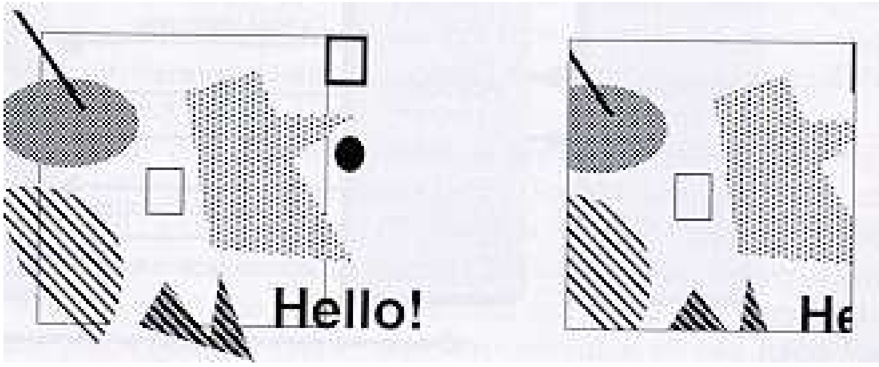
Clipping



- **We will discuss:**

  - Line clipping

  - Polygon clipping

  (Clipping window will be rectangle)

- **Approaches**

  - Perform clipping analytically first and then apply scan conversion

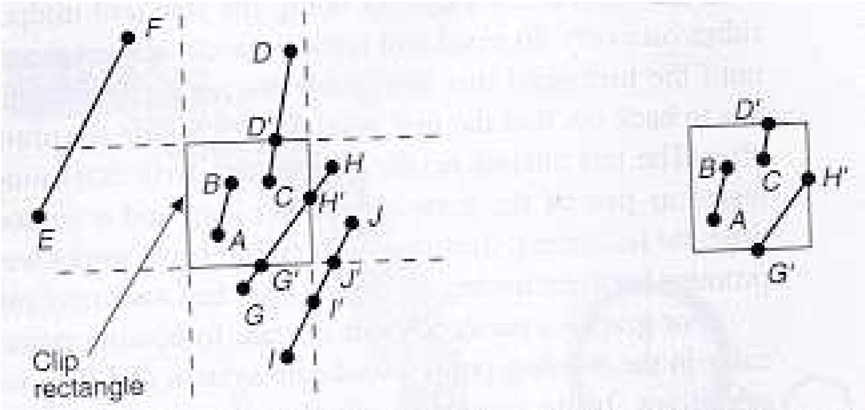
  - Perform clipping during scan conversion

- **When to use each approach**

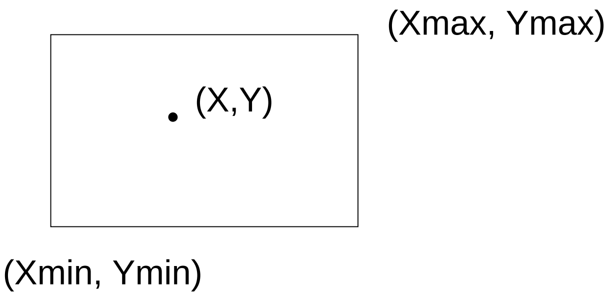
  - Allying clipping before scan conversion is best for simple primitives

  - For more complicated primitives, it is faster to clip during scan conversion

Clipping lines against rectangles



- **Clipping points**



$$x_{\min} \le x \le x_{\max},\ y_{\min} \le y \le y_{\max}$$

- **Clipping lines**

  - Consider only its endpoints !!

    i) Both endpoints are inside: entire line lies inside

    ii) One endpoint is inside and one outside: we must compute the intersection point

    iii) Both endpoints are outside: line may or may not intersect the clip rectangle

- **Brute-force approach**

  - Steps

    * If case (i) applies, the line is trivially accepted

    * If cases (ii) and (iii) apply, intersect the line with each of the four edges of the clip rectangle to see whether any intersection point lines on those edges

    * If there is at least one intersection point which lies on an edge of the clip rectangle, the line is partially inside

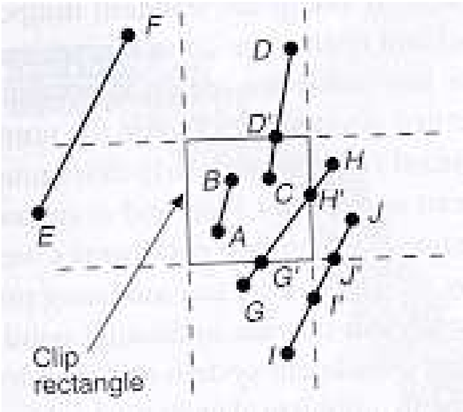
  - Not efficient: requires the solution of two equations/edge

