Liang-Barsky Line Clipping Algorithm

Computer Graphics

Liang-Barsky Line Clipping

- Line clipping approach is given by the Liang and Barsky is faster then cohen-sutherland line clipping.
- Which is based on analysis of the **parametric equation of the line** which are,

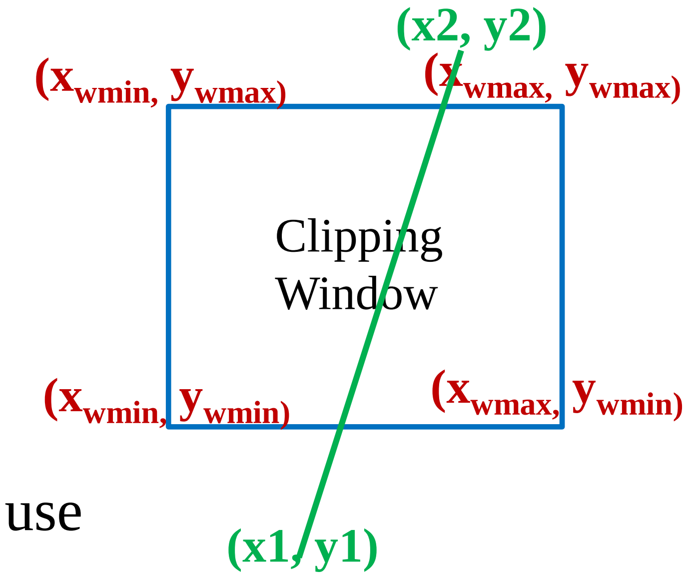
$$x = x_1 + t\Delta x$$

$$y = y_1 + t\Delta y$$

Where $0 \le t \le 1$ **,** $\Delta x = x_2 - x_1$ **and** $\Delta y = y_2 - y_1$.

if t=0 then x=x1,y=y1(starting point)

If t=1 then x=x2,y=y2(ending point)



- From point clipping we already know;
- To find weather given point is inside or outside the clipping window we use following inequality:

$$x_{wmin} \le x \le x_{wmax}, \qquad y_{wmin} \le y \le y_{wmax}$$

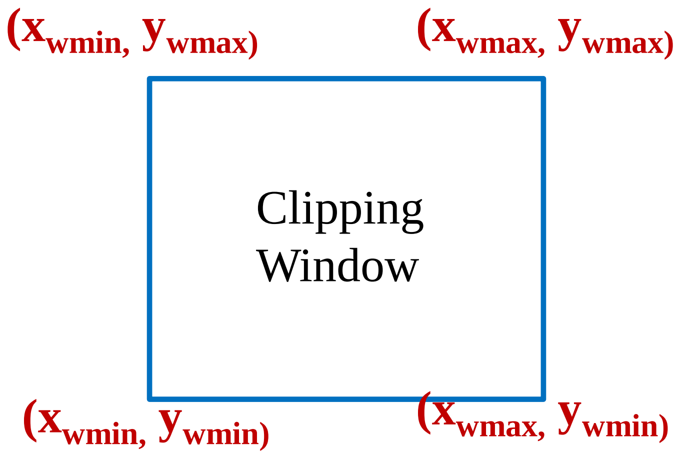
- $x_{wmin} \le x_1 + t\Delta x \le x_{wmax}, y_{wmin} \le y_1 + t\Delta y \le y_{wmax}$

Liang-Barsky Line Clipping

- $x_{wmin} \leq x_1 + t\Delta x \leq x_{wmax}, y_{wmin} \leq y_1 + t\Delta y \leq y_{wmax}$

Can be written as:

- $t\Delta x \geq x_{wmin} - x_1,$ *left*
- $t\Delta x \leq x_{wmax} - x_1,$ *right*
- $t\Delta y \geq y_{wmin} - y_1,$ *bottom*
- $t\Delta y \leq y_{wmax} - y_1,$ *top*



