AND outcode2 = 0000

so partially inside

outcode1 != 0000

outcode1 has bottom bit

Applying bottom intersection:

$$y_1 = y_{min} = 0$$

$$x_1 = x_1 + \frac{1}{m} \cdot (y_{min} - y_1) = 50 + \frac{-150}{350} \cdot (0 + 125) = -3.57$$

outcode1 = 0001 [recalculated]

outcode2 = 1001

outcode1 AND outcode2 = 0001

so completely outside

cohen-Sutherland(x1, y1, x2, y2):

oc1 = calculate\_outcode(x1, y1),

oc2 = calculate\_outcode(x2, y2);

while(true) {

if (oc1 == oc2 == 0000) {

//declare completely inside

output (x1, y1), (x2, y2) as clipped line

break

}

else if ((oc1 AND oc2) != 0000) { // condition to check matching bit

//declare completely outside and clip

break

}

else{

if(oc1 != 0000){

(x1, y1) = find intersection point of line

and the boundary corresponding

to non-zero bit of oc1

oc1 = calculate\_outcode(x1, y1)

}

else{

(x2, y2) = find intersection point of line

and the boundary corresponding

to non-zero bit of oc2

oc2 = calculate\_outcode(x2, y2)

}

continue

}

b)(ii) boundary:

$$x_{min} = 0, x_{max} = 300, y_{min} = 0, y_{max} = 200$$

points:

$$x_1 = -250, y_1 = 200, x_2 = 250, y_2 = -200$$

Outcode calculation:

$$\text{outcode1} = 0001$$

$$\text{outcode2} = 0100$$

$$\text{outcode1 AND outcode2} = 0000$$

so partially inside

$$\text{outcode1} \neq 0000$$

outcode1 has left bit

applying left intersection:

$$x_1 = x_{min} = 0$$

$$y_1 = y_1 + m \cdot (x_{min} - x_1) = 200 + \frac{-400}{500} \cdot (0 + 250) = 0$$

$$\text{outcode1} = 0000 \text{ [recalculated]}$$

so (x1, y1) has been clipped to (0, 0)

$$\text{outcode2} = 0100$$

so partially inside

cohen-Sutherland(x1, y1, x2, y2):

oc1 = calculate\_outcode(x1, y1),

oc2 = calculate\_outcode(x2, y2);

```
while(true) {
    if (oc1 == oc2 == 0000) {
        //declare completely inside
        output (x1, y1), (x2, y2) as clipped line
        break
    }
    else if ((oc1 AND oc2) != 0000) { // condition to check matching bit
        //declare completely outside and clip
        break
    }
    else{
        if(oc1 != 0000){
            (x1, y1) = find intersection point of line
                        and the boundary corresponding
                        to non-zero bit of oc1
            oc1 = calculate_outcode(x1, y1)
        }
        else{
            (x2, y2) = find intersection point of line
                        and the boundary corresponding
                        to non-zero bit of oc2
            oc2 = calculate_outcode(x2, y2)
        }
        continue
    }
}
```



b)(ii) boundary:

$$x_{min} = 0, x_{max} = 300, y_{min} = 0, y_{max} = 200$$

points:

$$x_1 = 0, y_1 = 0, x_2 = 250, y_2 = -200$$

(continued)

outcode1 = 0000 [recalculated]

outcode2 = 0100

so partially inside

outcode1 = 0000

so going into else codeblock,

outcode 2 has bottom bit

applying bottom intersection:

$$y_2 = y_{min} = 0$$

$$x_2 = x_2 + \frac{1}{m} \cdot (y_{min} - y_2) = 250 + \frac{250}{-200} \cdot (0 + 200) = 0$$

[Note: m has been recalculated, you can skip recalculation too]

outcode2 = 0000

outcode1=0000

so completely inside

The clipped segment is between (0, 0) to (0, 0) which is just a single point.

cohen-Sutherland(x1, y1, x2, y2):

oc1 = calculate\_outcode(x1, y1),

oc2 = calculate\_outcode(x2, y2);

while(true) {

if (oc1 == oc2 == 0000) {

//declare completely inside

output (x1, y1), (x2, y2) as clipped line

break

}

else if ((oc1 AND oc2) != 0000) { // condition to check matching bit

//declare completely outside and clip

break

}

else{

if(oc1 != 0000){

(x1, y1) = find intersection point of line

and the boundary corresponding

to non-zero bit of oc1

oc1 = calculate\_outcode(x1, y1)

}

else{

(x2, y2) = find intersection point of line

and the boundary corresponding

to non-zero bit of oc2

oc2 = calculate\_outcode(x2, y2)

}

continue

}

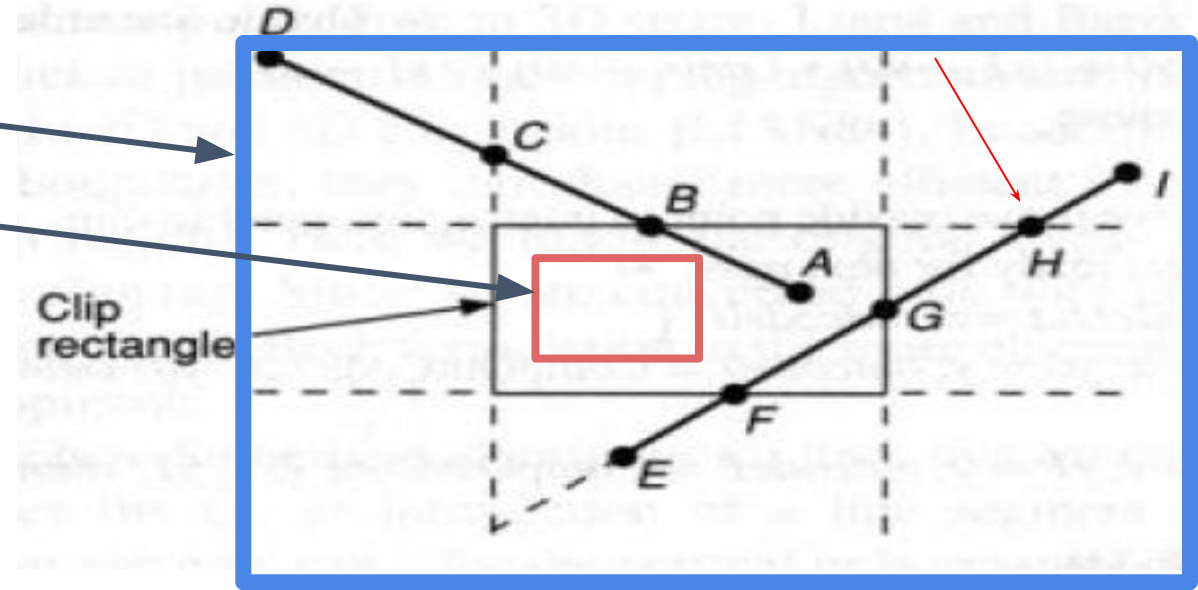
# Cohen-Sutherland Line Clipping Algorithm

Works well for two cases:

1. Very large clip region
2. Very small clip region

- Why?

[For many trivial accept and many trivial reject]



## Where is the problem?

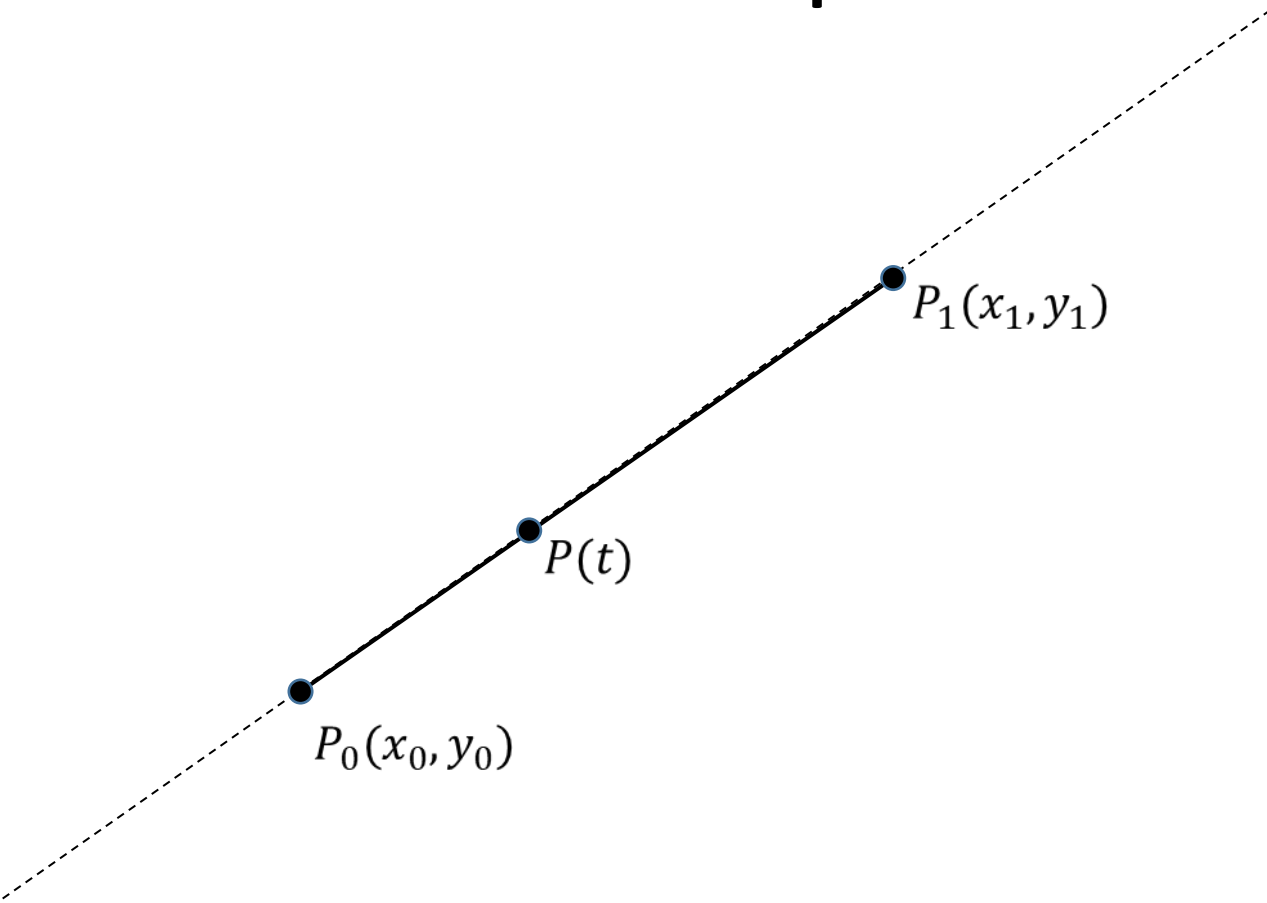
- Only rectangular clipping region
- Unnecessary clipping is done
- Different clipping order may take less iterations to finish

