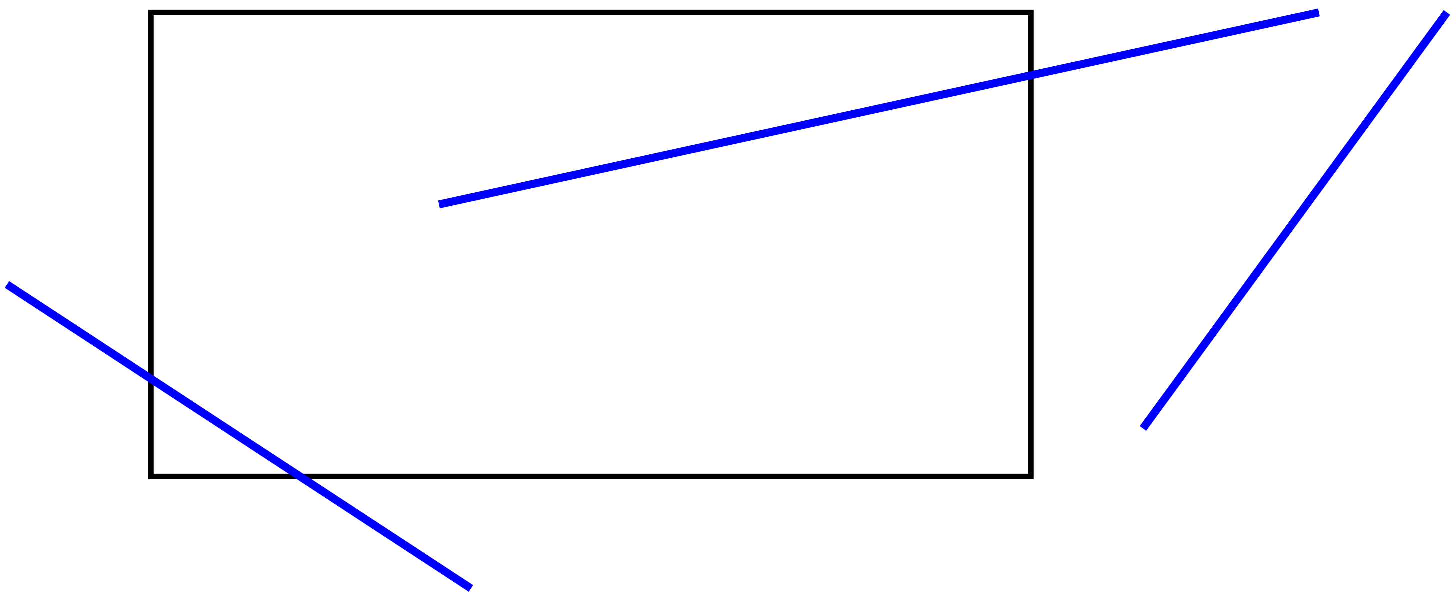
Clipping Lines

Dr. Scott Schaefer

TAMU
Computer
Science

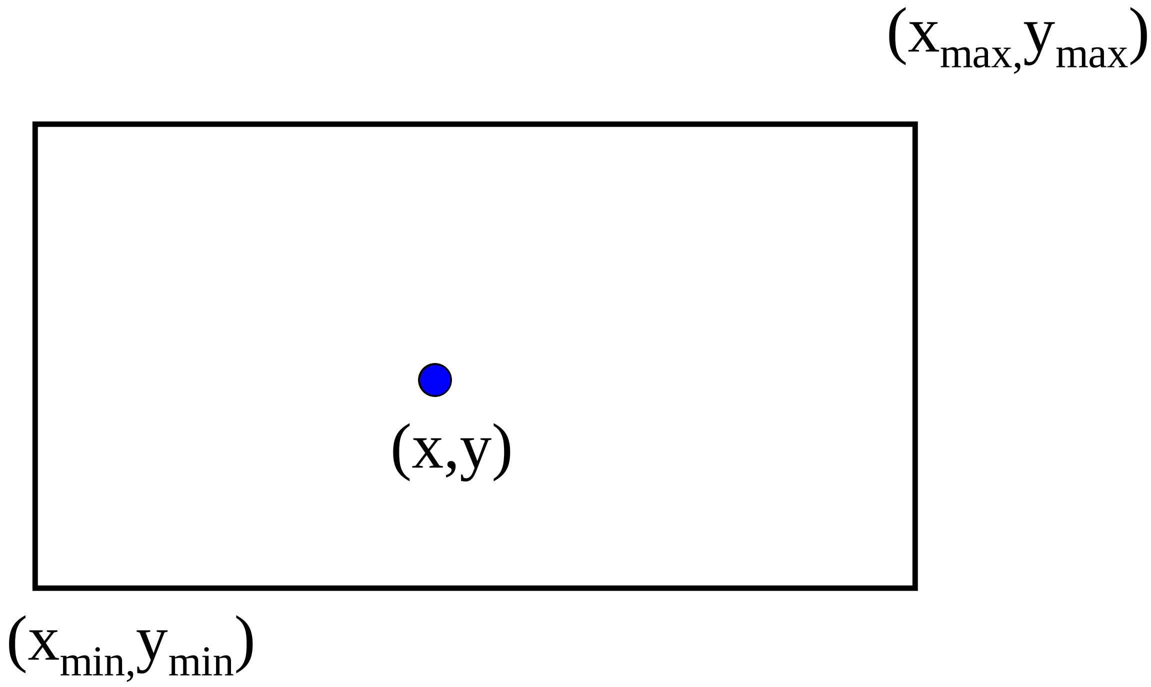
Why Clip?

- We do not want to waste time drawing objects that are outside of viewing window (or clipping window)



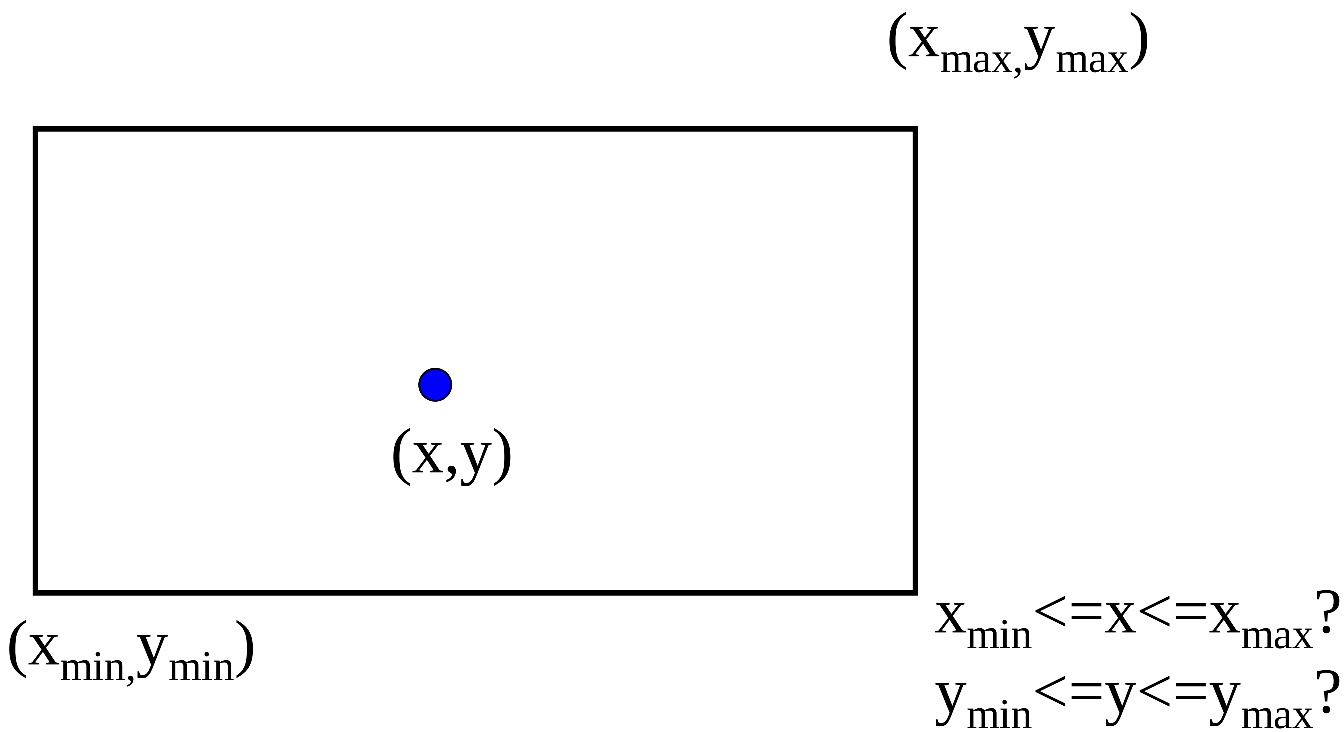
Clipping Points

- Given a point (x, y) and clipping window (x_{min}, y_{min}), (x_{max}, y_{max}), determine if the point should be drawn



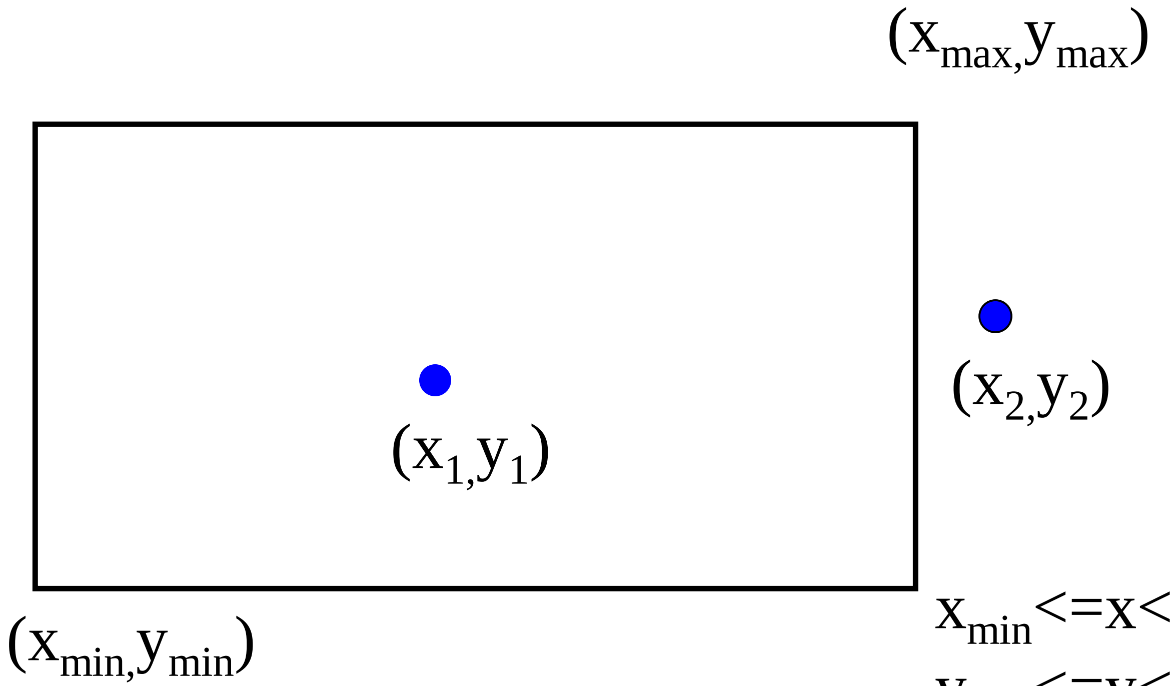
Clipping Points

- Given a point (x, y) and clipping window (x_{min}, y_{min}), (x_{max}, y_{max}), determine if the point should be drawn



Clipping Points

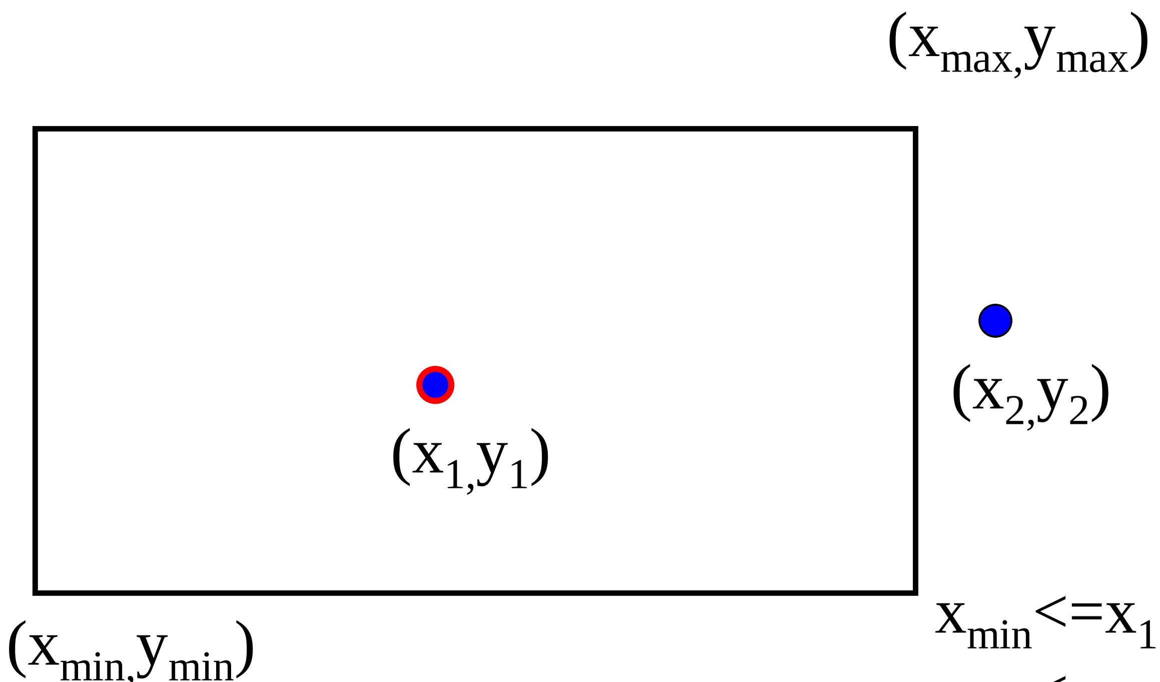
- Given a point (x, y) and clipping window (x_{min}, y_{min}), (x_{max}, y_{max}), determine if the point should be drawn



$x_{min} <= x <= x_{max}$?
$y_{min} <= y <= y_{max}$?

Clipping Points

- Given a point (x, y) and clipping window (x_{min}, y_{min}), (x_{max}, y_{max}), determine if the point should be drawn

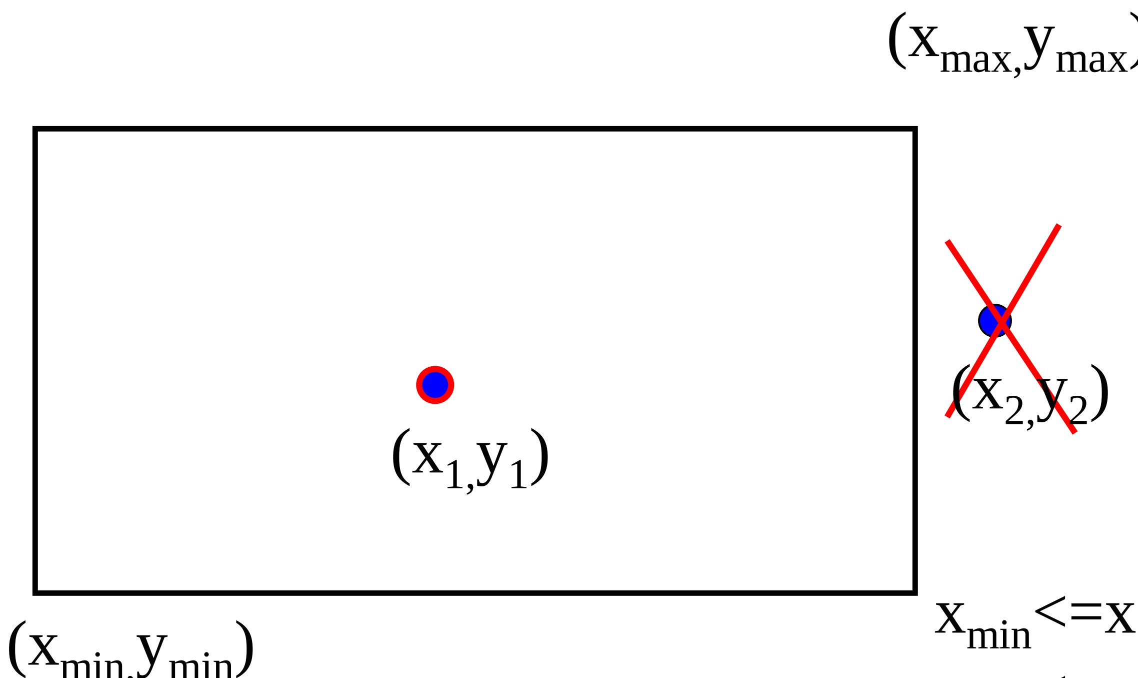


$x_{min} <= x_1 <= x_{max}$ **Yes**

$y_{min} <= y_1 <= y_{max}$ **Yes**

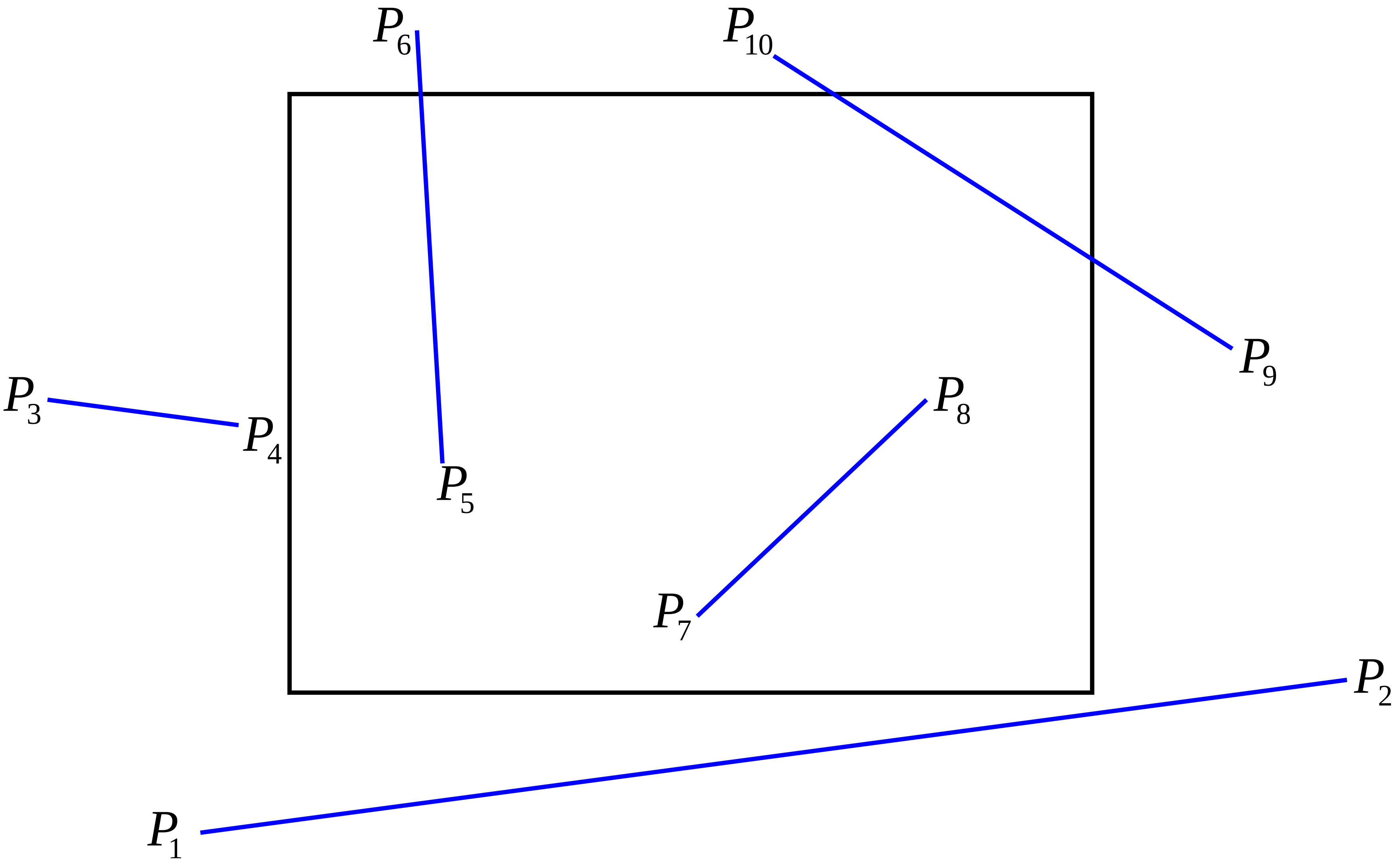
Clipping Points

- Given a point (x, y) and clipping window (x_{min}, y_{min}), (x_{max}, y_{max}), determine if the point should be drawn