- **Cohen-Sutherland algorithm**

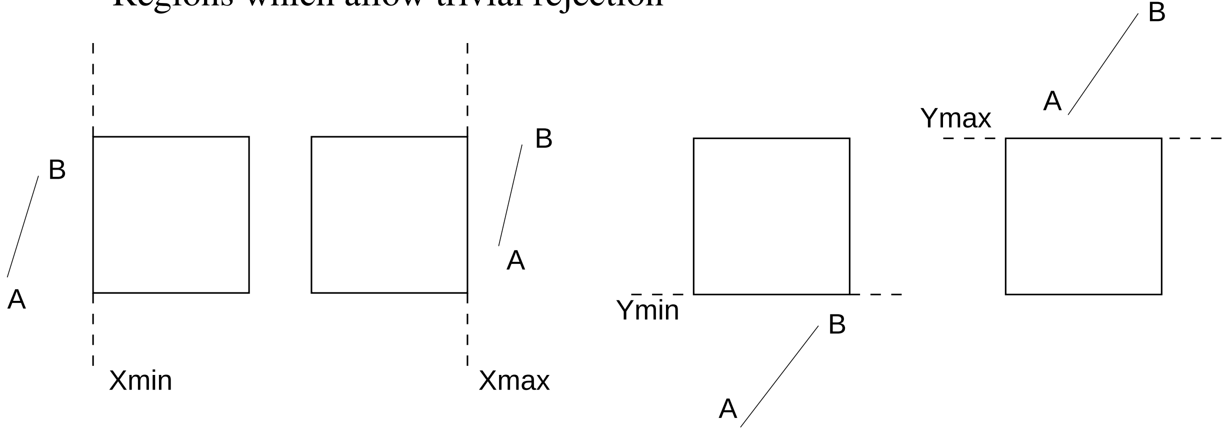
  - Idea: perform initial tests to trivially accept or trivially reject lines

    * Tests for trivial acceptance involve the endpoints of the line

    * Test for trivial rejection involve regions (e.g., EF)



  - Regions which allow trivial rejection

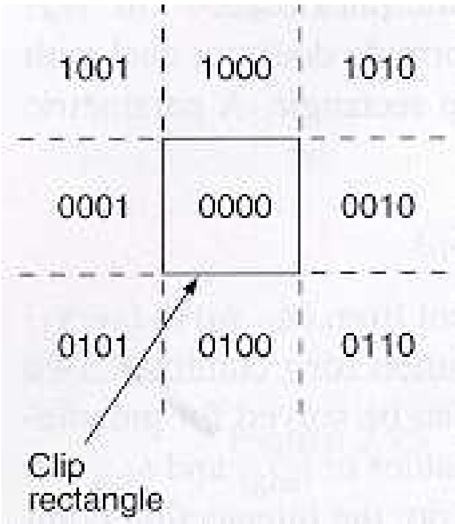


X1,X2 < Xmin   X1,X2 > Xmax   Y1,Y2 < Ymin   Y1,Y2 > Ymax

- Lines which cannot be trivially accepted or rejected, are iteratively clipped

- Iterative clipping: split the line into two segments such that at least one of them can be trivially rejected

- **Region outcodes**



- Divide the plane of the clip rectangle into 9 regions

- Assign a 4-bit code to each region

bit1: the point is above the top edge

bit2: the point is below the bottom edge

bit3: the point is to the right of the right edge

bit4: the point is to the left of the left edge

- **Computation of outcodes**

- bit1: sign bit of $y_{max} - y$

- bit2: sign bit of $y - y_{min}$

- bit3: sign bit of $x_{max} - x$

- bit4: sign bit of $x - x_{min}$

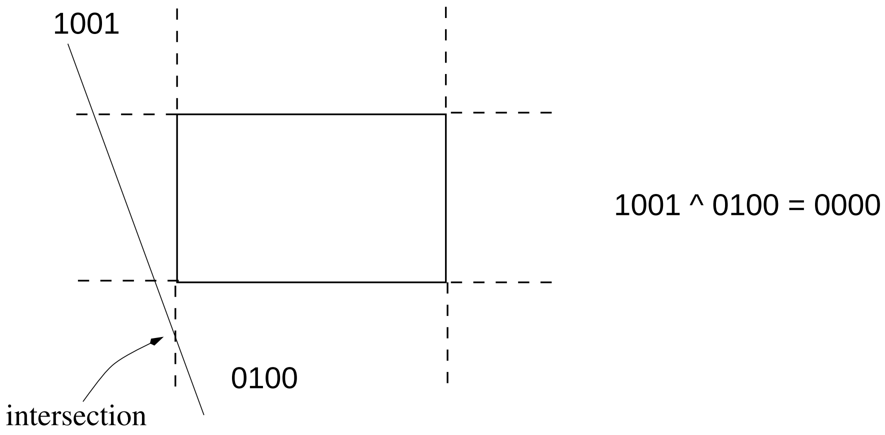
- **Acceptance and rejection tests using outcodes**