Represent all equations in ≤ form

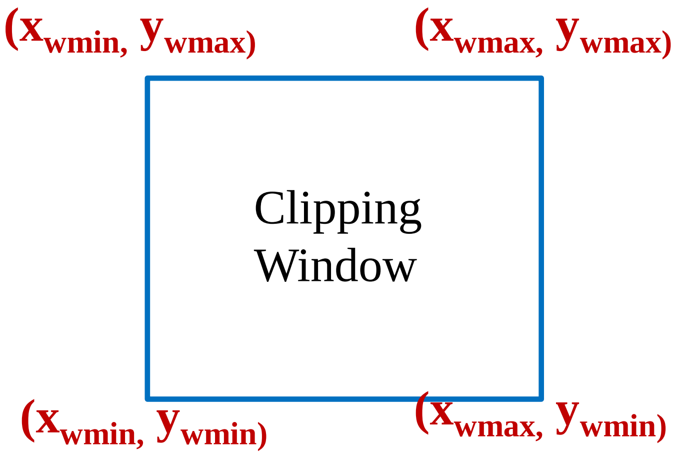
- $-t\Delta x \leq x_1 - x_{wmin}$
- $t\Delta x \leq x_{wmax} - x_1$
- $-t\Delta y \leq y_1 - y_{wmin}$
- $t\Delta y \leq y_{wmax} - y_1$

Liang-Barsky Line Clipping

- $-t\Delta x \leq x_1 - x_{wmin}$
- $t\Delta x \leq x_{wmax} - x_1$
- $-t\Delta y \leq y_1 - y_{wmin}$
- $t\Delta y \leq y_{wmax} - y_1$

General form for equations is: $t^*p_k \leq q_k$, where k=1,2,3,4(left, right, bottom & top edge)

- $p_1 = -\Delta x$, $q_1 = x_1 - x_{wmin}$, $t = q_1/p_1$
- $p_2 = \Delta x$, $q_2 = x_{wmax} - x_1$, $t = q_2/p_2$
- $p_3 = -\Delta y$, $q_3 = y_1 - y_{wmin}$, $t = q_3/p_3$
- $p_4 = \Delta y$, $q_4 = y_{wmax} - y_1$, $t = q_4/p_4$



Liang-Barsky Line Clipping Algorithm

Parametric definition of line

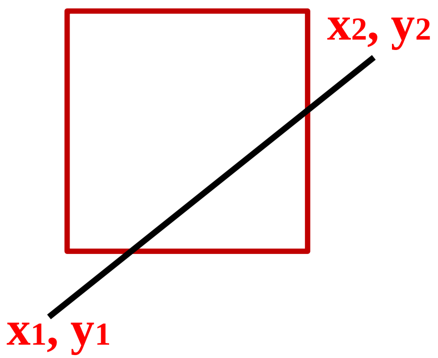
$x_{min} \leq x_1 + t\,\Delta x \leq x_{max}$

$y_{min} \leq y_1 + t\,\Delta y \leq y_{max}$

$\Delta x = x_2 - x_1$

$\Delta y = y_2 - y_1$

$0 \leq t \leq 1$



Initializations

- Set viewport points as x_{min}, x_{max}, y_{min} and y_{max}
- Set the line with 2 points $A(x_1,y_1)$ and $B(x_2,y_2)$
- Set line intersection parameters t_{entry} $(t_1) = 0.0$ and $t_{leaving}$ $(t_2) = 1.0$

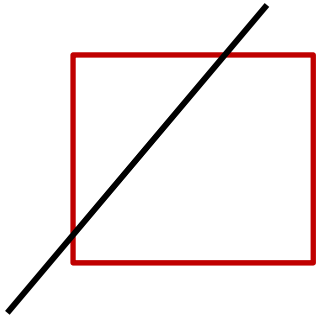
Calculations

- Find $dx = x_2 - x_1$ and $dy = y_2 - y_1$
- Update t_1 or t_2 depending upon dx or dy

Liang-Barsky Line Clipping Algorithm

Step 1 :

Set line intersection parameters t_{entry} (t_1) = 0.0 and $t_{leaving}$ (t_2) = 1.0



Step 2 :

Obtain p_k and q_k for k =1: 4

Edge & K value	p	q	t
Left **k=1**	p1 = -dx	q1 = x0 - xwmin	t = q1/p1
Right **k=2**	p2 = dx	q2 = xwmax - x0	t = q2/p2
Bottom **k=3**	p3 = -dy	q3 = y0 - ywmin	t = q3/p3
Top **k=4**	p4 = dy	q4= ywmax - y0	t = q4/p4