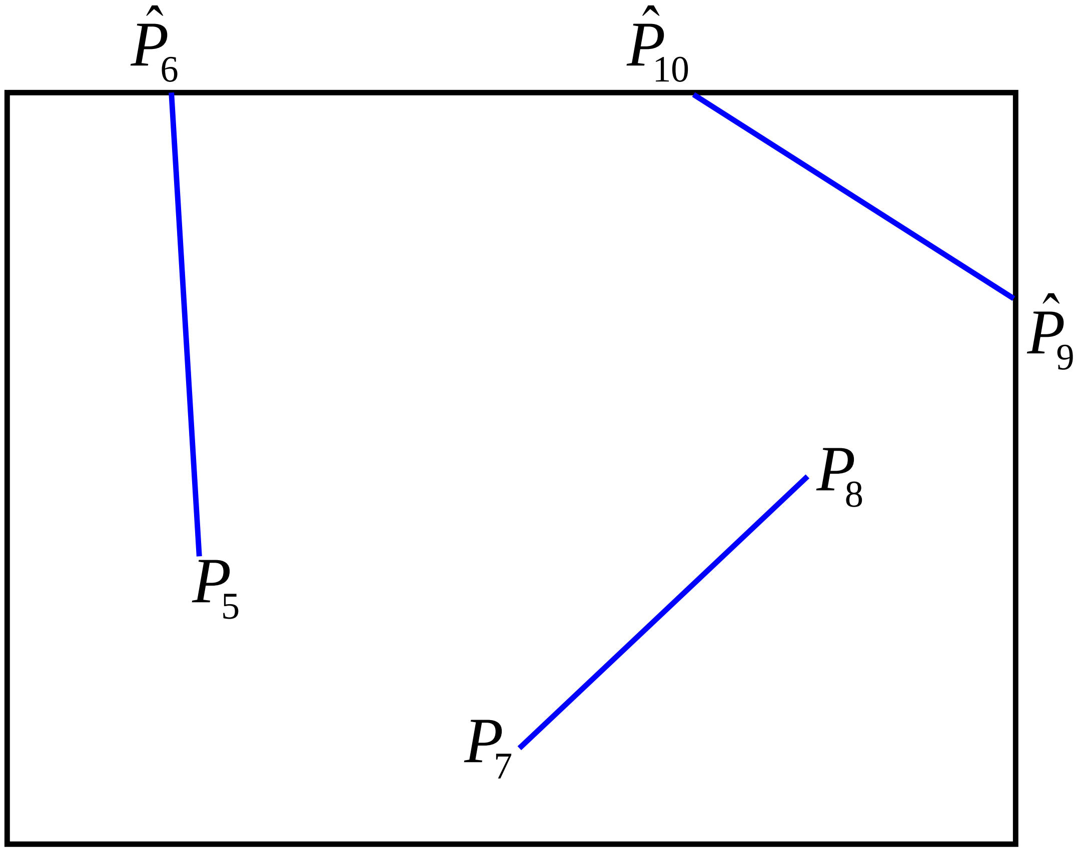


$x_{min}<=x_2<=x_{max}$ **No**
$y_{min}<=y_2<=y_{max}$ **Yes**

Clipping Lines

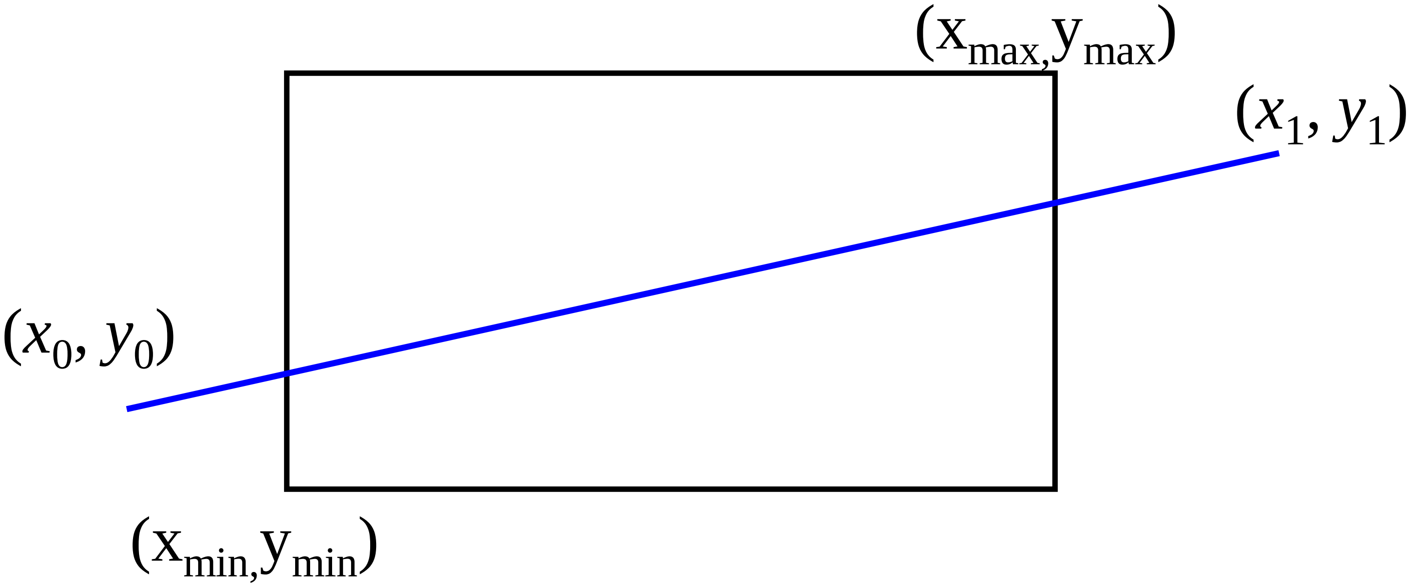


Clipping Lines

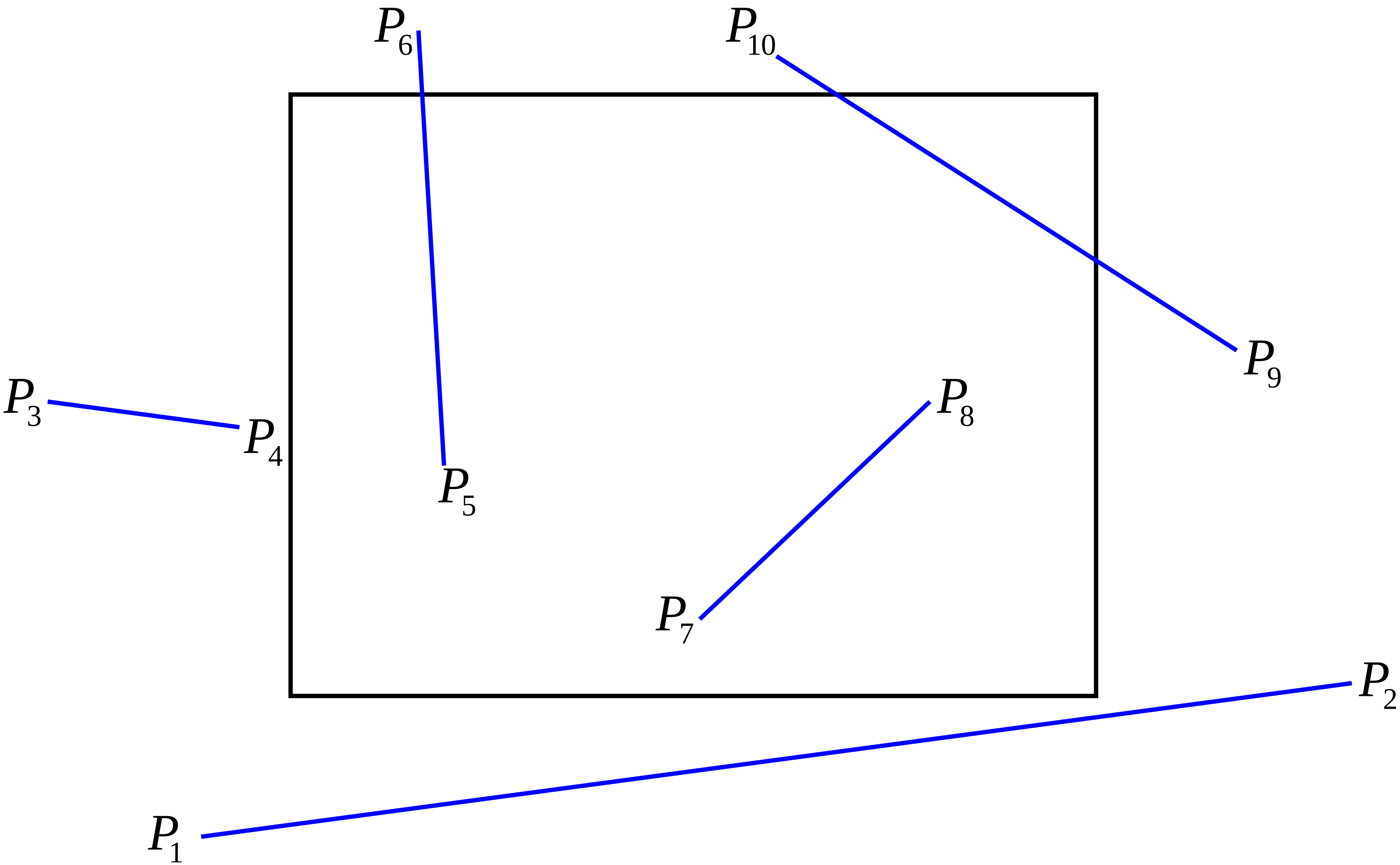


Clipping Lines

- Given a line with end-points (x_0, y_0), (x_1, y_1) and clipping window (x_{min}, y_{min}), (x_{max}, y_{max}), determine if line should be drawn and clipped end-points of line to draw.



Clipping Lines



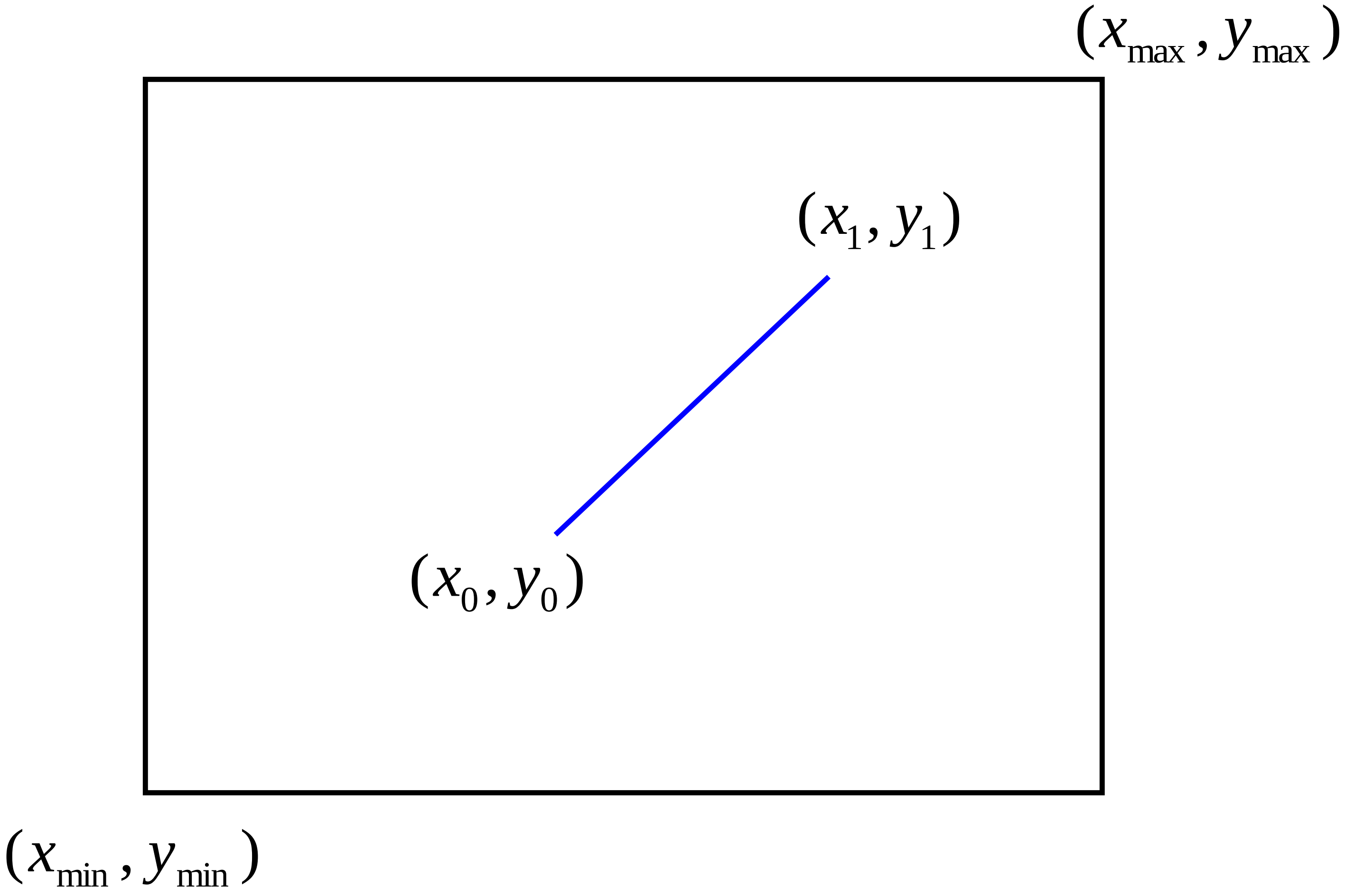
Clipping Lines – Simple Algorithm

- If both end-points inside rectangle, draw line
- Otherwise,

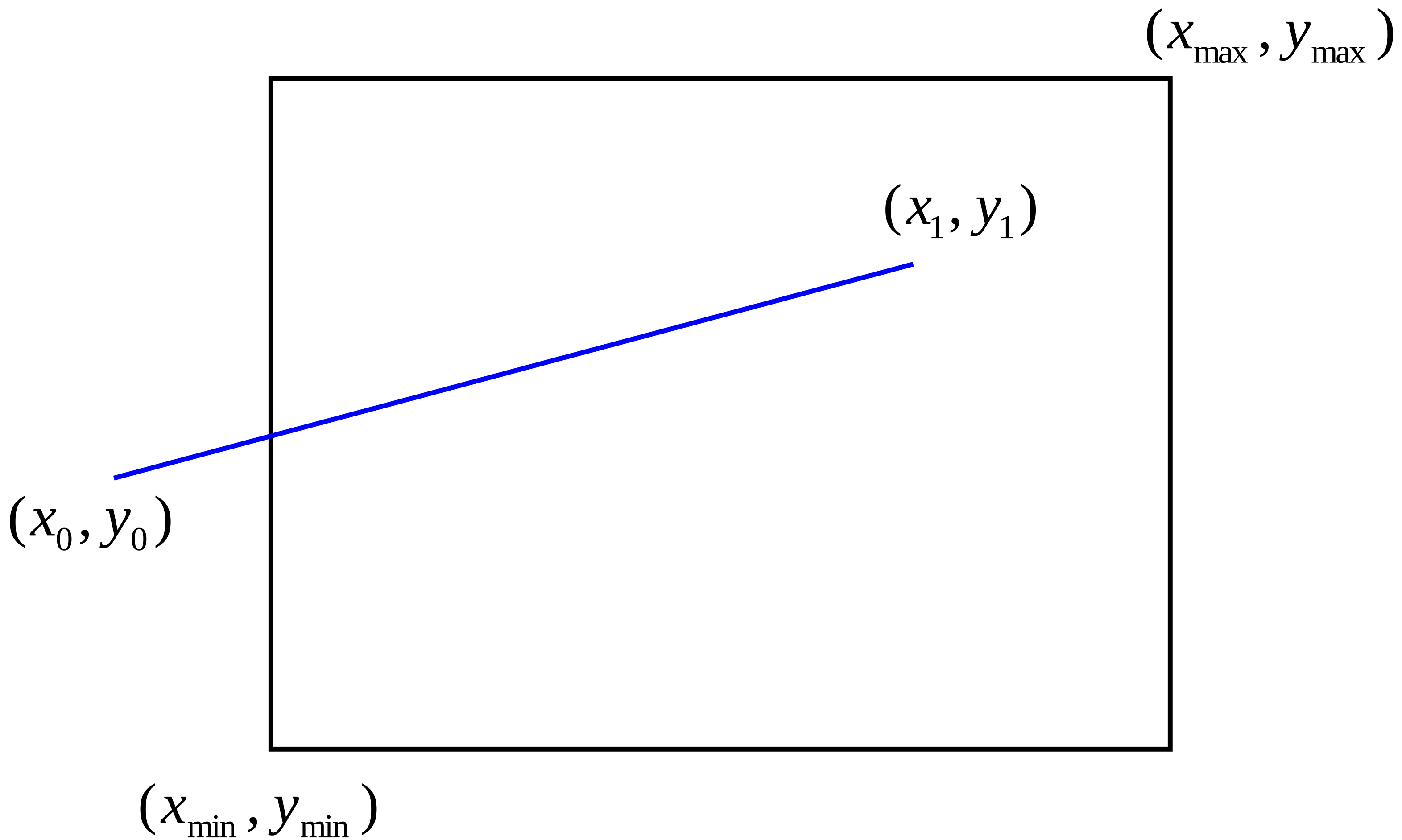
  intersect line with all edges of rectangle