- Both outcodes are zero: the line is trivially accepted

- The logical AND of the outcodes is not zero: the line is trivially rejected

- The logical AND of the outcodes is zero: the line cannot be trivially rejected

(one point lies outside, one point lies inside: the segment intersects an edge !!)



- **Algoritm**

1. Compute the outcodes of both endpoints

2. Check for trivial acceptance and rejection

3. If neither test is successful (line intersects with clip window):

    3.1 Pick an end point that lies outside the clip window

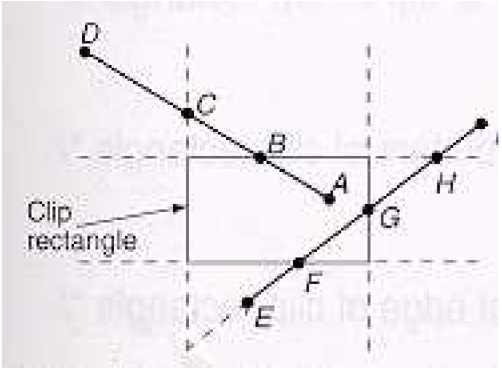
    3.2 Compute the intersection point

    (Edges are tested in the following order: top, down, right, left)

    (The non-zero bits in an outcode show the edges crossed !!)

4. Clip off the segment from the outside point to the intersection point and compute the outcode of the new endpoint

5. Goto step 2



- Line AD:

  * A's outcode: 0000, D's outcode: 1001

  * 0000 ^ 1001 = 0000 (no decision)

  * Choose D

  * Find intersection point B (B's outcode: 0000)

  * AB is accepted

- Line EI:

  * E's outcode: 0100, I's outcode: 1010

  * 0100 ^ 1010 = 0000 (no decision)

  * Choose E

  * Find intersection point F (F's outcode: 0000)

  * 0000 ^ 1010 = 0000 (no decision)

  * Choose I

  * Find intersection point H (H's outcode: 0010)

  * 0000 ^ 0010 = 0000 (no decision)

  * Choose H

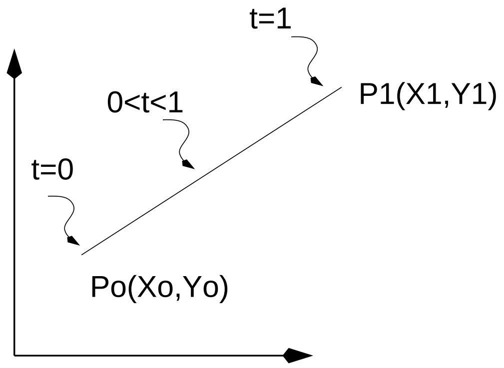
  * Find intersection point G (G's outcode: 0000)

  * FG is accepted

- Disadvantage: the algorithm will sometimes perform needless clipping due to the fact that testing and clipping are done in a fixed order

- **Parametric line clipping**

- Parametric representation of lines



$P(t) = P_0 + (P_1 - P_0)t, \; t \in [0,1]$

$t = 0$: $P(0) = P_0 + (P_1 - P_0)0 = P_0$

$t = 1$: $P(1) = P_0 + (P_1 - P_0)1 = P_1$

$x(t) = x_0 + (x_1 - x_0)t$

$y(t) = y_0 + (y_1 - y_0)t$

- The above equations describe a directed line segment

- Parametric equation vs slope-intercept equation

* Slope-intercept equation describes infinite lines

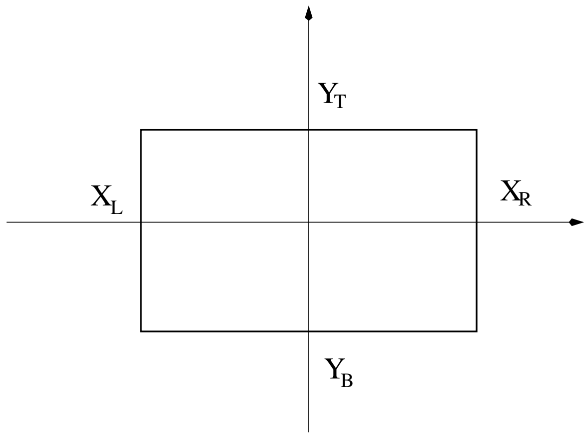
* Parametric equations describe segments (finite lines)