# Clipping

Cyrus-Beck Algo



# Parametric equation of Line



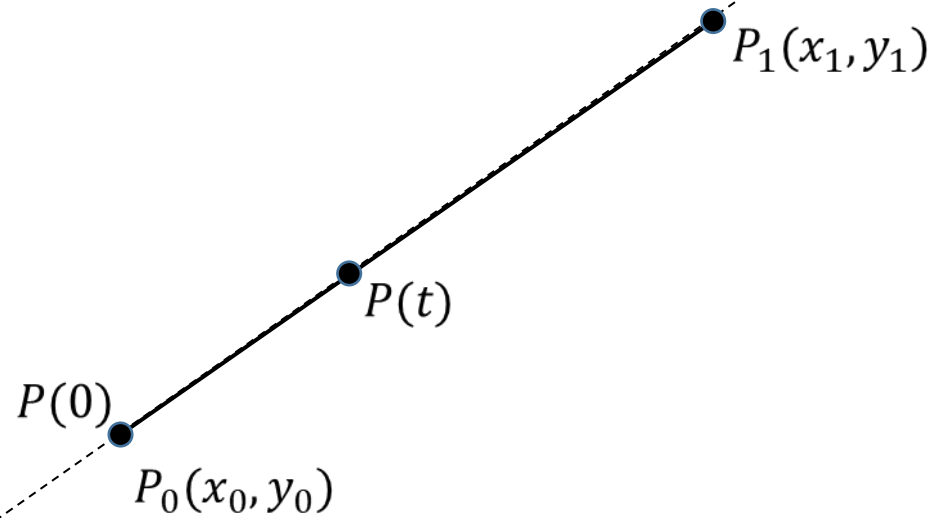
Parametric equation,

$$P(t) = P_0 + t \cdot (P_1 - P_0)$$

$$= (x_0, y_0) + t(x_1 - x_0, y_1 - y_0)$$

$$= (x_0 + t(x_1 - x_0), y_0 + t(y_1 - y_0))$$

# Parametric equation of Line

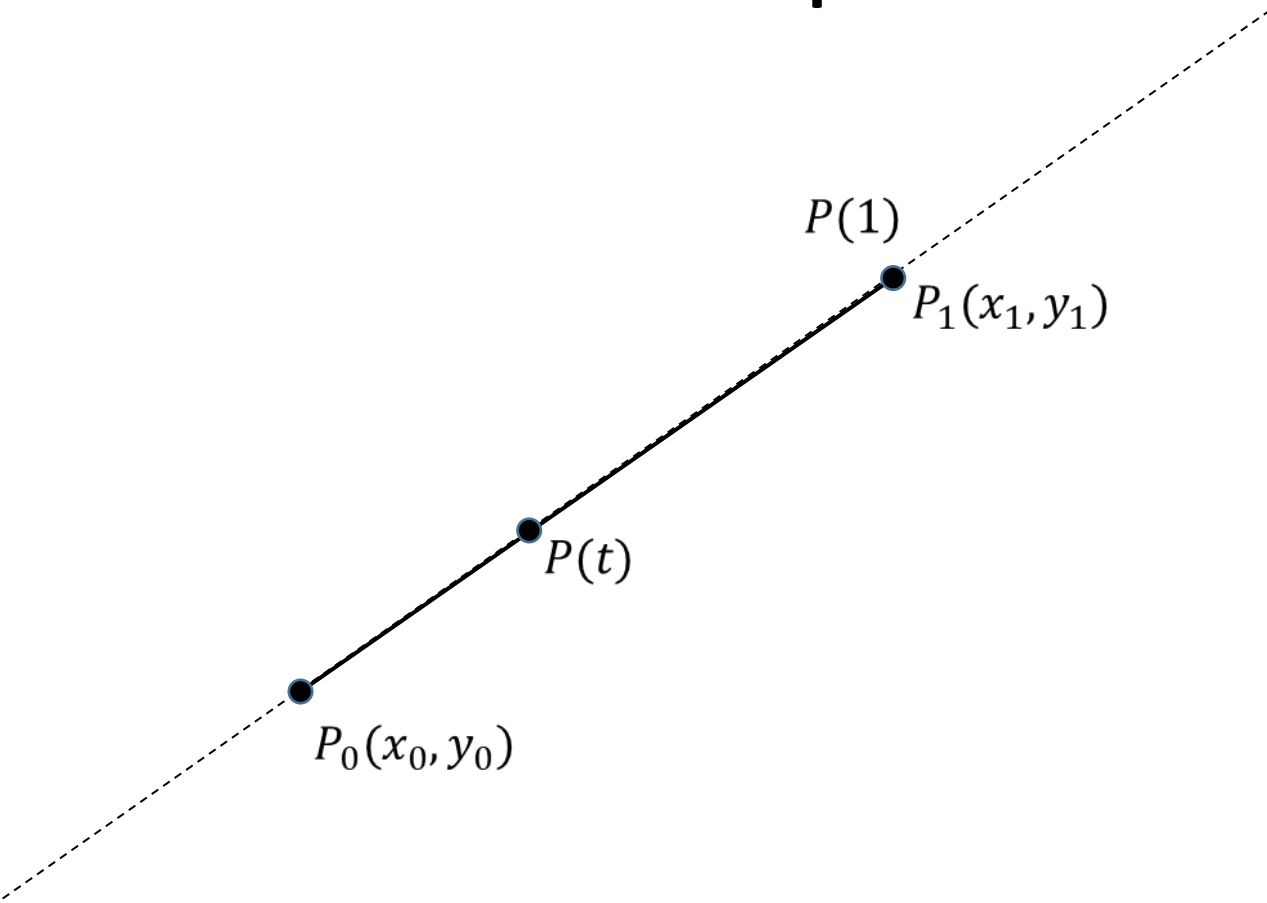


Parametric equation,

$$\begin{aligned} P(t) &= P_0 + t \cdot (P_1 - P_0) \\ &= (x_0, y_0) + t(x_1 - x_0, y_1 - y_0) \\ &= (x_0 + t(x_1 - x_0), y_0 + t(y_1 - y_0)) \end{aligned}$$

$$\begin{aligned} P(0) &= (x_0, y_0) + 0 \cdot (x_1 - x_0, y_1 - y_0) \\ &= (x_0, y_0) \end{aligned}$$

# Parametric equation of Line

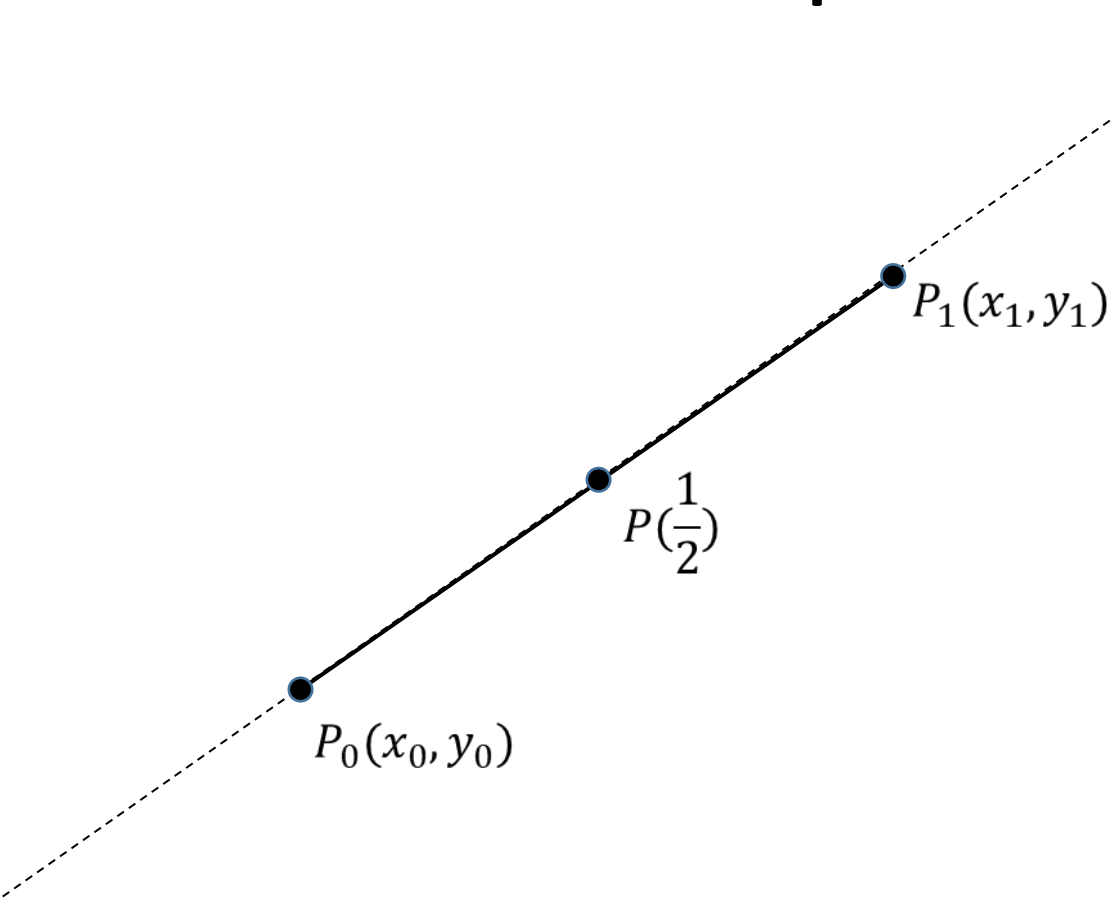


Parametric equation,

$$\begin{aligned} P(t) &= P_0 + t \cdot (P_1 - P_0) \\ &= (x_0, y_0) + t(x_1 - x_0, y_1 - y_0) \\ &= (x_0 + t(x_1 - x_0), y_0 + t(y_1 - y_0)) \end{aligned}$$

$$\begin{aligned} P(1) &= (x_0, y_0) + 1 \cdot (x_1 - x_0, y_1 - y_0) \\ &= (x_0, y_0) + (x_1 - x_0, y_1 - y_0) \\ &= (x_0 + x_1 - x_0, y_0 + y_1 - y_0) \\ &= (x_1, y_1) \end{aligned}$$

# Parametric equation of Line

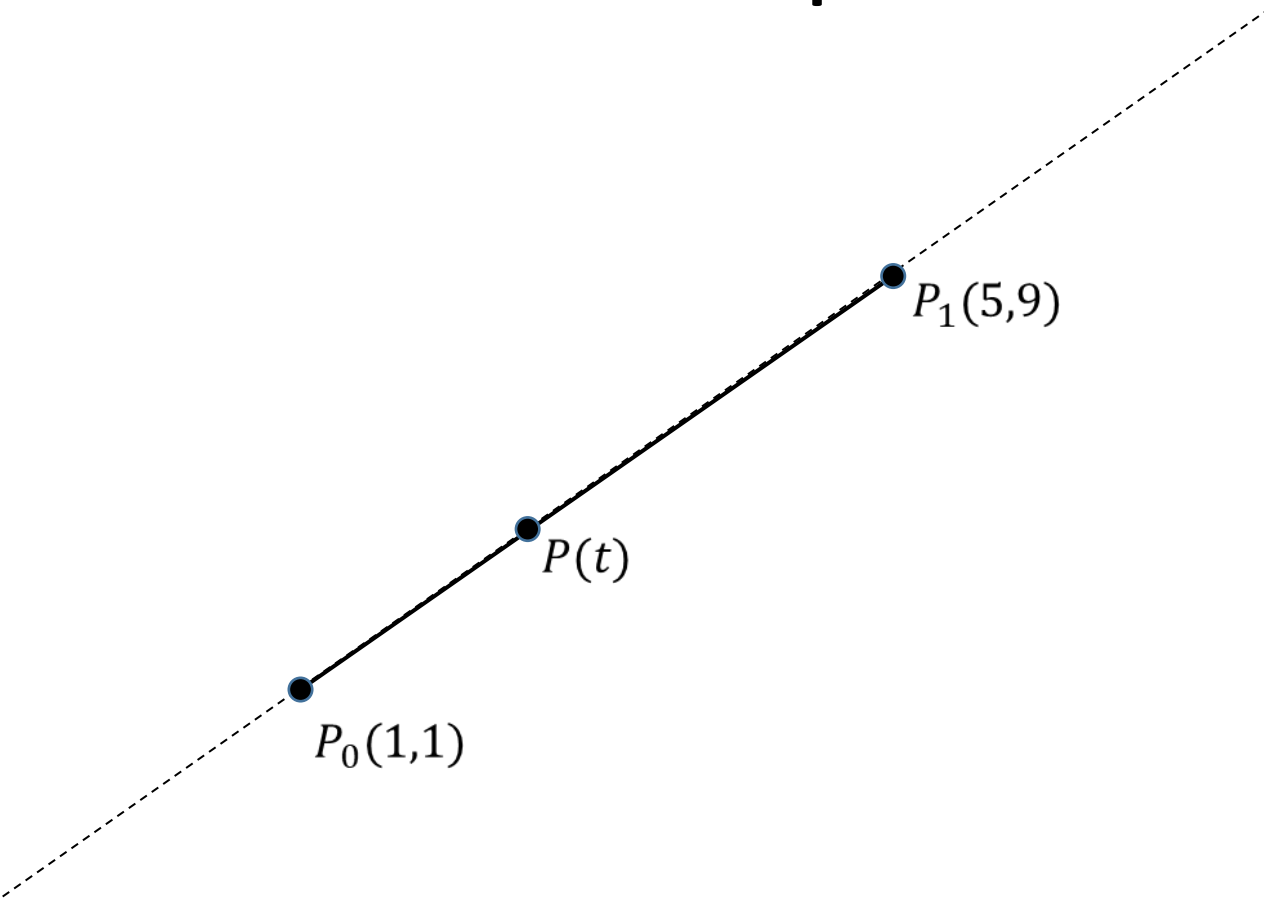


Parametric equation,

$$\begin{aligned} P(t) &= P_0 + t \cdot (P_1 - P_0) \\ &= (x_0, y_0) + t(x_1 - x_0, y_1 - y_0) \\ &= (x_0 + t(x_1 - x_0), y_0 + t(y_1 - y_0)) \end{aligned}$$

$$\begin{aligned} P\left(\frac{1}{2}\right) &= (x_0, y_0) + \frac{1}{2} \cdot (x_1 - x_0, y_1 - y_0) \\ &= \left(\frac{x_0 + x_1}{2}, \frac{y_0 + y_1}{2}\right) \end{aligned}$$