  clip that point and repeat test

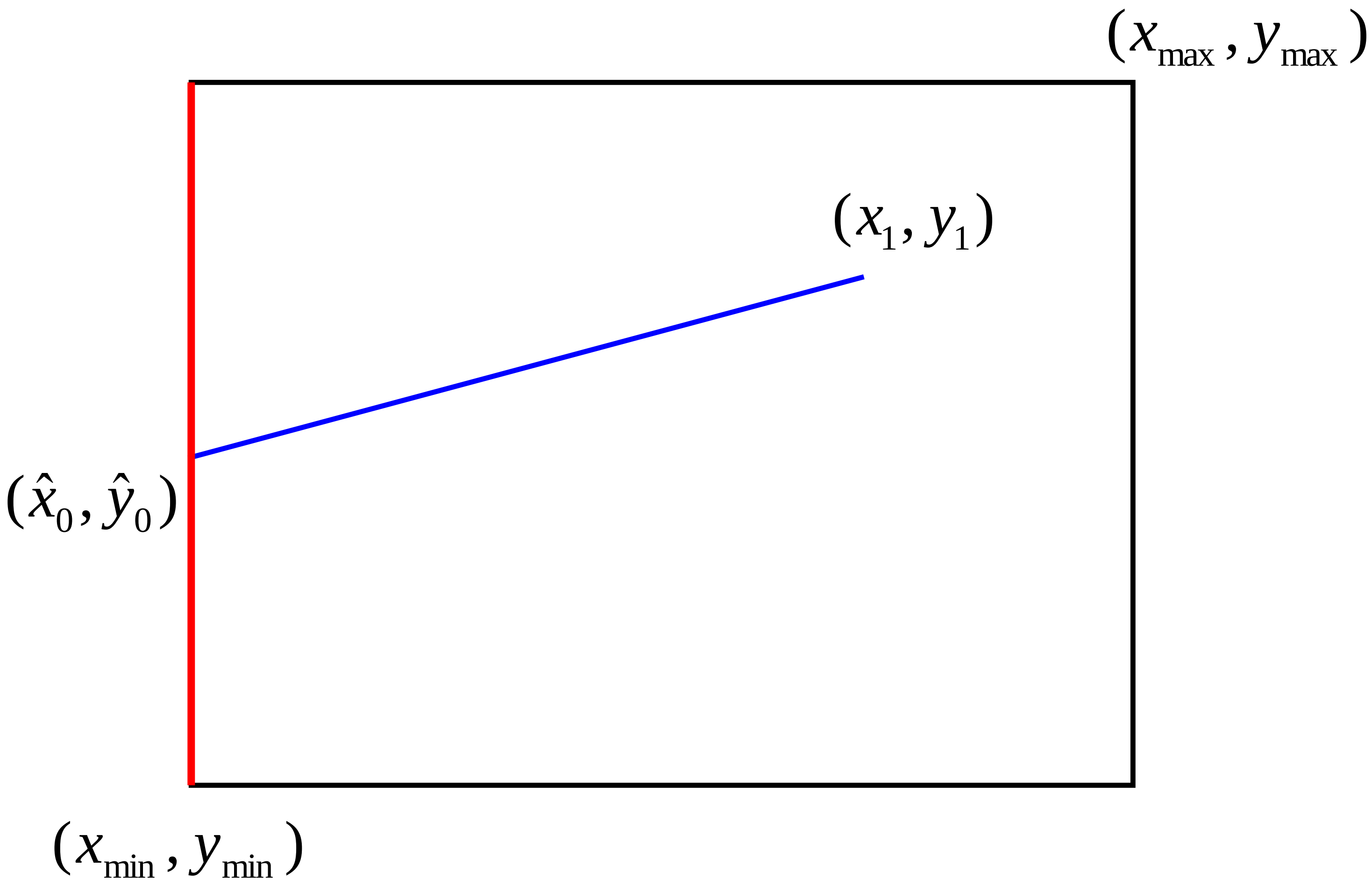
Clipping Lines – Simple Algorithm



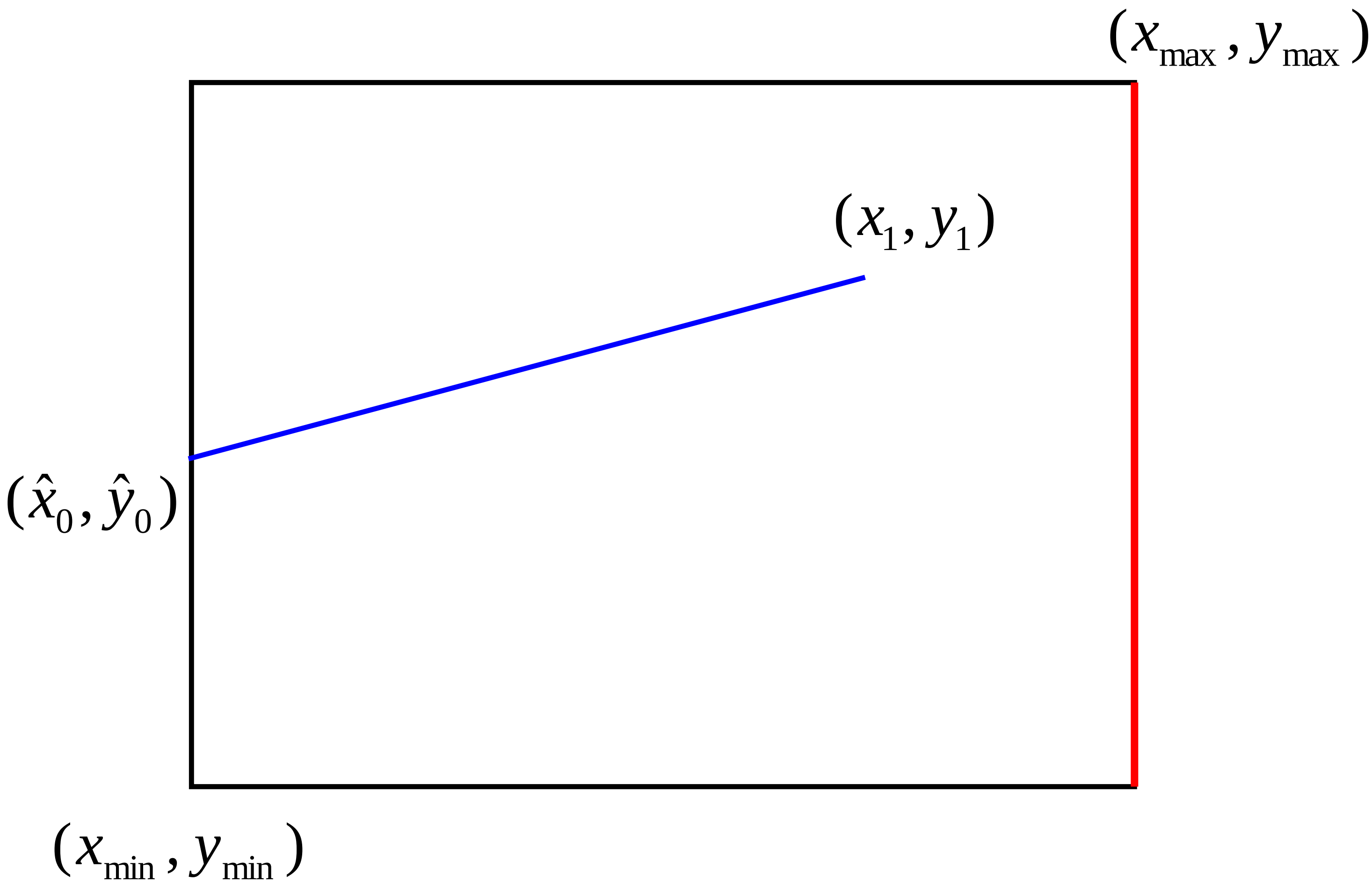
Clipping Lines – Simple Algorithm



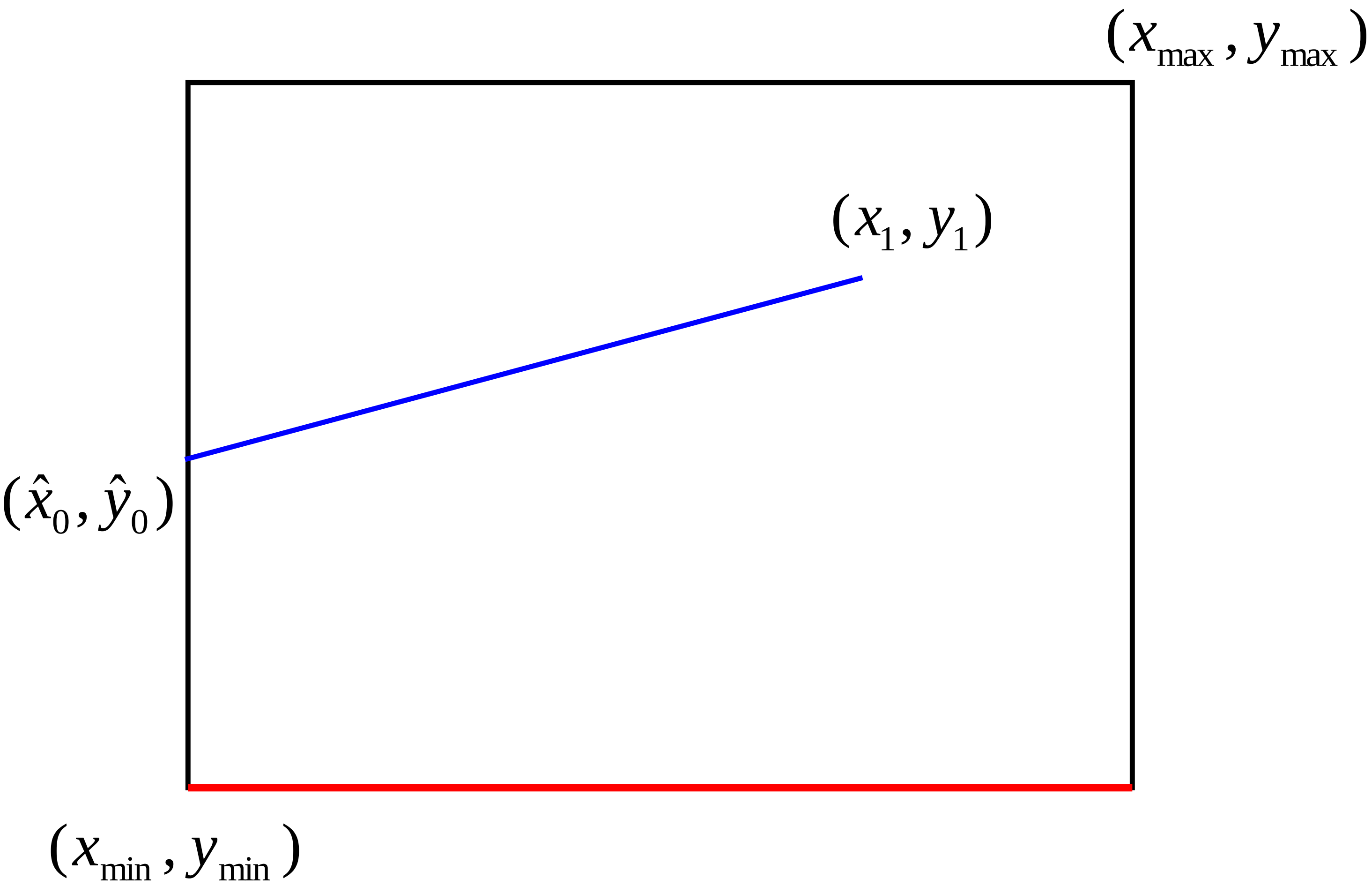
Clipping Lines – Simple Algorithm



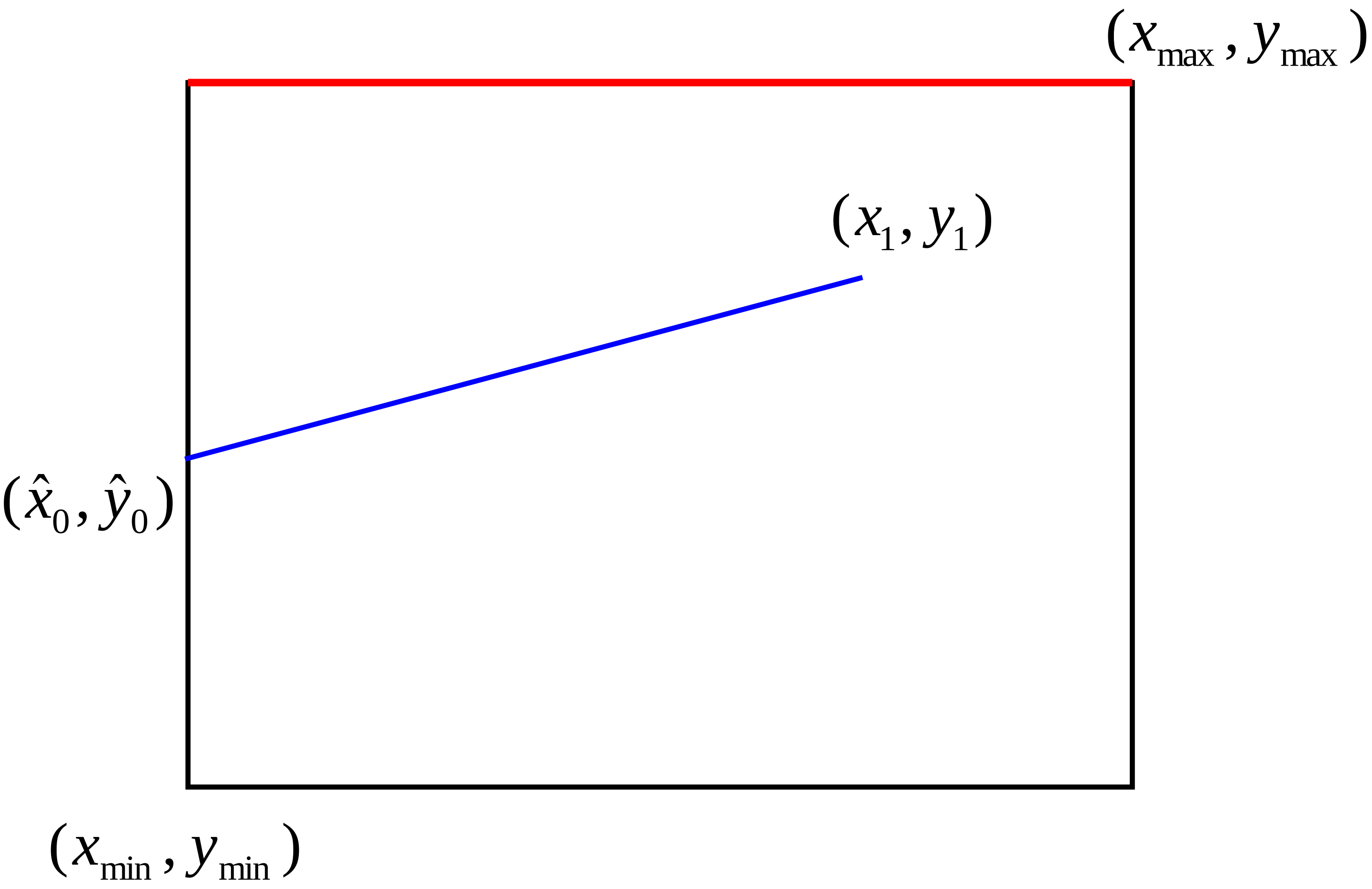
Clipping Lines – Simple Algorithm



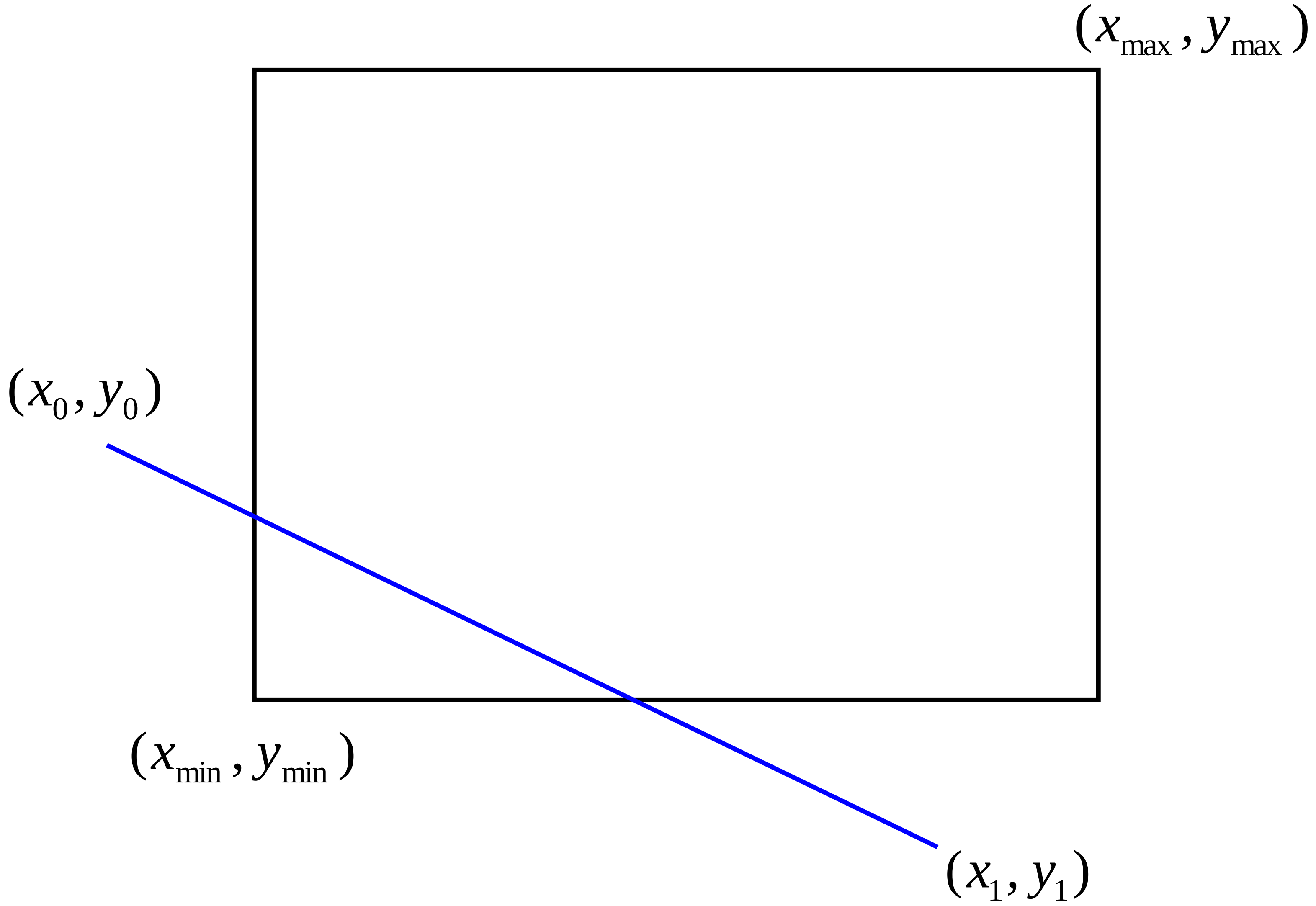
Clipping Lines – Simple Algorithm



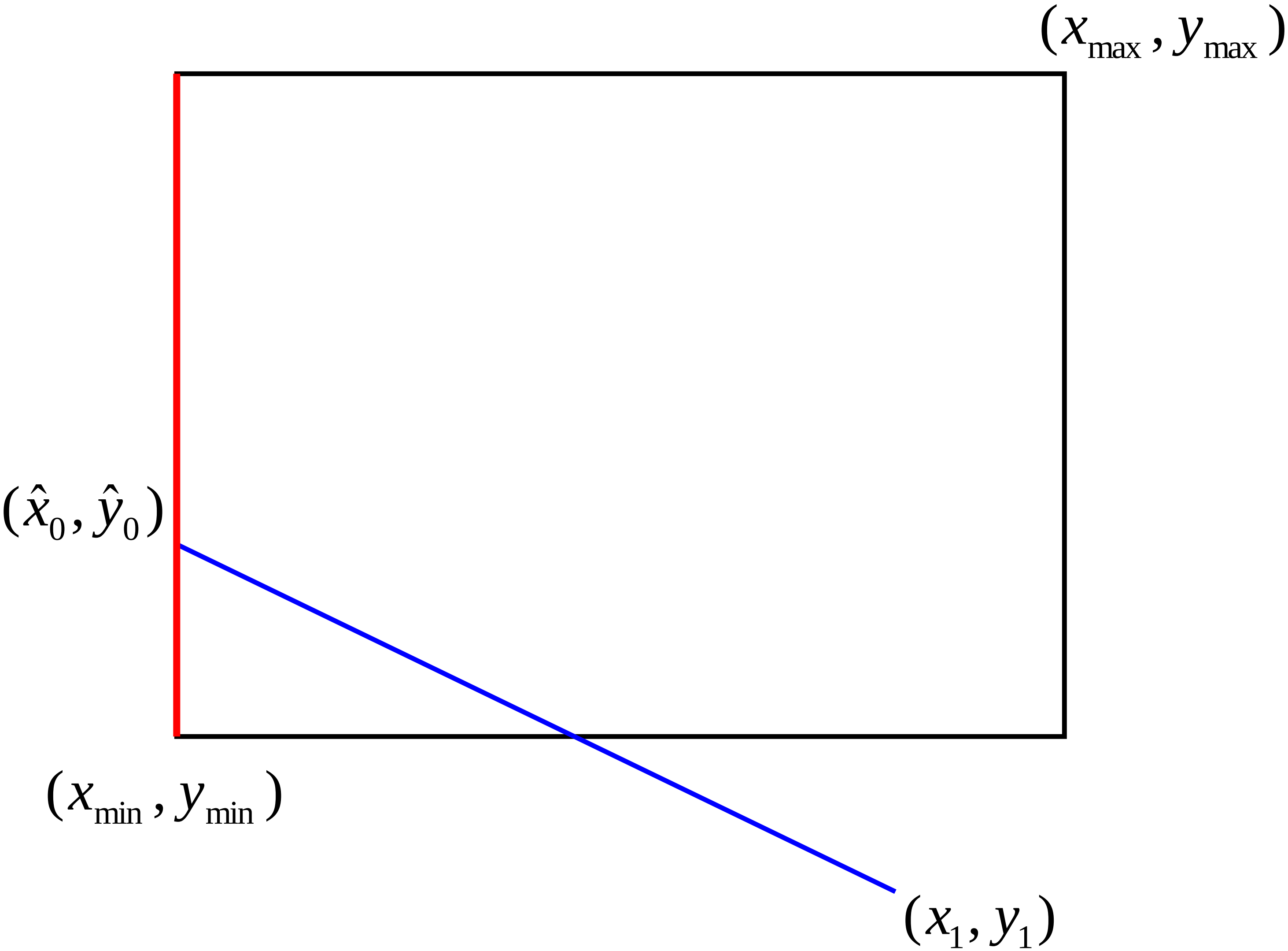
Clipping Lines – Simple Algorithm



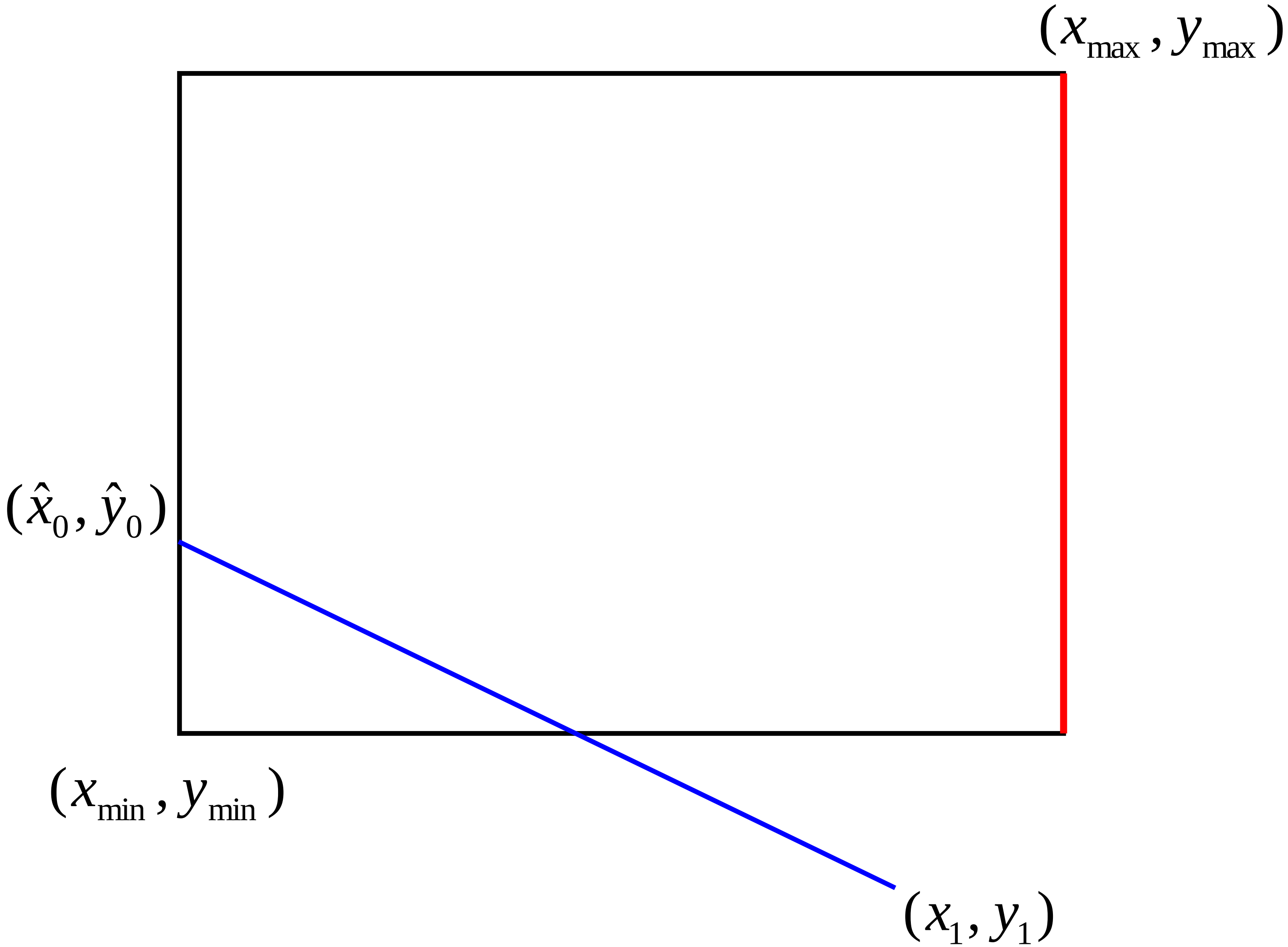
Clipping Lines – Simple Algorithm



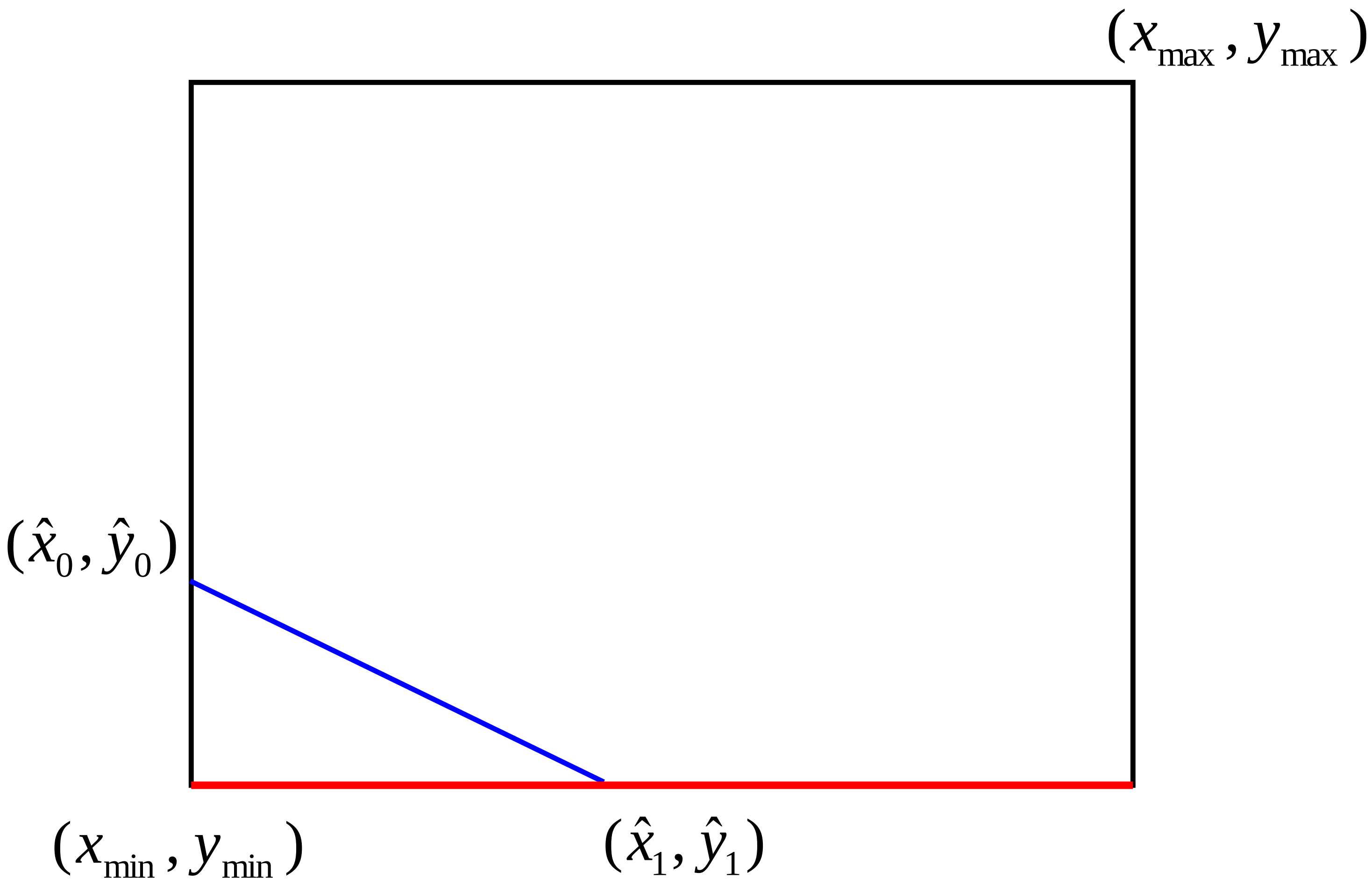
Clipping Lines – Simple Algorithm



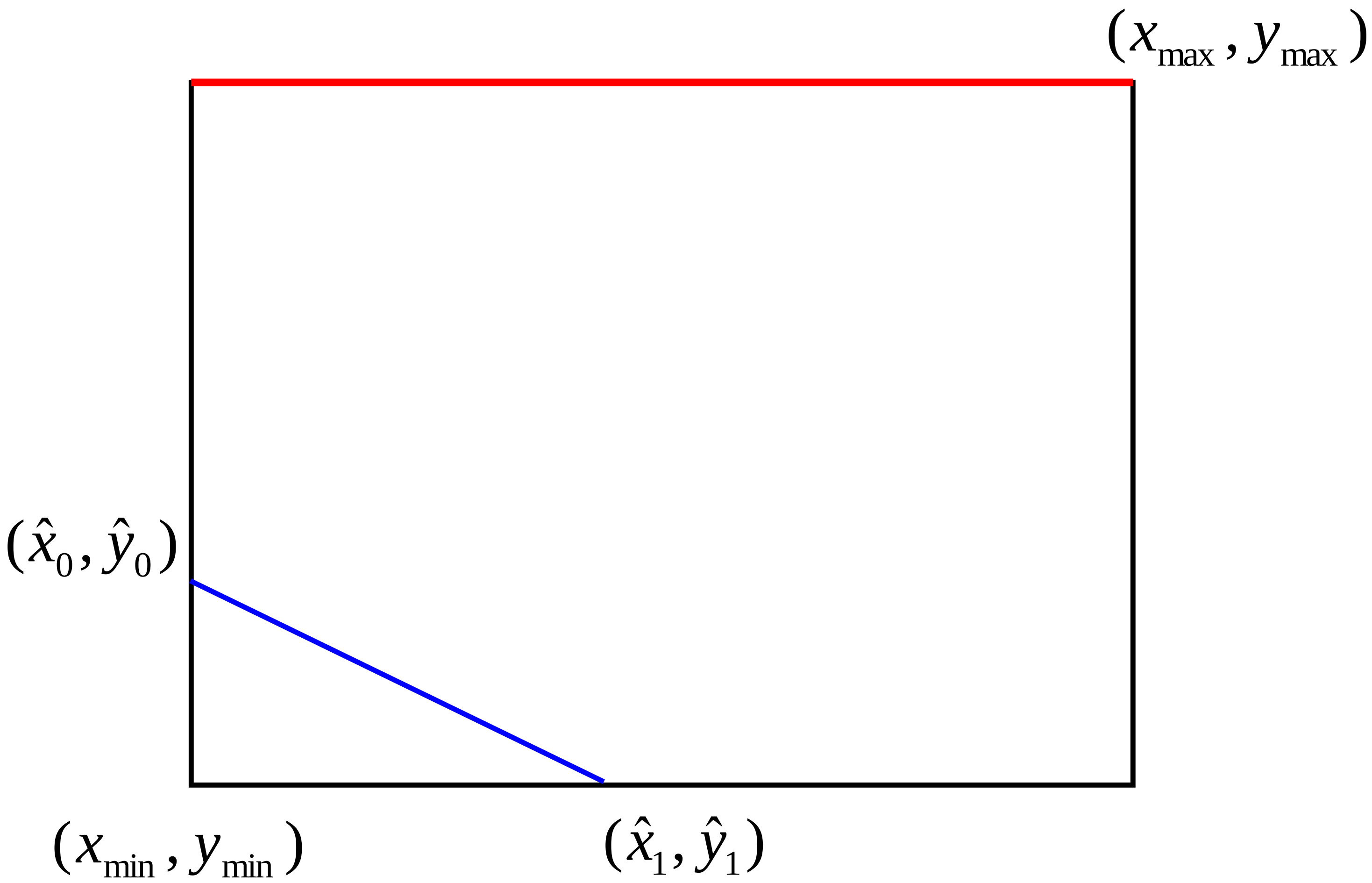
Clipping Lines – Simple Algorithm



Clipping Lines – Simple Algorithm



Clipping Lines – Simple Algorithm



Window Intersection

$(x_1, y_1), (x_2, y_2)$ intersect with vertical edge at x_{right}

- $y_{intersect} = y_1 + m(x_{right} - x_1)$
  where $m=(y_2-y_1)/(x_2-x_1)$

$(x_1, y_1), (x_2, y_2)$ intersect with horizontal edge at y_{bottom}

- $x_{intersect} = x_1 + (y_{bottom} - y_1)/m$
  where $m=(y_2-y_1)/(x_2-x_1)$

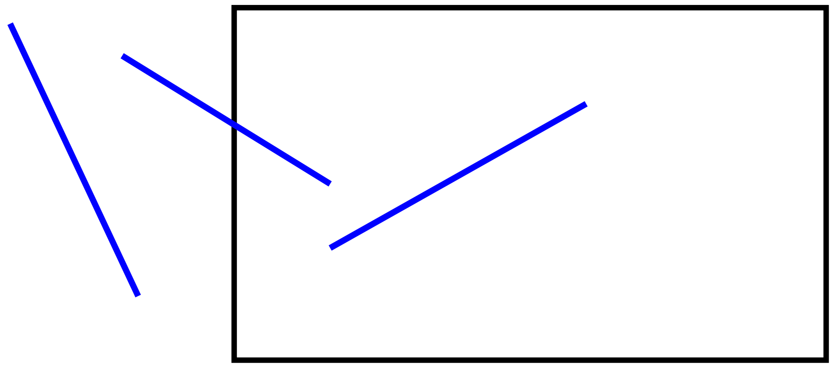
Clipping Lines – Simple Algorithm

- Lots of intersection tests makes algorithm expensive
- Complicated tests to determine if intersecting rectangle

- Is there a better way?

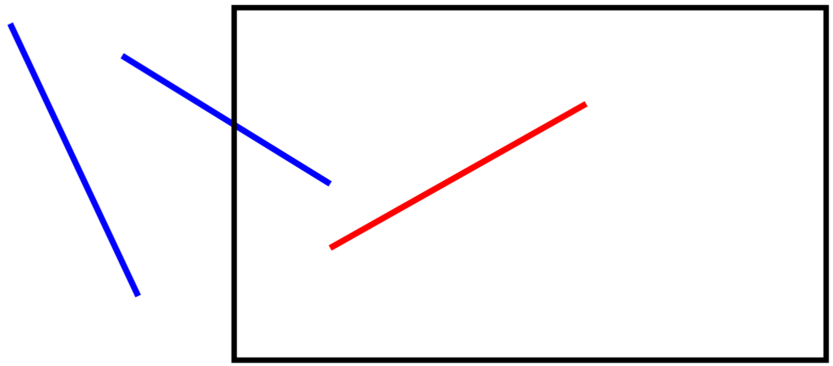
Trivial Accepts

- Big Optimization: trivial accepts/rejects
- How can we quickly decide whether line segment is entirely inside window
- Answer: test both endpoints



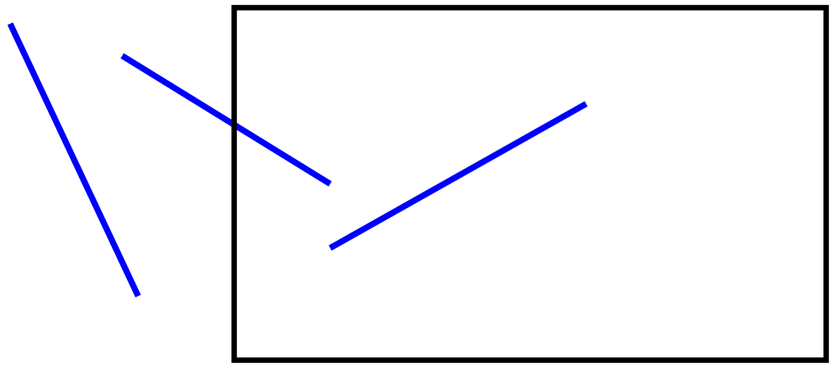
Trivial Accepts

- Big Optimization: trivial accepts/rejects
- How can we quickly decide whether line segment is entirely inside window
