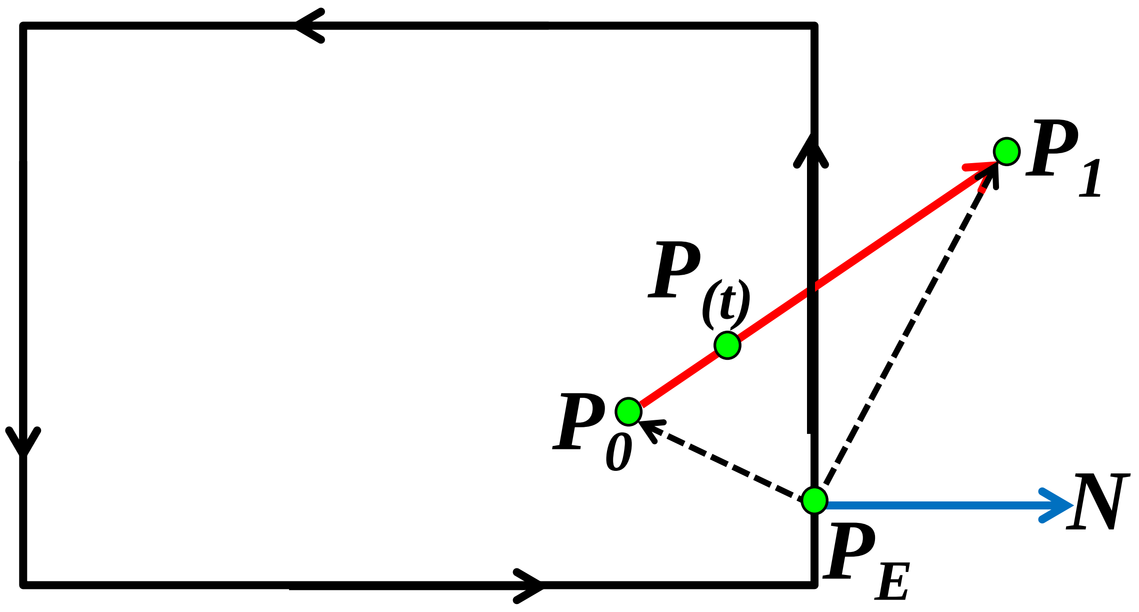
Solving Simultaneous Equations

- Equation of a line:
  - Slope-intercept: $y = mx + c$
  - Implicit Equation: $Ax + By + C = 0$
  - Parametric: Line defined by two points, P_0 and P_1
    - $P(t) = P_0 + (P_1 - P_0)t$
    - $x(t) = x_0 + (x_1 - x_0)t$
    - $y(t) = x_0 + (y_1 - y_0)t$

At $t = 0 \rightarrow P_{(t)} = P_0$, at $t = 1 \rightarrow P_{(t)} = P_1$ and if t is known $P_{(t)} \rightarrow (x_{(t)}, y_{(t)})$

Parametric Lines and Intersections



Position of a point:

$(P_{(t)} - P_E).N > 0$ means point $P_{(t)}$ is outside

$(P_{(t)} - P_E).N < 0$ means point $P_{(t)}$ is inside

$(P_{(t)} - P_E).N = 0$ means point $P_{(t)}$ on the edge

Line Entering or Leaving:

$(P_1 - P_0).N > 0$ means intersecting line is leaving

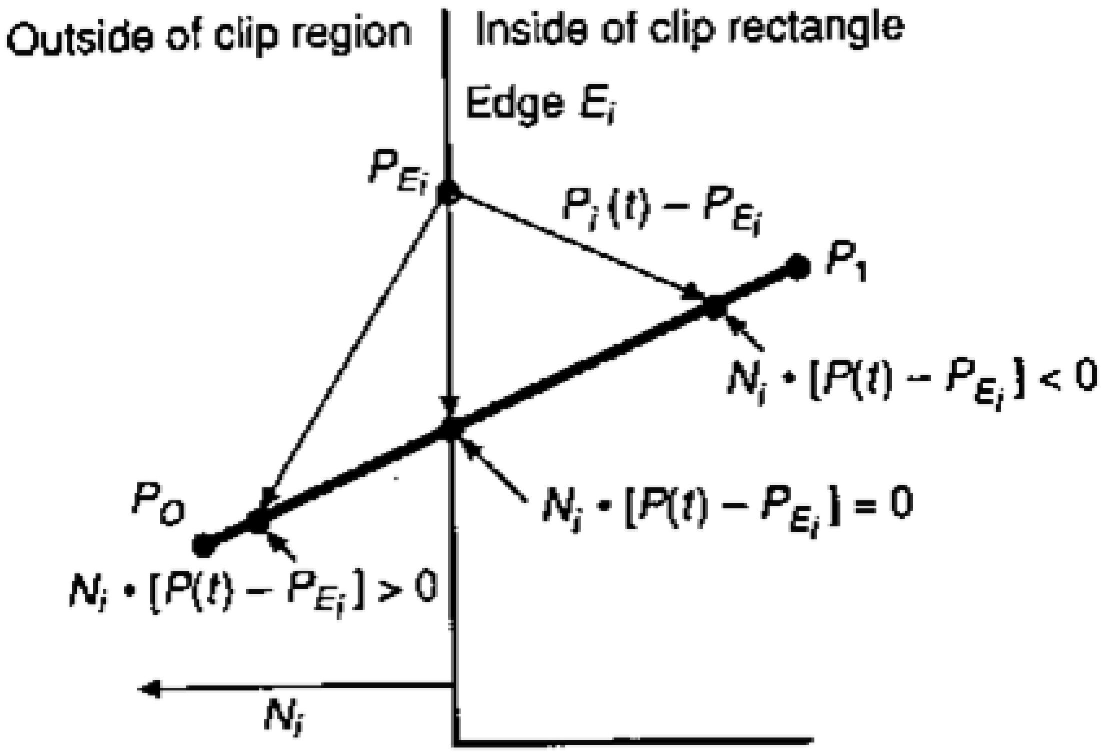
$(P_1 - P_0).N < 0$ means intersecting line is Entering