# Parametric equation of Line



Suppose, endpoints of a line are  $(1, 1)$  and  $(5, 9)$

Parametric equation,

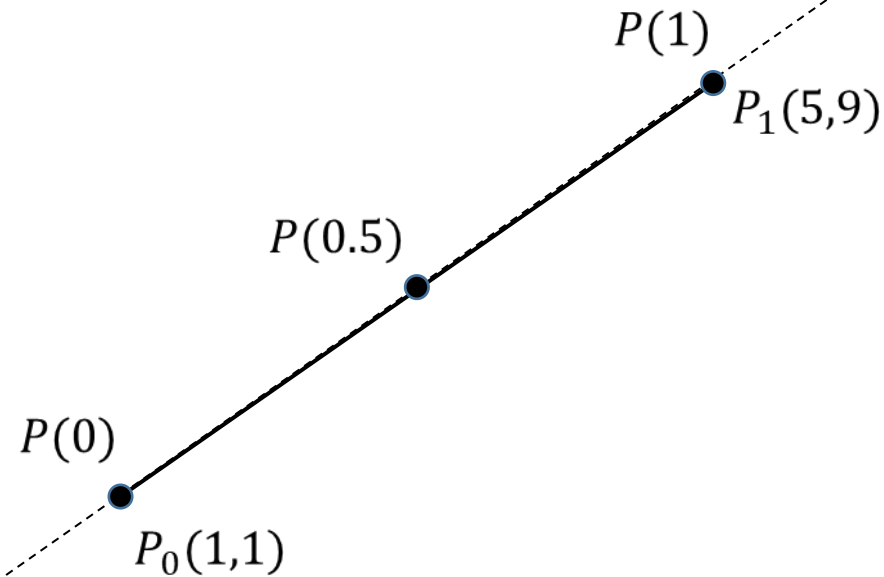
$$\begin{aligned} P(t) &= P_0 + t \cdot (P_1 - P_0) \\ &= (x_0, y_0) + t(x_1 - x_0, y_1 - y_0) \\ &= (1, 1) + t(5 - 1, 9 - 1) \\ &= (1, 1) + t(4, 8) \\ &= (1 + 4t, 1 + 8t) \end{aligned}$$

$$P(0) = (1, 1)$$

$$P(1) = (5, 9)$$



# Parametric equation of Line



Suppose, endpoints of a line are  $(1, 1)$  and  $(5, 9)$

Parametric equation,

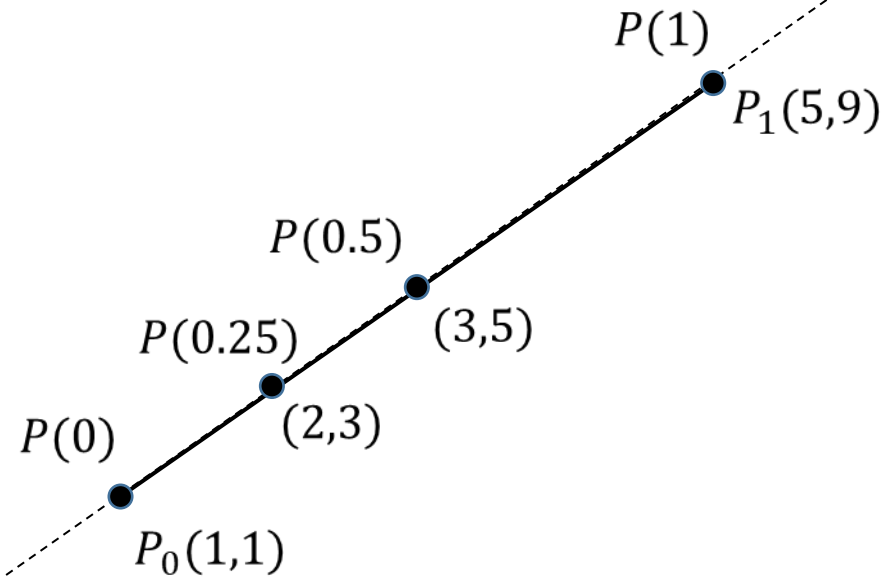
$$\begin{aligned}
 P(t) &= P_0 + t \cdot (P_1 - P_0) \\
 &= (x_0, y_0) + t(x_1 - x_0, y_1 - y_0) \\
 &= (1, 1) + t(5 - 1, 9 - 1) \\
 &= (1, 1) + t(4, 8) \\
 &= (1 + 4t, 1 + 8t)
 \end{aligned}$$

$$P(0) = (1, 1)$$

$$P(1) = (5, 9)$$

$$P(0.5) = (1 + 4 \times 0.5, 1 + 8 \times 0.5) = (3, 5)$$

# Parametric equation of Line



Suppose, endpoints of a line are  $(1, 1)$  and  $(5, 9)$

Parametric equation,

$$\begin{aligned}
 P(t) &= P_0 + t \cdot (P_1 - P_0) \\
 &= (x_0, y_0) + t(x_1 - x_0, y_1 - y_0) \\
 &= (1, 1) + t(5 - 1, 9 - 1) \\
 &= (1, 1) + t(4, 8) \\
 &= (1 + 4t, 1 + 8t)
 \end{aligned}$$

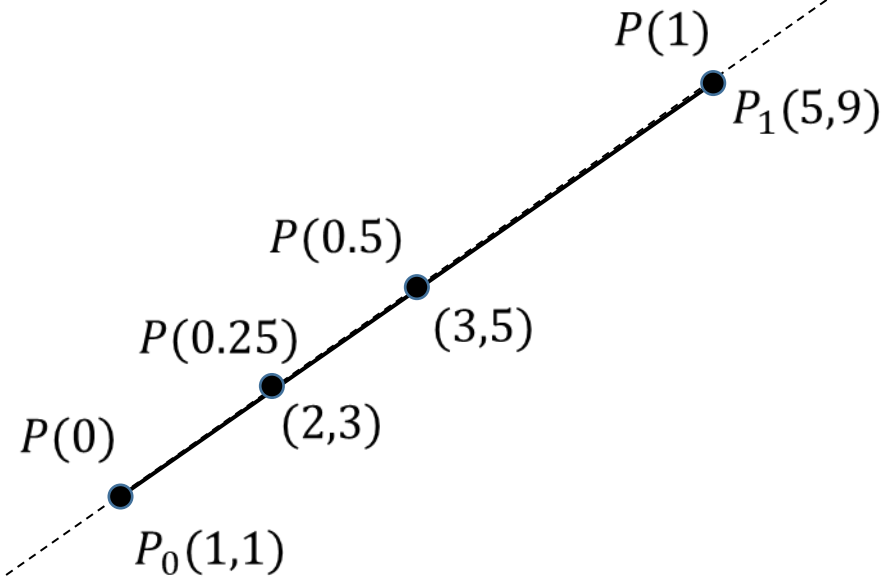
$$P(0) = (1, 1)$$

$$P(1) = (5, 9)$$

$$P(0.5) = (1 + 4 \times 0.5, 1 + 8 \times 0.5) = (3, 5)$$

$$P(0.25) = (1 + 4 \times 0.25, 1 + 8 \times 0.25) = (2, 3)$$

# Parametric equation of Line



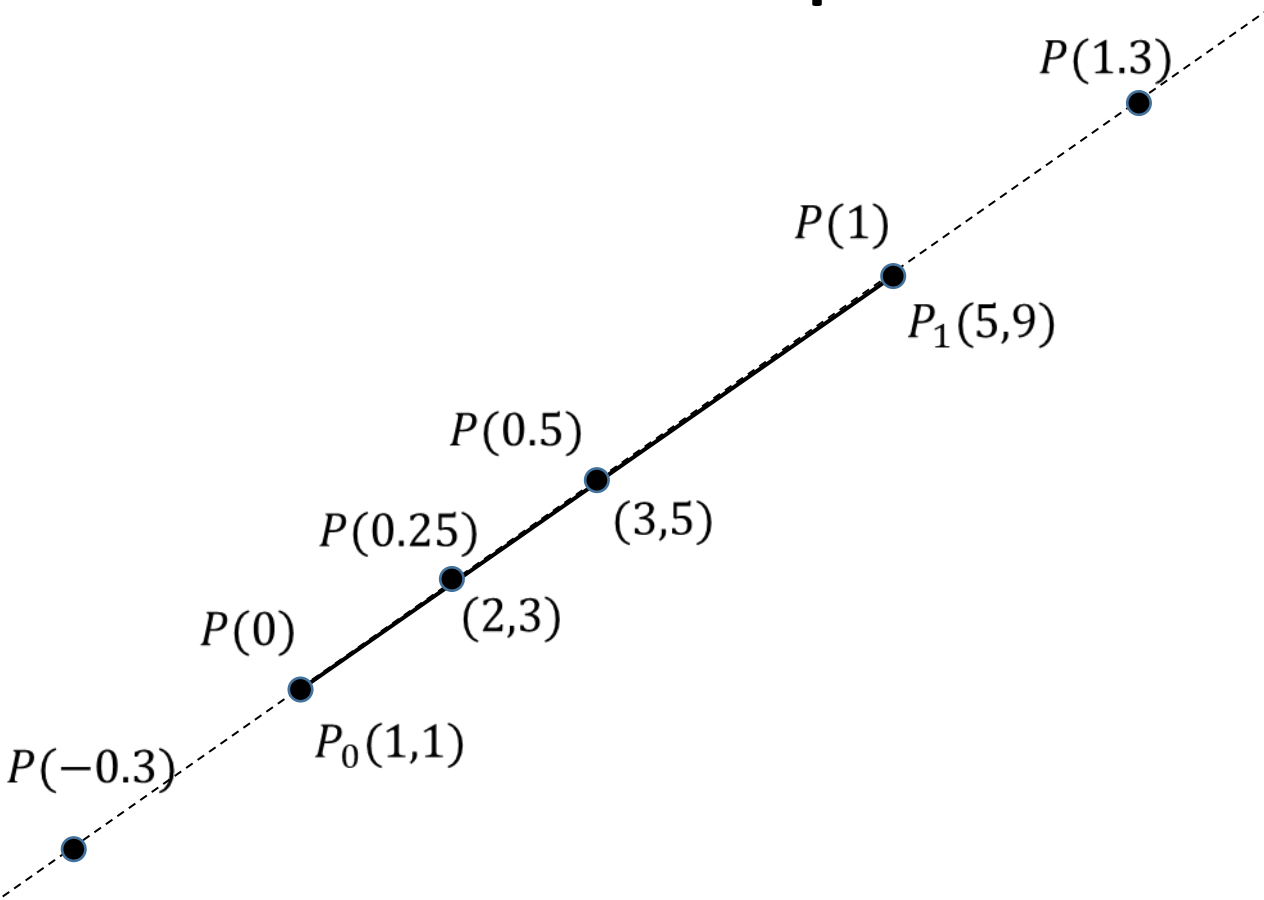
Suppose, endpoints of a line are  $(1, 1)$  and  $(5, 9)$

Parametric equation,

$$\begin{aligned}
 P(t) &= P_0 + t \cdot (P_1 - P_0) \\
 &= (x_0, y_0) + t(x_1 - x_0, y_1 - y_0) \\
 &= (1, 1) + t(5 - 1, 9 - 1) \\
 &= (1, 1) + t(4, 8) \\
 &= (1 + 4t, 1 + 8t)
 \end{aligned}$$

