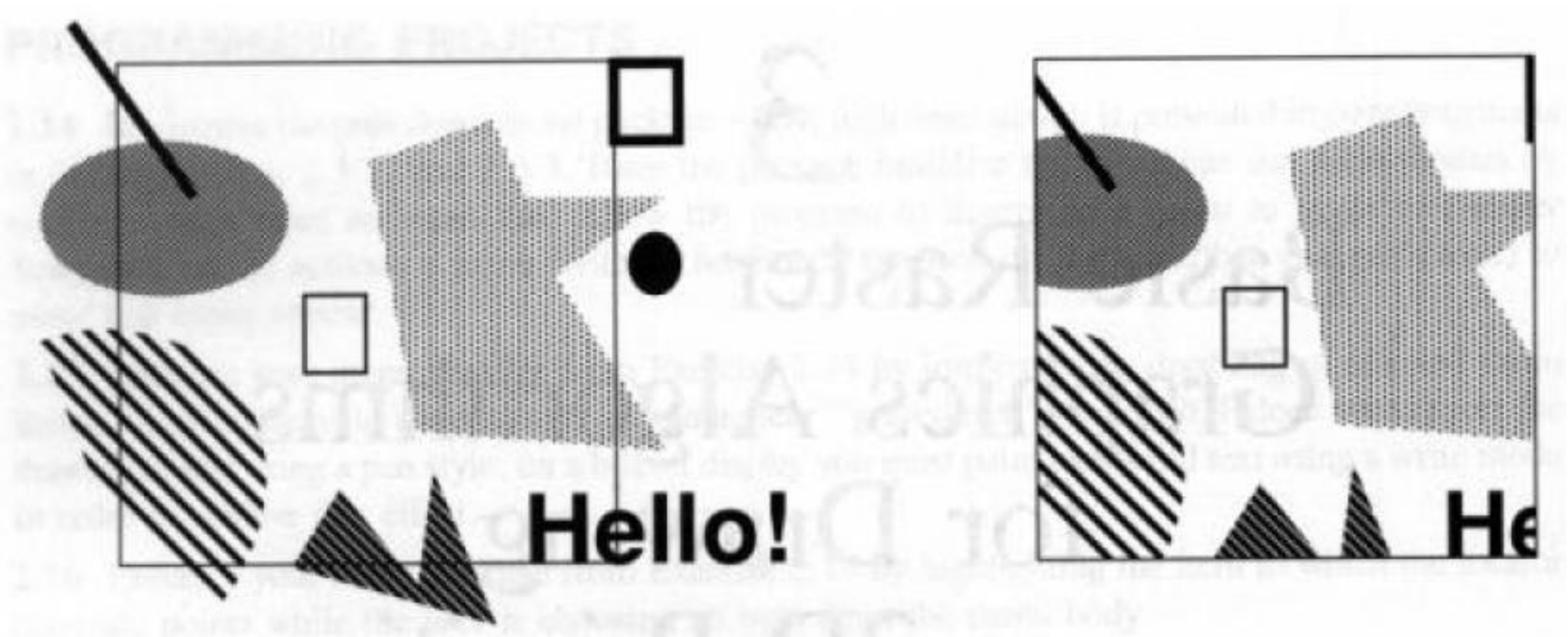
Clipping

Foley : Ch 3

3.11 - 3.14

What is Clipping?

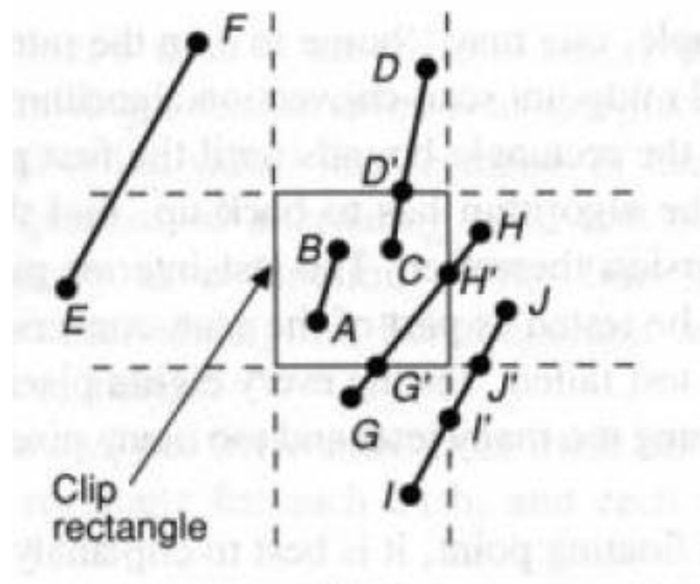


Clipping in a Raster World

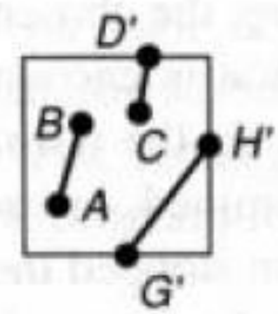
- Clipping techniques
  - **Analytical**
    - Lines, polygons etc.
    - Floating point graphics package
  - During scan conversion (scissoring)
    - Checking extrema suffices, internal points can be ignored
    - Circles, curves etc.
  - During writing a pixel
    - Outline primitive not much larger,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

Clipping Lines

- Clip rectangle
  - x_{min} to x_{max}
  - y_{min} to y_{max}
- Clipping endpoints
  - A point (x,y) lies within a clip rectangle and thus displayed if following conditions are hold
    - $x_{min} <= x <= x_{max}$
    - $y_{min} <= y <= y_{max}$

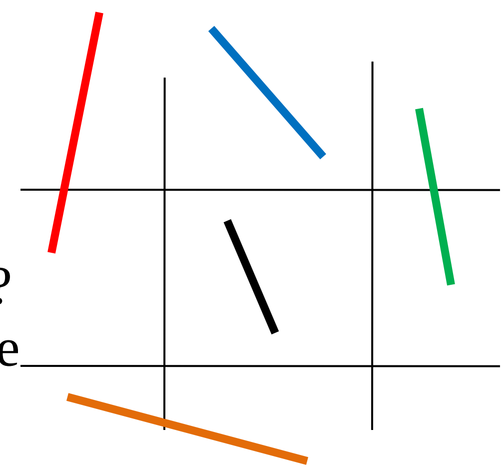


(a)



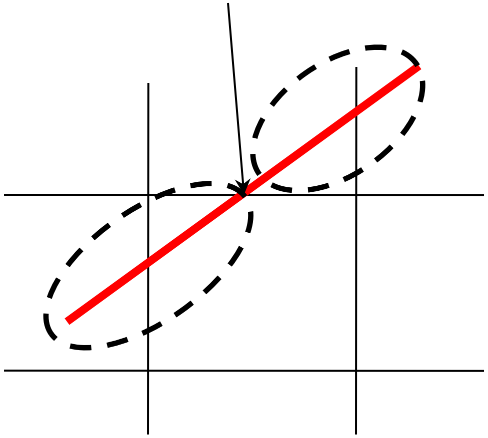
Cohen-Sutherland Line Clipping Algorithm

1. End points are checked to for **trivial acceptance**
   – Both endpoints are inside the clip rectangle boundary?
2. If not trivially accepted, **region check** is done for **trivial rejection**
   – Both endpoints have x coordinate less than x_{min} ?
     □ region to the left of left edge of clip rectangle
   – Both endpoints have x coordinate greater than x_{max} ?
     □ region to the right of right edge of clip rectangle
   – Both endpoints have y coordinate less than y_{min} ?
     □ region below the bottom edge of clip rectangle
   – Both endpoints have y coordinate greater than y_{max} ?
     □ region above the top edge of clip rectangle



Cohen-Sutherland Line Clipping Algorithm

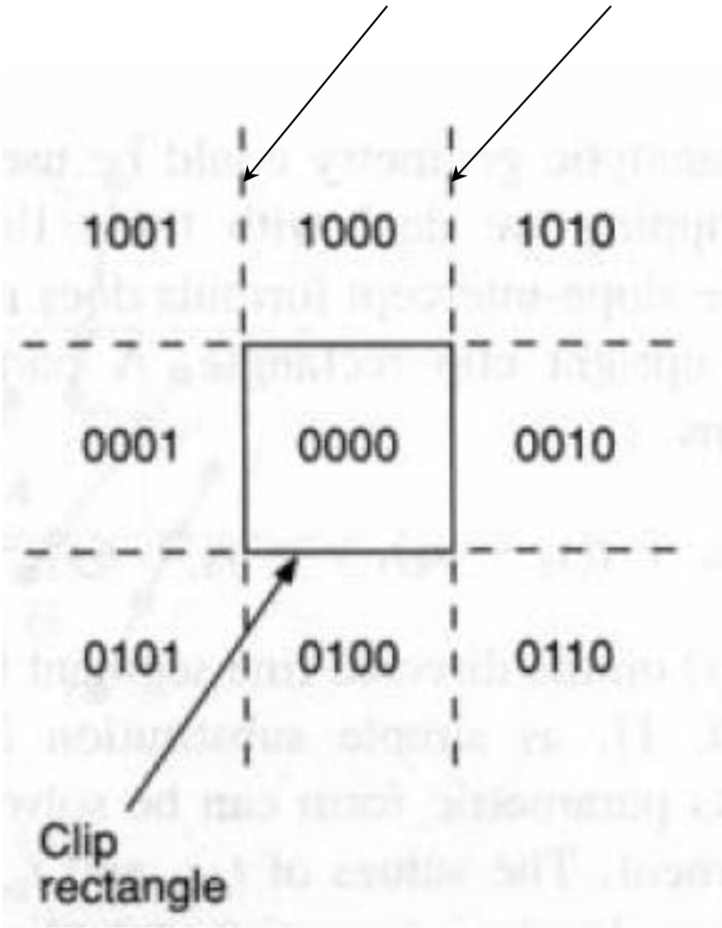
3. If neither trivially accepted, nor trivially rejected
   - Divided into two segments at the intersection point of a clip edge, such that
     - one segment can be trivially rejected
     - Another segment is subdivided iteratively until cannot be trivially rejected or accepted.



- How are the trivial acceptance/rejection tests done?
- In what order are the intersections with the clip edges considered?