- Answer: test both endpoints



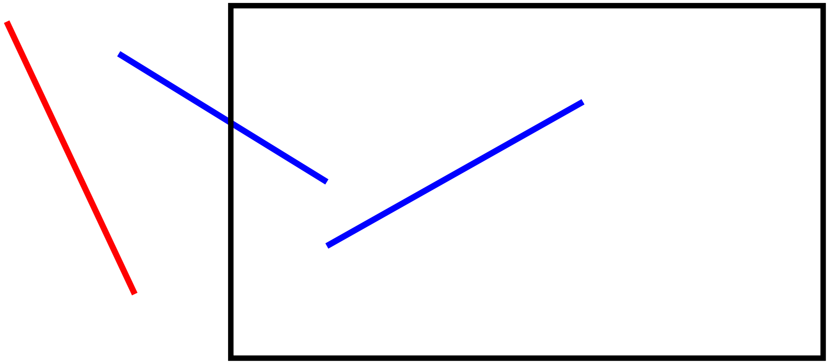
Trivial Rejects

- How can we know a line is outside of the window
- Answer: both endpoints on wrong side of same edge, can trivially reject the line



Trivial Rejects

- How can we know a line is outside of the window
- Answer: both endpoints on wrong side of same edge, can trivially reject the line



Cohen-Sutherland Algorithm

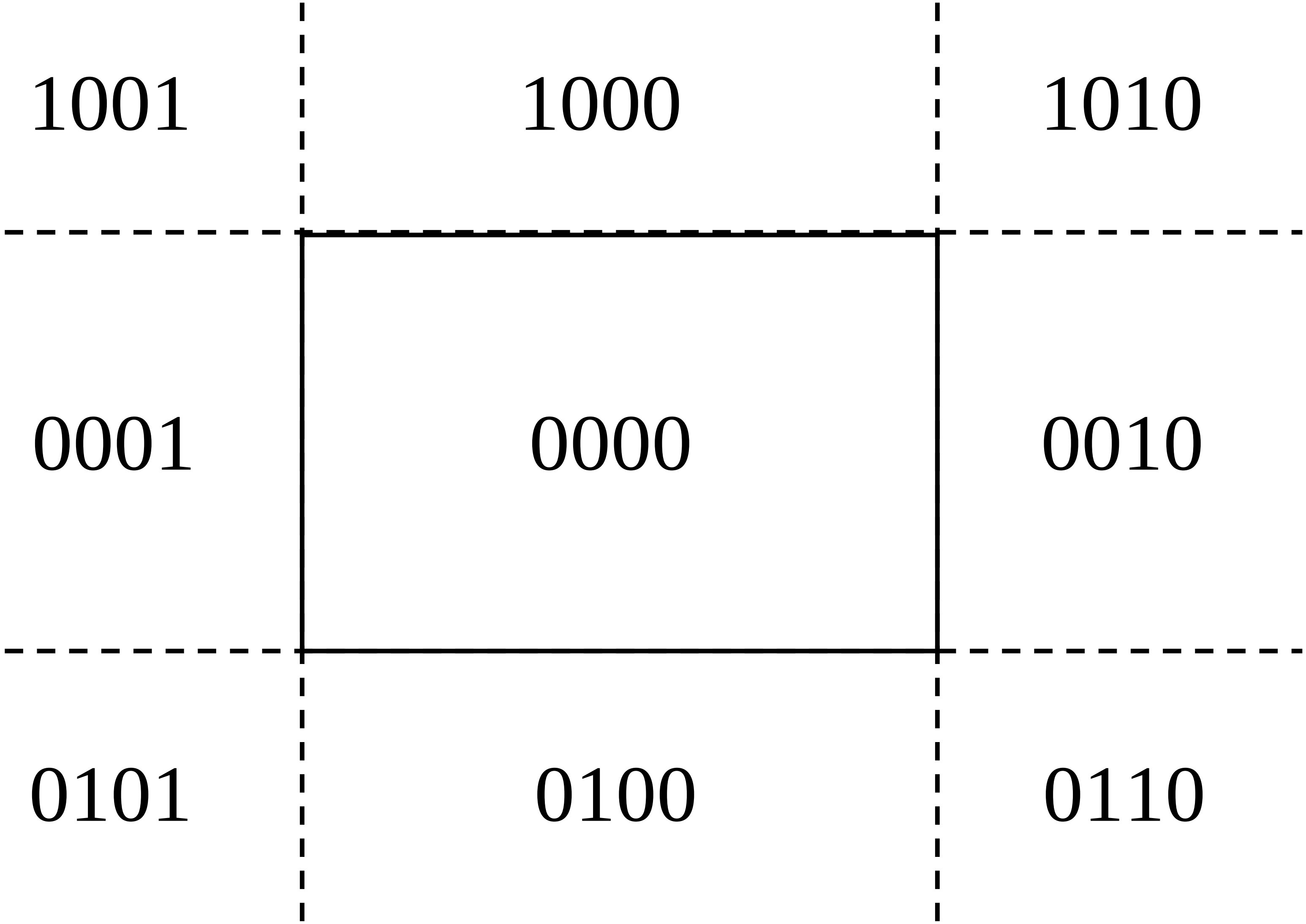
- Classify p_0, p_1 using region codes c_0, c_1
- If $c_0 \wedge c_1 \neq 0$, trivially reject
- If $c_0 \vee c_1 = 0$, trivially accept
- Otherwise reduce to trivial cases by splitting into two segments

Cohen-Sutherland Algorithm

- Every end point is assigned to a four-digit binary value, i.e. Region code
- Each bit position indicates whether the point is inside or outside of a specific window edge

bit 4	bit 3	bit 2	bit 1
top	bottom	right	left

Cohen-Sutherland Algorithm



Cohen-Sutherland Algorithm

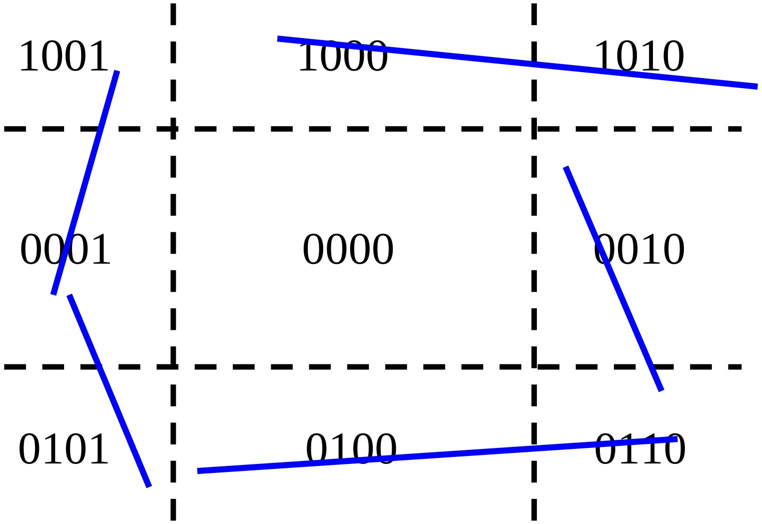
- Classify p_0, p_1 using region codes c_0, c_1
- If $c_0 \wedge c_1 \neq 0$, trivially reject
- If $c_0 \vee c_1 = 0$, trivially accept
- Otherwise reduce to trivial cases by splitting into two segments

Cohen-Sutherland Algorithm

- Classify p_0, p_1 using region codes c_0, c_1
- $\boxed{\text{If } c_0 \wedge c_1 \neq 0\text{, trivially reject}}$
- If $c_0 \vee c_1 = 0$, trivially accept
- Otherwise reduce to trivial cases by splitting into two segments

Cohen-Sutherland Algorithm

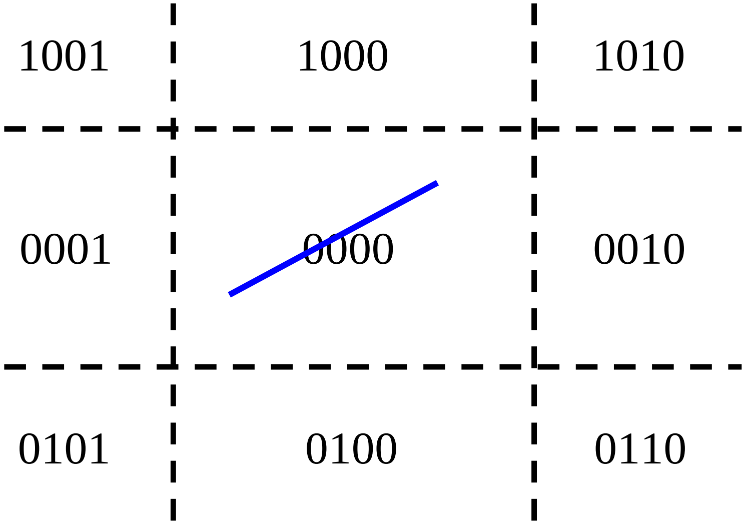
- Classify p_0, p_1 using region codes c_0, c_1
- If $c_0 \wedge c_1 \neq 0$, trivially reject
- If $c_0 \vee c_1 = 0$, trivially accept
- Otherwise reduce to trivial cases by splitting into two segments