So all points between  $0 \leq t \leq 1$  are inside the line segment

# Parametric equation of Line

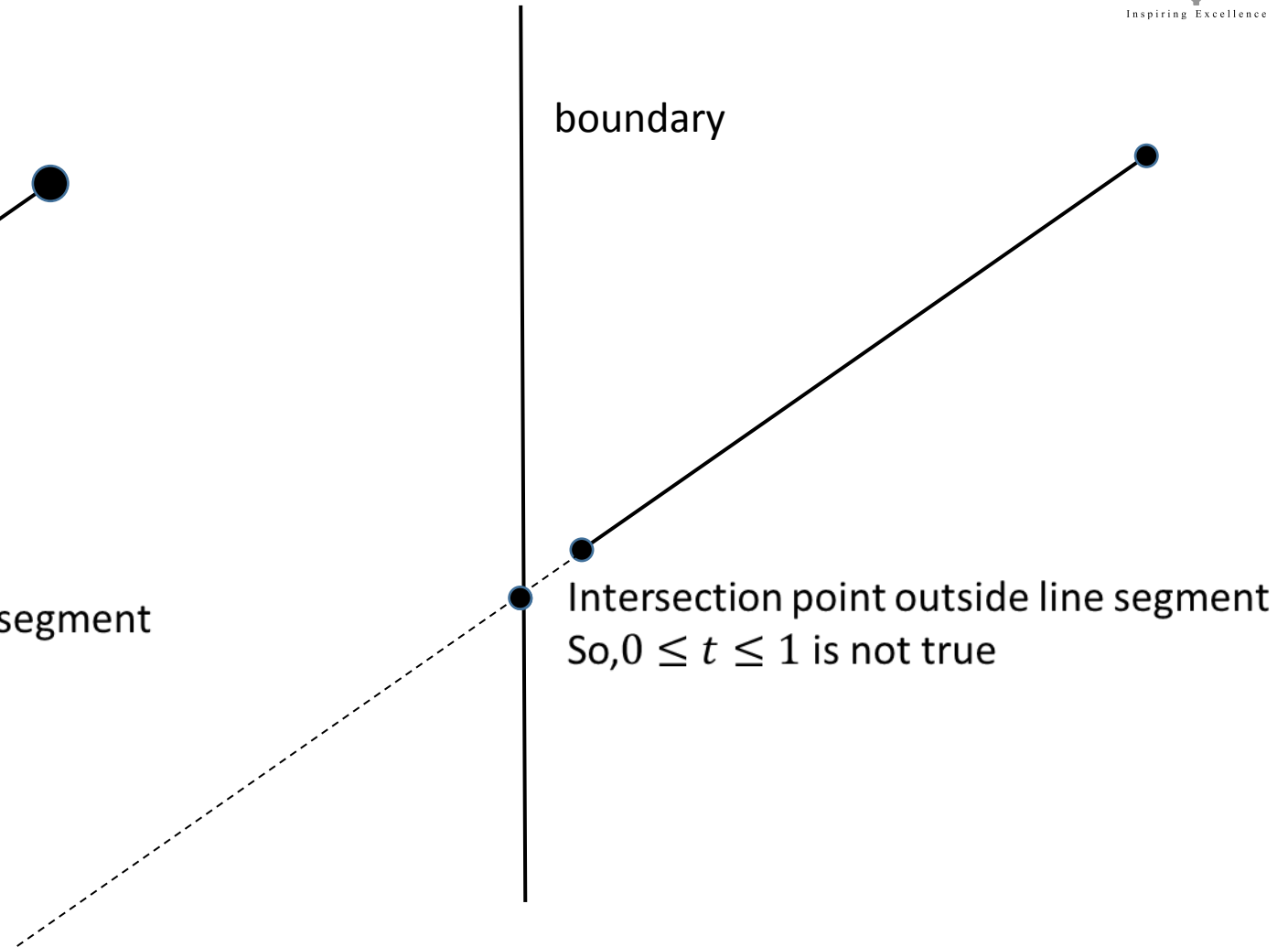
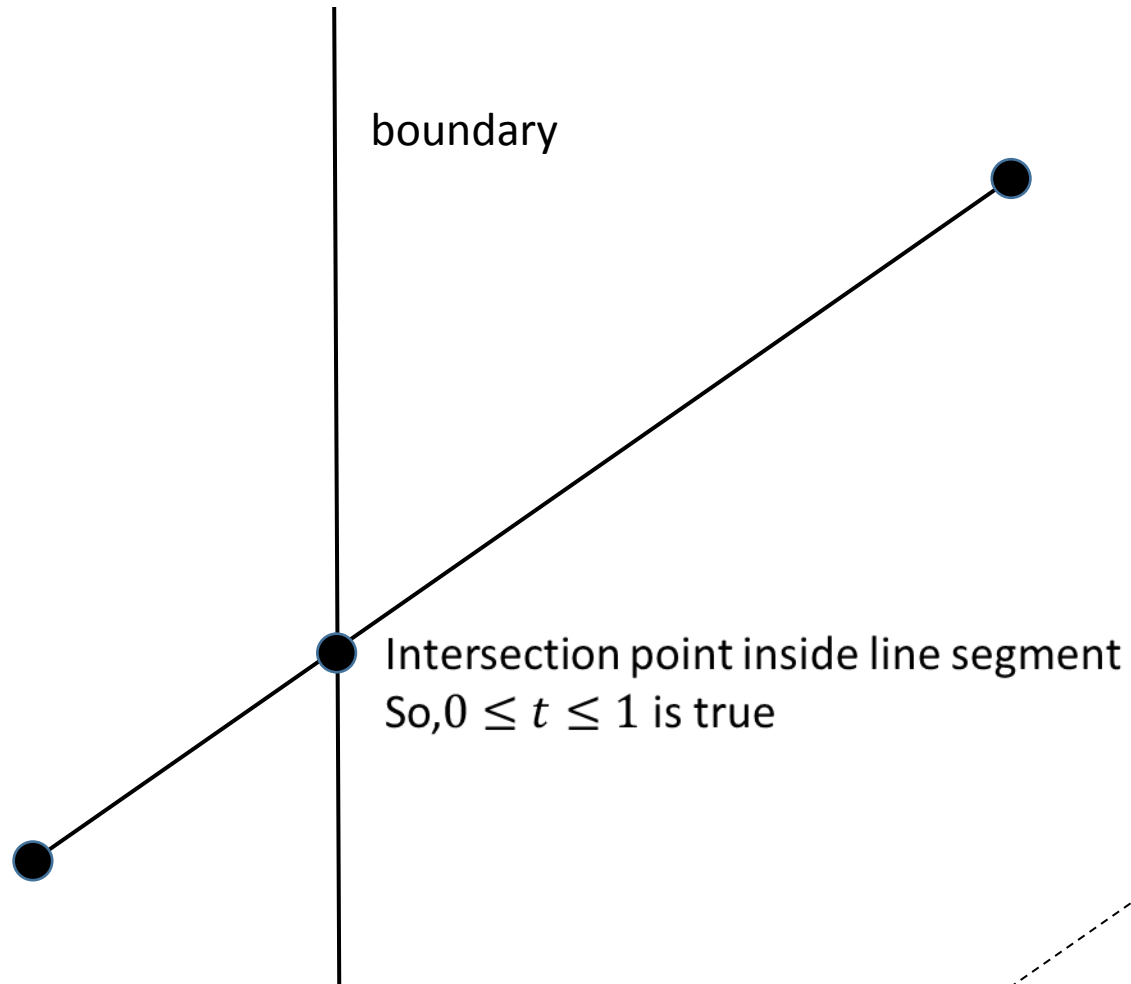


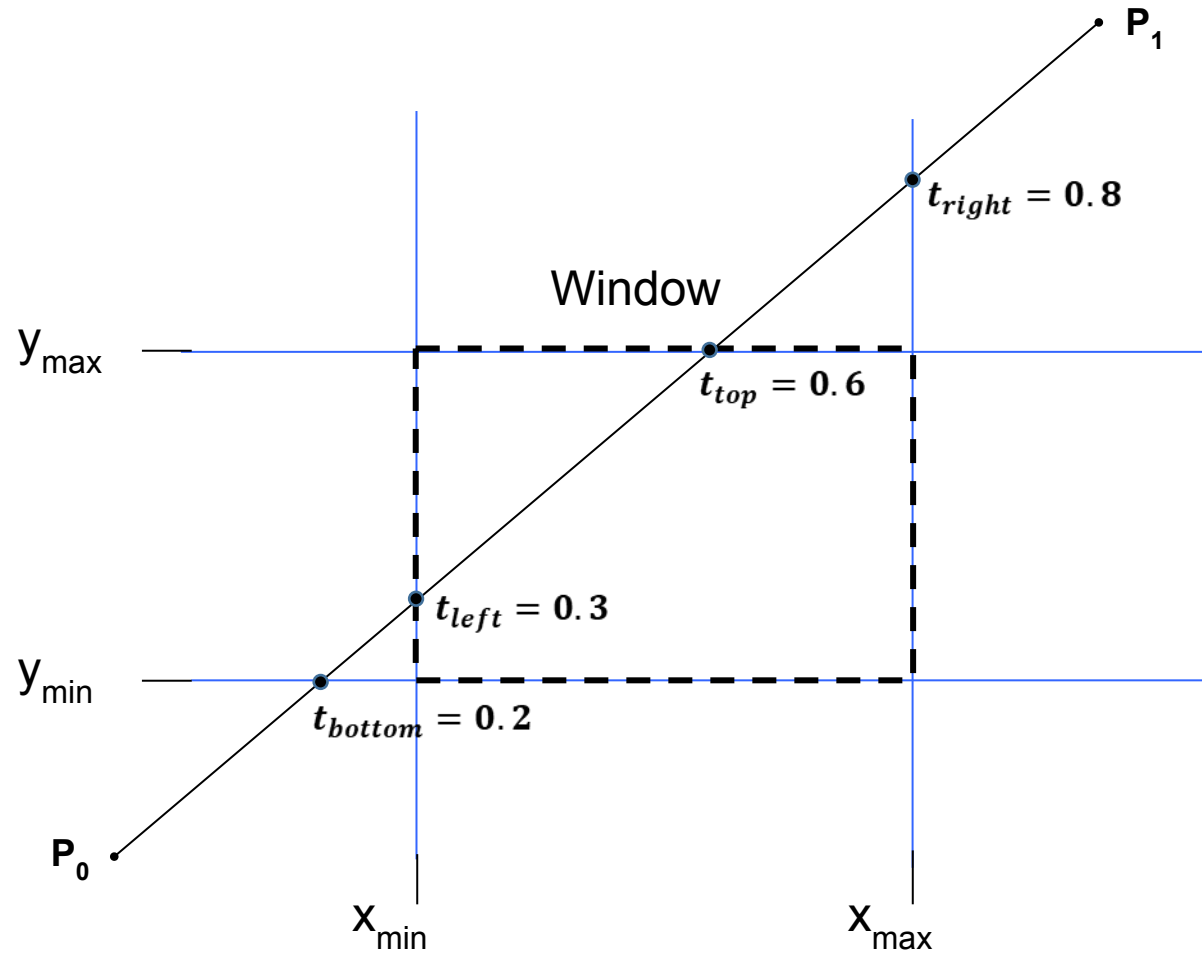
Suppose, endpoints of a line are (1, 1) and (5, 9)

Parametric equation,

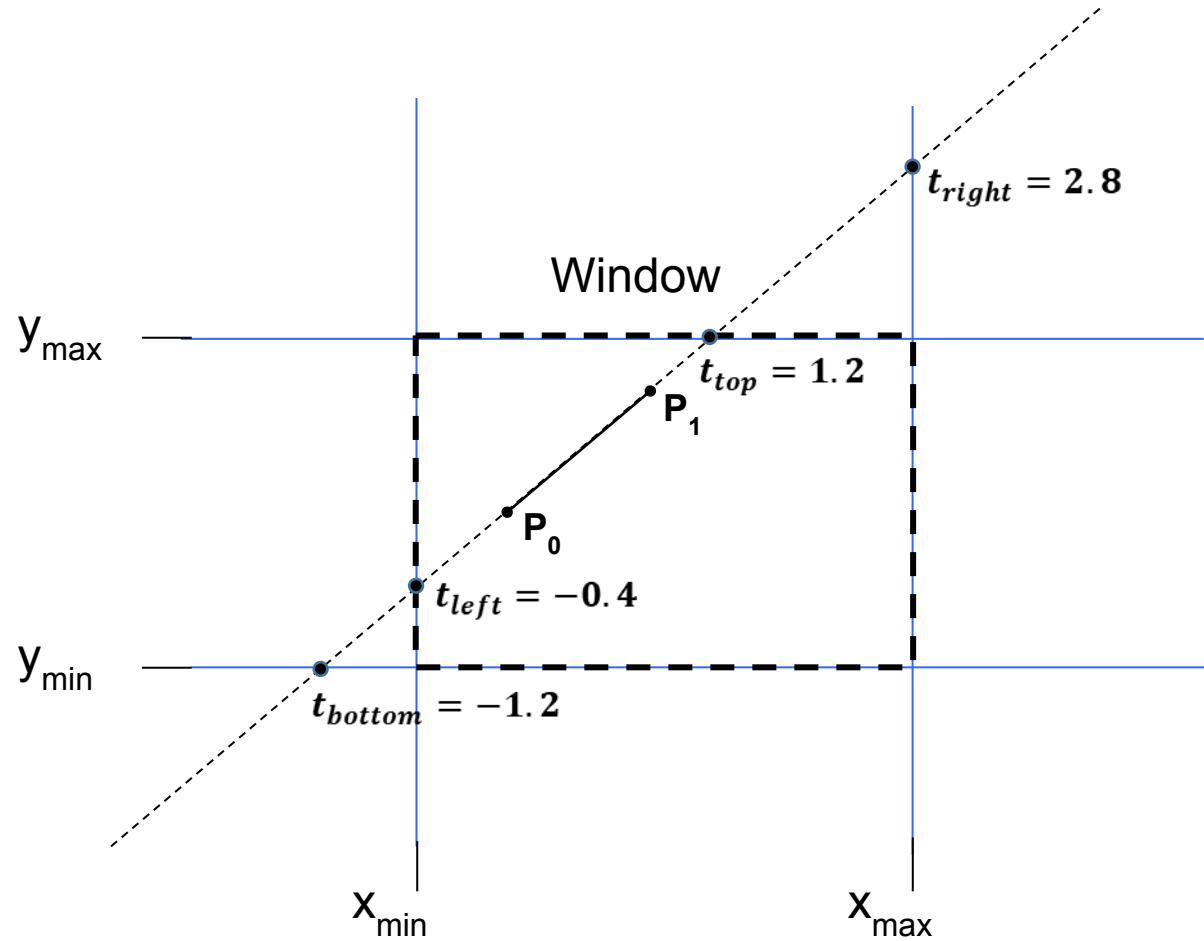
$$\begin{aligned}
 P(t) &= P_0 + t \cdot (P_1 - P_0) \\
 &= (x_0, y_0) + t(x_1 - x_0, y_1 - y_0) \\
 &= (1, 1) + t(5 - 1, 9 - 1) \\
 &= (1, 1) + t(4, 8) \\
 &= (1 + 4t, 1 + 8t)
 \end{aligned}$$