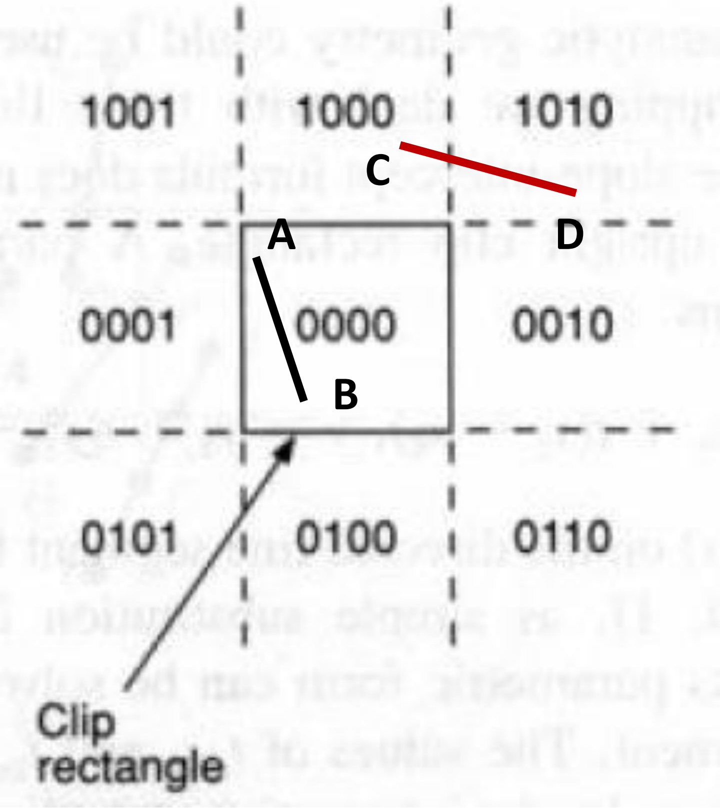
Cohen-Sutherland Line Clipping Algorithm

- Extend the edges of the clip rectangle
  - Divide the plane into 9 regions
- Each region is assigned a 4 bit code called outcode
  - First bit is 1 if the region is above the top edge, 0 otherwise
  - Second bit is 1 if the region is below the bottom edge, 0 otherwise
  - Third bit is 1 if the region is right to the right edge, 0 otherwise
  - Fourth bit is 1 if the region is left to the left edge, 0 otherwise



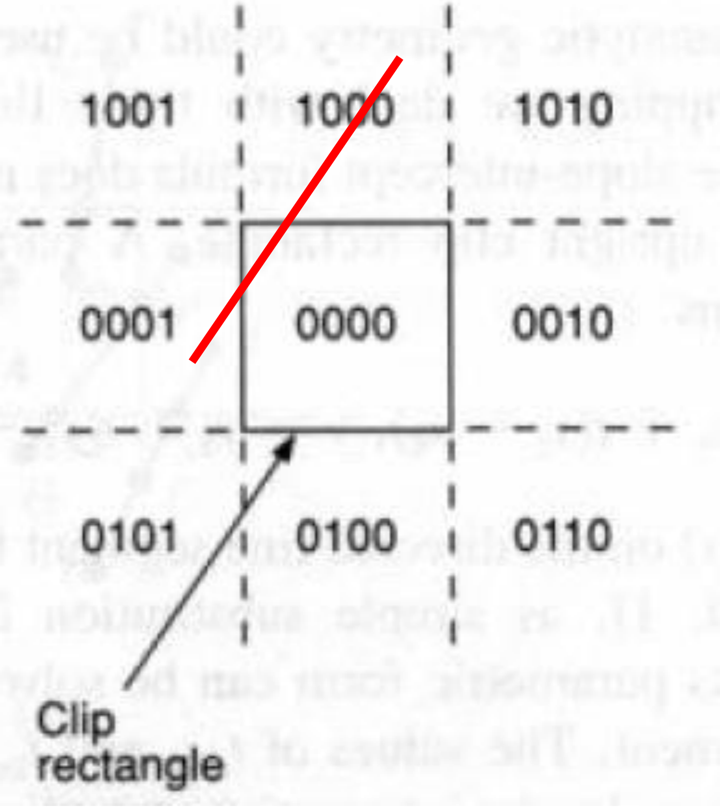
Cohen-Sutherland Line Clipping Algorithm

- Each endpoint of the line segment is assigned the outcode
  - If both outcodes are 0000 then it completely lies inside the clip rectangle
    □ **trivial acceptance**
    A:0000
    B:0000
  - Otherwise perform the logical AND of the outcodes. If results in non zero
    □ **trivial rejection**
    C:1000
    D:1010
    AND gives 1000 (non zero)
    □ above the top edge



Cohen-Sutherland Line Clipping Algorithm

- If neither trivially accepted nor rejected, subdivide it based on a intersecting edge
  - Select the outcode of the endpoint that lies outside
  - Choose a set bit in that outcode for selecting the edge for subdivision
  - Order: top bottom right left
  - Leftmost set bit in the outcode is used for selecting the edge for subdivision



Cohen-Sutherland Line Clipping Algorithm

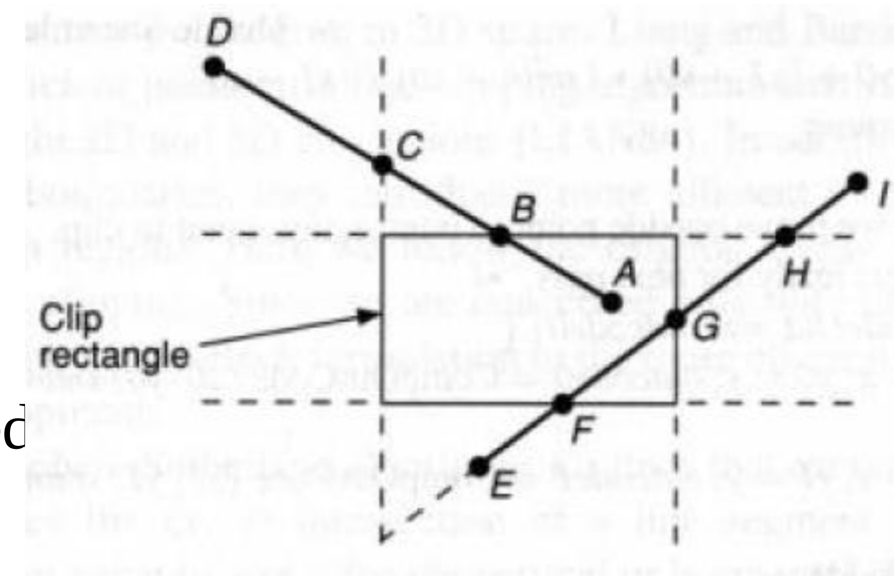
<u>Example: Consider AD</u>

<u>Iteration 1:</u>

1. A=0000
   D=1001

Both endpoints are not 0000

So cannot be trivially accepted



2. AND gives zero
   – Cannot trivially rejected
3. Select the outcode that lies outside: D=1001
   – Select leftmost set bit: first bit = top edge for subdivision
     □ new end point B is found
   – Assign outcode 0000 to B

Cohen-Sutherland Line Clipping Algorithm

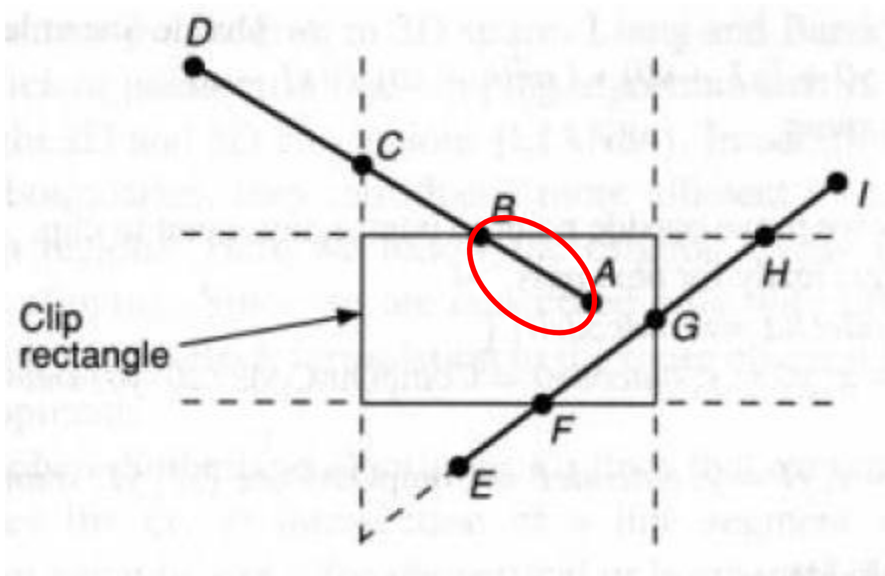
<u>Example: Consider AD</u>

<u>Iteration 2:</u>

1. A=0000
   B=0000

Both endpoints are 0000

Trivially accepted.



Cohen-Sutherland Line Clipping Algorithm

<u>Example: Consider IE</u>

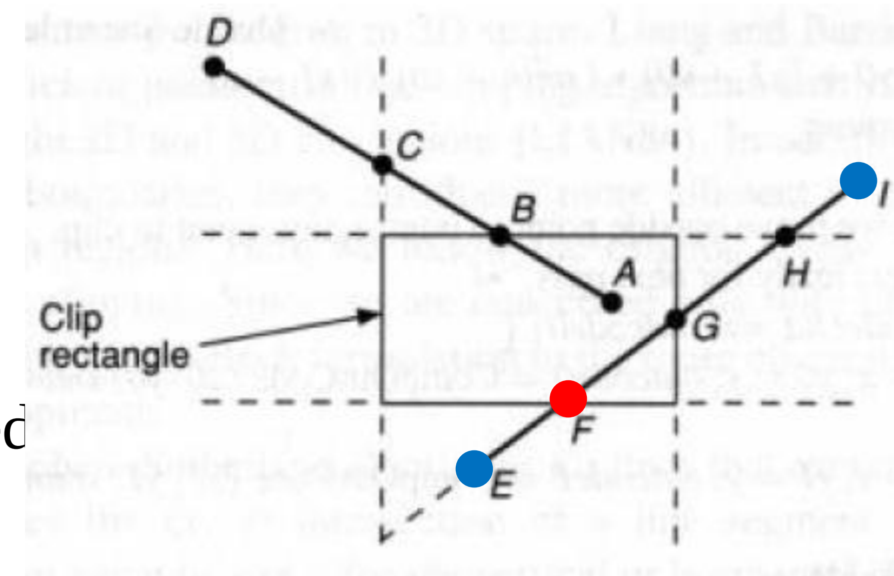
<u>Iteration 1:</u>

1. E=0100
   I=1010

Both endpoints are not 0000

So cannot be trivially accepted

2. AND gives zero
   – Cannot trivially rejected



3. Select the outcode that lies outside the clip rectangle:
   E=0**1**00 (or I)
   – Select leftmost set bit: second bit = bottom edge for subdivision
     □ **new end point F is found**
   – Assign outcode 0000 to F