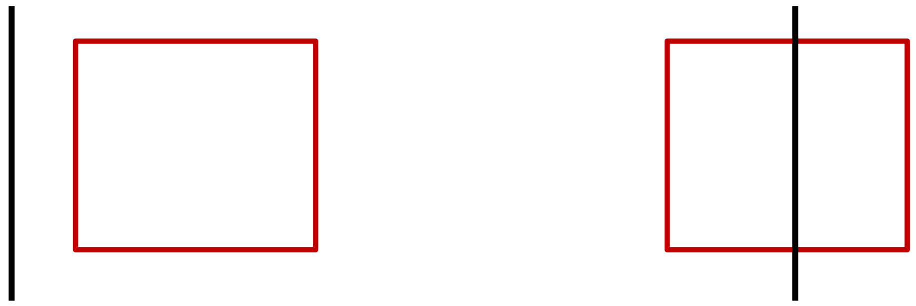
Liang-Barsky Line Clipping Algorithm

Step 3 :

If pk = 0, the line is parallel to the corresponding clipping boundary.

a) If pk = 0 and qk<0, the line is completely outside the boundary.

b) If pk = 0 and qk>=0, the line is inside the parallel clipping boundary.



(a) **(b)**

Liang-Barsky Line Clipping Algorithm

Step 4 :

Using p_k and q_k ($k = 1:4$), find if the line can be rejected or intersection parameters (t_1 and t_2) must be adjusted.

- if $p_k < 0$, update t_1 as ***max***$[0, q_k/p_k]$ where $k = 1:4$
- if $p_k > 0$, update t_2 as ***min***$[1, q_k/p_k]$ where $k = 1:4$

After the update,

- If $t_1 > t_2$, reject the line
- If $t_1 > 0$, calculate new values of x1, y1
- If $t_2 < 1$, calculate new values of x2, y2