* Slope-intercept equation cannot deal with vertical lines

- What is the value of t for any point on the line segment ?

$$t = (P(t) - P_0)/(P_1 - P_0)$$

- **Computing intersection points using parametric line equation**



* Left edge: $t_L = (x_L - x_0)/(x_1 - x_0)$

$(x(t_L) = x_L,\ y(t_L) = y_0 + (y_1 - y_0)t_L)$

* Right edge: $t_R = (x_R - x_0)/(x_1 - x_0)$

$(x(t_R) = x_R,\ y(t_R) = y_0 + (y_1 - y_0)t_R)$

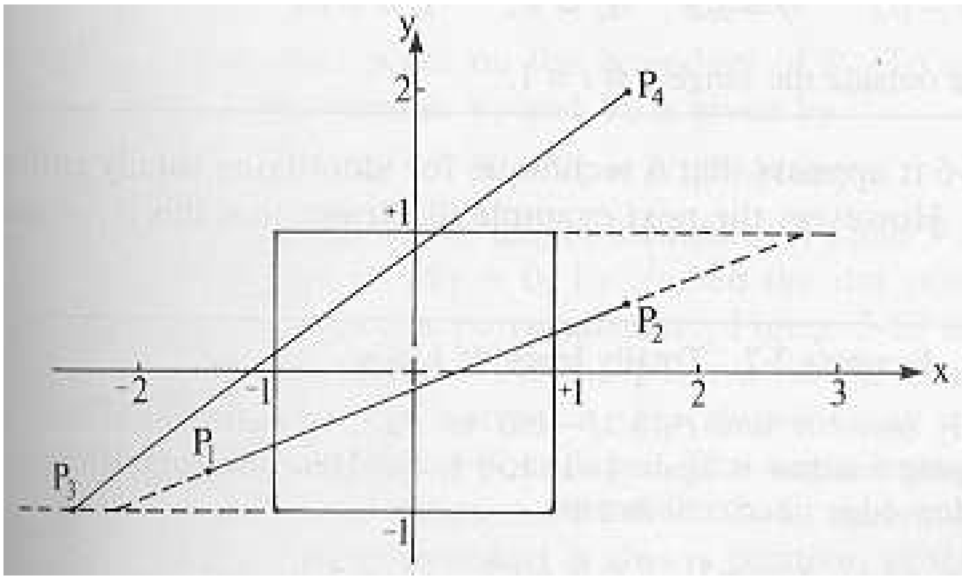
* Top edge: $t_T = (y_T - y_0)/(y_1 - y_0)$

$(x(t_T) = x_0 + (x_1 - x_0)t_T,\ y(t_T) = y_T)$

* Bottom edge: $t_B = (y_B - y_0)/(y_1 - y_0)$

$(x(t_B) = x_0 + (x_1 - x_0)t_B,\ y(t_B) = y_B)$

- Example



$$P_1 = (-3/2, -3/4),\ P_2 = (3/2, 1/2),\ P_3 = (-5/2, -1),\ P_4 = (3/2, 2)$$

* P_1P_2

$t_L = (-1 - (-3/2))/(3/2 - (-3/2)) = 1/6,\ x(1/6) = -1,\ y(-1/6) = -13/24$

$t_R = (1 - (-3/2))/(3/2 - (-3/2)) = 5/6,\ x(5/6) = 1,\ y(5/6) = 7/24$

$t_T = (1 - (-3/4))/(1/2 - (-3/4)) = 7/5 > 1$!!

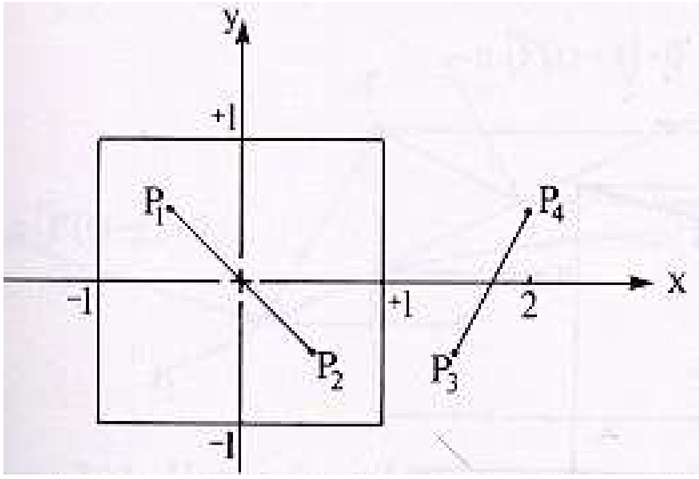
$t_B = (-1 - (-3/4))/(1/2 - (-3/4)) = -1/5 < 0$!!