Cohen-Sutherland Line Clipping Algorithm

<u>**Example: Consider IE**</u>

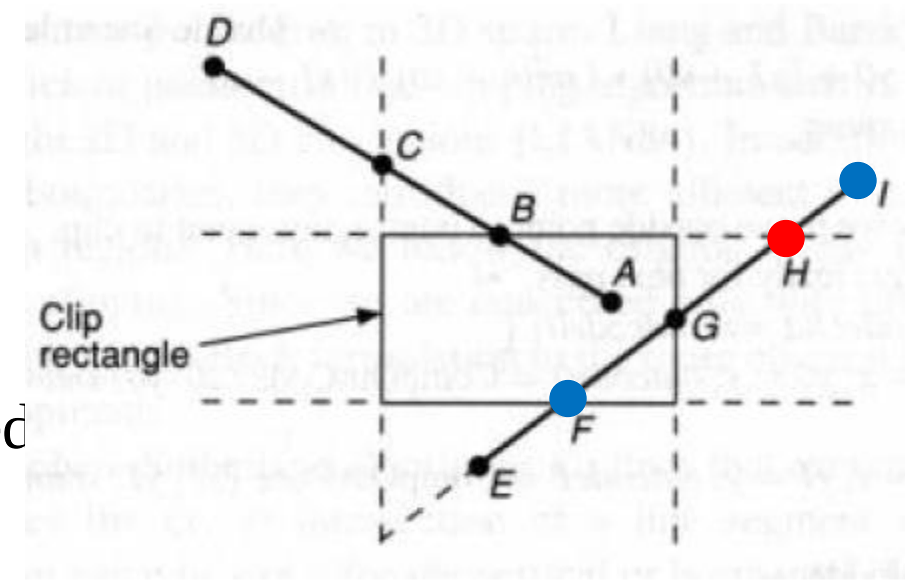
<u>Iteration 2:</u>

1. F=0000
   I=1010

Both endpoints are not 0000

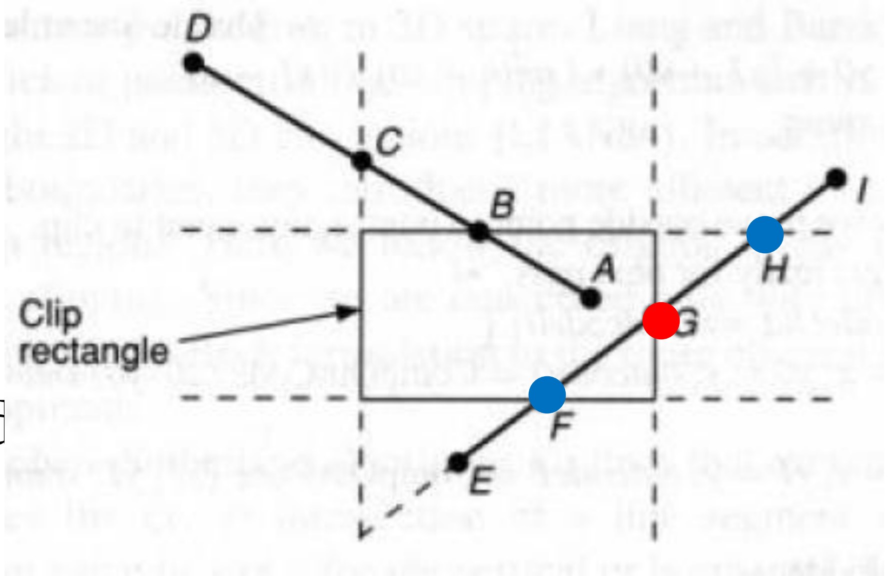
So cannot be trivially accepted

2. AND gives zero
   – Cannot trivially rejected



3. Select the outcode that lies outside the clip rectangle: I=1010
   – Select leftmost set bit: first bit = top edge for subdivision
     - **new end point H is found**
   – Assign outcode 0010 to H

Cohen-Sutherland Line Clipping Algorithm



<u>Example: Consider IE</u>

<u>Iteration 3:</u>

1. F=0000
   H=0010

Both endpoints are not 0000

So cannot be trivially accepted

2. AND gives zero
   – Cannot trivially rejected
3. Select the outcode that lies outside the clip rectangle: H=0010
   – Select leftmost set bit: third bit = right edge for subdivision
     - **new end point G is found**
   – Assign outcode 0000 to H

Cohen-Sutherland Line Clipping Algorithm

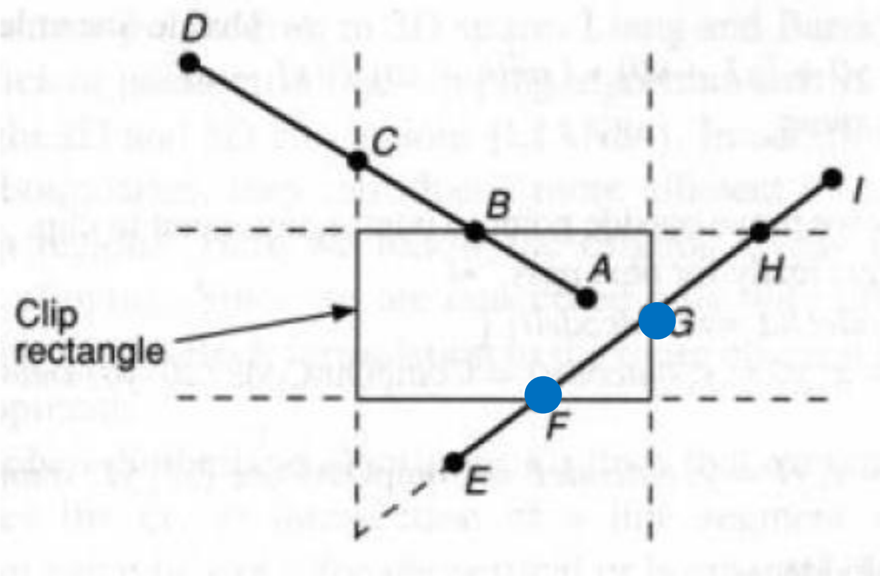
<u>Example: Consider IE</u>

<u>Iteration 4:</u>

1. F=0000
   G=0000

Both endpoints are 0000

Trivially accepted.



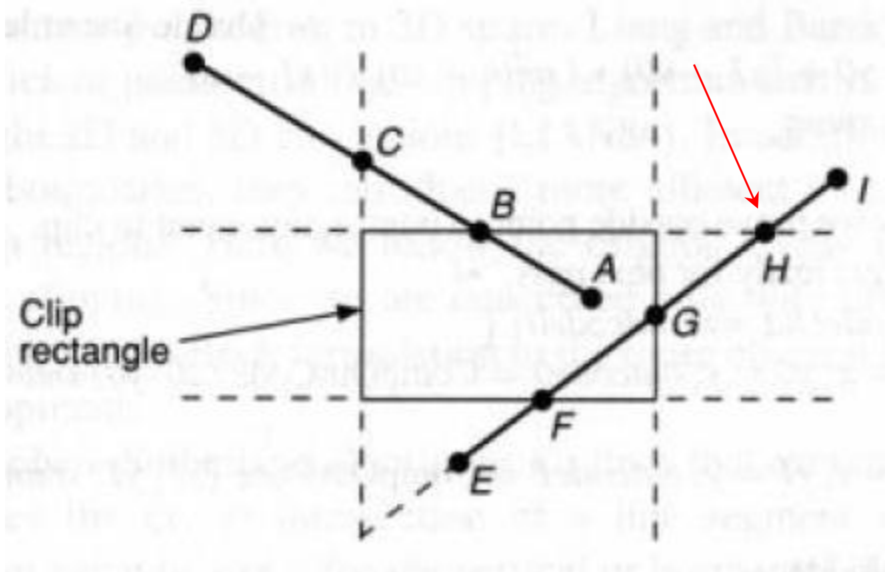
Cohen-Sutherland Line Clipping Algorithm

Works well for two cases:

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

- Why?

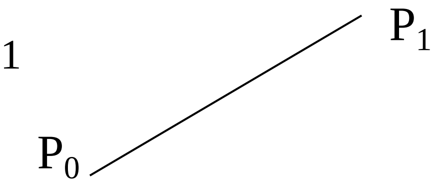
[For many trivial accept and many trivial reject]



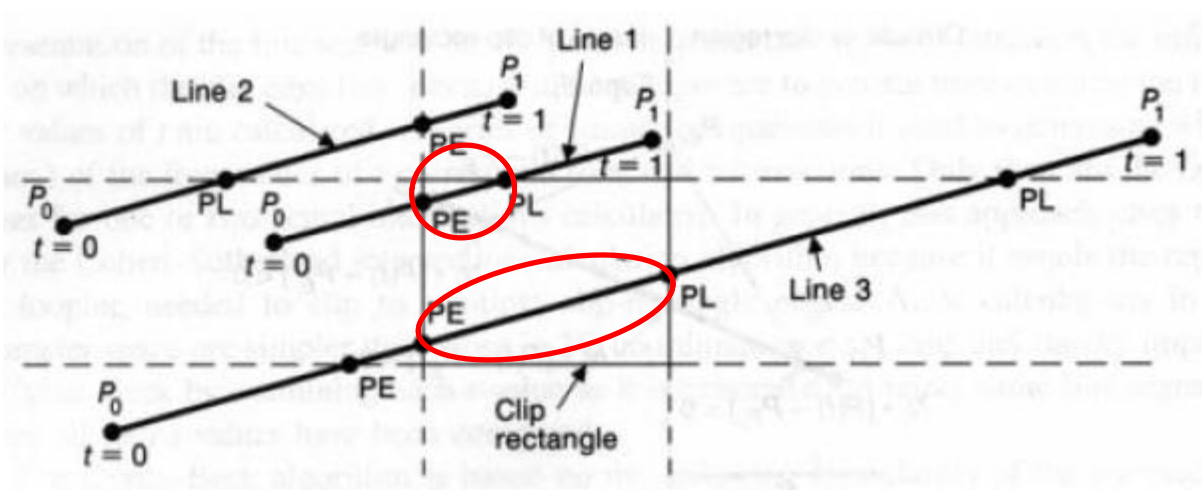
<u>Where is the problem?</u>

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

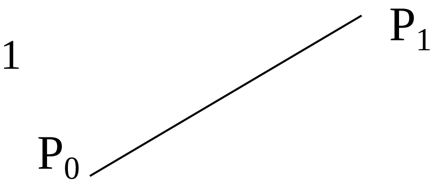
Cyrus-Beck Parametric Line Clipping Algorithm