Line is outside the window! reject

Cohen-Sutherland Algorithm

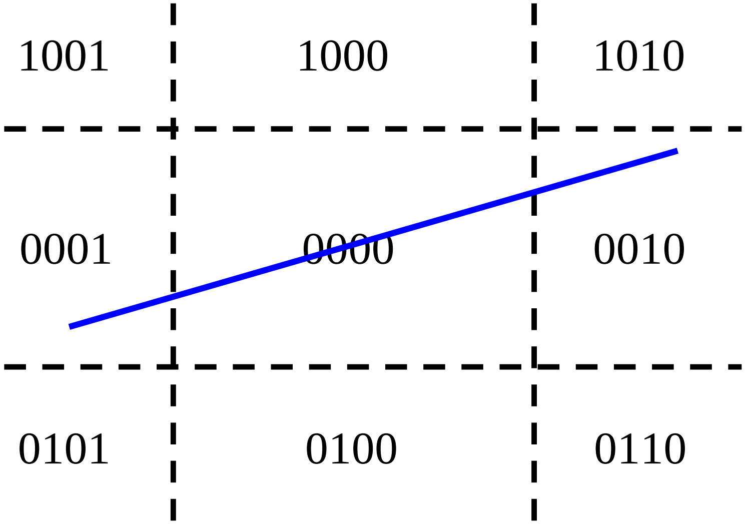
- Classify p_0, p_1 using region codes c_0, c_1
- If $c_0 \wedge c_1 \neq 0$, trivially reject
- If $c_0 \vee c_1 = 0$, trivially accept
- Otherwise reduce to trivial cases by splitting into two segments



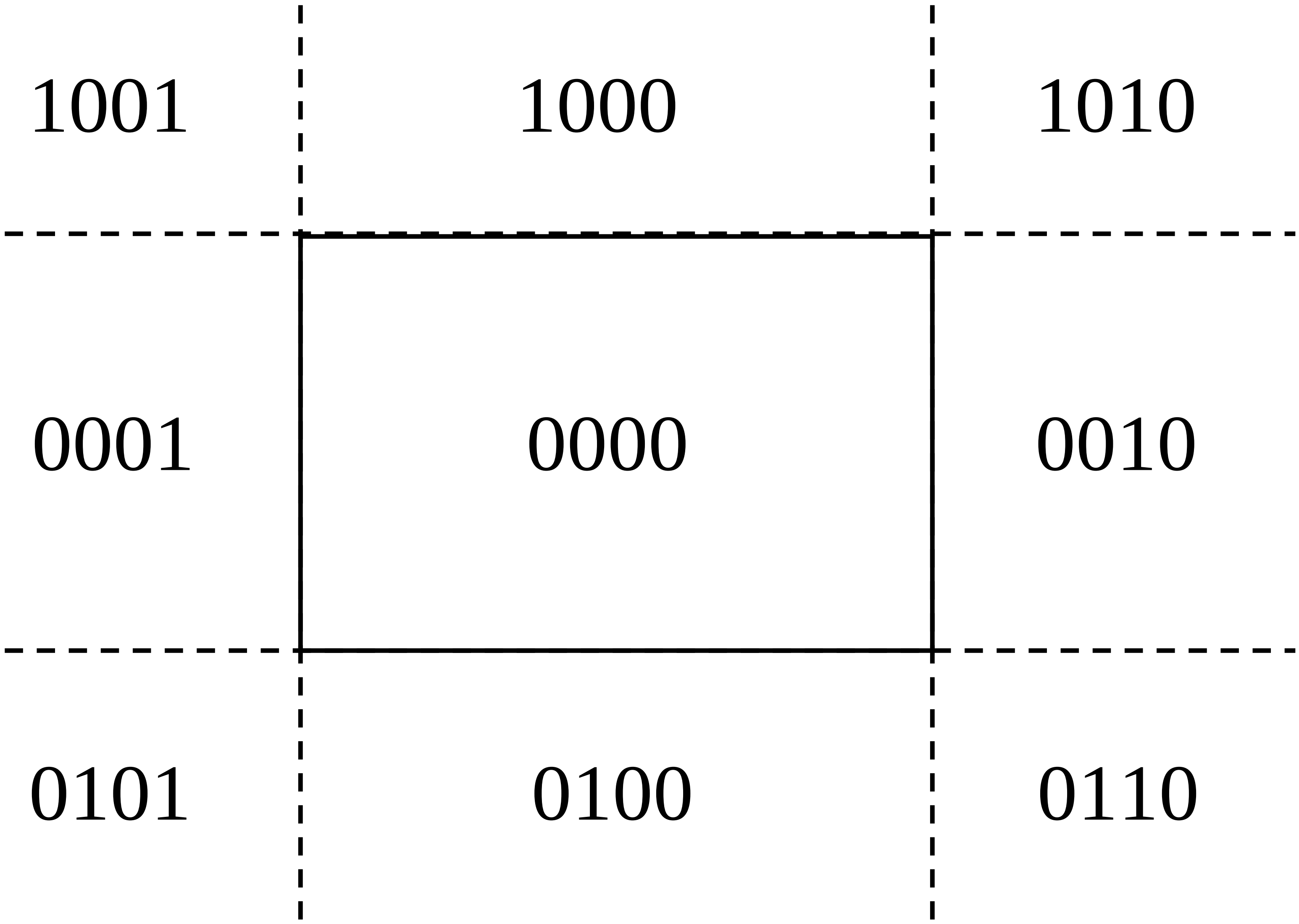
Line is inside the window! draw

Cohen-Sutherland Algorithm

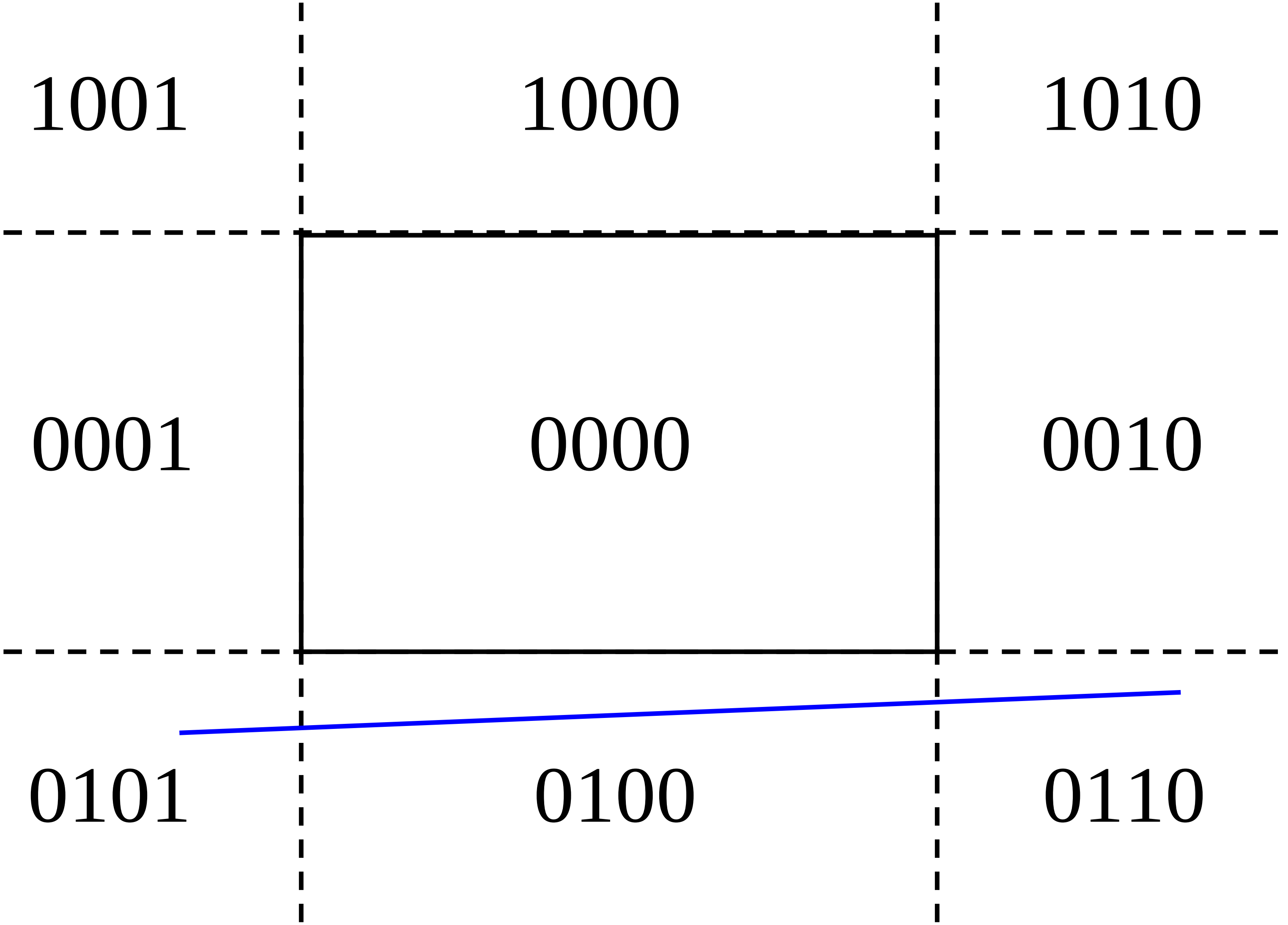
- Classify p_0, p_1 using region codes c_0, c_1
- If $c_0 \wedge c_1 \neq 0$, trivially reject
- If $c_0 \vee c_1 = 0$, trivially accept
- Otherwise reduce to trivial cases by splitting into two segments