* P_3P_4

$t_L = 3/8,\ t_R = 7/8,\ t_B = 0,\ t_T = 2/3$, all t values $\in [0,1]$

Further testing is needed, for example, sort t values and choose intermediate values

- Example



$$P_1 = (-1/2, 1/2),\ P_2 = (1/2, -1/2),\ P_3 = (3/2, -1/2),\ P_4 = (2, 1/2)$$

* P_1P_2

$t_L = -1/2,\ t_R = 3/2,\ t_B = 3/2,\ t_T = -1/2,\ t \notin [0,1]$

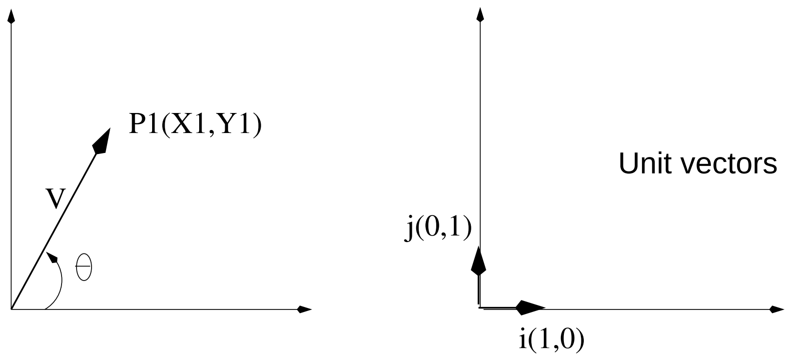
* P_3P_4

$t_L = -5,\ t_R = -1,\ t_B = -1/2,\ t_T = 3/2$, none of $t \in [0,1]$

<u>Review on vectors</u>

- **Vector**

- A directed line with two attributes: *magnitude* and *direction*



representation: $v = (v_x, v_y) = (x_1, y_1)$ (note: P_0=(0,0))

magnitude: $\|v\| = \sqrt{{v_x}^2 + {v_y}^2}$, direction: $\theta = \tan^{-1}(v_y/v_x)$

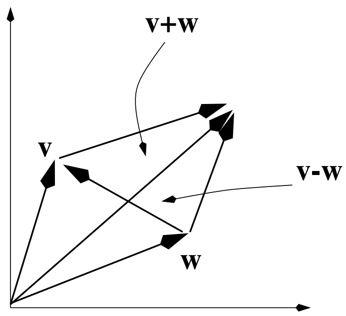
- **Unit vectors**

  - Unit vector in the x direction: i=(1,0)

  - Unit vector in the y direction: j=(0,1)

  - Vector normalization: $v/\|v\| = (v_x/\|v\|, v_y/\|v\|)$

- **Properties**



  - Addition: $v + w = (v_x, v_y) + (w_x, w_y) = (v_x + w_x, v_y + w_y)$

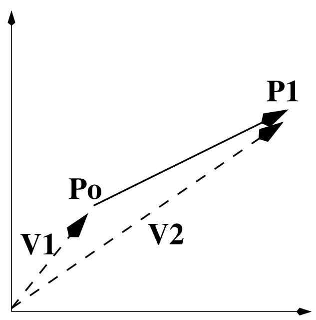
  - Subtraction: $v - w = (v_x, v_y) - (w_x, w_y) = (v_x - w_x, v_y - w_y)$

  - Scalar multiplication: $av = (av_x, av_y)$

- **Every vector can be expressed in terms of *i* and *j***

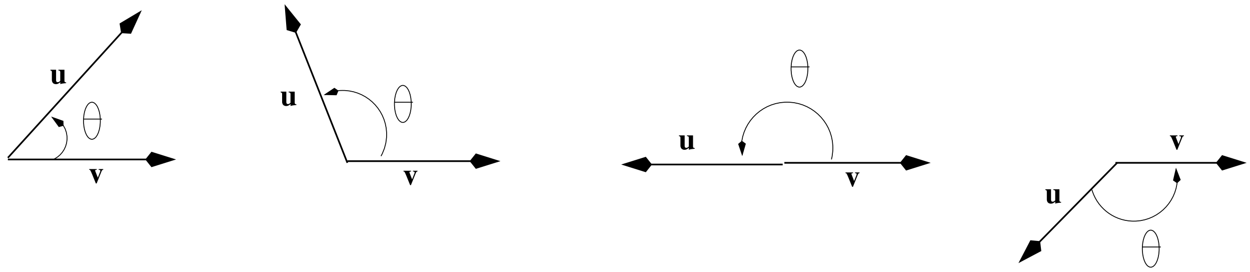
  $$V = (v_x, v_y) = (v_x, 0) + (0, v_y) = v_x(1,0) + v_y(0,1) = v_x i + v_y j$$