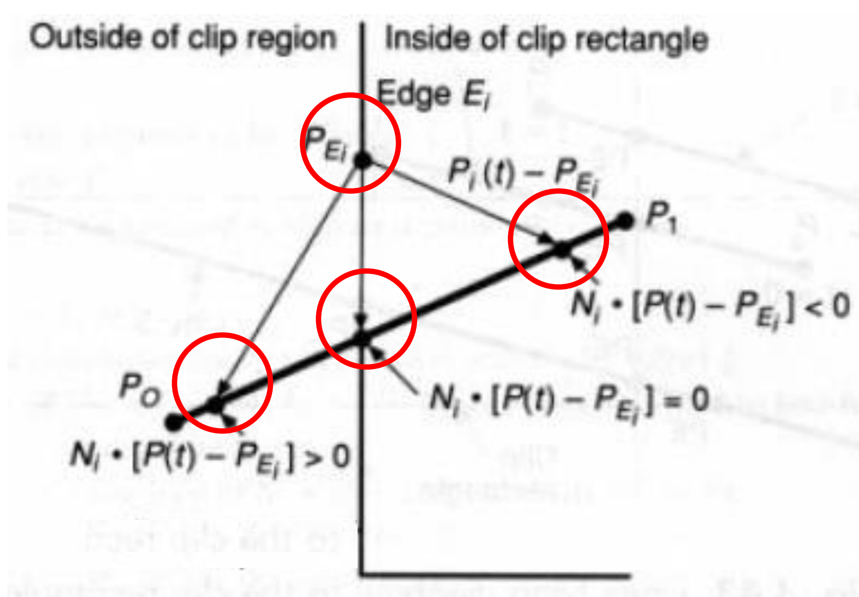
- Parametric representation of a line from P_o to P_1
  $P(t) = P_0 + (P_1 - P_0)t; \quad t = [0,1]$
- Extend the clip edges and the line
- If the line is not parallel to any edge, it will eventually intersect all four edges
- Four intersection points, that is 4 values of t
- Only valid intersection points (presenting line segment inside the clip rectangle) are considered



Cyrus-Beck Parametric Line Clipping Algorithm

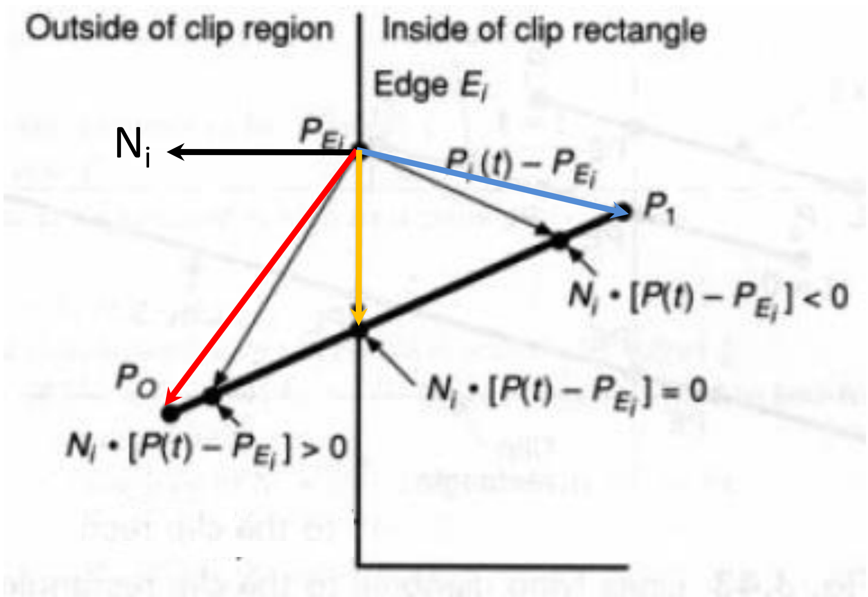


- Parametric representation of a line from P_o to P_1
  $P(t) = P_0 + (P_1 - P_0)t; \quad t = [0,1]$
- Let us consider the left edge of clip rectangle
- Select a point, say P_{Ei} on that edge
- Vector from P_{Ei} to any point on the line by $P(t) - P_{Ei}$



Cyrus-Beck Parametric Line Clipping Algorithm

- Vector from P_{Ei} to any point on the line by $P(t) - P_{Ei}$
  - t = 0
  - t = 1
  - t = ? □ for intersecting point

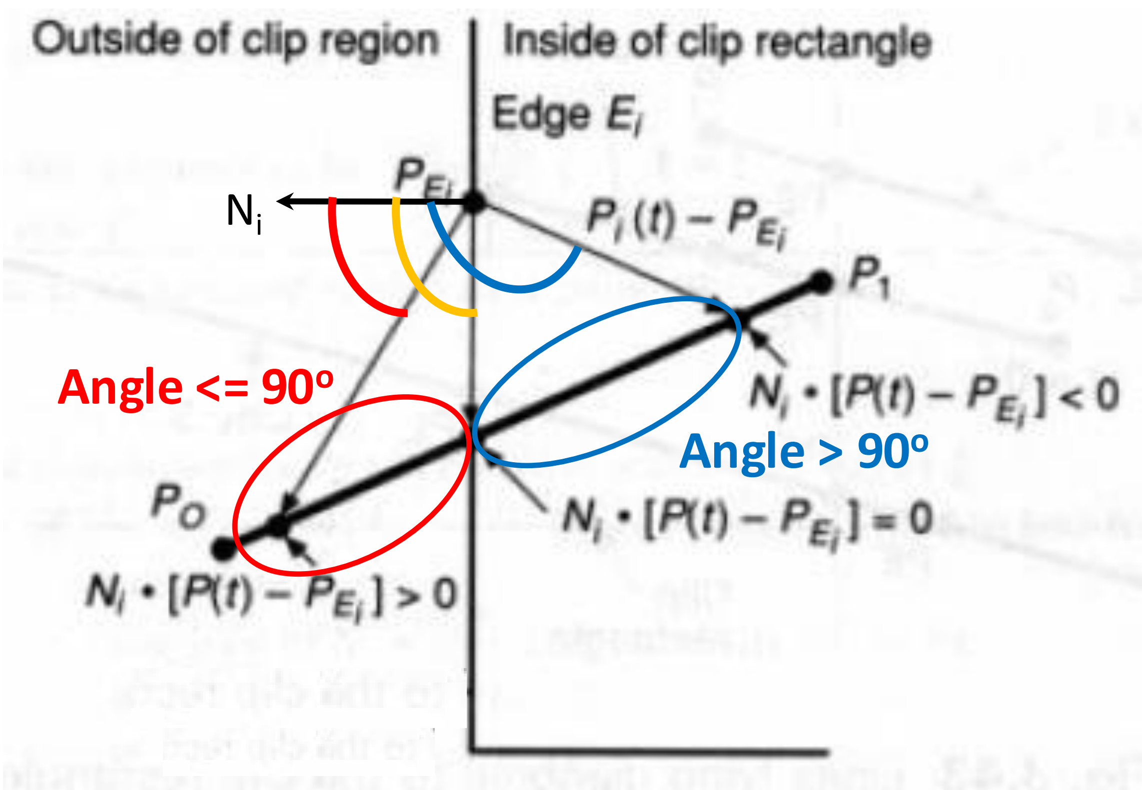


Cyrus-Beck Parametric Line Clipping Algorithm

How to know whether we are outside or inside the clip region?

Consider a normal vector N_i

Consider the ***angle*** between N_i and vector: $P(t) - P_{Ei}$

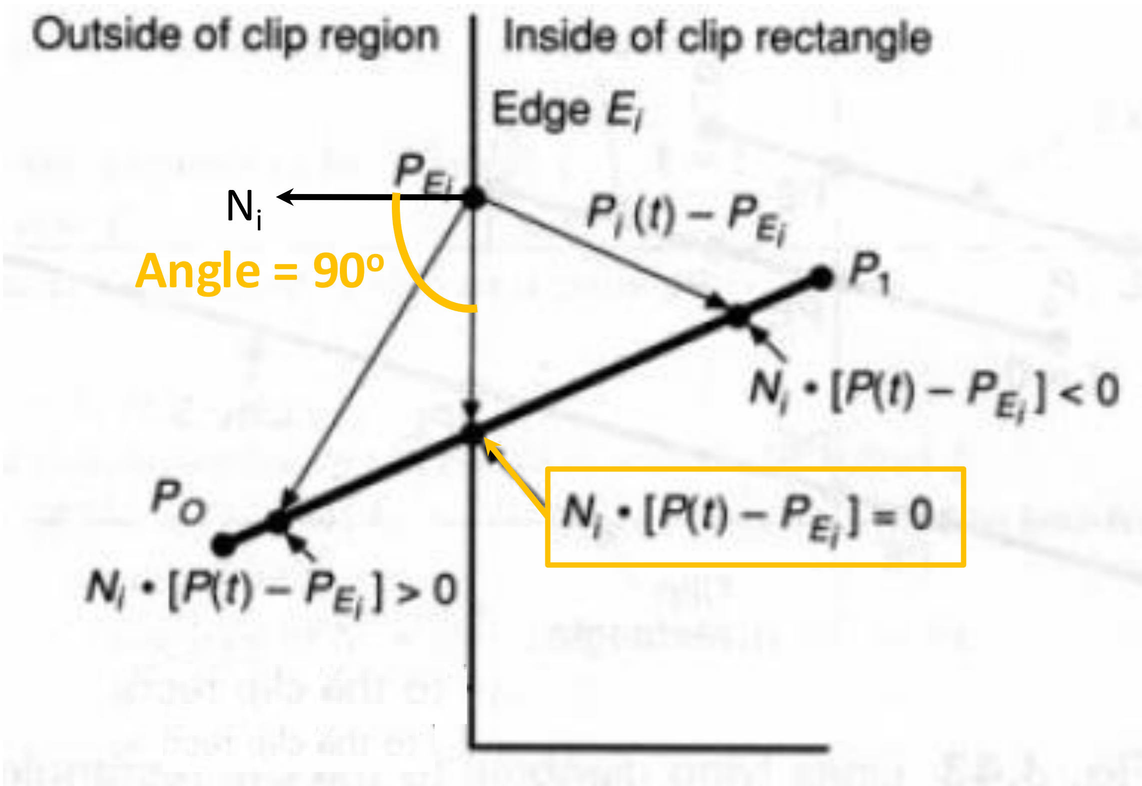


Cyrus-Beck Parametric Line Clipping Algorithm

Angle between Ni and the vector at the intersection point is 90°

Solve the equation: $N_i \,.\, [P(t) - P_{Ei}] = 0$

- for the value of t at **intersection point**



Cyrus-Beck Parametric Line Clipping Algorithm

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

First, substitute for $P(t)$:

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

Next, group terms and distribute the dot product:

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

Let $D = (P_1 - P_0)$ be the vector from P_0 to P_1, and solve for t:

$$t = \frac{N_i \cdot [P_0 - P_{E_i}]}{-N_i \cdot D}.$$

Cyrus-Beck Parametric Line Clipping Algorithm

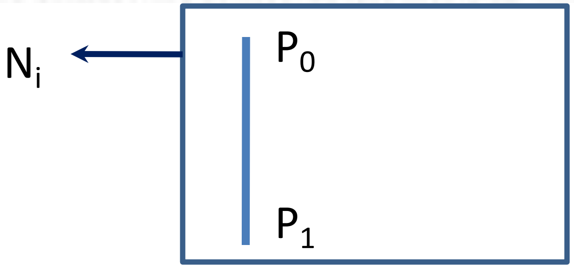
Let $D = (P_1 - P_0)$ be the vector from P_0 to P_1, and solve for t:

$$t = \frac{N_i \cdot [P_0 - P_{E_i}]}{-N_i \cdot D}.$$

$N_i \neq 0$ (that is, the normal should not be 0; this could occur only as a mistake),

$D \neq 0$ (that is, $P_1 \neq P_0$),

$N_i \cdot D \neq 0$ (that is, the edge E_i and the line from P_0 to P_1 are not parallel. If they were parallel, there can be no single intersection for this edge, so the algorithm moves on to the next case.).



Cyrus-Beck Parametric Line Clipping Algorithm

Let $D = (P_1 - P_0)$ be the vector from P_0 to P_1, and solve for t:

$$t = \frac{N_i \cdot [P_0 - P_{E_i}]}{-N_i \cdot D}.$$

- This equation can be used to find intersection of P_0P_1 with each clip edge
  - Consider any arbitrary point, say endpoint of each edge as P_{Ei}
  - Outward Normal N_i for that edge.

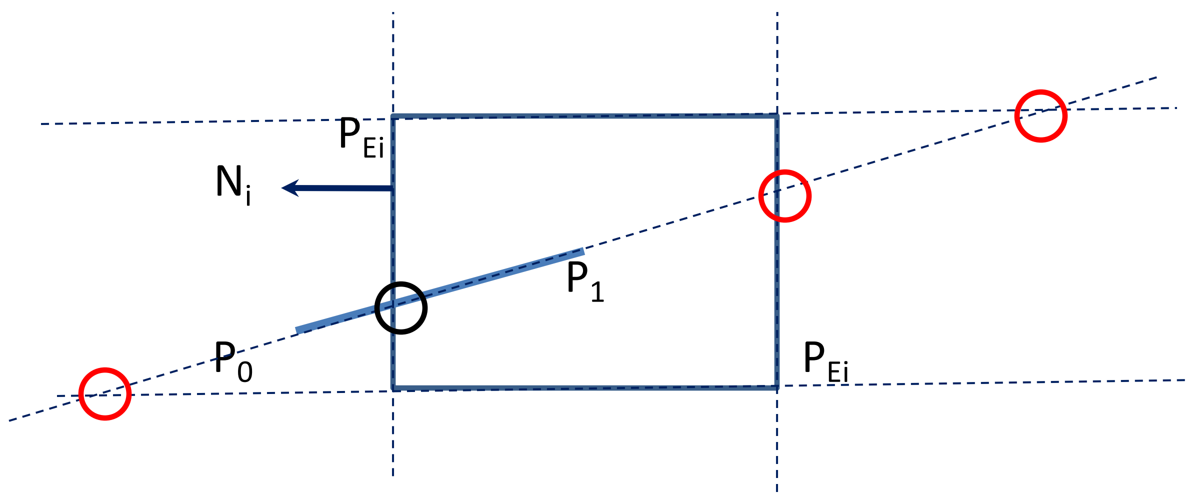
Cyrus-Beck Parametric Line Clipping Algorithm

Let $D = (P_1 - P_0)$ be the vector from P_0 to P_1, and solve for t:

$$t = \frac{N_i \cdot [P_0 - P_{E_i}]}{-N_i \cdot D}.$$

Trivial Rejection:

- Intersection points which has t not in [0,1]



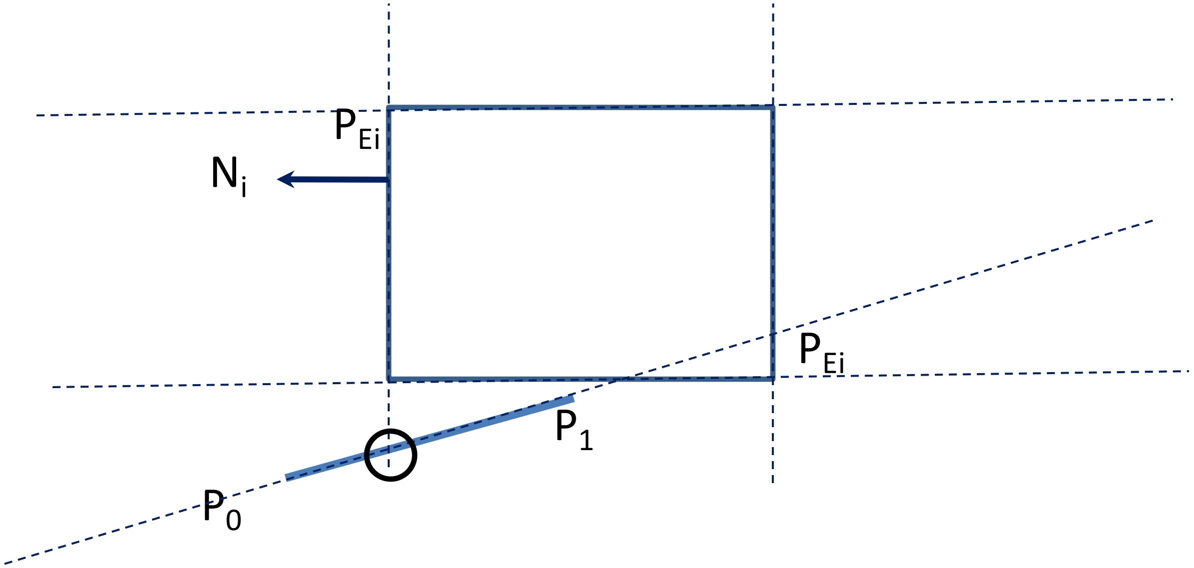
Cyrus-Beck Parametric Line Clipping Algorithm

Let $D = (P_1 - P_0)$ be the vector from P_0 to P_1, and solve for t:

$$t = \frac{N_i \cdot [P_0 - P_{E_i}]}{-N_i \cdot D}.$$

What about remaining points?

- May not always give the internal segment



Cyrus-Beck Parametric Line Clipping Algorithm

How to identify internal segments?

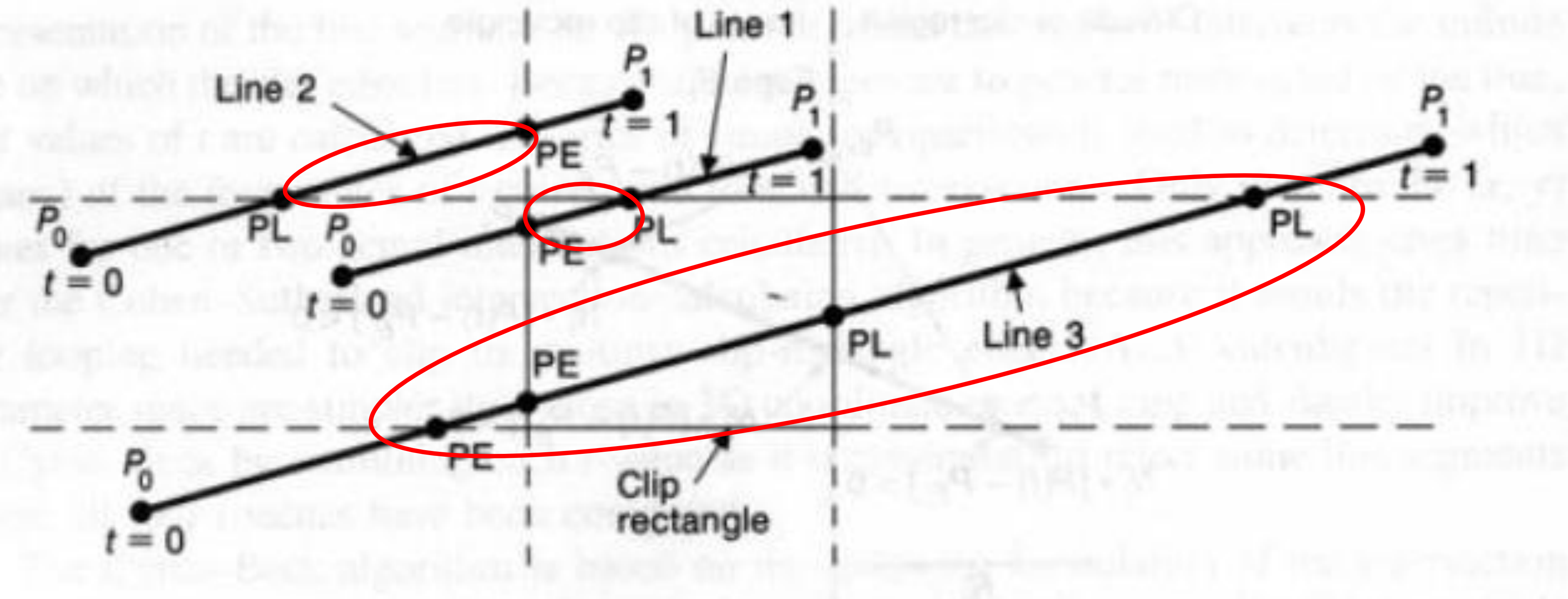


Fig. 3.43 Lines lying diagonal to the clip rectangle.

Cyrus-Beck Parametric Line Clipping Algorithm

New Idea!!