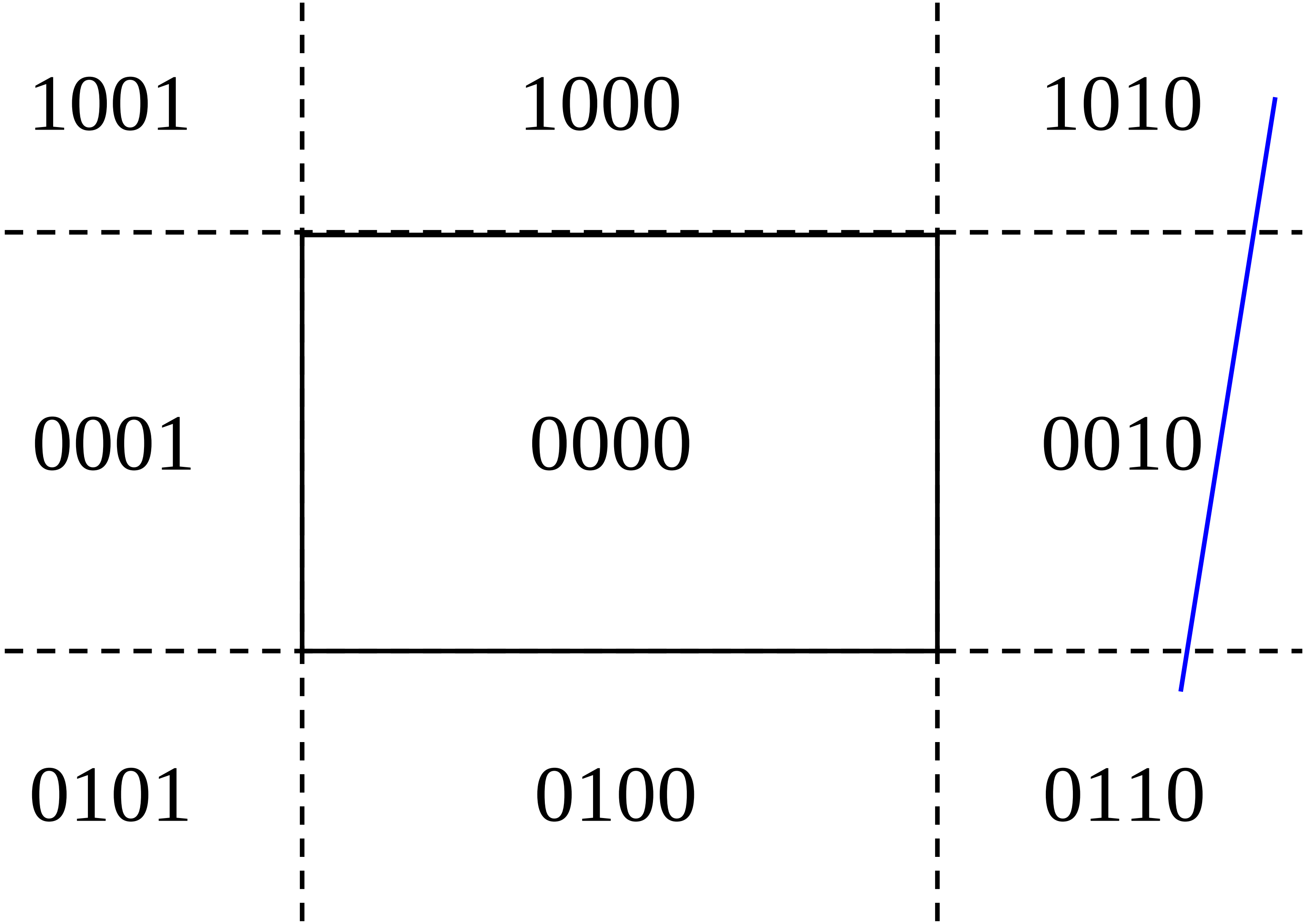
Cohen-Sutherland Algorithm



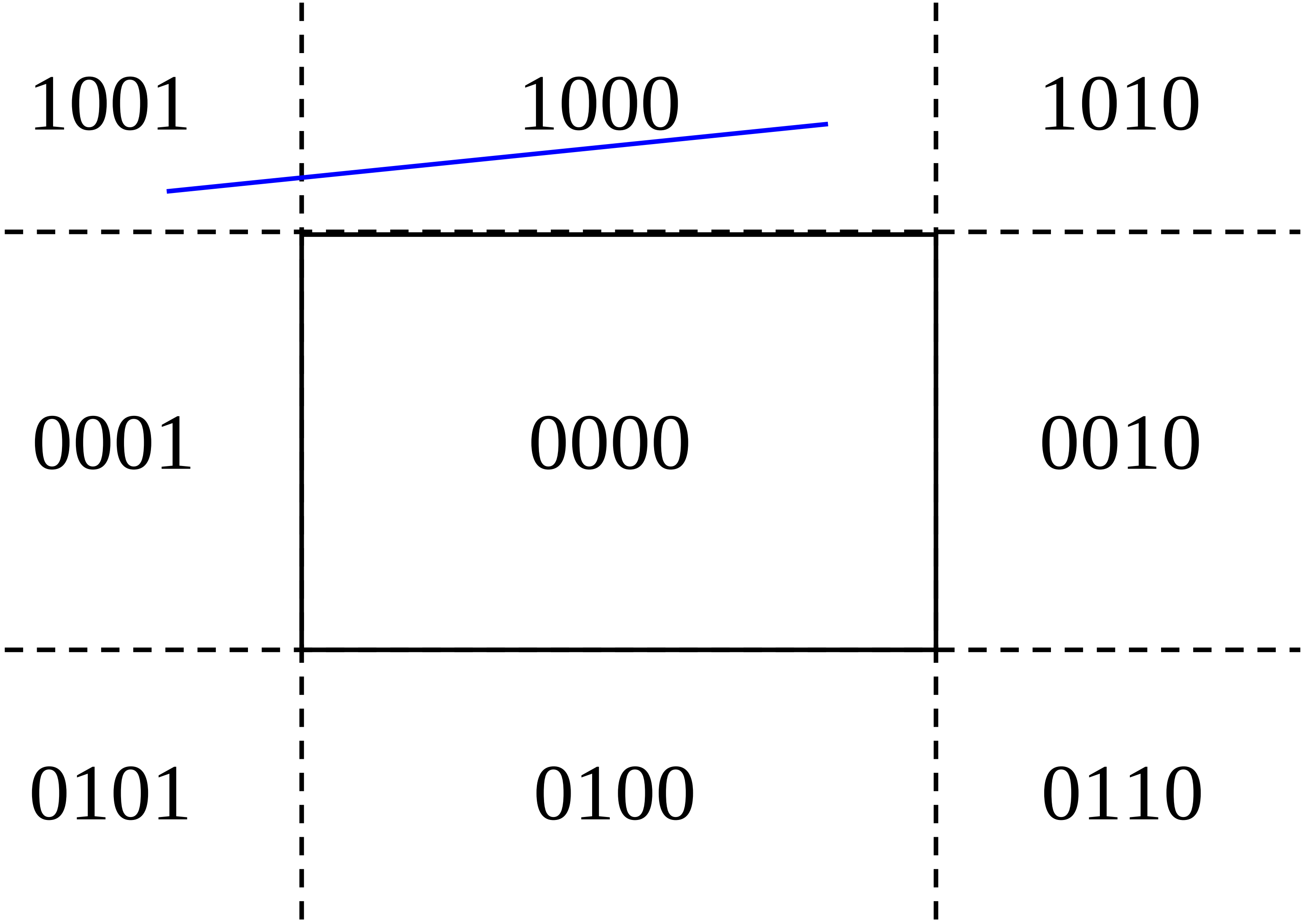
Cohen-Sutherland Algorithm



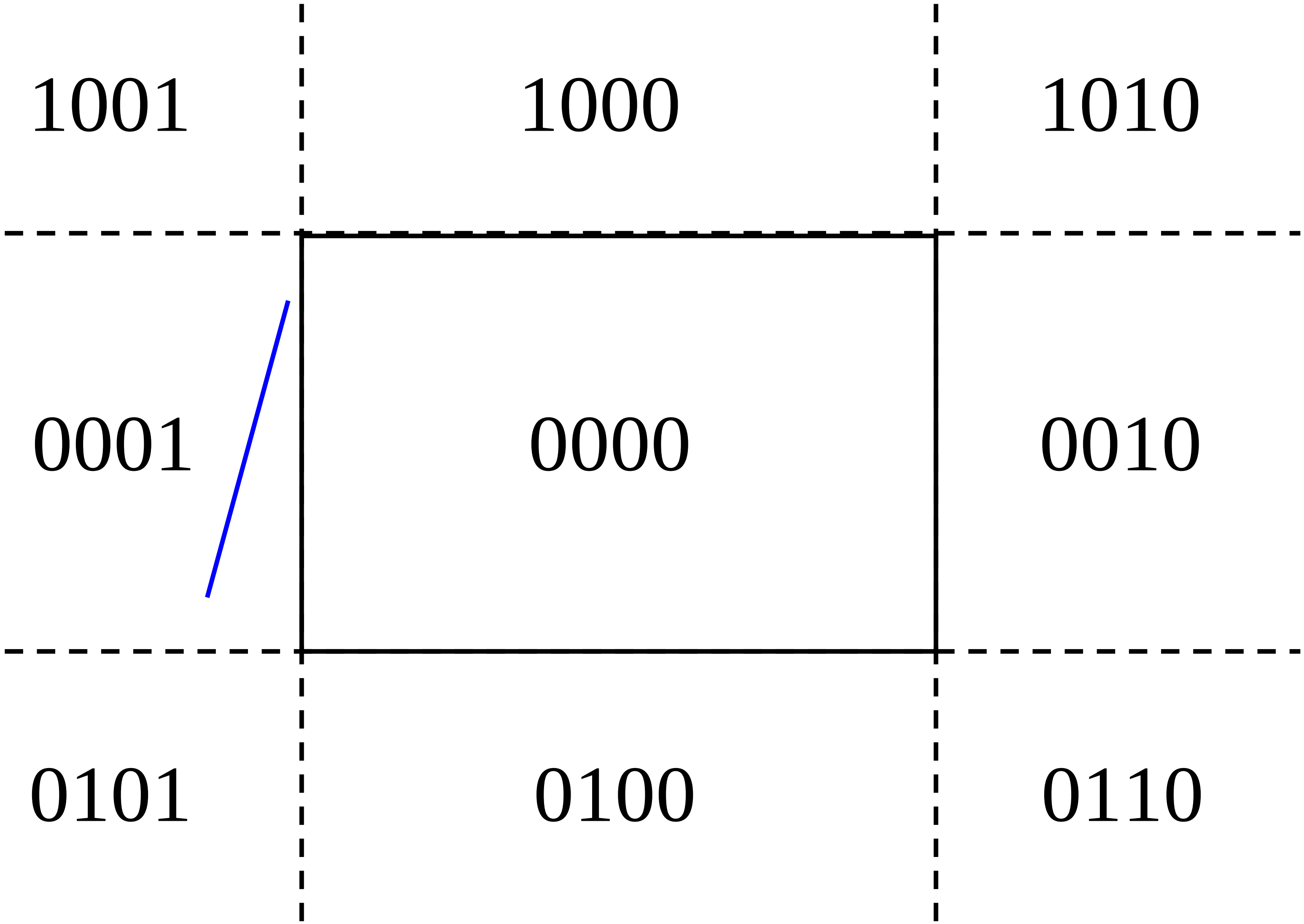
Cohen-Sutherland Algorithm



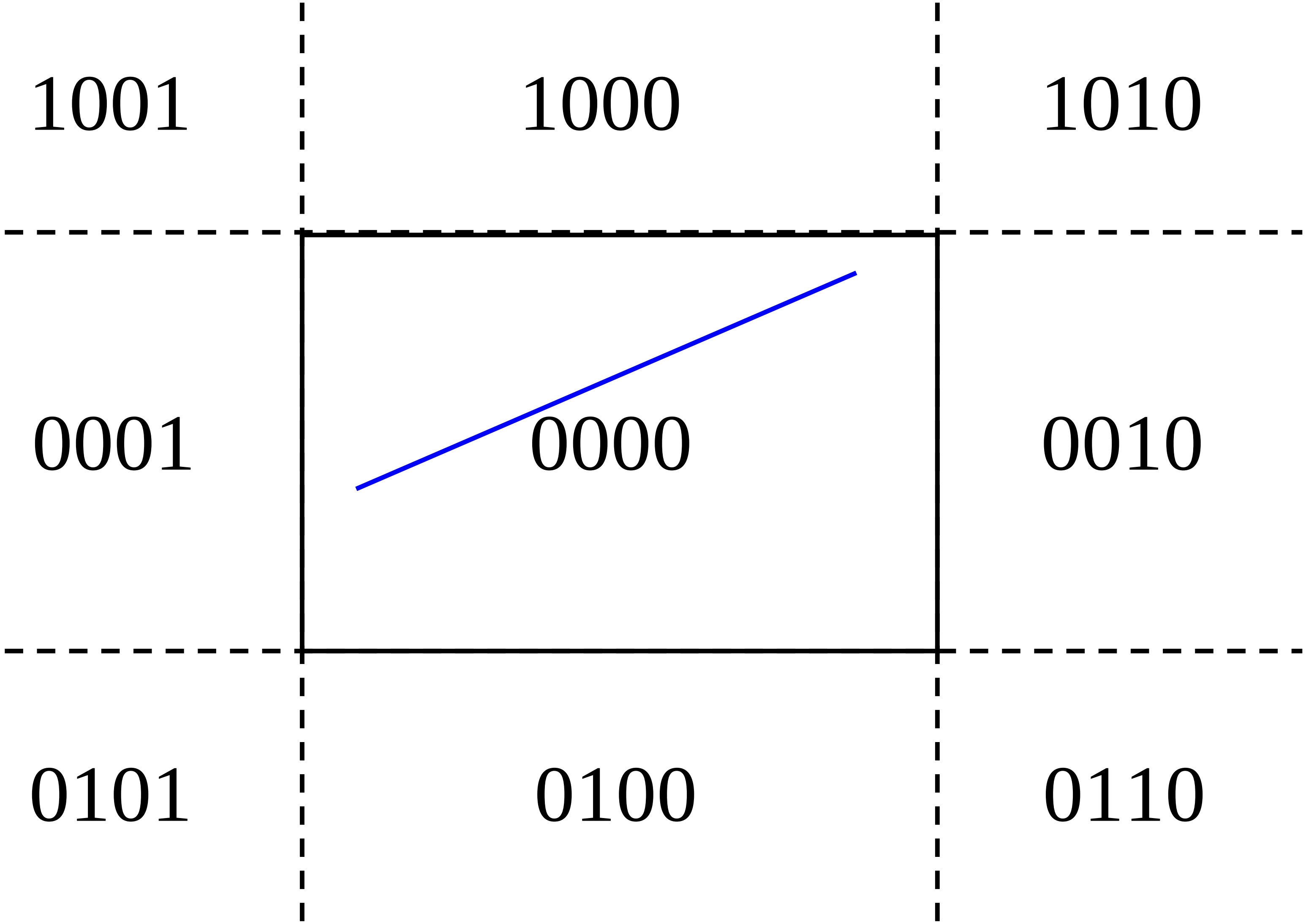
Cohen-Sutherland Algorithm



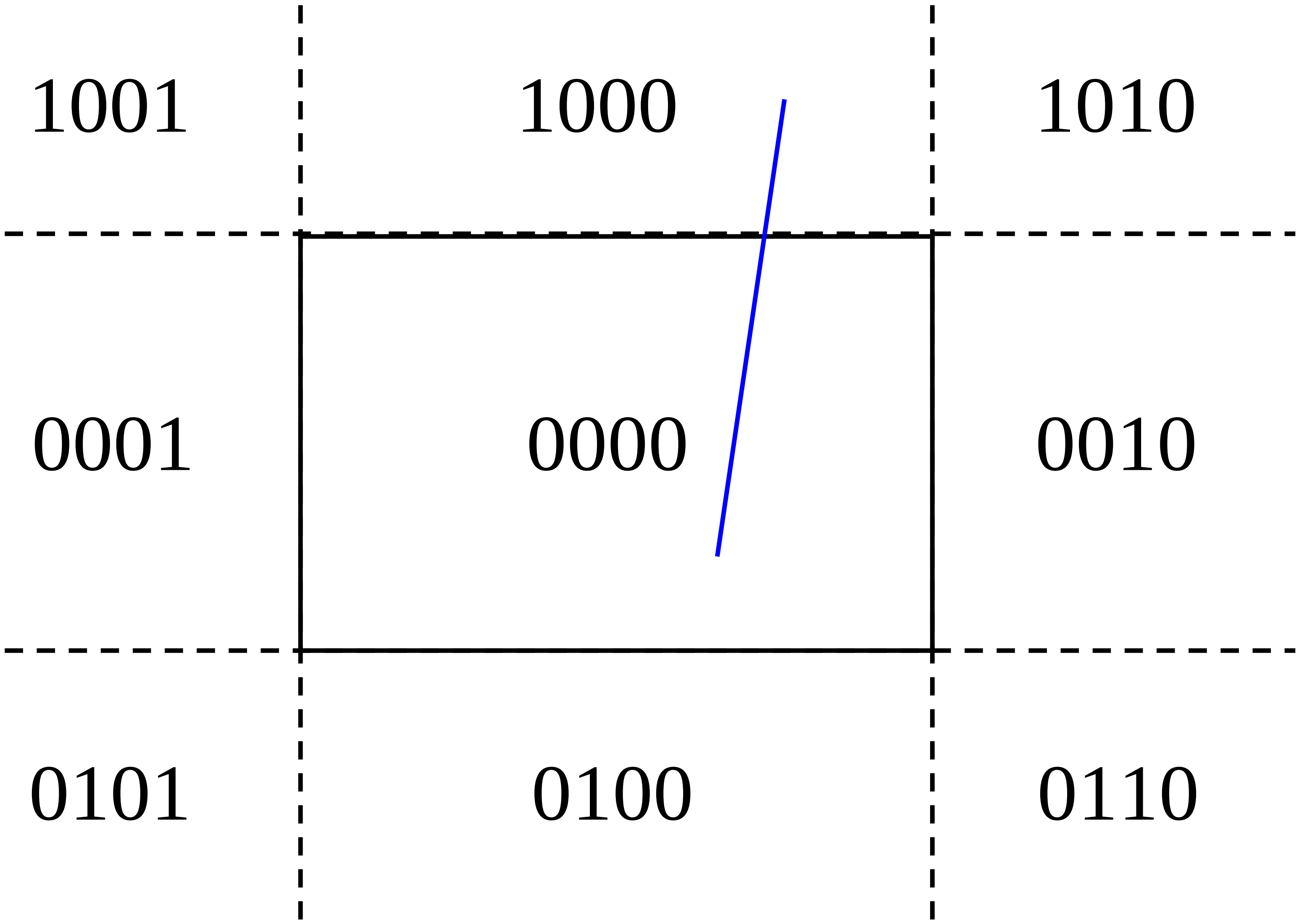
Cohen-Sutherland Algorithm



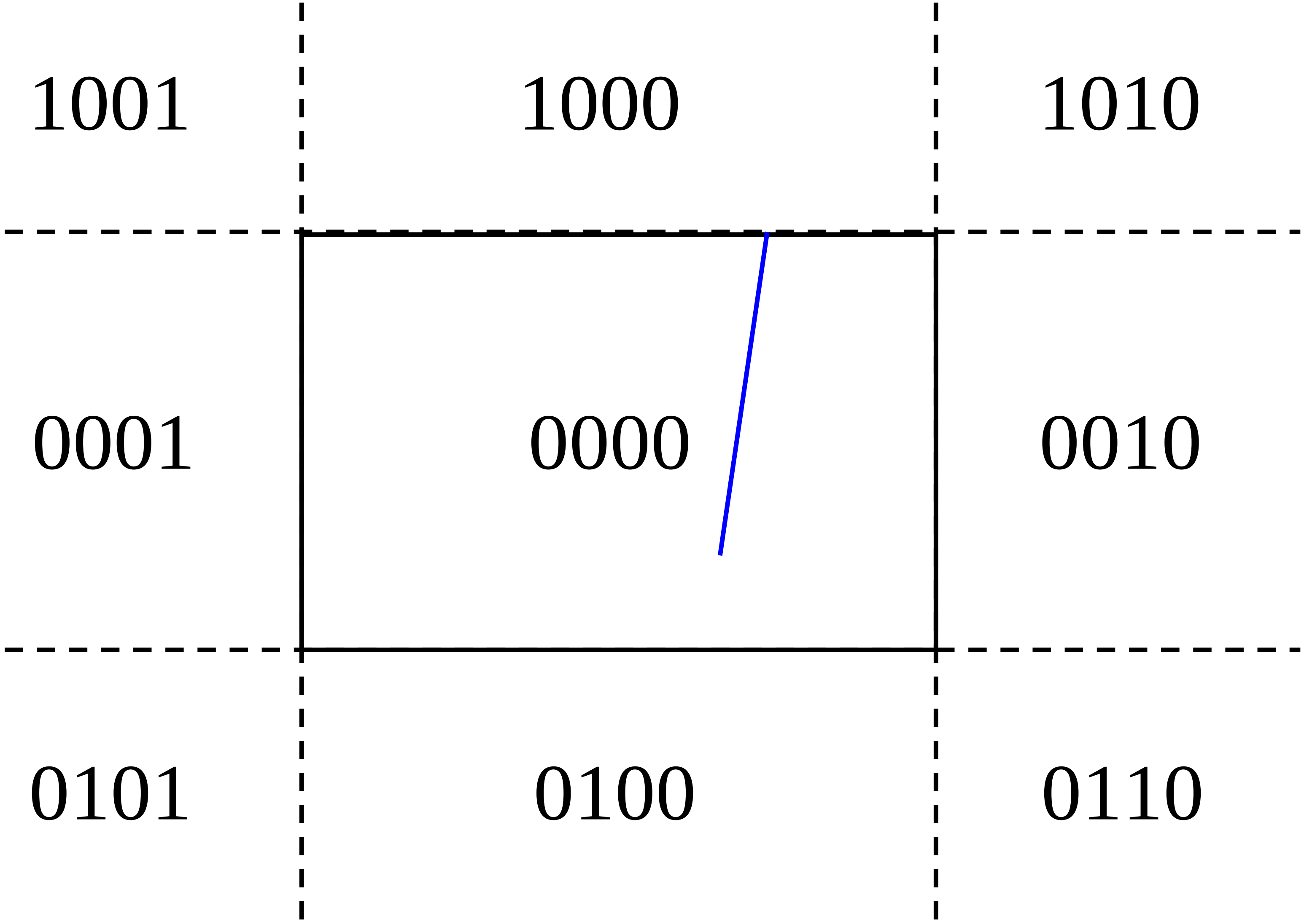
Cohen-Sutherland Algorithm



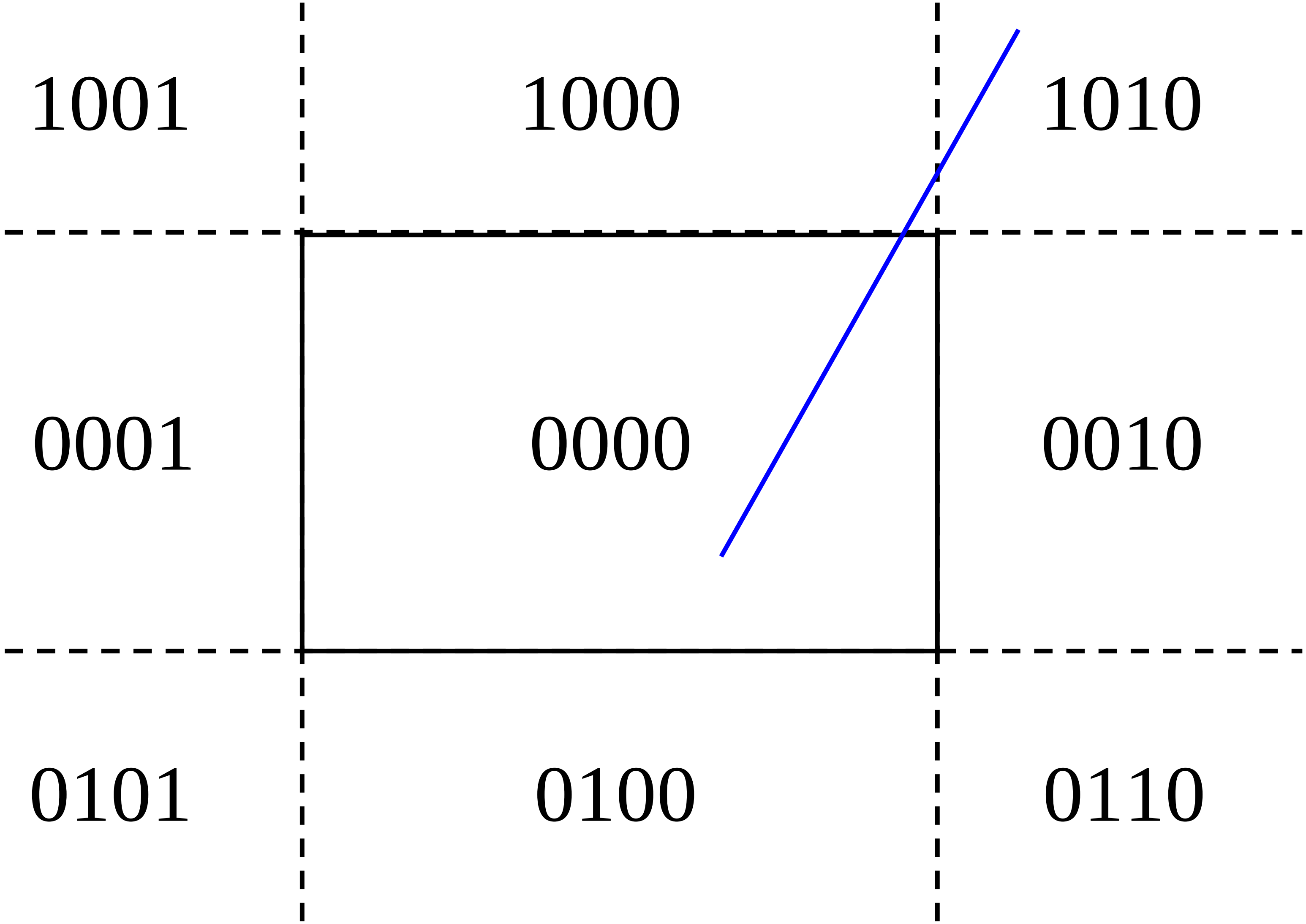
Cohen-Sutherland Algorithm



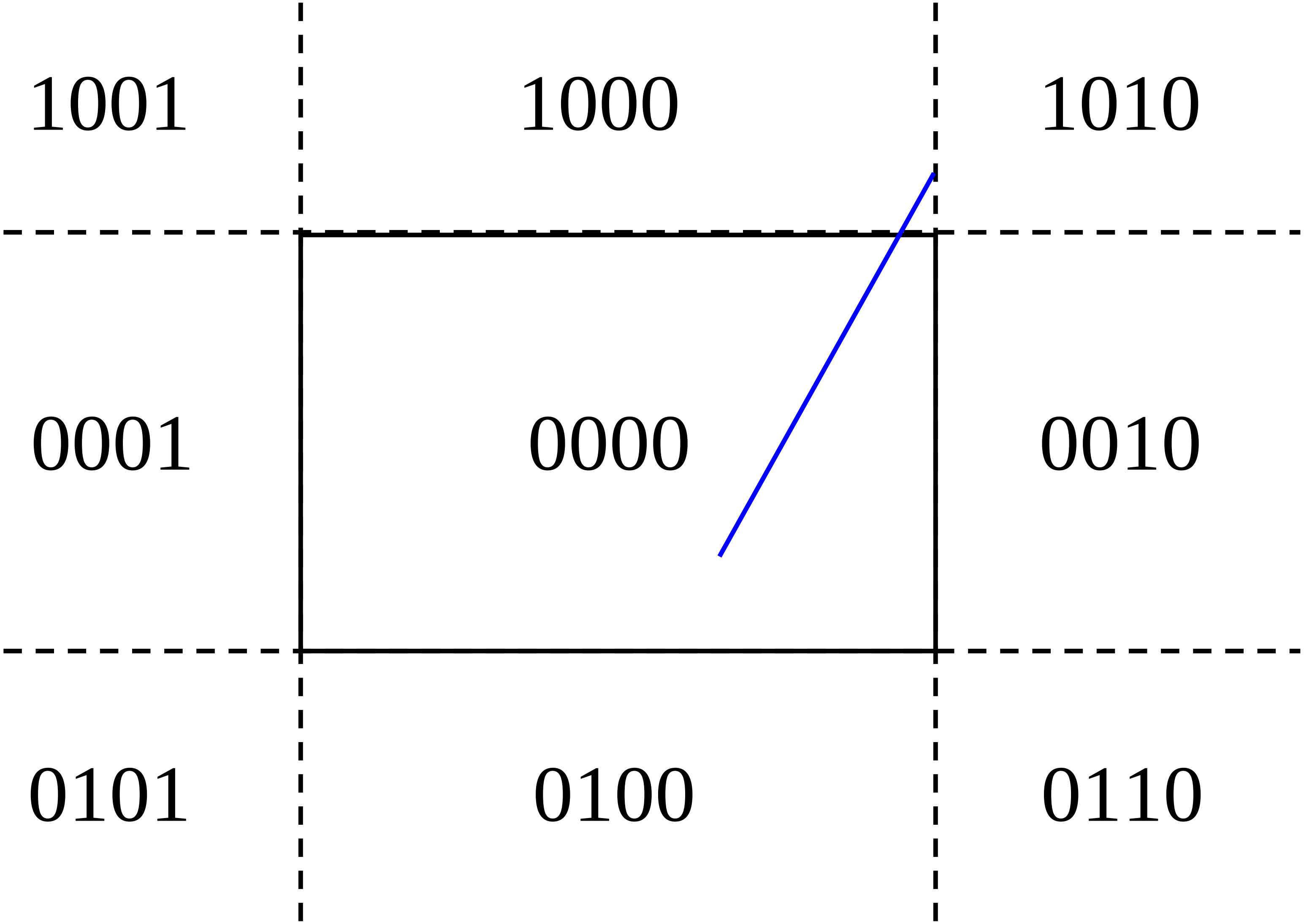
Cohen-Sutherland Algorithm



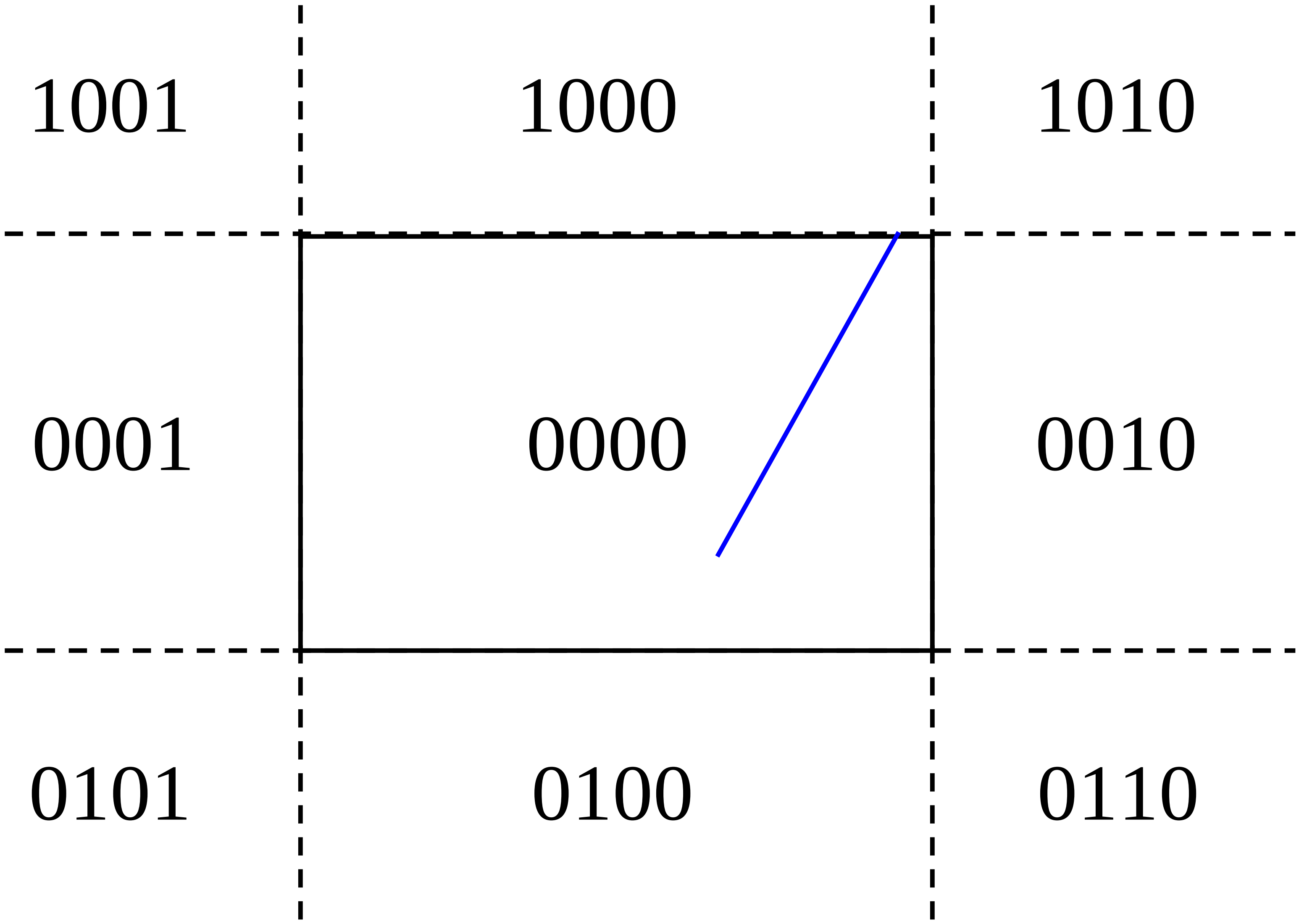
Cohen-Sutherland Algorithm



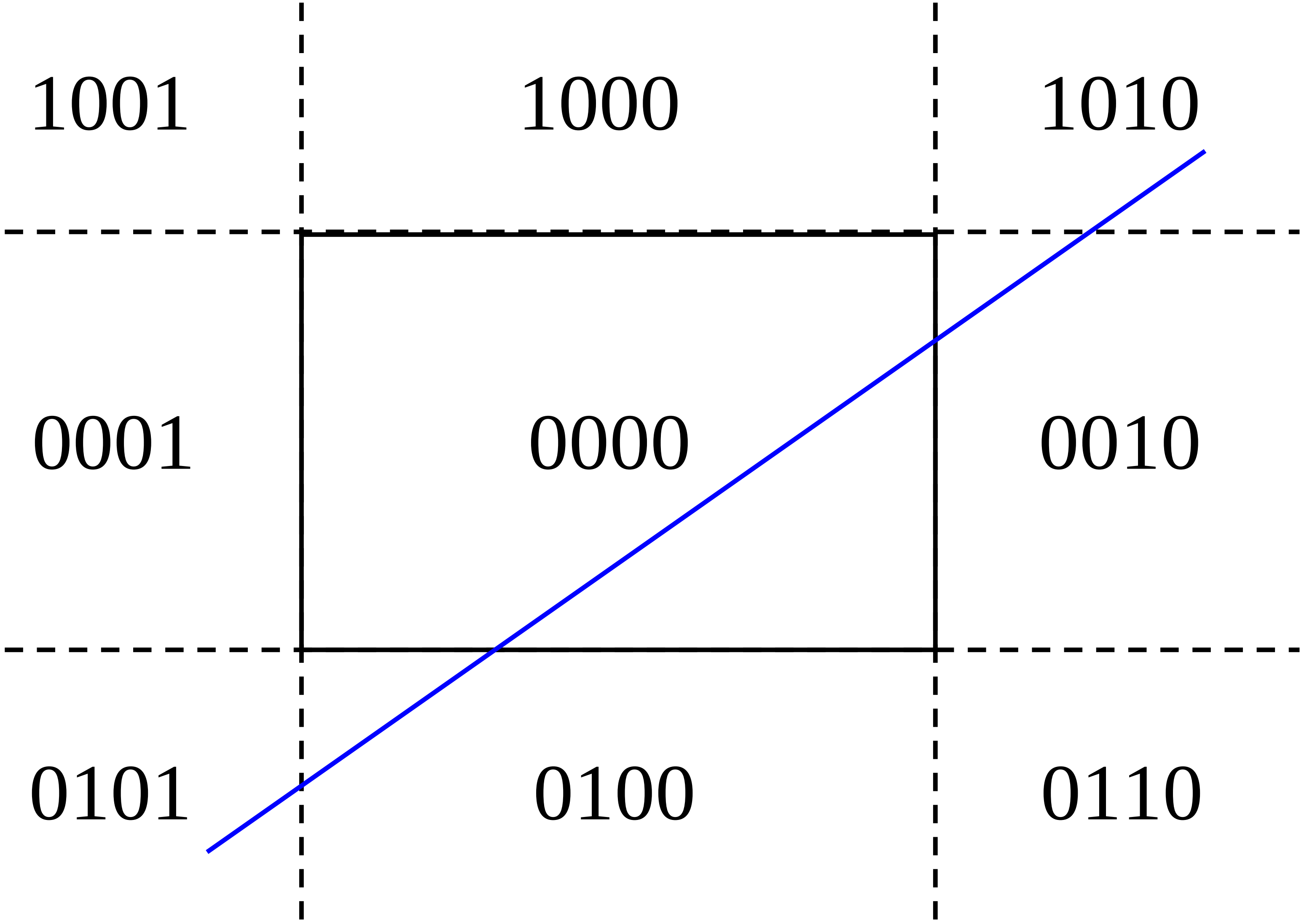
Cohen-Sutherland Algorithm