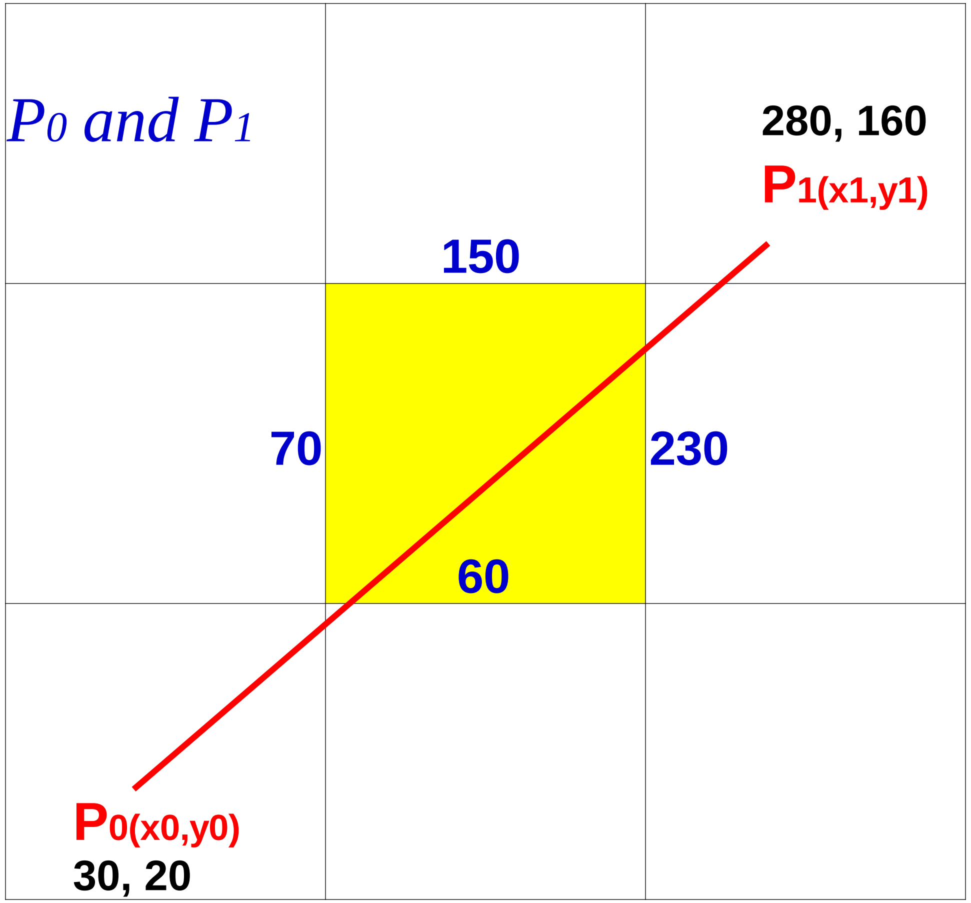
Liang-Barsky Line Clipping - Example

$dx = P_{1}x - P_{0}x$ *// Horizontal diff between P_0 and P_1*

$dx = x_1 - x_0 = 280-30 = 250$

$dy = P_{1}y - P_{0}y$ *// Vertical diff between P_0 and P_1*

$dy = y_1 - y_0 = 160-20 = 140$



dx = 250 , dy = 140

Initially

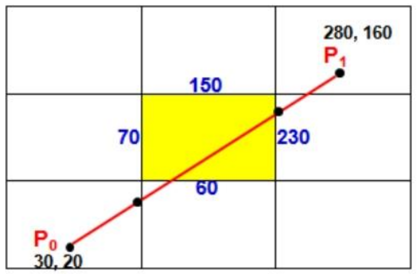
t_{entry} (t1) = 0.0

$t_{leaving}$(t2) = 1.0

dx = 250

dy = 140

Liang-Barsky Line Clipping - Example



Left edge check:

p = -dx = -250

q = x0 - xmin = 30-70 = -40

t = q/p = 0.16

- **If p< 0** *// update tentry (t1)*

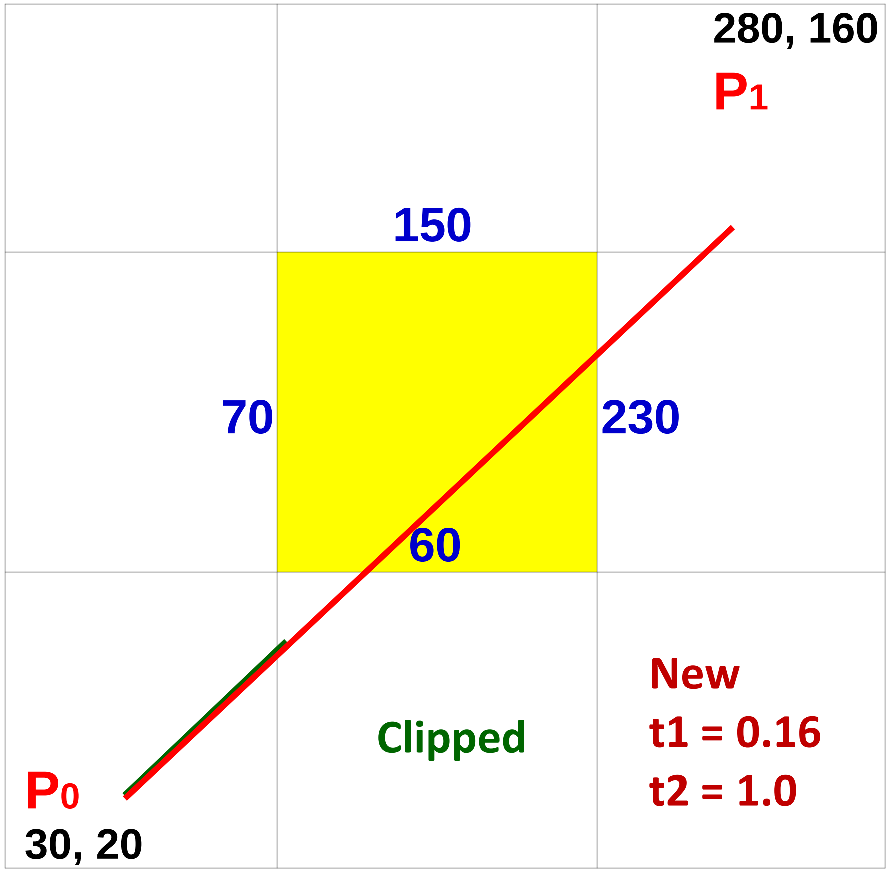
  if t > t1 set t1 = t

-250 < 0 TRUE

check if t > t1 --> 0.16 > 0.0 TRUE

t1 = t

t1 = 0.16

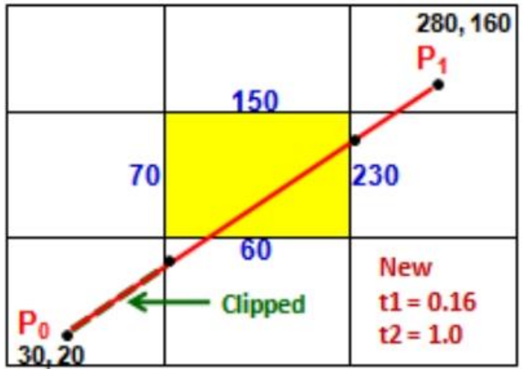


t1 = 0.16 t2 = 1.0

dx = 250

dy = 140

Liang-Barsky Line Clipping - Example



Right edge check:

p = dx = 250

q = $x_{max} - x_0$ = 230-30 = 200 t

= q/p = 0.8

- **If p< 0 FALSE**
- **If p > 0** *// update $t_{leaving}$ (t2)*

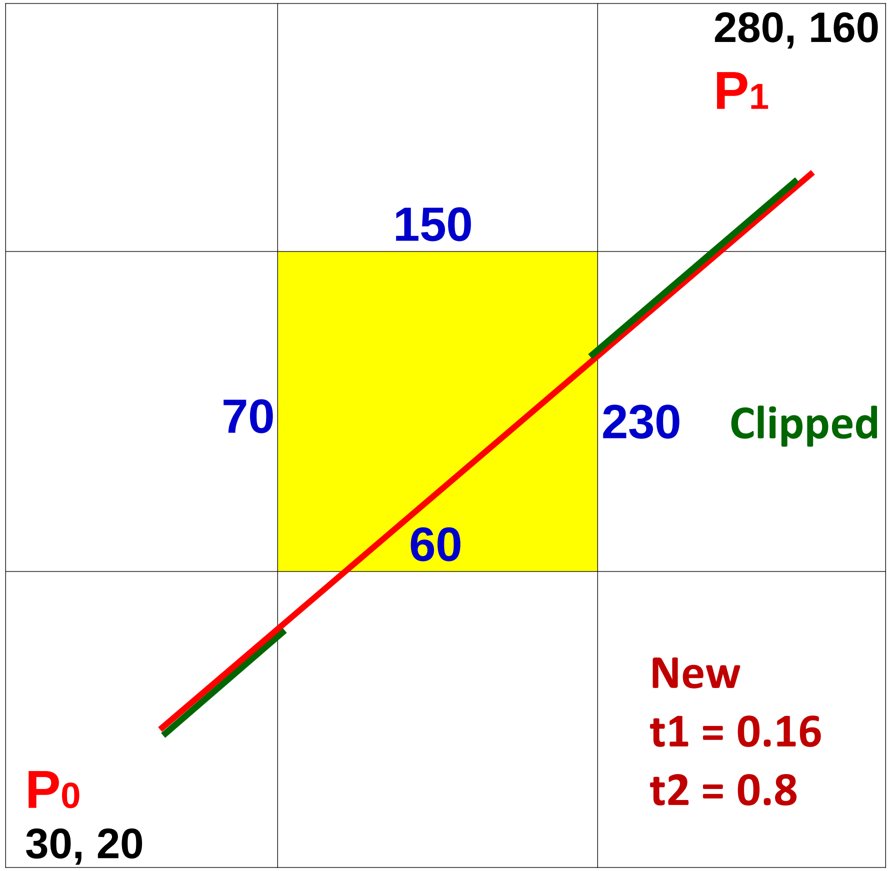
  if t < t2 set t2 = t

250 > 0 TRUE

check if t < t2 ---> 0.8 < 1.0 TRUE

t2 = t = 0.8

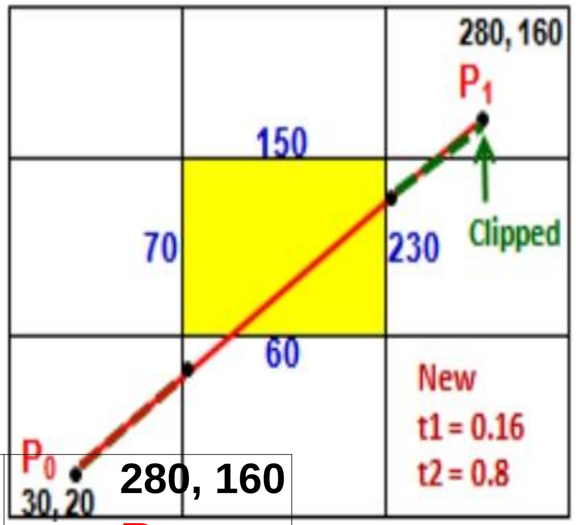
t1 = 0.16 t2 = 0.8



dx = 250

dy = 140

Liang-Barsky Line Clipping - Example



Bottom edge check:

p = -dy = -140

q = y_0 - y_{min} = 20-60 = -40 t

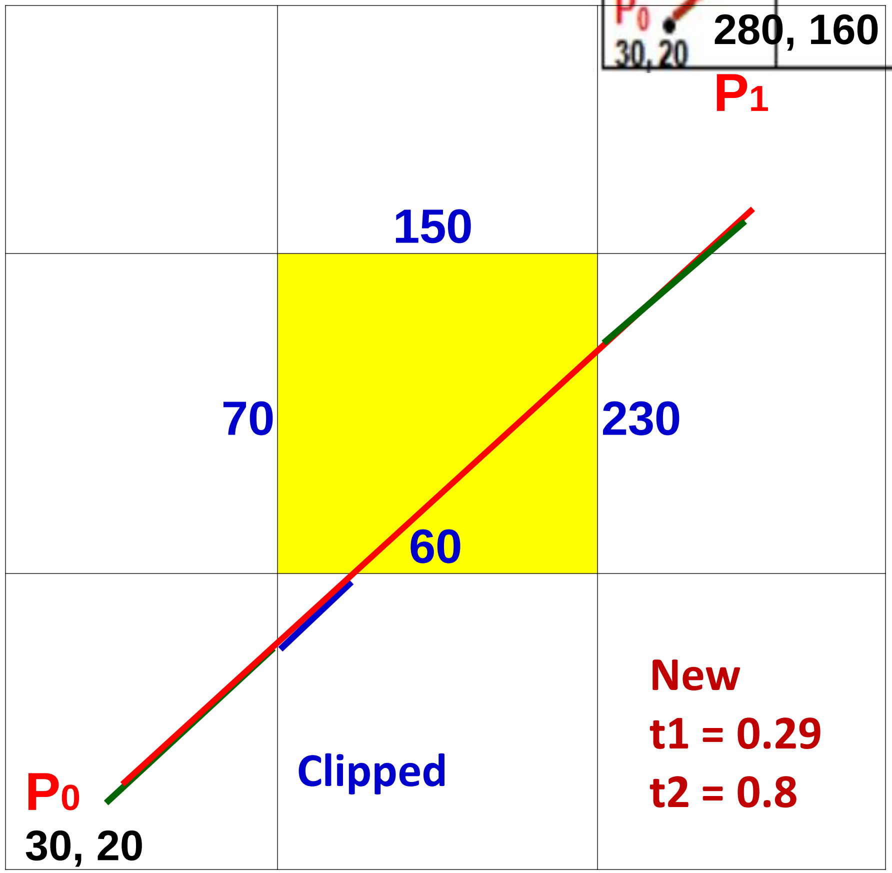
= q/p = 0.2857

- **If p< 0** *// update t_{entry} (t1)* **if t > t1 set t1 = t**

-140 < 0 TRUE

check if t > t1 -----> 0.2857 > 0.16 TRUE

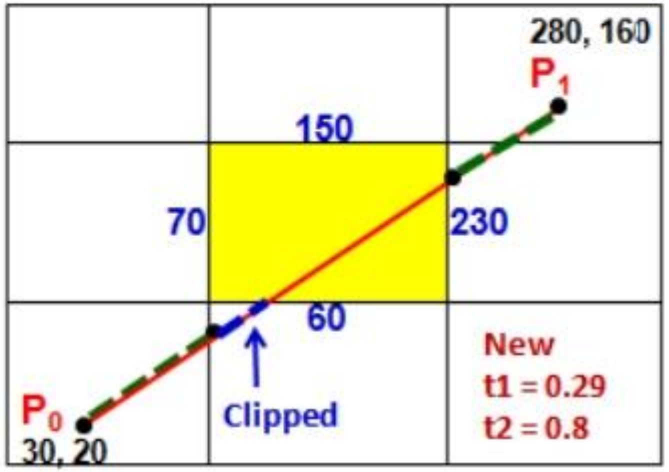
t1 = t----> t1 = 0.2857 t1 = 0.2857 t2 = 0.8