- **Representation of a vector with $P_0 \neq (0,0)$**



$v_1 + P_0P_1 = v_2$ **or** $P_0P_1 = v_2 - v_1$ **or** $P_0P_1 = (x_1 - x_0, y_1 - y_0)$

- **Dot product between two vectors**



$u.v = \|u\|\ \|v\|\ cos(\theta)$, $0 \le \theta \le \pi$ (θ is the smaller angle between u,v)

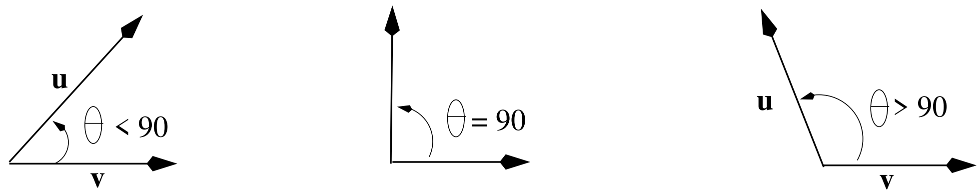
Alternatively: $u.v = u_x v_x + u_y v_y$

- **Properties of the dot product**

- $v.v = 0$ only when $v = 0$

- $v.(u + aw) = v.u + a(v.w)$

- The sign of $u.v$ depends on the $cos(\theta)$

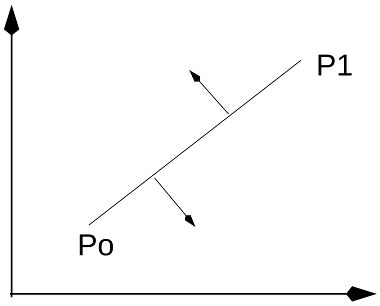


cos (θ) > 0   cos (θ) = 0   cos (θ) < 0

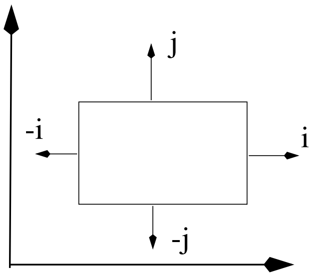
u . v > 0   u . v = 0   u . v < 0

- **Normal of a line**

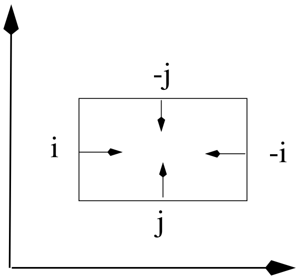
- A unit vector, perpendicular to the line



- **Normal vectors - clip edges**

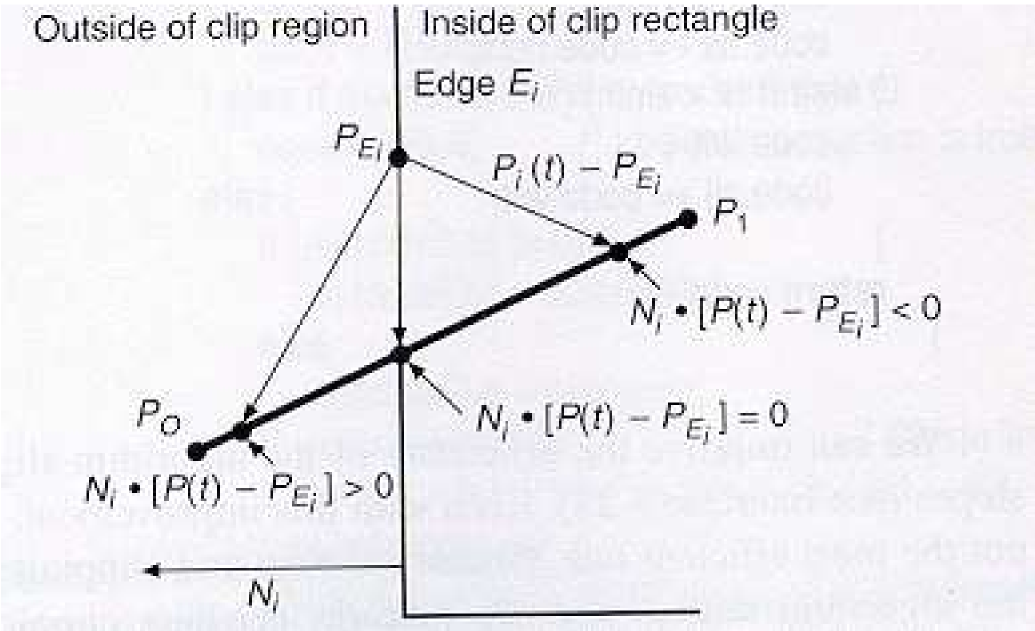


outward normals



inward normals

- **Cyrus-Beck algorithm** (uses outward normals)