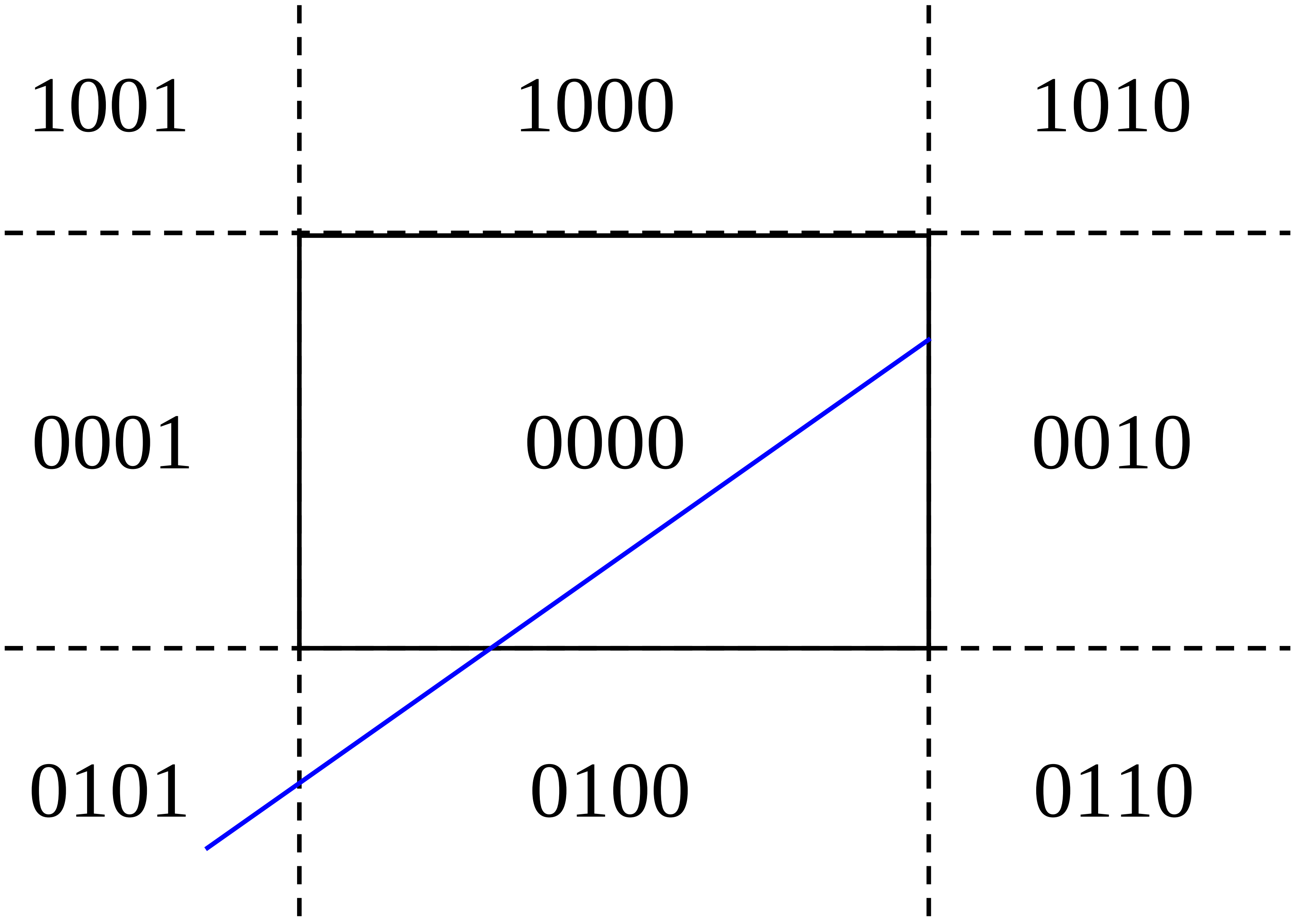
Cohen-Sutherland Algorithm



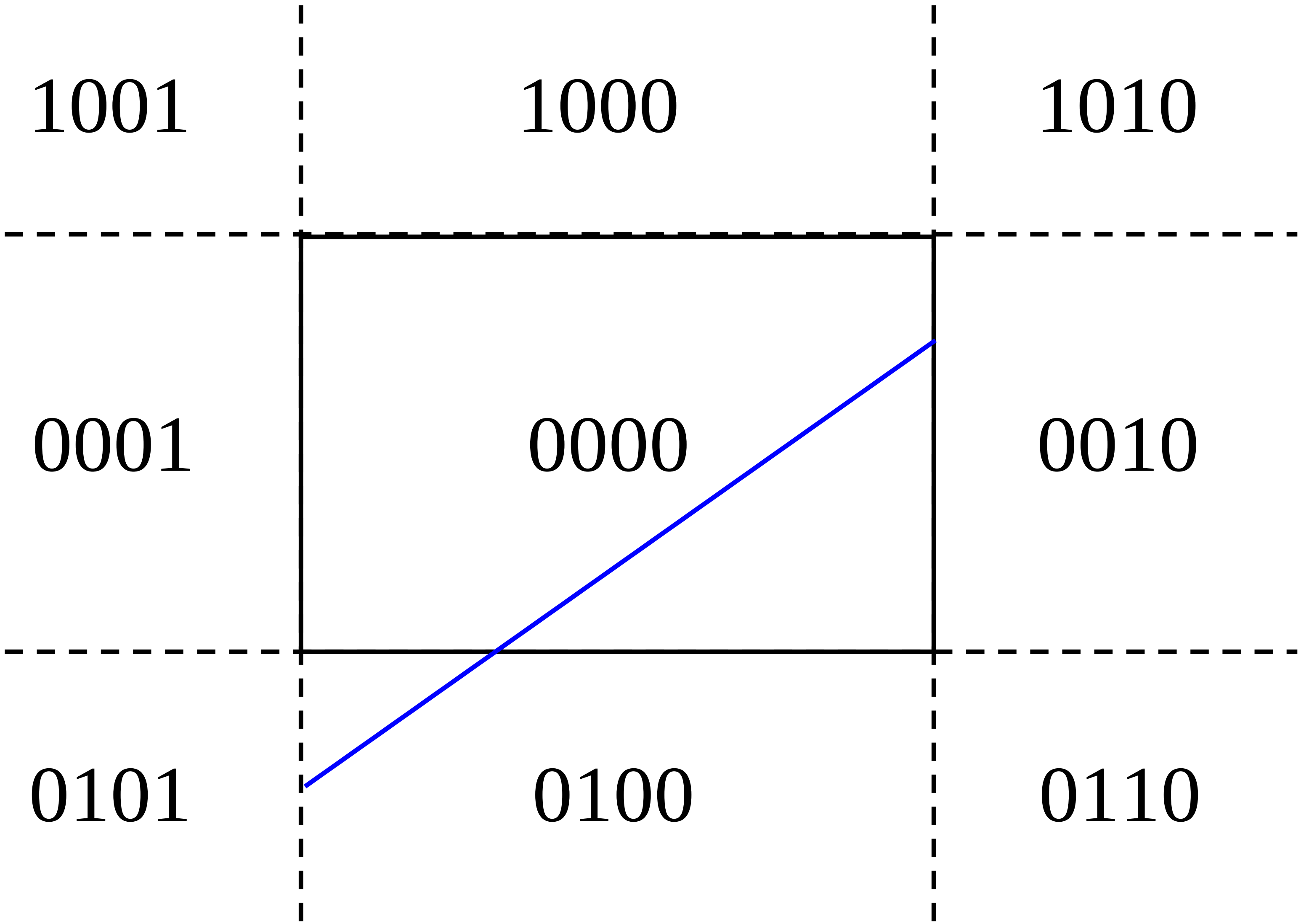
Cohen-Sutherland Algorithm



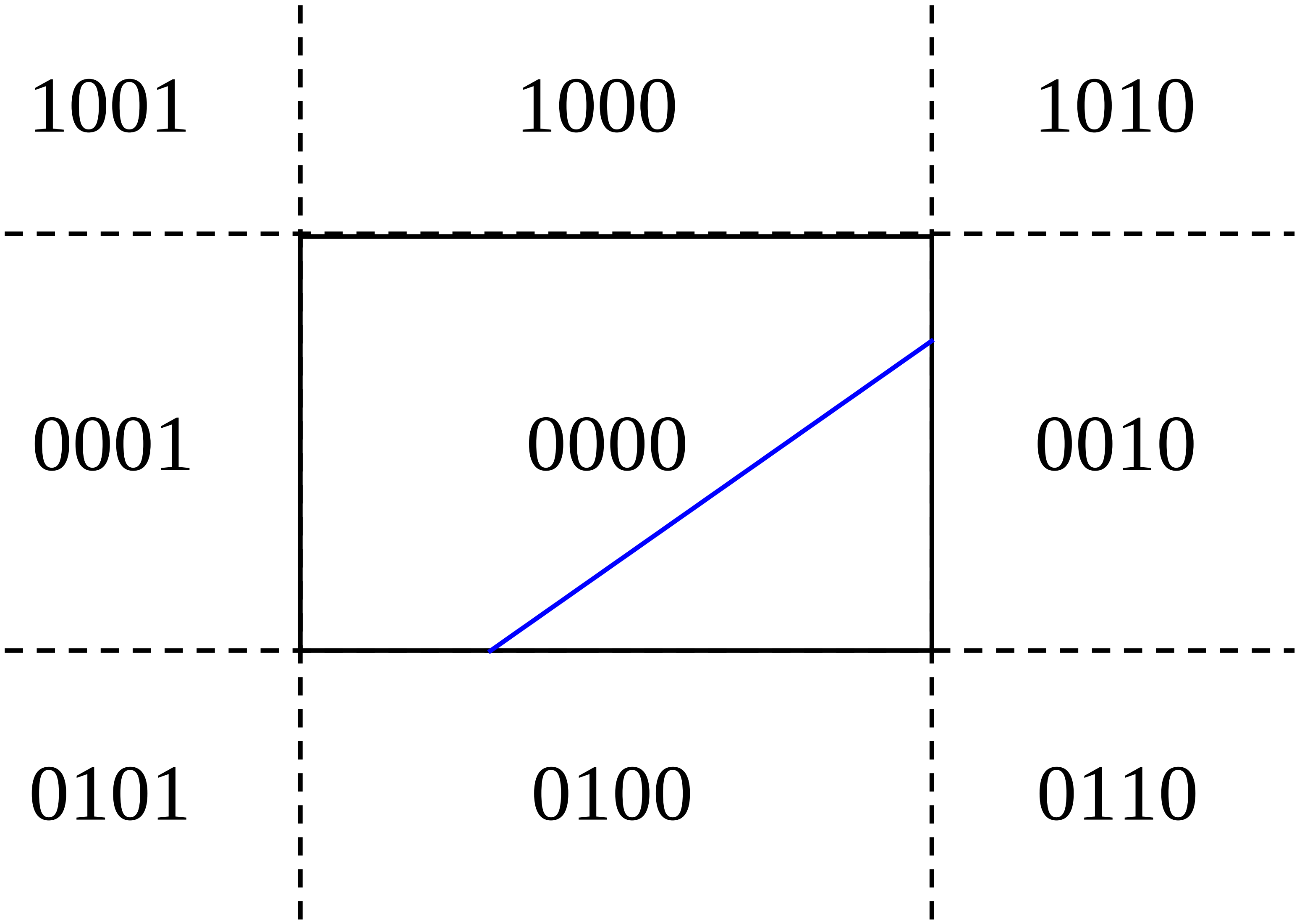
Cohen-Sutherland Algorithm



Cohen-Sutherland Algorithm



Cohen-Sutherland Algorithm



Liang-Barsky Algorithm

- Uses parametric form of line for clipping
- Lines are oriented

  Classify lines as moving inside to out or outside to in
- Don't find actual intersection points

  Find parameter values on line to draw

Liang-Barsky Algorithm

- Initialize interval to $[t_{min}, t_{max}]=[0,1]$
- For each boundary

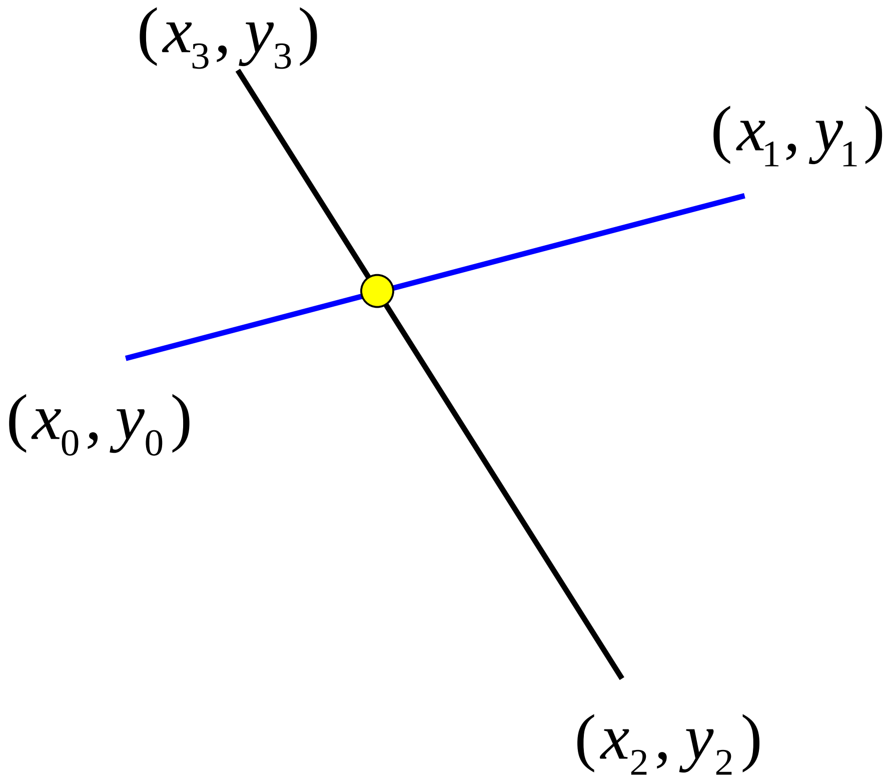
  Find parametric intersection t with boundary