$P(-0.3)$  and  $P(1.3)$  are outside the line segment



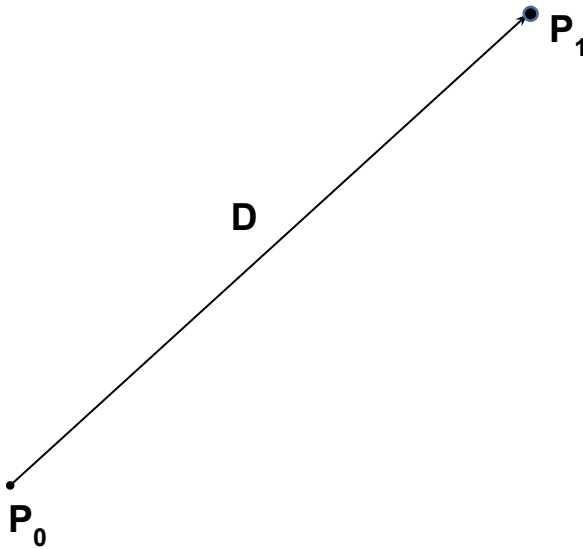


Both endpoints outside  
All intersections have  $t$  between  
0 and 1



Both endpoints inside  
All intersections have  $t$  outside 0  
to 1

# Vector

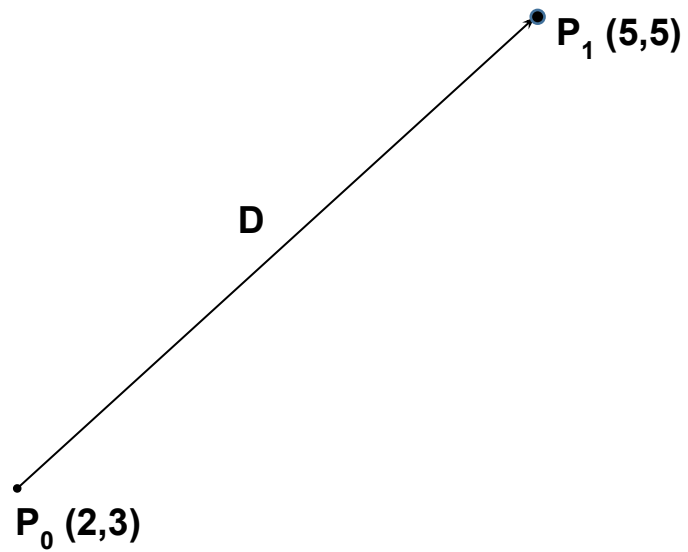


$D$  is a vector from  $P_0$  to  $P_1$

$$D = P_1 - P_0$$
$$= (x_1 - x_0, y_1 - y_0)$$

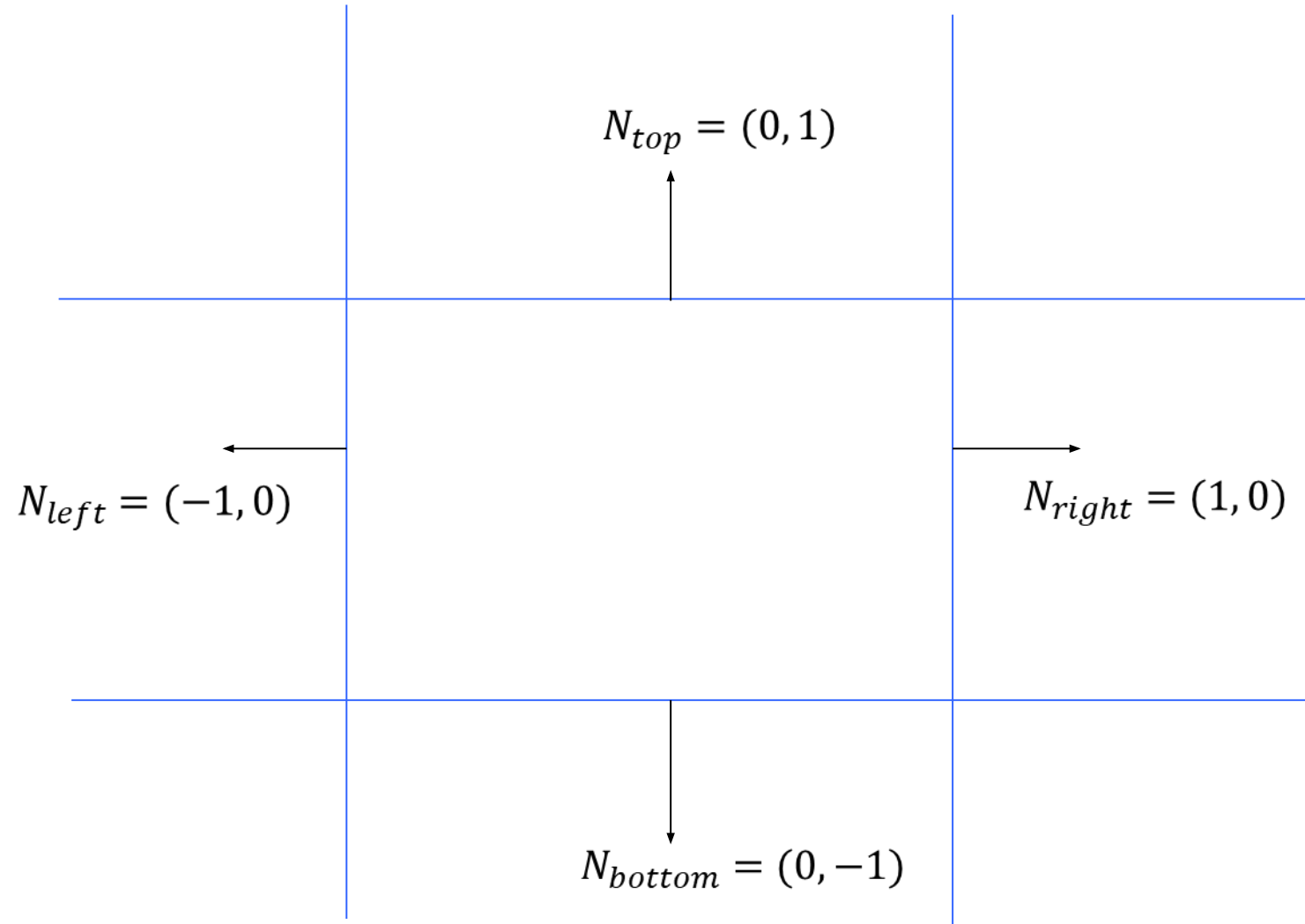


# Vector



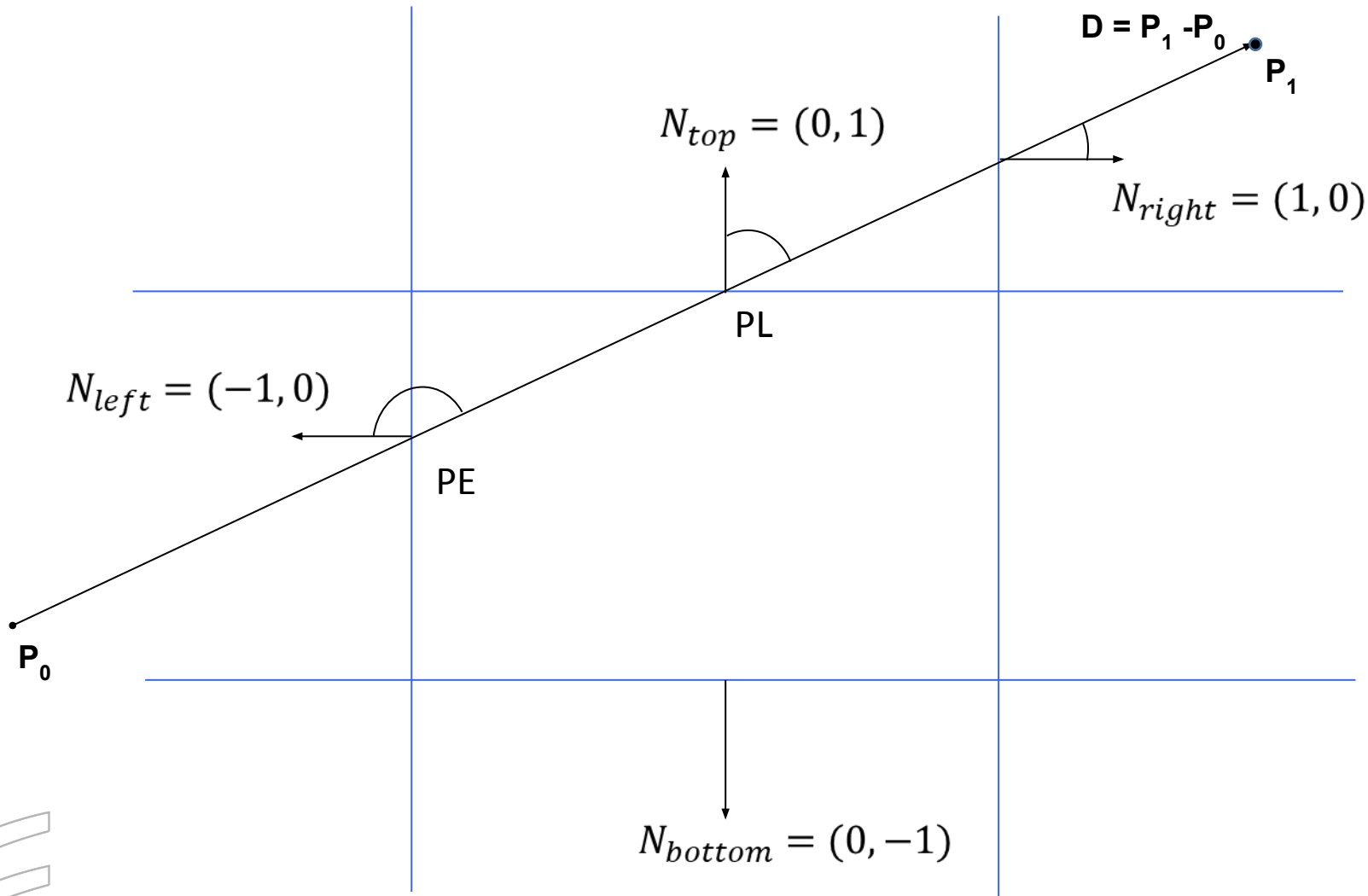
$$\begin{aligned} D &= P_1 - P_0 \\ &= (x_1 - x_0, y_1 - y_0) \\ &= (5 - 2, 5 - 3) \\ &= (3, 2) \\ &\cong 3i + 2j \end{aligned}$$