- PE = Potentially entering the clip rectangle
  = Moving from P_0 to P_1 causes us to cross a particular edge to **enter the edge's inside half plane**
- PL = Potentially leaving the clip rectangle
  = Moving from P_0 to P_1 causes us to cross a particular edge to **leave the edge's inside half plane**

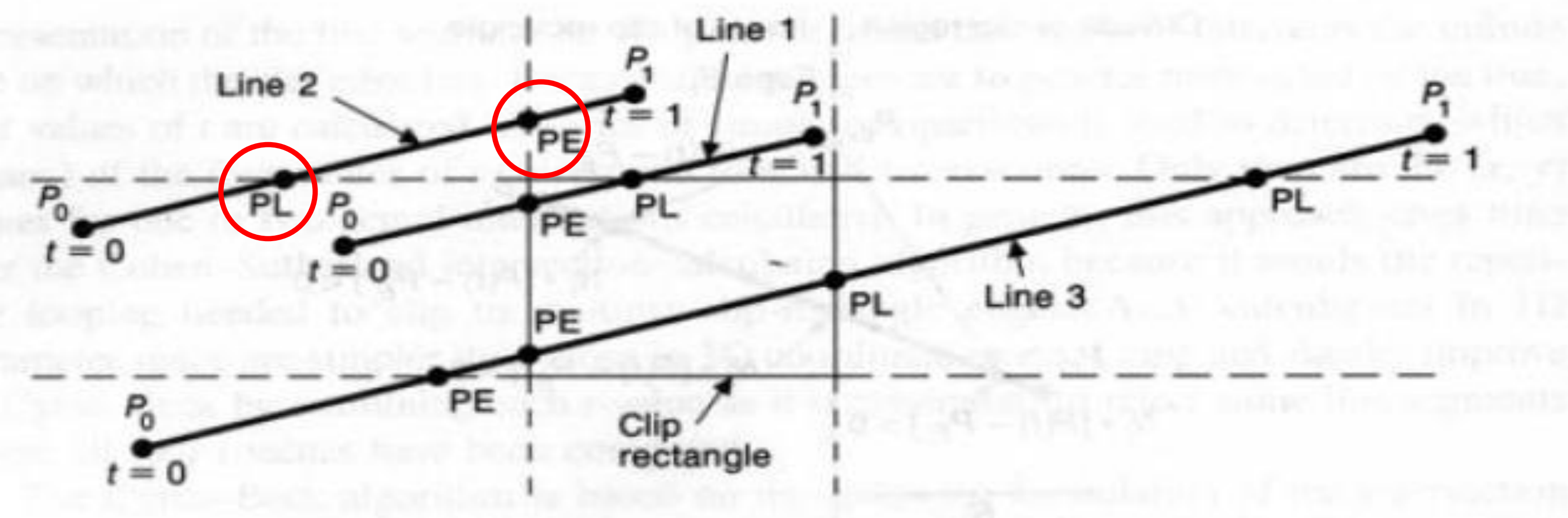


Fig. 3.43 Lines lying diagonal to the clip rectangle.

Cyrus-Beck Parametric Line Clipping Algorithm

New Idea!!

- PE = Angle of the line with the outward normal at that intersection point is > 90
- PL = Angle of the line with the outward normal at that intersection point is < 90

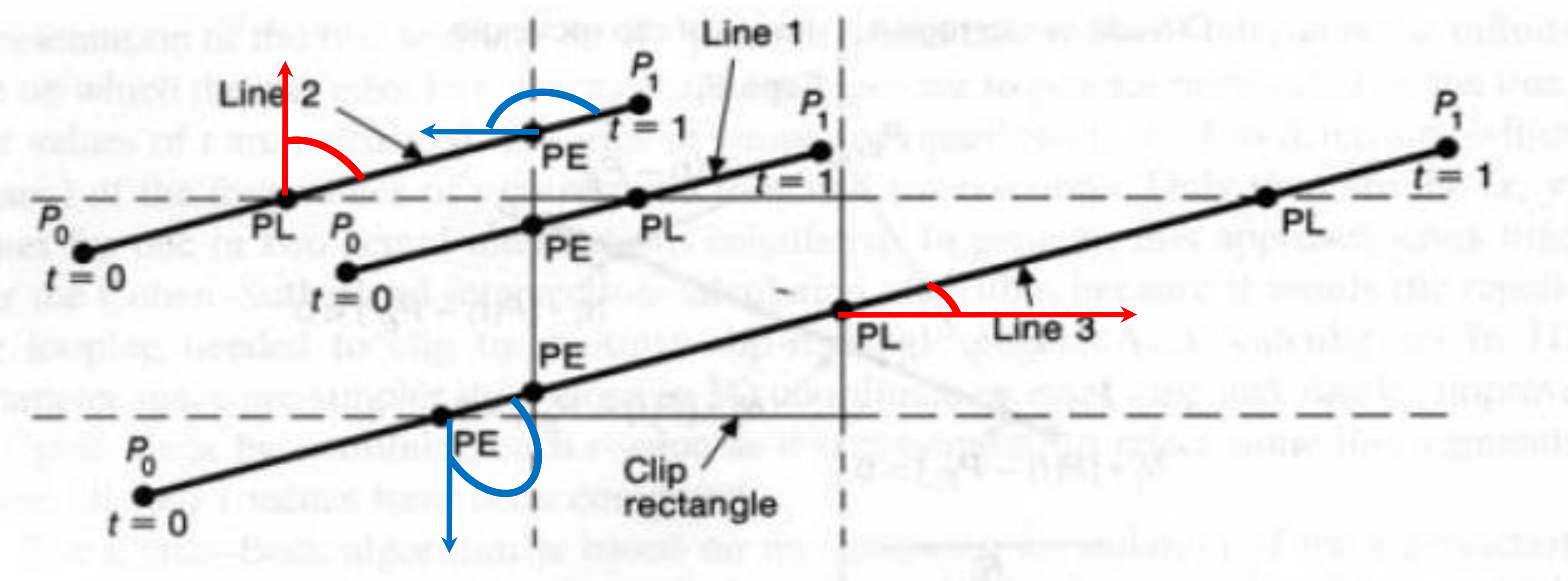


Fig. 3.43 Lines lying diagonal to the clip rectangle.

Cyrus-Beck Parametric Line Clipping Algorithm

How to know if an intersection point is a PL or PE ?

$$t = \frac{N_i \cdot [P_0 - P_{E_i}]}{-N_i \cdot D}.$$

$$N_i \cdot D < 0 \Rightarrow \text{PE (angle greater than 90)},$$
$$N_i \cdot D > 0 \Rightarrow \text{PL (angle less than 90)}.$$

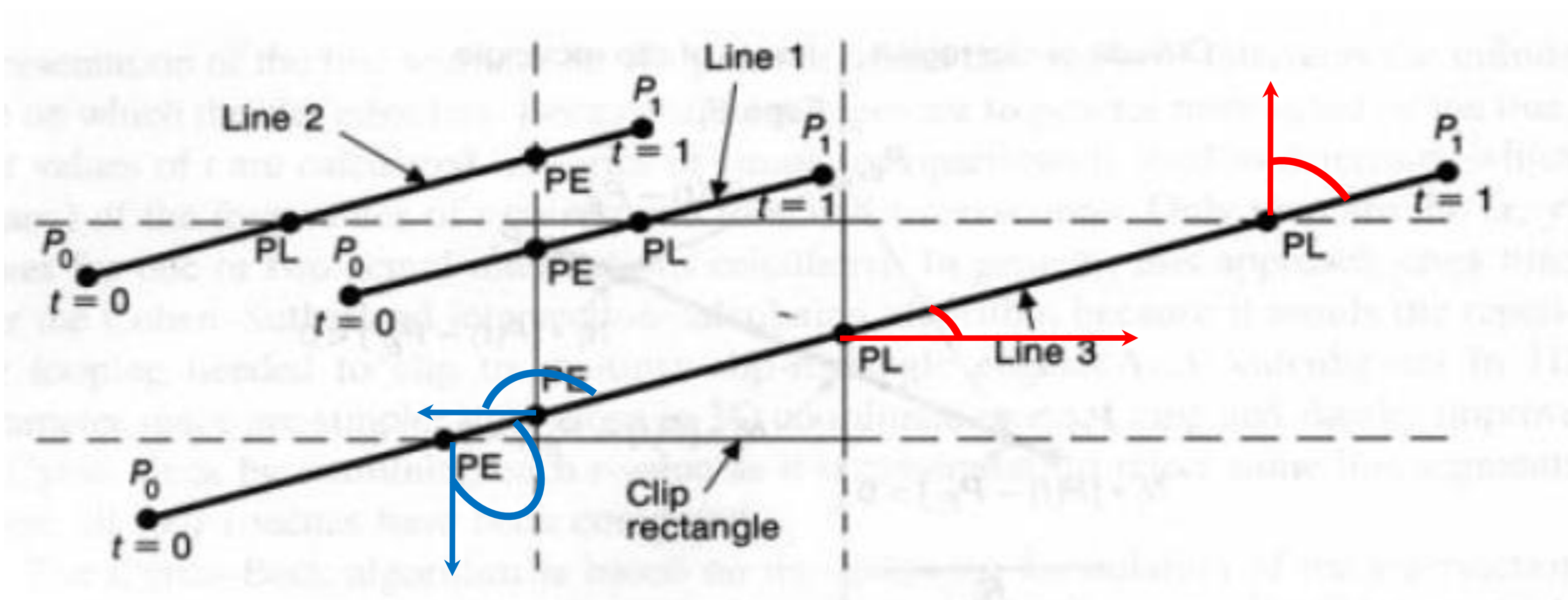


Fig. 3.43 Lines lying diagonal to the clip rectangle.

Cyrus-Beck Parametric Line Clipping Algorithm

Find the intersection points as shown in the table

TABLE 3.1 CALCULATIONS FOR PARAMETRIC LINE CLIPPING ALGORITHM*

Clip edge$_i$	Normal N_i	P_{E_i}	$P_0 - P_{E_i}$	$t = \frac{N_i \cdot (P_0 - P_{E_i})}{-N_i \cdot D}$
left: $x = x_{min}$	$(-1, 0)$	(x_{min}, y)	$(x_0 - x_{min}, y_0 - y)$	$\frac{-(x_0 - x_{min})}{(x_1 - x_0)}$
right: $x = x_{max}$	$(1, 0)$	(x_{max}, y)	$(x_0 - x_{max}, y_0 - y)$	$\frac{(x_0 - x_{max})}{-(x_1 - x_0)}$
bottom: $y = y_{min}$	$(0, -1)$	(x, y_{min})	$(x_0 - x, y_0 - y_{min})$	$\frac{-(y_0 - y_{min})}{(y_1 - y_0)}$
top: $y = y_{max}$	$(0, 1)$	(x, y_{max})	$(x_0 - x, y_0 - y_{max})$	$\frac{(y_0 - y_{max})}{-(y_1 - y_0)}$

*The exact coordinates of the point P_{E_i} on each edge are irrelevant to the computation, so they have been denoted by variables x and y. For a point on the right edge, $x = x_{min}$ as indicated in the first row, third entry.