- Cases:

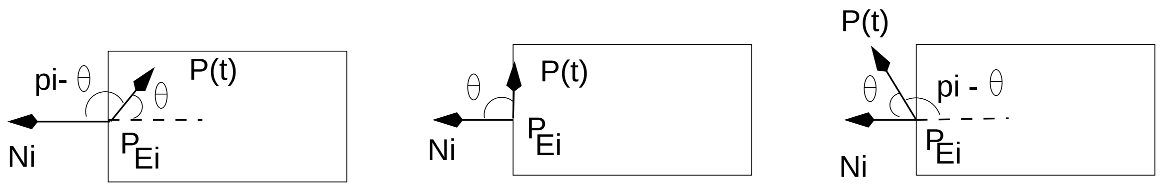
Consider P_2 and vector $P_2(t) - P_{E_i}$: $N_i.[P_2(t) - P_{E_i}] < 0$

Consider P_3 and vector $P_3(t) - P_{E_i}$: $N_i.[P_3(t) - P_{E_i}] = 0$

Consider P_4 and vector $P_4(t) - P_{E_i}$: $N_i.[P_4(t) - P_{E_i}] > 0$

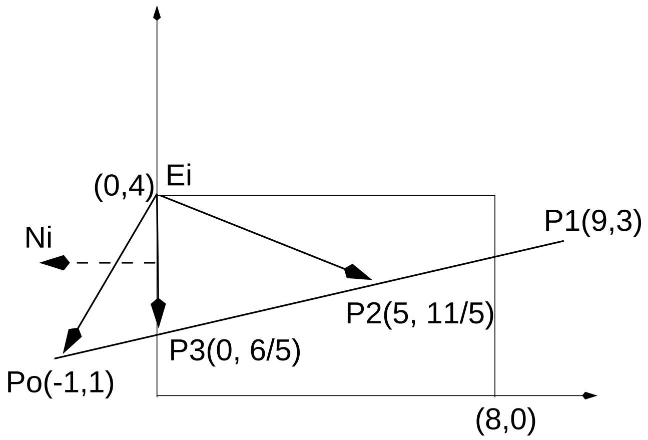
- Why ?

P(t) is inside | P(t) lies on Ei | P(t) is outside



P(t) is inside	P(t) lies on Ei	P(t) is outside
$-pi/2 < \theta < pi/2$	$\theta = pi/2$, $\cos\theta = 0$	$-pi/2 < \theta < pi/2$
$\cos\theta > 0$, $\cos(pi - \theta) = -\cos\theta < 0$		$\cos\theta > 0$
$\boxed{Ni \,.\, [\,P(t) - P_{Ei}] < 0}$	$\boxed{Ni \,.\, [P(t) - P_{Ei}] = 0}$	$\boxed{Ni \,.\, [P(t) - P_{Ei}] > 0}$

- Example



$N_i = -i$ ((-1,0))

$$P_0 - P_{E_i} = (-i + j) - (4j) = -i - 3j \quad ((-1, -3))$$
$$N_i.\,[P_0 - P_{E_i}] = (-i).\,(-i - 3j) = (-1)(-1) + 0(-3) = 1 > 0$$

$$P_3 - P_{E_i} = ((6/5)j) - (4j) = -(14/5)j$$
$$N_i.\,[P_3 - P_{E_i}] = (-i).\,(-(14/5)j) = (-1)0 + 0(-14/5) = 0$$

$$P_2 - P_{E_i} = (5i + (11/5)j) - (4j) = 5i - (9/5)j$$
$$N_i.\,[P_2 - P_{E_i}] = (-i).\,(5i - (9/5)j) = (-1)5 + 0(-9/5) = -5 < 0$$