# Normal vectors to boundary



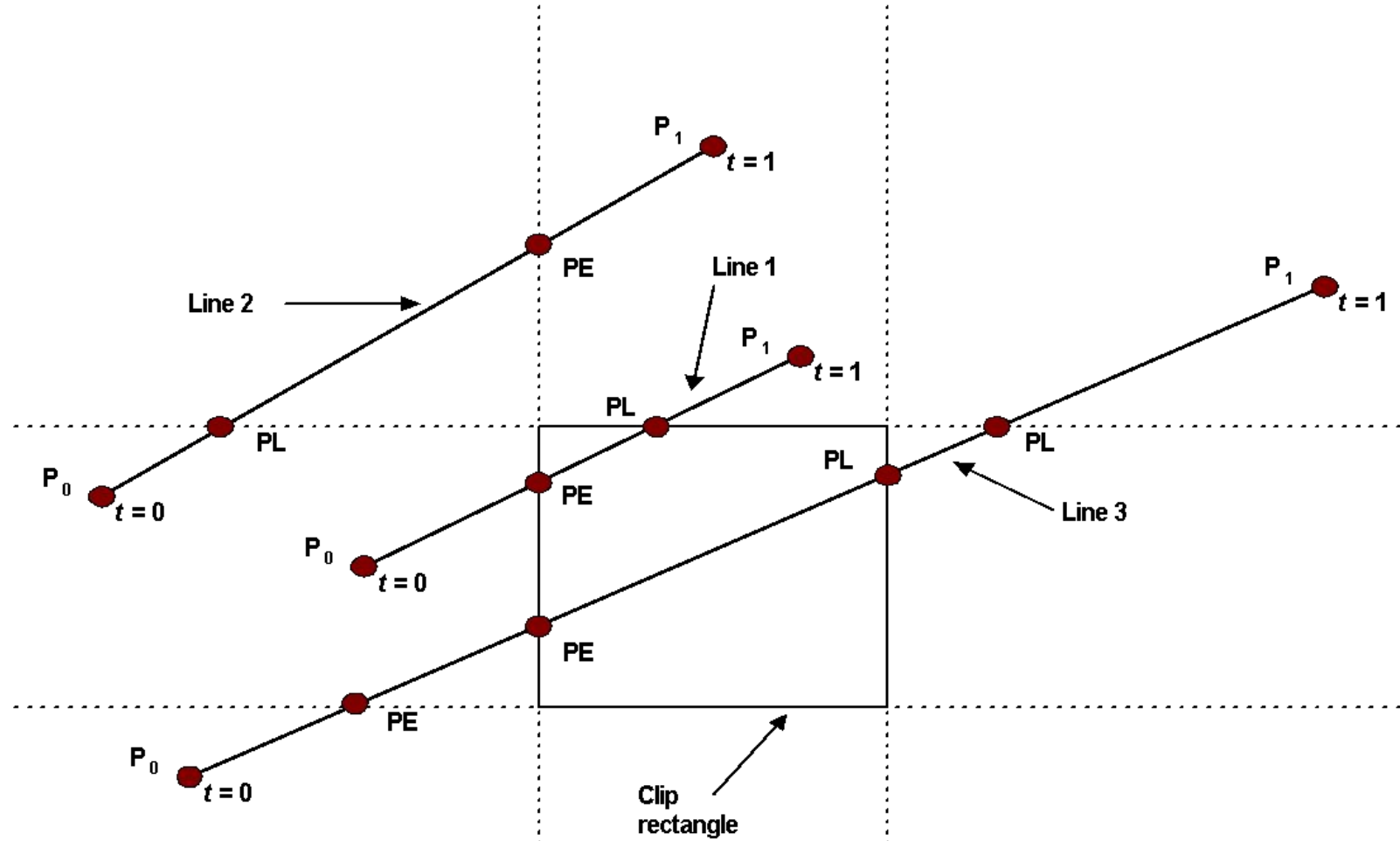
Each  $N_i$  is a perpendicular unit vector to each of the boundaries: left, right, top, bottom.  $N_i$  points away from the clip window

# Normal vectors to boundary



When angle between  $N_i$  and  $D$  is more than 90 degree-  
Or,  $N_i \cdot D < 0$   
The intersection point is PE  
(Potentially entering)

When angle between  $N_i$  and  $D$  is less than 90 degree-  
Or,  $N_i \cdot D > 0$   
The intersection point is PL  
(Potentially leaving)



PE = Potentially Entering

$$N_i \cdot D < 0 \Rightarrow PE \\ \Rightarrow Angle > 90^\circ$$

PL = Potentially Leaving

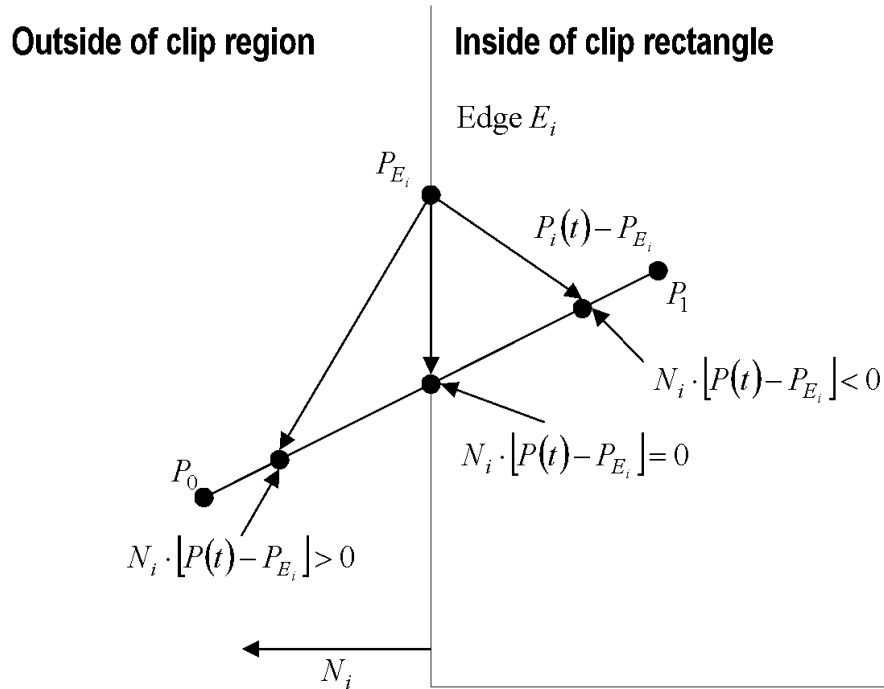
$$N_i \cdot D > 0 \Rightarrow PL \\ \Rightarrow Angle < 90^\circ$$

# Cyrus-Beck algo

- Calculate  $t$  values of intersection points with each 4 boundaries
- Classify intersection points whether PE/PL
- Select the PE with highest  $t$  and the PL with the lowest  $t$
- Using parametric line eqn. find the clipped points



# The Cyrus-Beck Algorithm



$$\text{Line } P_0P_1 : P(t) = P_0 + (P_1 - P_0)t$$

$$N_i \cdot [P(t) - P_{E_i}] = 0$$

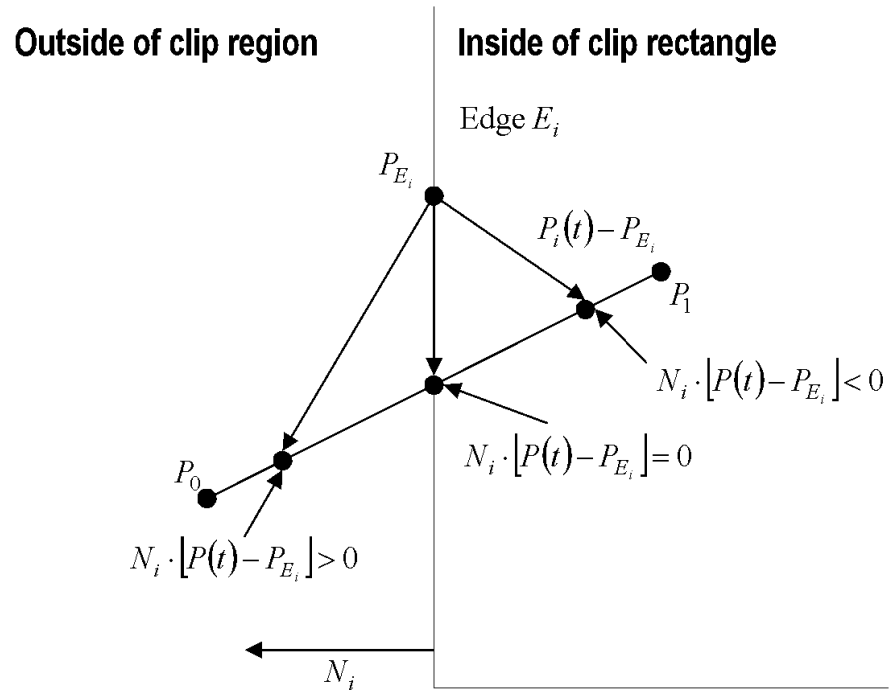
$$\Rightarrow N_i \cdot [P_0 + (P_1 - P_0)t - P_{E_i}] = 0$$

$$\Rightarrow N_i \cdot [P_0 + (P_1 - P_0)t - P_{E_i}] = 0$$

$$\Rightarrow t = \frac{N_i \cdot [P_0 - P_{E_i}]}{-N_i \cdot (P_1 - P_0)}$$

$$\Rightarrow t = \frac{N_i \cdot [P_0 - P_{E_i}]}{-N_i \cdot D}, \quad D = P_1 - P_0$$

# The Cyrus-Beck Algorithm



For left boundary,

$$\begin{aligned}
 t_{left} &= \frac{N_i \cdot [P_0 - P_E]}{-N_i \cdot [P_1 - P_0]} \\
 &= \frac{(-1, 0) \cdot [(x_0, y_0) - (x_{min}, y)]}{-(-1, 0) \cdot [(x_1, y_1) - (x_0, y_0)]} \\
 &= \frac{(-1, 0) \cdot (x_0 - x_{min}, y_0 - y)}{-(-1, 0) \cdot (x_1 - x_0, y_1 - y_0)} \\
 &= \frac{-1 \times (x_0 - x_{min}) + 0 \times (y_0 - y)}{-(-1 \times (x_1 - x_0) + 0 \times (y_1 - y_0))} \\
 &= \frac{-(x_0 - x_{min})}{(x_1 - x_0)}
 \end{aligned}$$

Similarly,

$$\begin{aligned}
 t_{right} &= \frac{-(x_0 - x_{max})}{(x_1 - x_0)} \\
 t_{top} &= \frac{-(y_0 - y_{max})}{(y_1 - y_0)} \\
 t_{bottom} &= \frac{-(y_0 - y_{min})}{(y_1 - y_0)}
 \end{aligned}$$

# The Cyrus-Beck Algorithm

```

Precalculate  $N_i$  and  $P_{Ei}$  for each edge
for (each line segment to be clipped) {
    if ( $P_1 == P_0$ )
        line is degenerated, so clip as a point;
    else {
         $t_E = 0$ ;  $t_L = 1$ ;
        for (each candidate intersection with a clip edge) {
            if ( $N_i \bullet D \neq 0$ ) { /* Ignore edges parallel to line */
                calculate  $t$ ;
                use sign of  $N_i \bullet D$  to categorize as PE or PL;
                if (PE)  $t_E = \max(t_E, t)$ ;
                if (PL)  $t_L = \min(t_L, t)$ ;
            }
        }
        if ( $t_E > t_L$ ) return NULL;
        else return  $P(t_E)$  and  $P(t_L)$  as true clip intersection;
    }
}

```



a) Calculate the value of  $t$  of the lines given below for all edges and specify whether they are entering or leaving  $t$ . [Given  $(0,0)$  to  $(300,200)$  be the clip region.]