$(P_1 - P_0).N = 0$ line is parallel to the edge

Cyrus-Beck Algorithm

- Introduced by Cyrud and Beck in 1978
- Efficiently improved by Liang and Barsky
- Essentially find the parameter t from $P(t) = P_0 + (P_1 - P_0)t$
- The conditions
  - (1)Position of a point on the line and
  - (2)Line Entering or Leaving,
- shown in the previous slide are properly utilized in Cyrus-Beak Line Clipping Algorithm.

Cyrus-Beck Algorithm



Intersection:

Line equation: $P_{(t)} = P_0 + t(P_1 - P_0)$ ... (1)

When $P(t)$ intersects boundary

$$(P_{(t)} - P_E) \bullet N = 0 \quad \text{... (2)}$$

Substitute line in $Eq.(2)$:

$$(P_0 + t(P_1 - P_0) - P_E) \bullet N = 0$$

$$\rightarrow (P_0 - P_E) \bullet N + t(P_1 - P_0)) \bullet N = 0$$

[Since $(P_0 - P_E)$ and $(P_1 - P_0)$ are vectors, the above eq. can be written]

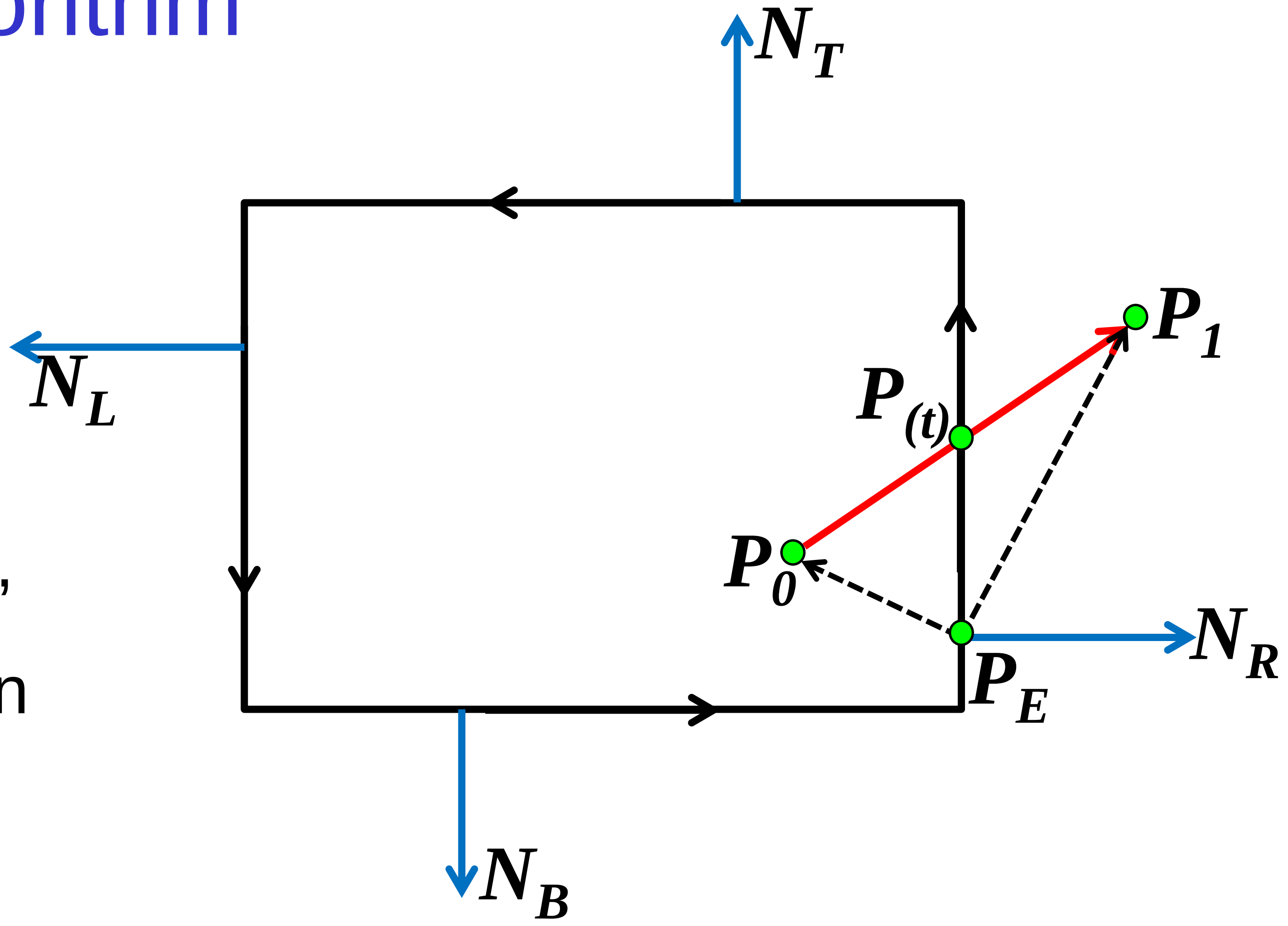
Solving for t:

$$\rightarrow t = -\frac{(P_0 - P_E)\cdot N}{(P_1 - P_0)\cdot N}$$

Cyrus-Beck Algorithm

$$t = -\frac{(P_0 - P_E)\cdot N}{(P_1 - P_0)\cdot N}$$

- The value of t is N dependent,
- For the same line the equation for t is different for different edges/boundaries.
- A list of t for all edges are given in the next slide



List of t for all Edges

Edge	***Normal***	P_E	$(P_0 - P_E).N$	$t = -\frac{N \cdot (P_0 - P_E)}{N \cdot (P_1 - P_0)}$
Left x=*Xmin*	$(-1,\ 0)$	$(Xmin,\ y)$	$-(x_0 - X_{min})$	$\frac{(X_{min} - x_0)}{(x_1 - x_0)}$
Right x=*Xmax*	$(1,\ 0)$	$(Xmax,\ y)$	$(x_0 - X_{max})$	$\frac{(X_{max} - x_0)}{(x_1 - x_0)}$
Bottom y=*Ymin*	$(0,\ -1)$	$(x,\ Ymin)$	$-(y_0 - Y_{min})$	$\frac{(Y_{min} - y_0)}{(y_1 - y_0)}$
Top y=*Ymax*	$(0,\ 1)$	$(x,\ Ymax)$	$(y_0 - Y_{max})$	$\frac{(Y_{max} - y_0)}{y_1 - y_0}$

Cyrus-Beck Algorithm

- Formally, intersections can be classified as
  - P_{Ent} (potentially entering) if $(P_1 - P_0).N < 0$ and
  - P_{Leav} (potentially leaving) if if $(P_1 - P_0).N > 0.$
- Similarly,
  - $t = t_E$ (potentially entering) if $(P_1 - P_0).N < 0$ and
  - $t = t_L$ (potentially leaving) if if $(P_1 - P_0).N > 0.$
- Determine t_E or t_L for all intersections
- Select the line segment that has maximum t_E (t_{Emax}) and minimum t_L (t_{Lmin})
- If $t_{Emax} > t_{Lmin}$, then trivially rejected

Algorithm

- Initialize t_{Emax} *as 0.0 and* t_{Lmin} *as 1.0*
- Compute *t* for line intersection with all edges;
- Discard all ($t < 0$) and ($t > 1$);
- Classify t for each remaining intersection as
  - Potentially Entering Line (t_E)
  - Potentially Leaving Line (t_L)
  - Find the maximum of t_{Emax} and minimum of t_{Lmin}
- IF($t_{Emax} > t_{Lmin}$):
- Line is outside the window (Rejected)
- Else:
- The line is from $P_{(tE)}$ to $P_{(tL)}$

Programming:

$t_{Emax}, t_{Lmin} = 1, 0$
for (i edges of clipping window):
    solve $N_i \cdot (P_1 - P_0)$
    solve $N_i \cdot (P_0 - Pi)$
    if ($N_i \cdot (P_1 - P_0) == 0$): #parallel to the edge
        go to next edge
    else:
        solve t_i
            if($N_i \cdot (P_1 - P_0) > 0$): #leaving t_L
                if($ti < t_{Lmin}$):
                    $t_{Lmin} = t_i$
            else: #entering t_E
                if($ti > t_{Emax}$):
                    $t_{Emax} = t_i$

Output:
if ($t_{Emax} > t_{Lmin}$): # outside the window
    return nil;
else:
    return
    $P_0 = P(t_{Emax})$ and $P_1 = P(t_{Lmin})$
    as the true clip intersections or new endpoints afterclipping;

Example:

Determine the coordinate of the end-points after clipping.