Cyrus-Beck Parametric Line Clipping Algorithm

How to get a PE, PL pair presenting a segment inside clip rectangle?

1. t_E = t value of the PE point having highest value of t
2. t_L = t value of the PL point having lowest value of t
3. $t_E < t_L$

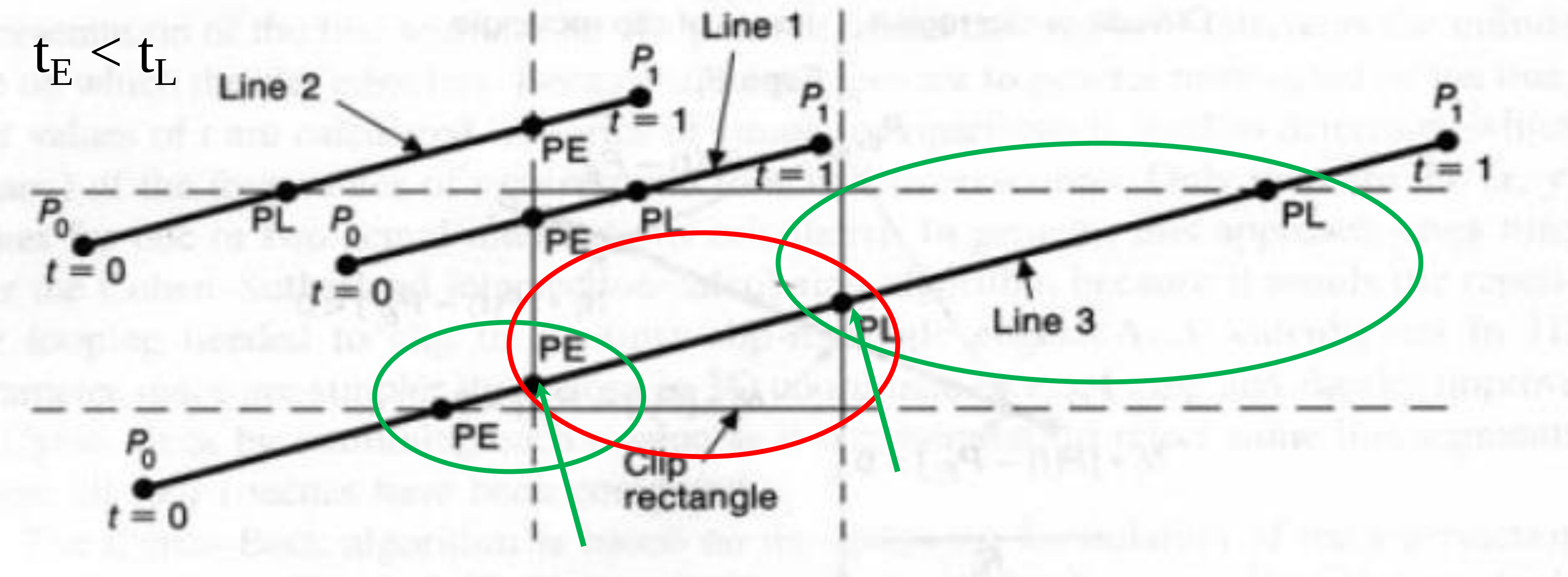


Fig. 3.43 Lines lying diagonal to the clip rectangle.

Cyrus-Beck Parametric Line Clipping Algorithm

How to get a PE, PL pair presenting a segment inside clip rectangle?

- If $t_E > t_L$ □ reject

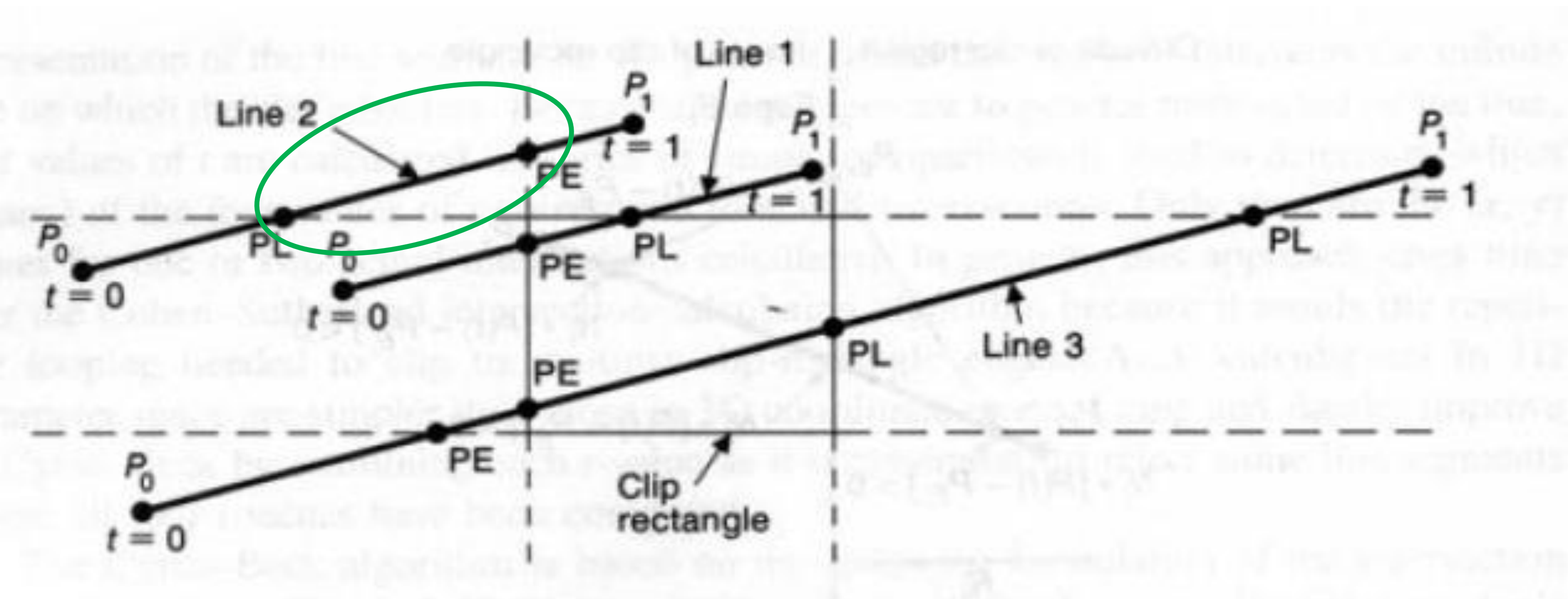
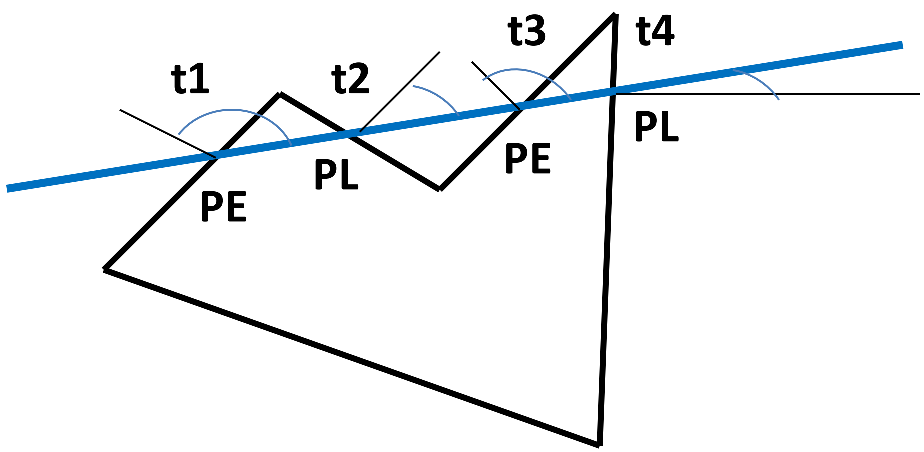


Fig. 3.43 Lines lying diagonal to the clip rectangle.

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



$t_E = t_3$
$t_L = t_2$
$t_E > t_L$
So the whole line is discarded though some segments should be displayed