- (i)  $(50, -125)$  to  $(-100, 225)$ .
- (ii)  $(-250, 200)$  to  $(250, -200)$ .
- (iii)  $(-250, 200)$  to  $(150, 100)$

Also, find the line segment within the clipping window



a)(i) boundary:

$$x_{min} = 0, x_{max} = 300, y_{min} = 0, y_{max} = 200$$

points:

$$x_0 = 50, y_0 = -125, x_1 = -100, y_1 = 225$$

$$D = (x_1 - x_0, y_1 - y_0) = (-150, 350)$$

Initially,  $t_E = 0, t_L = 1$

Boundary	$N_i$	$N_i \cdot D$	t	PE/PL	$t_E$	$t_L$
Left	(-1,0)	150	$\frac{-(50 - 0)}{-100 - 50} = 0.33$	PL	0	0.33
Right	(1,0)	-150	$\frac{-(50 - 300)}{-100 - 50} = -1.67$	PE	0	0.33
Bottom	(0, -1)	-350	$\frac{-(-125 - 0)}{225 - (-125)} = 0.357$	PE	0.357	0.33
Top	(0, 1)	350	$\frac{-(-125 - 200)}{225 - (-125)} = 0.93$	PL	0.357	0.33

Since  $t_E > t_L$ , line segment is outside clip window

$$t_{left} = \frac{-(x_0 - x_{min})}{(x_1 - x_0)}$$

$$t_{right} = \frac{-(x_0 - x_{max})}{(x_1 - x_0)}$$

$$t_{top} = \frac{-(y_0 - y_{max})}{(y_1 - y_0)}$$

$$t_{bottom} = \frac{-(y_0 - y_{min})}{(y_1 - y_0)}$$

### Algorithm

Precalculate  $N_i$  and  $P_{Ei}$  for each edge

for (each line segment to be clipped) {

if ( $P_1 == P_0$ )

line is degenerated, so clip as a point;

else {

$t_E = 0; t_L = 1;$

for (each candidate intersection with a clip edge) {

if ( $N_i \cdot D \neq 0$ ) { /\* Ignore edges parallel to line \*/  
calculate t;

use sign of  $N_i \cdot D$  to categorize as PE or PL;

if (PE)  $t_E = \max(t_E, t);$

if (PL)  $t_L = \min(t_L, t);$

}

}

if ( $t_E > t_L$ ) return NULL;

else return  $P(t_E)$  and  $P(t_L)$  as true clip intersection;

}

a)(ii) boundary:

$$x_{min} = 0, x_{max} = 300, y_{min} = 0, y_{max} = 200$$

points:

$$x_0 = -250, y_0 = 200, x_1 = 250, y_1 = -200$$

$$D = (x_1 - x_0, y_1 - y_0) = (500, -400)$$

$$\text{Initially, } t_E = 0, t_L = 1$$

Boundary	$N_i$	$N_i \cdot D$	t	PE/PL	$t_E$	$t_L$
Left	(-1,0)	-500	$\frac{-(-250 - 0)}{500}$ = 0.5	PE	0.5	1
Right	(1,0)	500	$\frac{-(-250 - 300)}{500}$ = 1.1	PL	0.5	1
Bottom	(0, -1)	400	$\frac{-(200 - 0)}{-400}$ = 0.5	PL	0.5	0.5
Top	(0, 1)	-400	$\frac{-(200 - 200)}{-400}$ = 0	PE	0.5	0.5

P(0.5) and P(0.5) (same point) are the true clip intersection

$$\begin{aligned}
 P(0.5) &= (x_0, y_0) + 0.5 \times D \\
 &= (-250, 200) + 0.5 \times (500, -400) \\
 &= (0, 0)
 \end{aligned}$$

$$\begin{aligned}
 t_{left} &= \frac{-(x_0 - x_{min})}{(x_1 - x_0)} \\
 t_{right} &= \frac{-(x_0 - x_{max})}{(x_1 - x_0)} \\
 t_{top} &= \frac{-(y_0 - y_{max})}{(y_1 - y_0)} \\
 t_{bottom} &= \frac{-(y_0 - y_{min})}{(y_1 - y_0)}
 \end{aligned}$$

### Algorithm

Precalculate  $N_i$  and  $P_{Ei}$  for each edge

for (each line segment to be clipped) {

if ( $P_1 == P_0$ )

line is degenerated, so clip as a point;

else {

$t_E = 0; t_L = 1;$

for (each candidate intersection with a clip edge) {

if ( $N_i \cdot D \neq 0$ ) { /\* Ignore edges parallel to line \*/

calculate t;

use sign of  $N_i \cdot D$  to categorize as PE or PL;

if (PE)  $t_E = \max(t_E, t);$

if (PL)  $t_L = \min(t_L, t);$

}

}

if ( $t_E > t_L$ ) return NULL;

else return  $P(t_E)$  and  $P(t_L)$  as true clip intersection;

}

}

a)(iii) boundary:

$$x_{min} = 0, x_{max} = 300, y_{min} = 0, y_{max} = 200$$

points:

$$x_0 = -250, y_0 = 200, x_1 = 150, y_1 = 100$$

$$D = (x_1 - x_0, y_1 - y_0) = (400, -100)$$

$$\text{Initially, } t_E = 0, t_L = 1$$

Boundary	$N_i$	$N_i \cdot D$	t	PE/PL	$t_E$	$t_L$
Left	(-1,0)	-400	$\frac{-(-250 - 0)}{400}$ $= 0.625$	PE	0.625	1
Right	(1,0)	400	$\frac{-(-250 - 300)}{400}$ $= 1.375$	PL	0.625	1
Bottom	(0, -1)	100	$\frac{-(200 - 0)}{-100}$ $= 2$	PL	0.625	1
Top	(0, 1)	-100	$\frac{-(200 - 200)}{-100}$ $= 0$	PE	0.625	1

P(0.625) and P(1) are the true clip intersection

$$\begin{aligned} P(0.625) &= (x_0, y_0) + 0.625 \times D \\ &= (-250, 200) + 0.625 \times (400, -100) \\ &= (0, 137.5) \end{aligned}$$

$$P(1) = (150, 100)$$

(0, 137.5) and (150, 100) are the endpoints of the clipped line

$$\begin{aligned} t_{left} &= \frac{-(x_0 - x_{min})}{(x_1 - x_0)} \\ t_{right} &= \frac{-(x_0 - x_{max})}{(x_1 - x_0)} \\ t_{top} &= \frac{-(y_0 - y_{max})}{(y_1 - y_0)} \\ t_{bottom} &= \frac{-(y_0 - y_{min})}{(y_1 - y_0)} \end{aligned}$$

### Algorithm

Precalculate  $N_i$  and  $P_{Ei}$  for each edge

for (each line segment to be clipped) {

if ( $P_1 == P_0$ )

line is degenerated, so clip as a point;

else {

$t_E = 0$ ;  $t_L = 1$ ;

for (each candidate intersection with a clip edge) {

if ( $N_i \cdot D \neq 0$ ) { /\* Ignore edges parallel to line \*/

calculate t;

use sign of  $N_i \cdot D$  to categorize as PE or PL;

if (PE)  $t_E = \max(t_E, t)$ ;

if (PL)  $t_L = \min(t_L, t)$ ;

}

}

if ( $t_E > t_L$ ) return NULL;

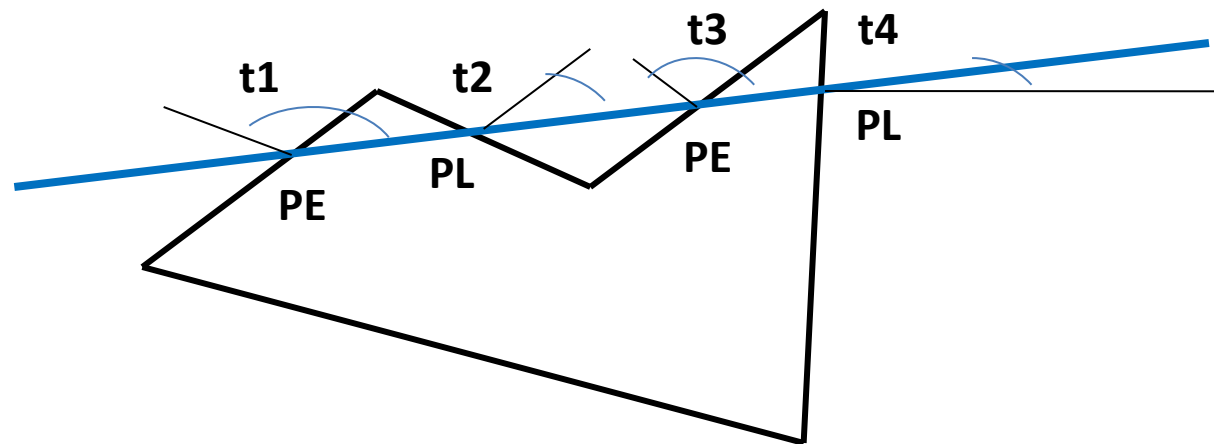
else return  $P(t_E)$  and  $P(t_L)$  as true clip intersection;

}

}

# Cyrus-Beck Parametric Line Clipping Algorithm

- Advantage
  - Works with polygons too (not only with clip rectangles)
  - Works in 3D scenario (polyhedrons)
- Problem
  - Does not work with **concave** polygon clip region



$$\begin{aligned} t_E &= t_3 \\ t_L &= t_2 \\ t_E &> t_L \end{aligned}$$

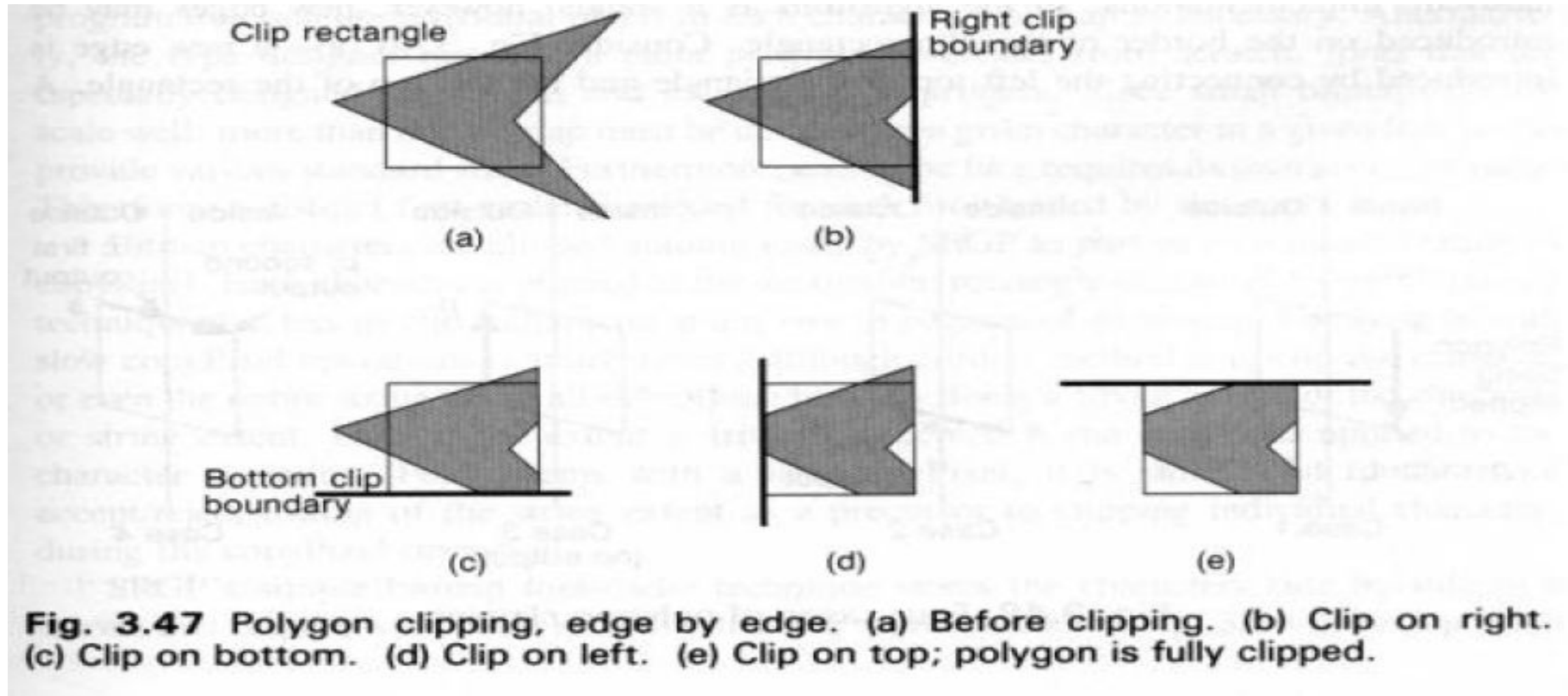
So the whole line is discarded though some segments should be displayed