dx = 250

dy = 140

Liang-Barsky Line Clipping - Example



Top edge check:

p = dy = 140

q = $y_{max} - y_0$ = 150 - 20 = 130 t

= q/p = 0.928

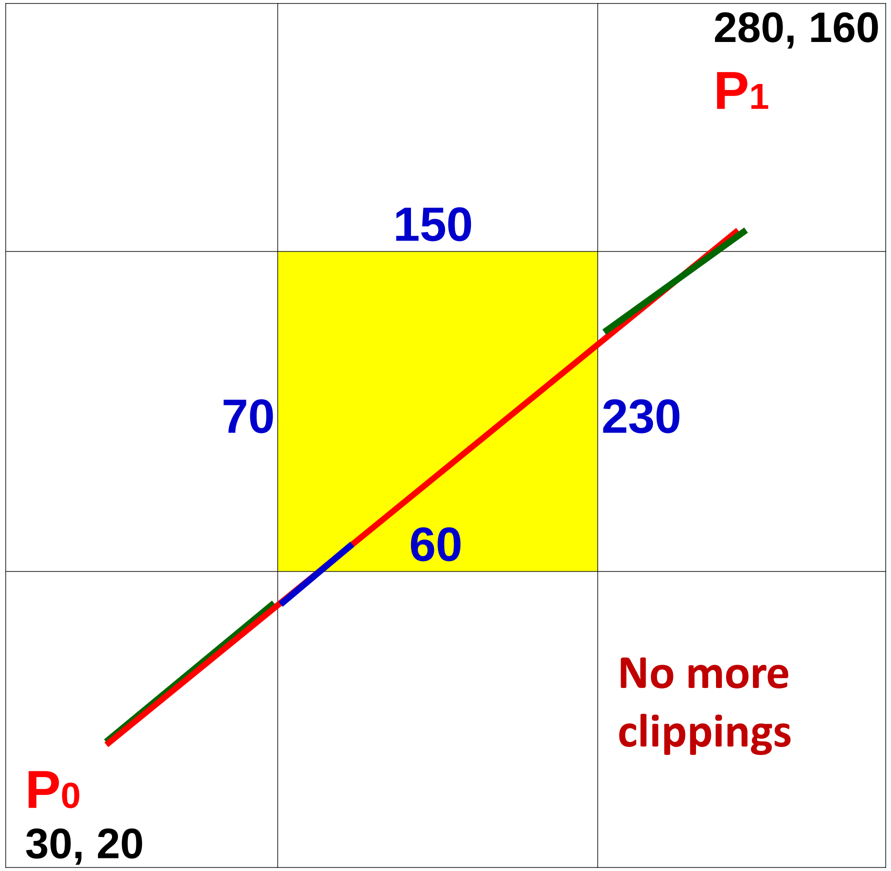
- **If p< 0 FALSE**
- **If p > 0** *// update $t_{leaving}$ (t2)* **if t < t2 set t2 = t**