- **Find t at the intersection point**

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

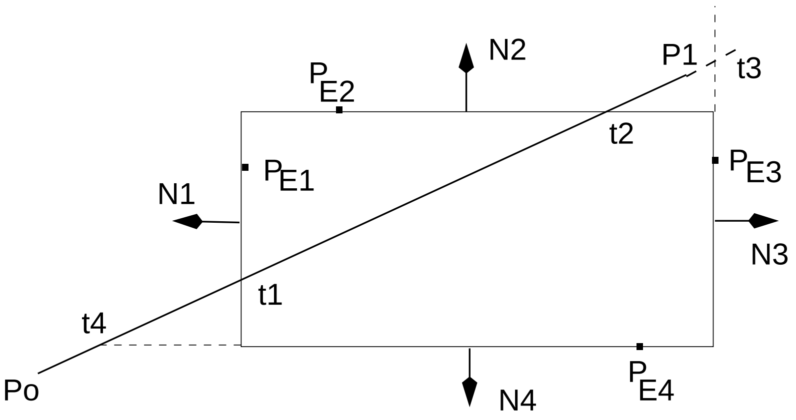
$N_i . [P_0 + (P_1 - P_0)t - P_{E_i}] = 0$ or $N_i . [P_0 - P_{E_i}] + N_i . t[P_1 - P_0] = 0$

$$t = N_i . [P_0 - P_{E_i}]/(-N_i . D) \text{ , where } D = [P_1 - P_0]$$

- **Algorithm**

- Determine normals and a point P_{E_i} for each clip edge

- Find intersections (four values of t) between P_0P_1 and the clip rectangle



(P_{Ei} is usually an endpoint)

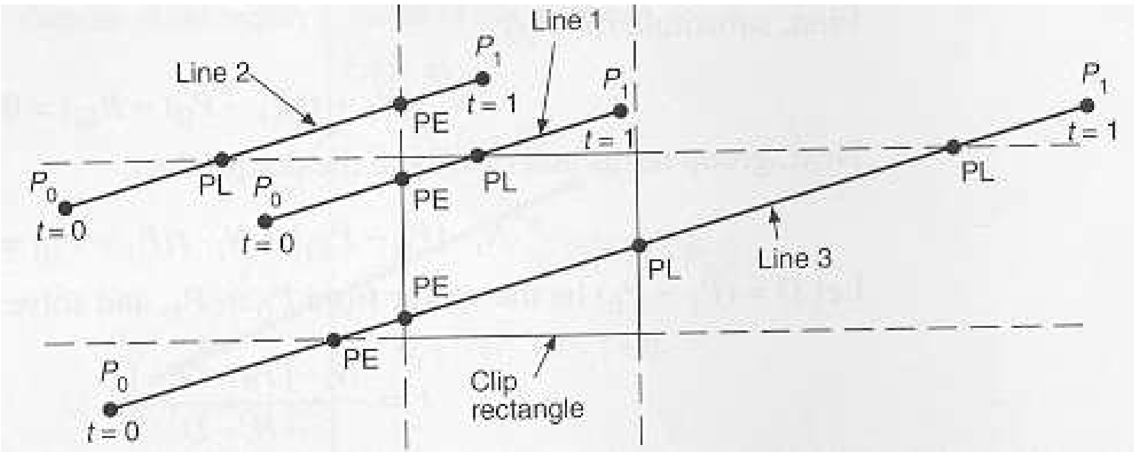
* If $t \notin [0,1]$, then reject this t value (for example, t_3)

* If $t \in [0,1]$, then characterize the intersections as:

<u>Potentially entering (PE):</u>
if $N_i . D < 0$, (angle between P_0P_1 and $N_i > 90$)

or

<u>Potentially leaving (PL):</u>
if $N_i . D > 0$, ==> PL (angle between P_0P_1 and $N_i < 90$)



- The portion of the line which is visible is bounded by (t_E, t_L):

  $t_E = \max_i \{t_i: t_i$ corresponds to a PE point$\}$

  $t_L = \min_i \{t_i: t_i$ corresponds to a PL point$\}$

  (note that $t_E \geq 0$, $t_L \leq 1$, and $t_E \leq t_L$)

- **Advantages of the parametric line clipping**

  - Good in cases where most lines cannot be trivially rejected or accepted

  - The (x, y) coordinates of an intersection point are calculated only if the corresponding $t \in [0,1]$ and $t = t_E$ or $t = t_L$

  - Works for any convex clip window (not for concave though !!)

precalculate N_i and select a P_{E_i} for each edge.

```
for(each line segment to be clipped) {
  if(P_1=P_0)
    line is degenerate so clip as a point;
  else {
   t_E = 0;
   t_L = 1;
   for(each candidate intersection with a clip edge) {
     if(N_i . D != 0) {
       calculate t;
       use sign of N_i . 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 nil;
   else
     return P(t_E) and P(t_L) as true clip intersections;
 }
}
```