# Clipping

Sutherland-Hodgman Polygon Clipping

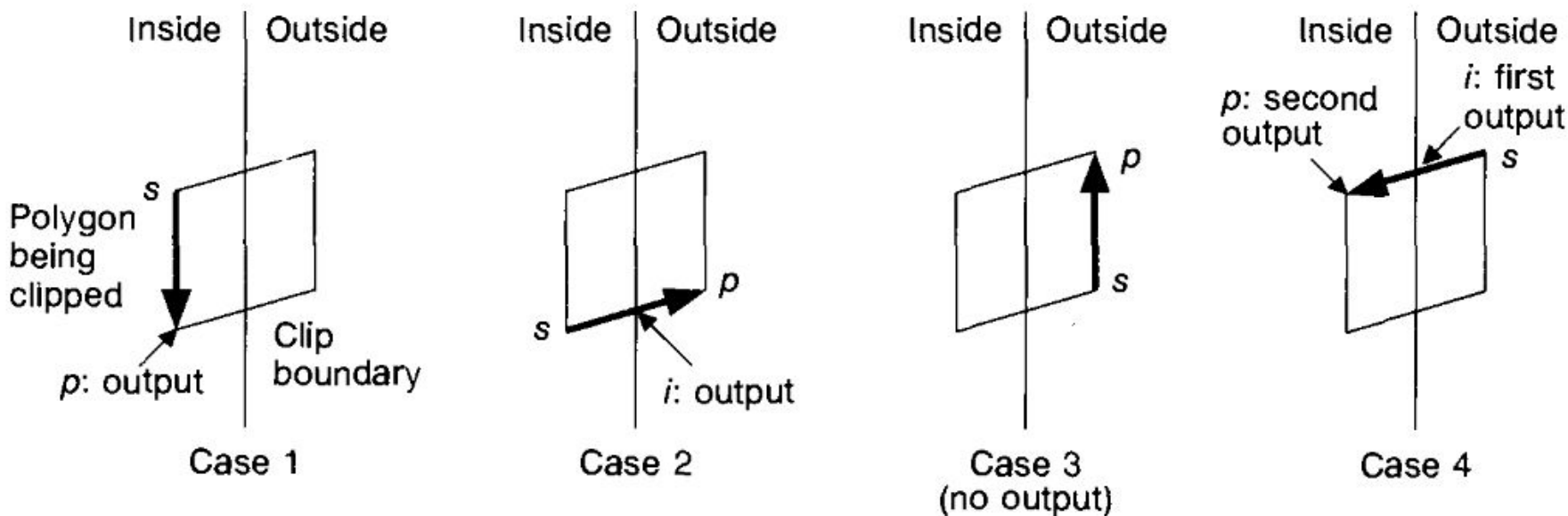


# Sutherland-Hodgman Polygon Clipping



# Sutherland-Hodgman Polygon Clipping

- Divide and Conquer Strategy
  - Clip against a single infinite clip edge and get new vertices
  - Repeat for next clip edge
- Adding Vertices to Output List



**Fig. 3.48** Four cases of polygon clipping.



# Sutherland-Hodgman Polygon Clipping

## Input:

1. Polygon described by an input of list of vertices:  $v_1, v_2, \dots, v_n$
2. Convex clip region C

## Algorithm:

Inputlist :  $v_1, v_2, \dots, v_n$

For each clip edge e in E do

$S \leftarrow v_n$

$P \leftarrow v_1$

$j \leftarrow 1$

While ( $j < n$ )

do, if both S & P inside the clip region,

Add p to output list

else if S inside & P outside, then

Find intersection point i

Add i to output list

else if S outside and P inside, then

find intersection point i

add i to output list

add P to output list

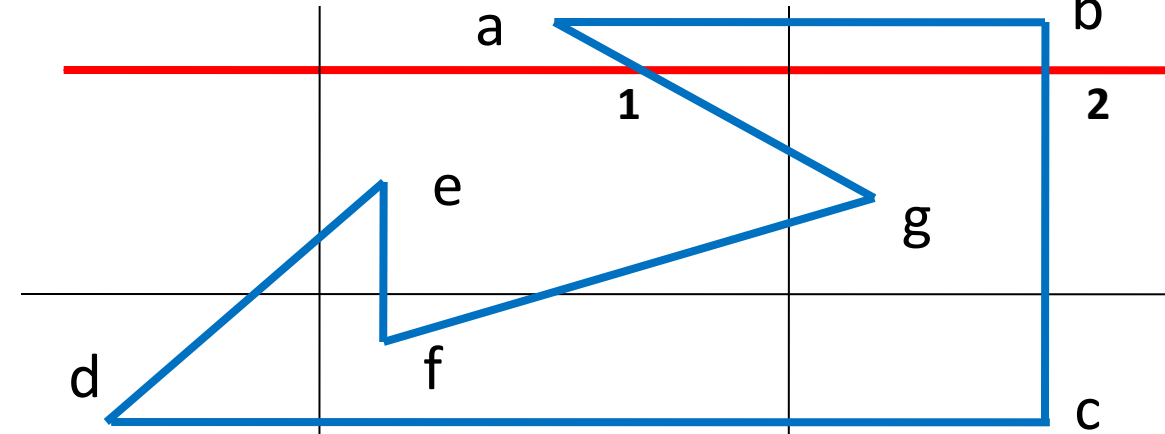
else if S and P both outside

do nothing

$S \leftarrow v_j$

# Sutherland-Hodgman Polygon Clipping

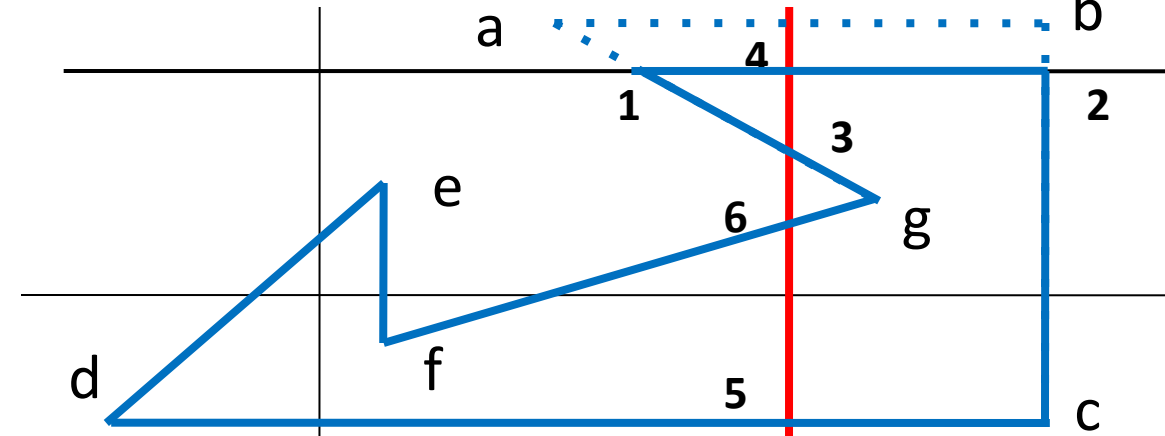
- a, b, c, d, e, f, g
- S = g, P = a
- Output: 1, 2, c, d, e, f, g



SP	Intersection	Output	Comments
g, a	1	1	g inside, a outside
a, b	-	-	Both outside
b, c	2	2,c	b outside, c inside
c, d	-	d	Both inside
d, e	-	e	Both inside
e, f	-	f	Both inside
f, g	-	g	Both inside

# Sutherland-Hodgman Polygon Clipping

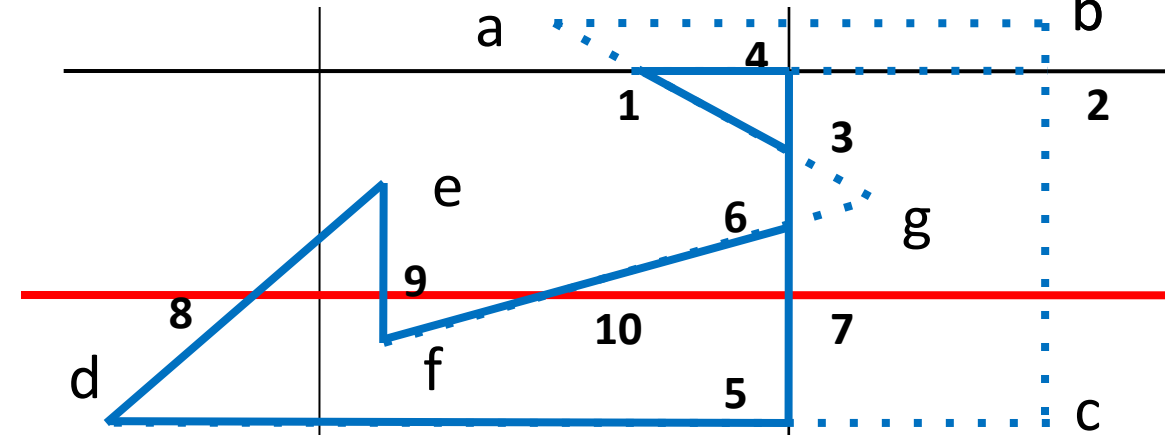
- Output of previous iteration  
1, 2, c, d, e, f, g
- $S = g$ ,  $P = 1$
- Output: 3, 1, 4, 5, d, e, f, 6



SP	Intersection	Output	Comments
g,1	3	3,1	g outside, 1 inside
1, 2	4	4	1 inside, 2 outside
2, c	-	-	Both outside
c, d	5	5,d	d inside, c outside
d, e	-	e	Both inside
e, f	-	f	Both inside
f, g	6	6	f inside, g outside

# Sutherland-Hodgman Polygon Clipping

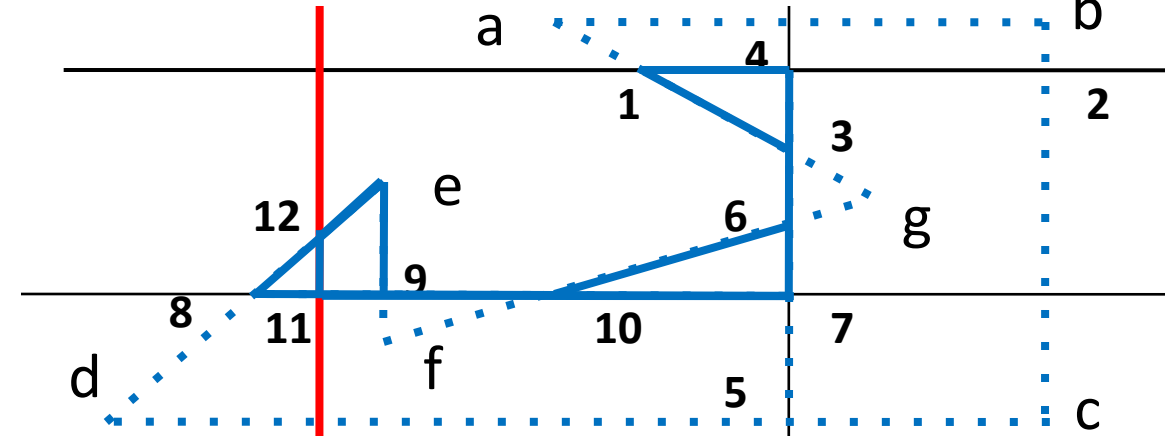
- Output of previous iteration  
3, 1, 4, 5, d, e, f, 6
- $S = 6$ ,  $P = 3$
- Output: 3, 1, 4, 7, 8, e, 9, 10, 6



SP	Intersection	Output	Comments
6, 3	-	3	Both inside
3, 1	-	1	Both inside
1, 4	-	4	Both inside
4, 5	7	7	4 inside, 5 outside
5, d	-	-	Both outside
d, e	8	8, e	e inside, d outside
e, f	9	9	e inside, f outside
f, 6	10	10, 6	6 inside, f outside

# Sutherland-Hodgman Polygon Clipping

- Output of previous iteration  
3, 1, 4, 7, 8, e, 9, 10, 6
- $S = 6$ ,  $P = 3$
- Output: 3, 1, 4, 7, 11, 12, e,  
9, 10, 6



SP	Intersection	Output	Comments
6, 3	-	3	Both inside
3, 1	-	1	Both inside
1, 4	-	4	Both inside
4, 7	-	7	Both inside
7, 8	11	11	7 inside, 8 outside
8, e	12	12, e	e inside, 8 outside
e, 9	-	9	Both inside
9, 10	-	10	Both inside
10, 6	-	6	Both inside

# Clipping Circles (and Ellipses)

- Analytical
  - Intersect circle's extent (square of size of circle's diameter) with clip rectangle
  - Run the algorithm of polygon clipping
    - No intersect : trivial reject
    - Intersect : divide into quadrants (and later octants if needed) and repeat
  - Compute intersection by solving equations
- During Scan Conversion
  - When circle is relatively small or scan conversion is fast
  - After extent checking scissor on a pixel by pixel basis

□ Similar Approach for Ellipses!



# Reference:

Computer Graphics: Principles and Practice: John F. Hughes, James D. Foley, Andries van Dam, Steven K. Feiner (2nd Edition)

**Chapter: 3.12, 3.14**