Sutherland-Hodgman Polygon Clipping

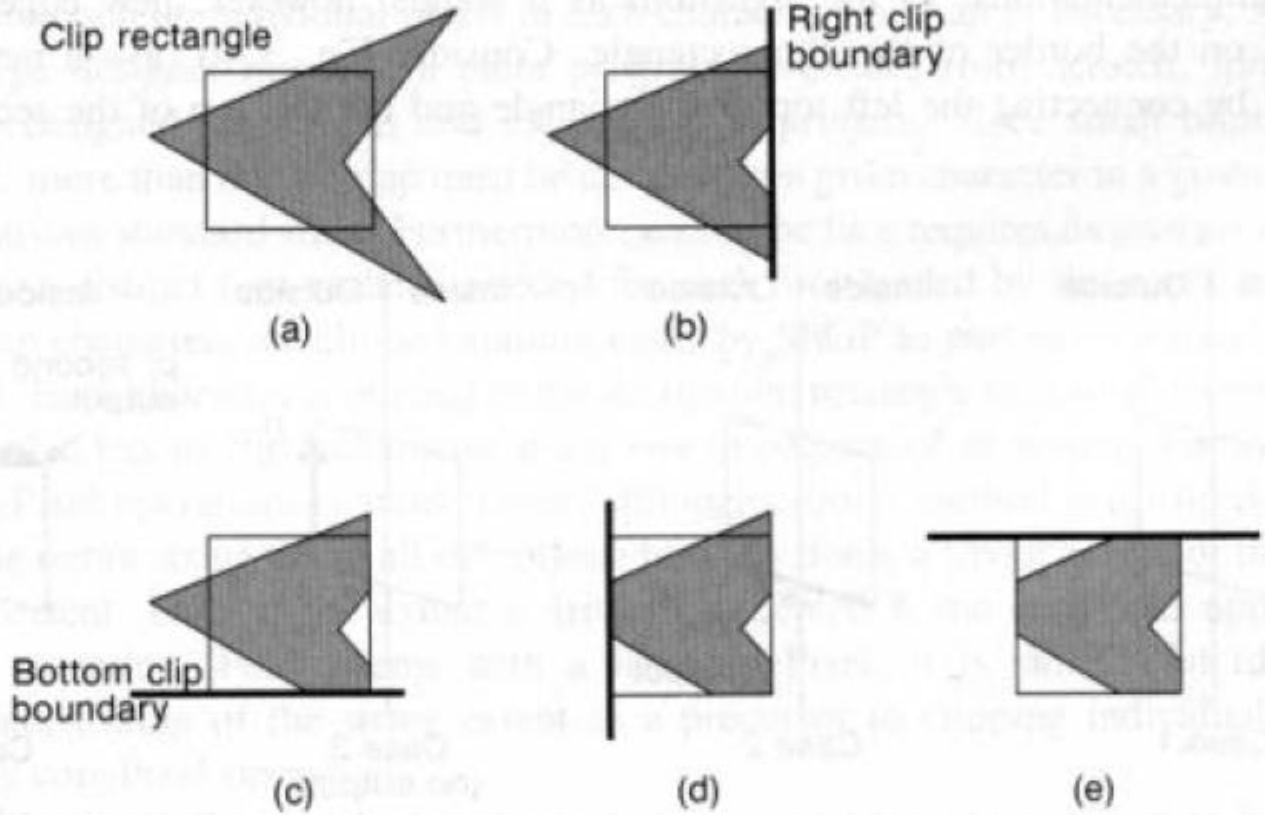


Fig. 3.47 Polygon clipping, edge by edge. (a) Before clipping. (b) Clip on right. (c) Clip on bottom. (d) Clip on left. (e) Clip on top; polygon is fully clipped.

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

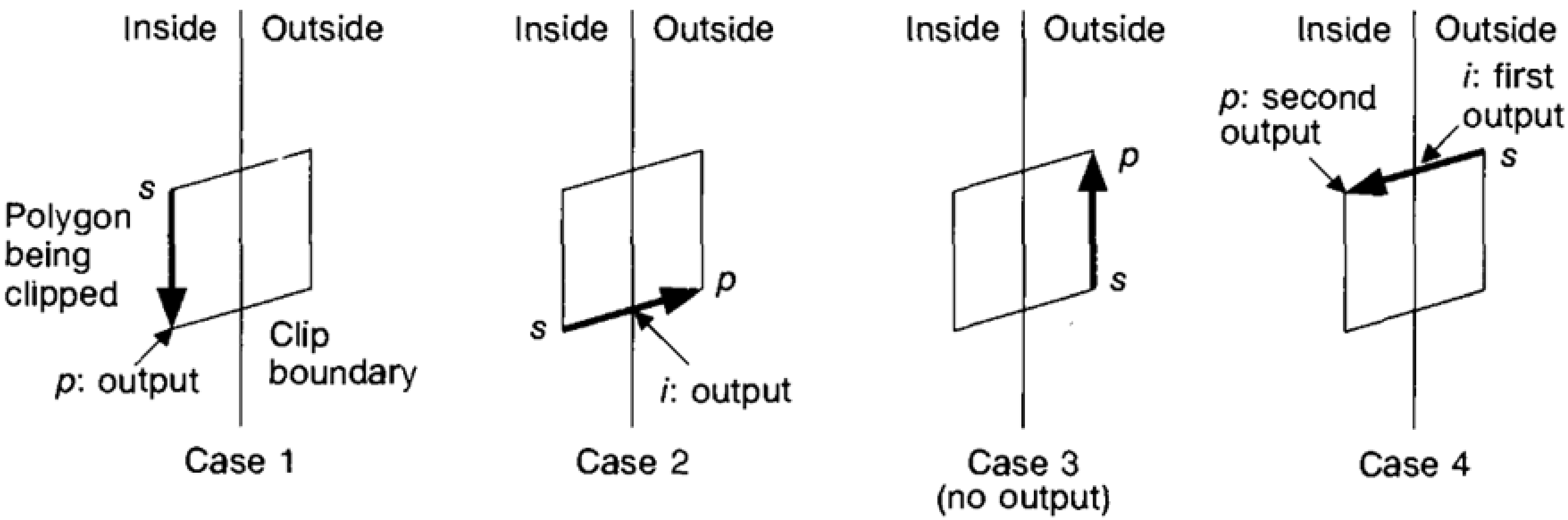


Fig. 3.48 Four cases of polygon clipping.

Sutherland-Hodgman Polygon Clipping

Input:

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

Algorithm:

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

For each clip edge e in E do

S □ v_n

P □ v_1

j □ 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

Sutherland-Hodgman Polygon Clipping

Algorithm:

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 □ v_j

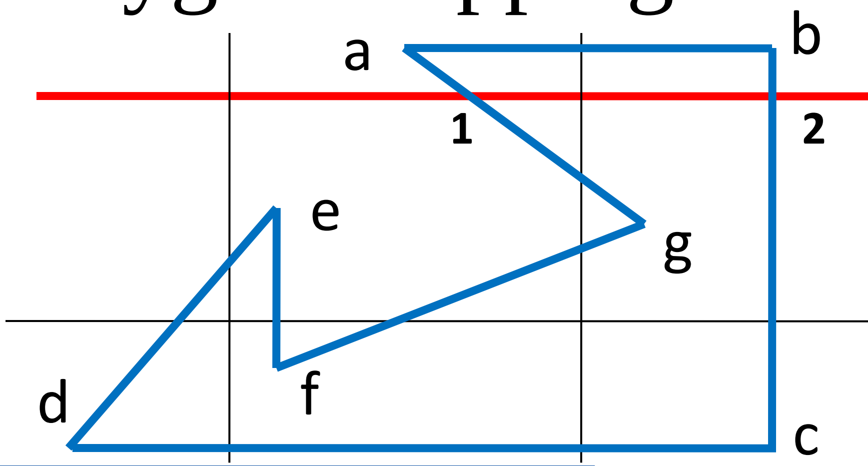
P □ v_{j+1}

i ++

inputlist □ current output list

Sutherland-Hodgman Polygon Clipping

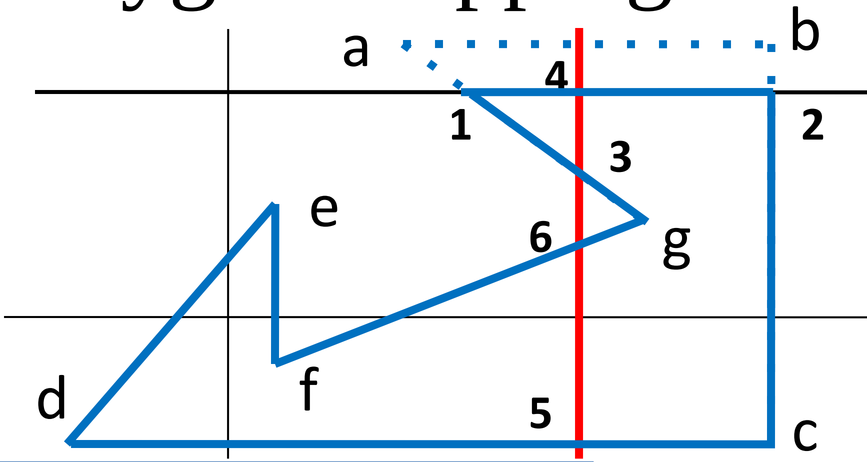
- a, b, c, d, e, f, g
- S = g, P = a
- Output: 1, 2, c, d, e, f, g



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

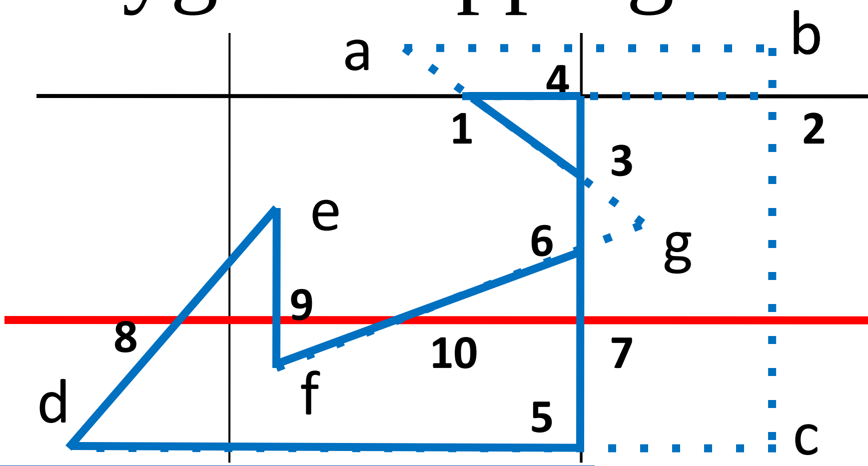
- Output of previous iteration
  1, 2, c, d, e, f, g
- S = g, P = 1
- Output: 3, 1, 4, 5, d, e, f, 6



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

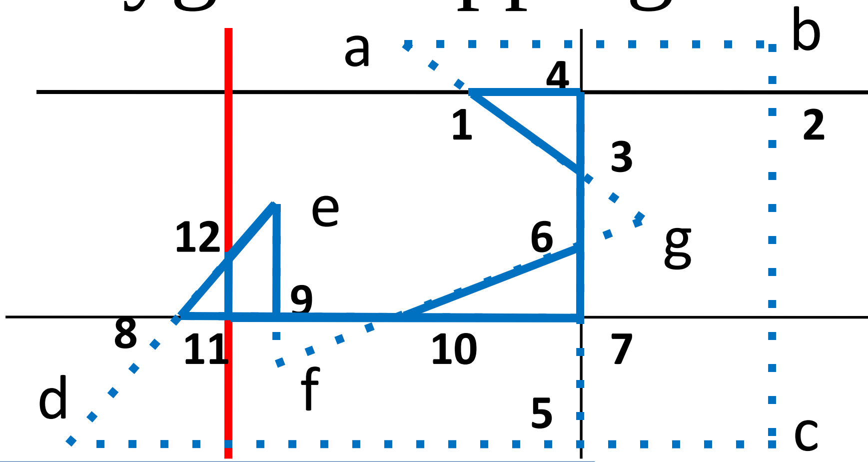
- Output of previous iteration
  3, 1, 4, 5, d, e, f, 6
- S = 6, P = 3
- Output: 3, 1, 4, 7, 8, e, 9, 10, 6



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!

Thank you ☺