140 > 0 TRUE

check if t < t2 ----->

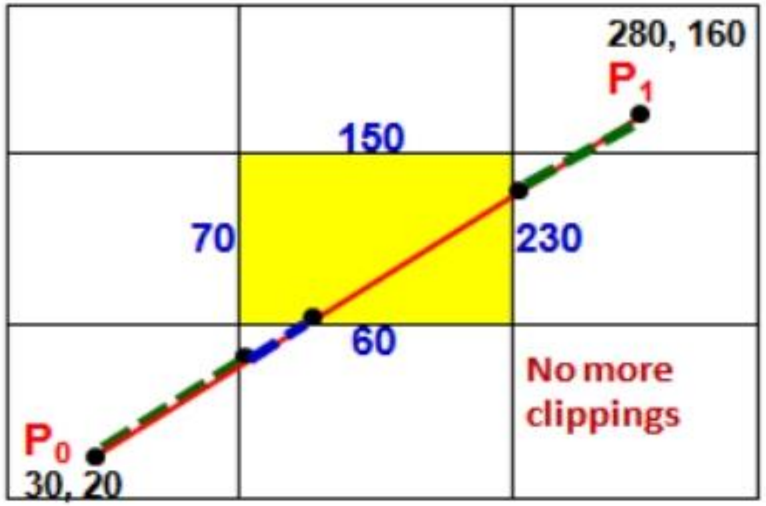
0.928 < 0.8 FALSE

t1 = 0.2857 t2 = 0.8



Liang-Barsky Line Clipping - Example

t1 = 0.2857 **t2 = 0.8** **dx = 250** **dy = 140**



if(t_{entry} > 0.0) calculate new x0, y0 0.2857 > 0.0 **TRUE**

$x0_{new} = x0 + t_{entry} * dx$

$x0_{new}$ = 30 + 0.2857 * 250

$y0_{new}$ = **101.425** = $y0 + t_{entry} * dy$

$y0_{new}$ = 20 + 0.2857 * 140 = **60.0**

Liang-Barsky Line Clipping - Example

$t1 = 0.2857$ **$t2 = 0.8$**

$dx = 250$ **$dy = 140$**

if($t_{leaving} < 1.0$) calculate new x1, y1

0.8 > 0.0 **TRUE**

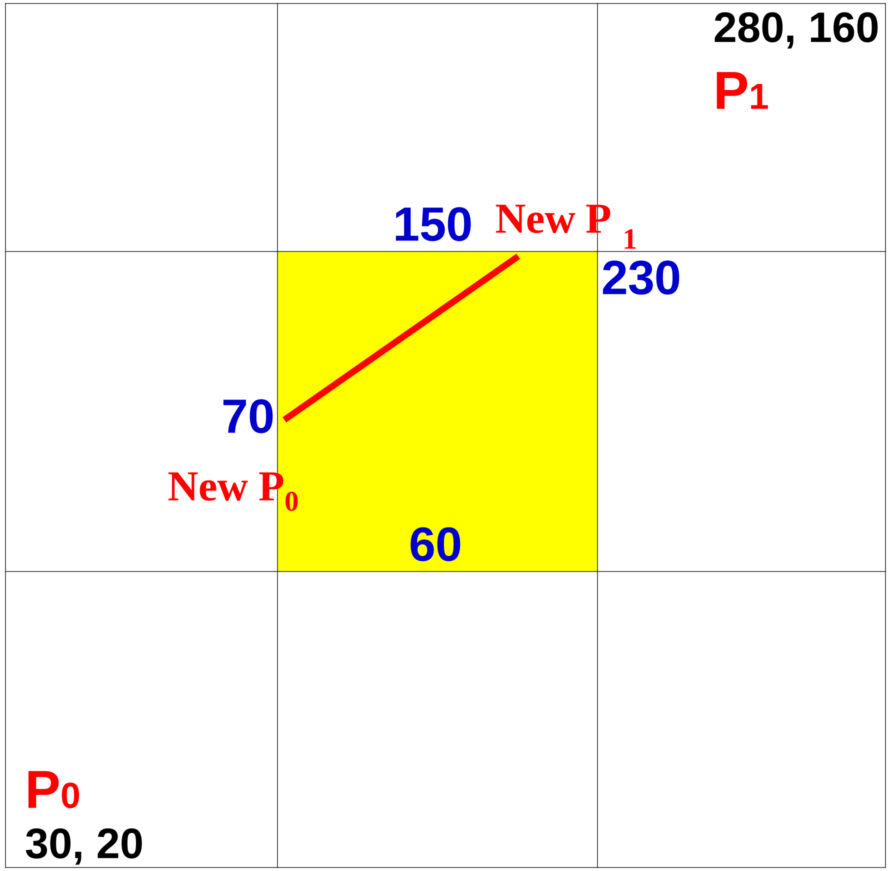
$x1_{new} = x0 + t_{leaving} * dx$

$x1_{new} = 30 + 0.8 * 250 = $ **230**

$y1_{new} = y0 + t_{leaving} * dy$

$y1_{new} = 20 + 0.8 * 140 = $ **132**

Draw clipped line with the points A($x0_{new}$, $y0_{new}$) and B($x1_{new}$, $y1_{new}$)



Advantages of Liang-Barsky Line Clipping

1. More efficient.
2. Only requires one division to update t_1 and t_2.
3. Window intersections of line are calculated just once.