  If moving in to out, $t_{max} = \min(t_{max}, t)$

  else $t_{min} = \max(t_{min}, t)$

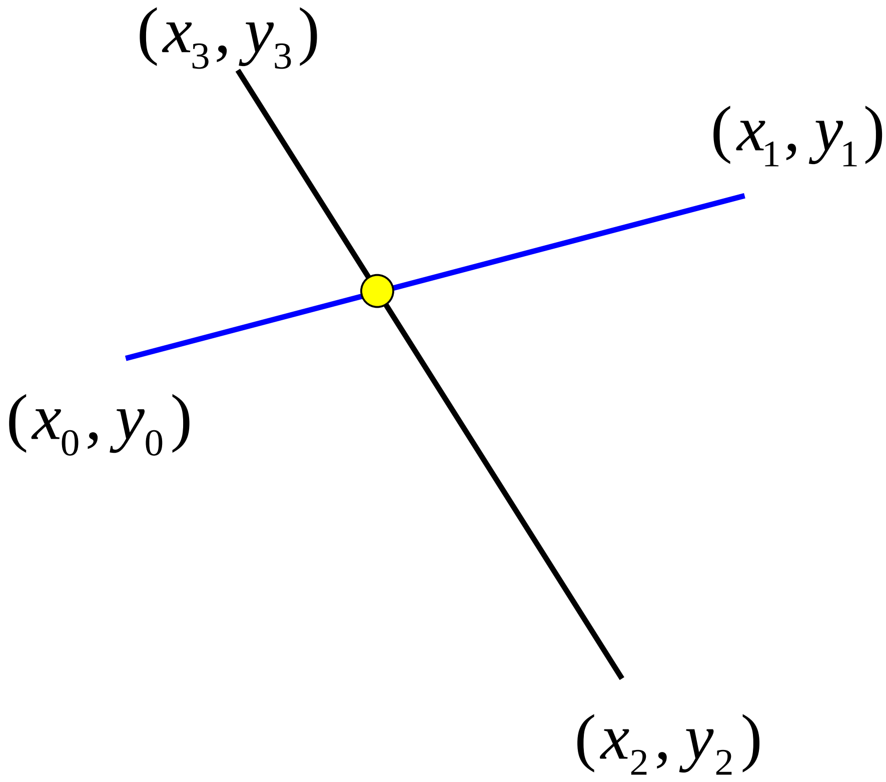
  If $t_{min} > t_{max}$, reject line

Intersecting Two Parametric Lines



Intersecting Two Parametric Lines

$$x(t) = x_0 + (x_1 - x_0)t$$
$$y(t) = y_0 + (y_1 - y_0)t$$
$$0 \le t \le 1$$

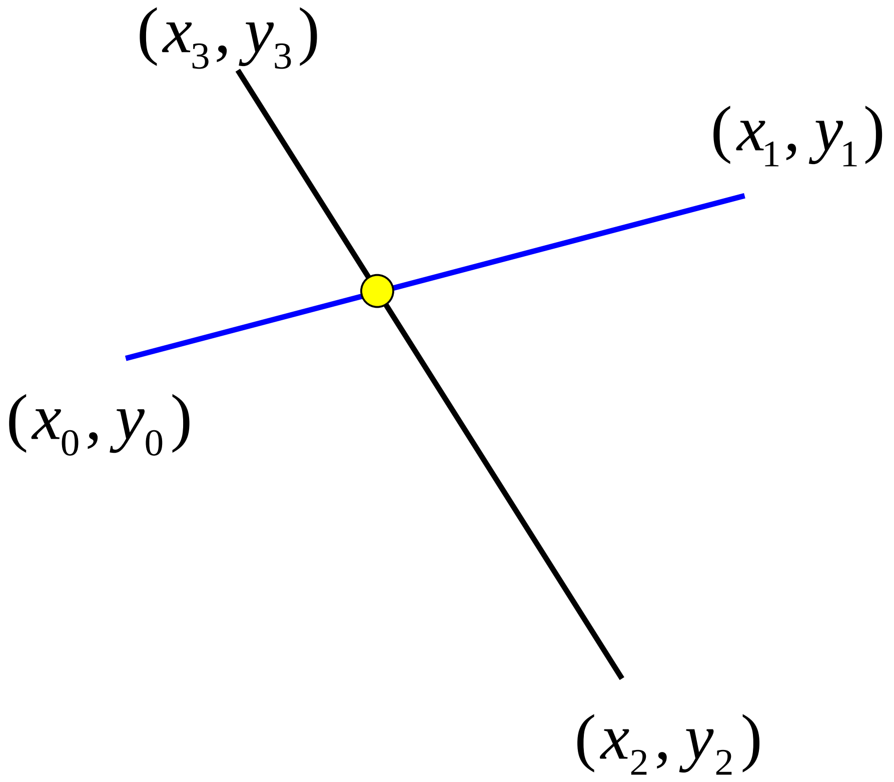


Intersecting Two Parametric Lines

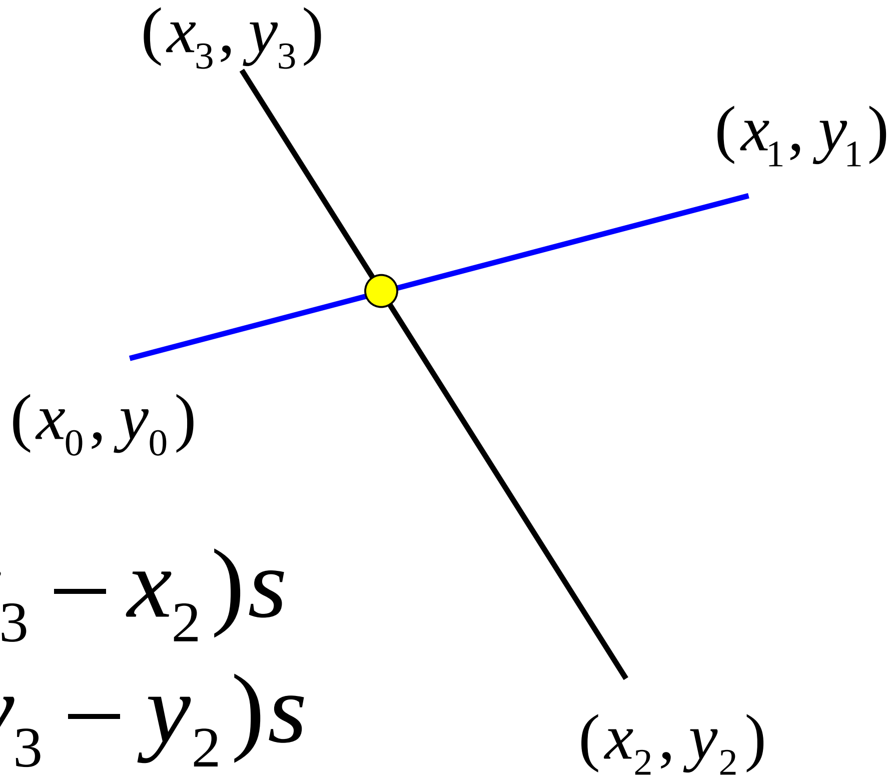
$$x(t) = x_0 + (x_1 - x_0)t$$
$$y(t) = y_0 + (y_1 - y_0)t$$

$$x(0) = x_0$$
$$y(0) = y_0$$

$$x(1) = x_1$$
$$y(1) = y_1$$

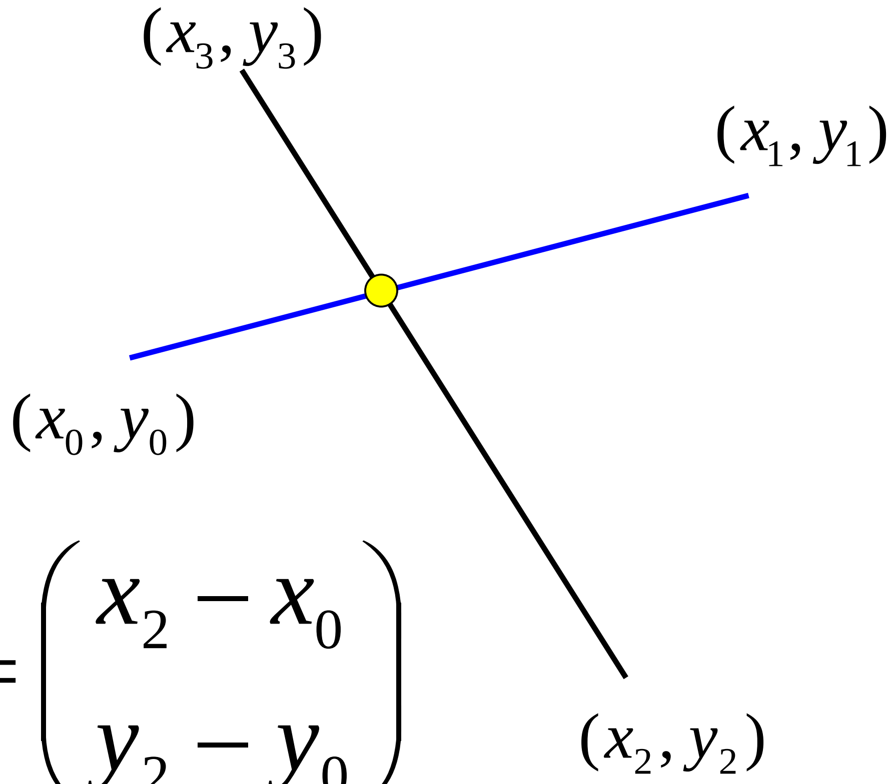


Intersecting Two Parametric Lines



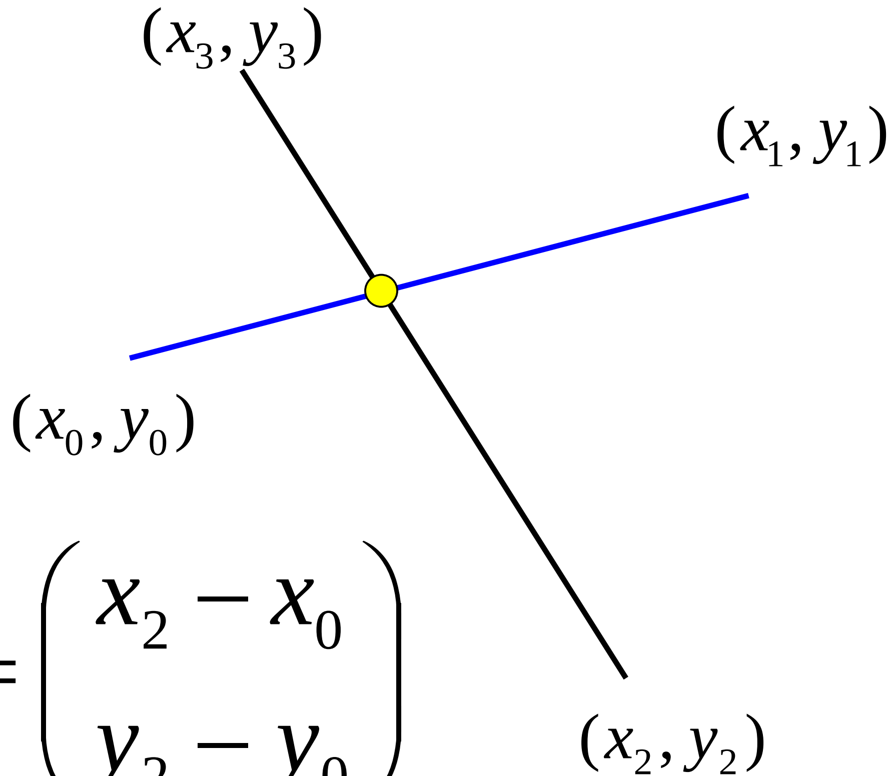
$$x_0 + (x_1 - x_0)t = x_2 + (x_3 - x_2)s$$
$$y_0 + (y_1 - y_0)t = y_2 + (y_3 - y_2)s$$

Intersecting Two Parametric Lines



$$\begin{pmatrix} x_1 - x_0 & x_2 - x_3 \\ y_1 - y_0 & y_2 - y_3 \end{pmatrix} \begin{pmatrix} t \\ s \end{pmatrix} = \begin{pmatrix} x_2 - x_0 \\ y_2 - y_0 \end{pmatrix}$$

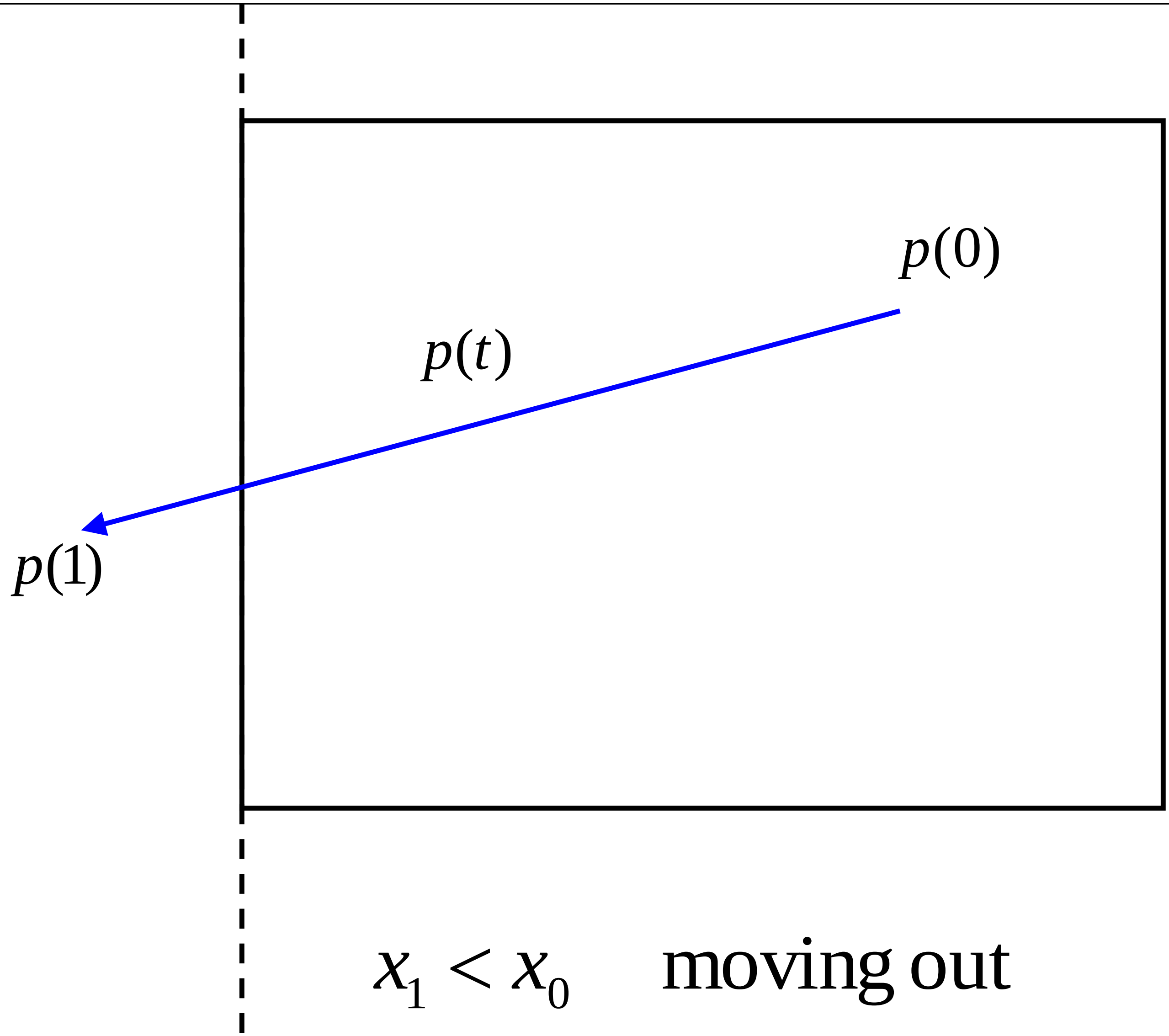
Intersecting Two Parametric Lines



$$\begin{pmatrix} x_1 - x_0 & x_2 - x_3 \\ y_1 - y_0 & y_2 - y_3 \end{pmatrix}\begin{pmatrix} t \\ s \end{pmatrix} = \begin{pmatrix} x_2 - x_0 \\ y_2 - y_0 \end{pmatrix}$$

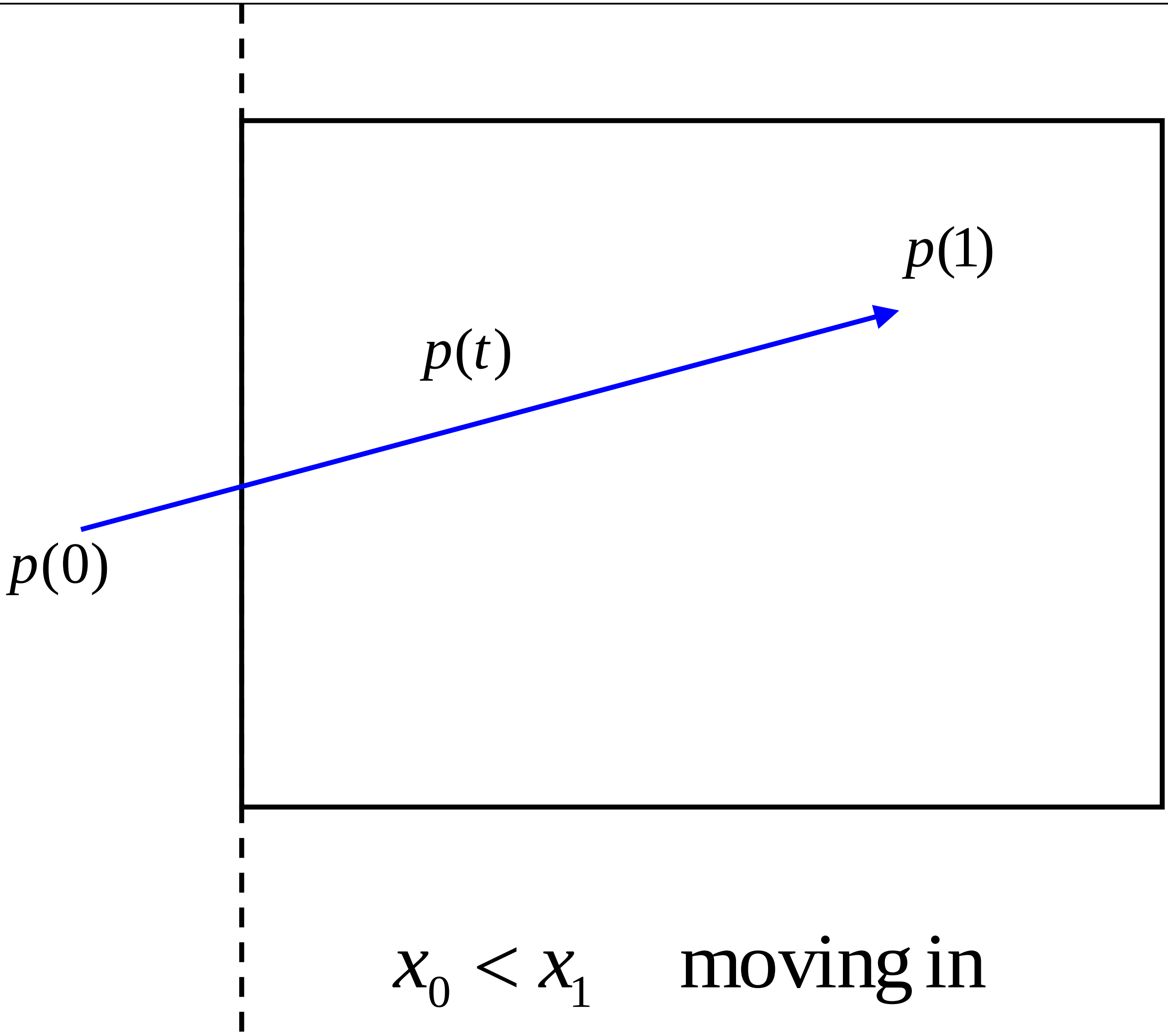
Substitute t or s back into equation to find intersection

Liang-Barsky: Classifying Lines



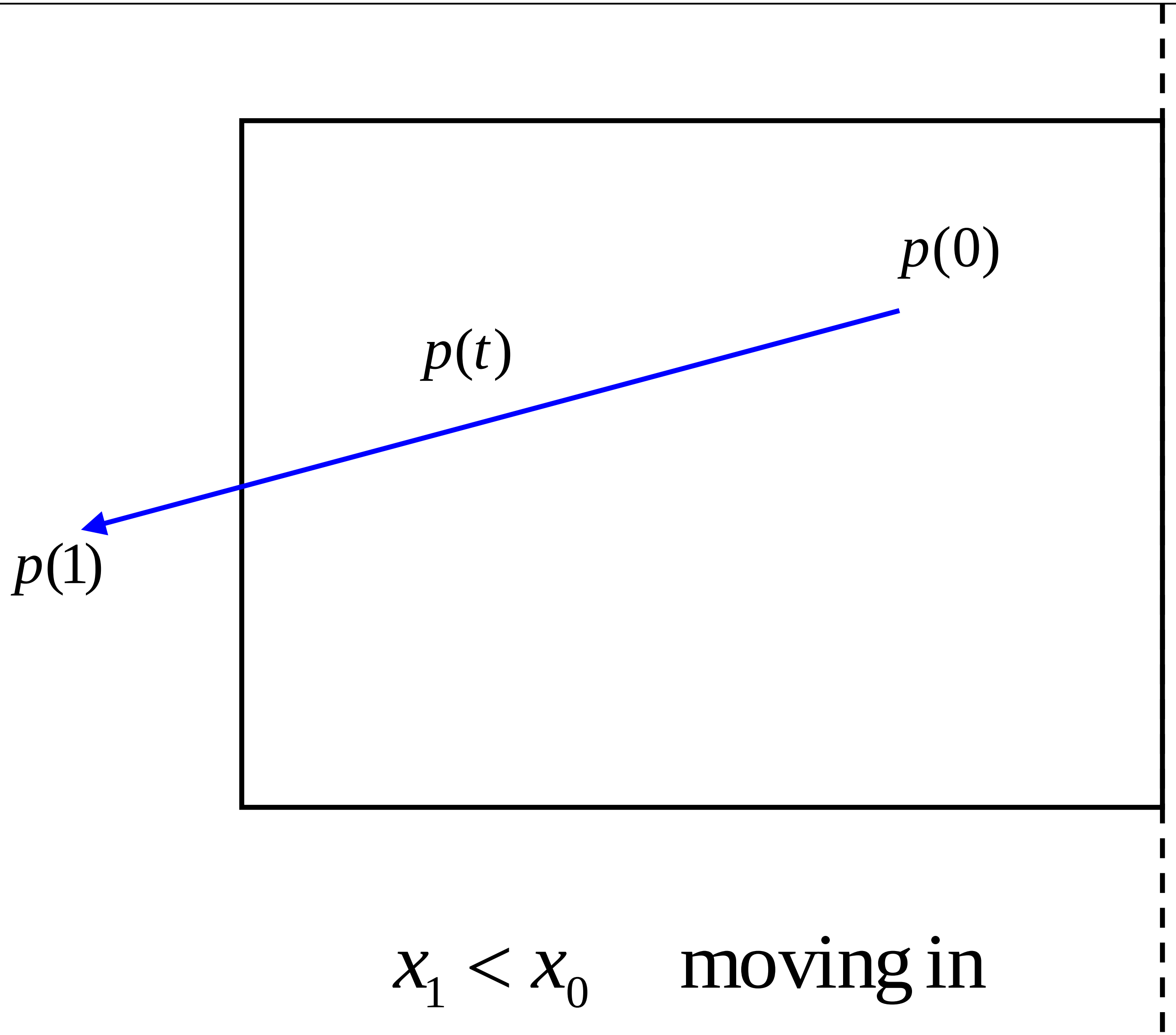
$x_1 < x_0$ moving out

Liang-Barsky: Classifying Lines



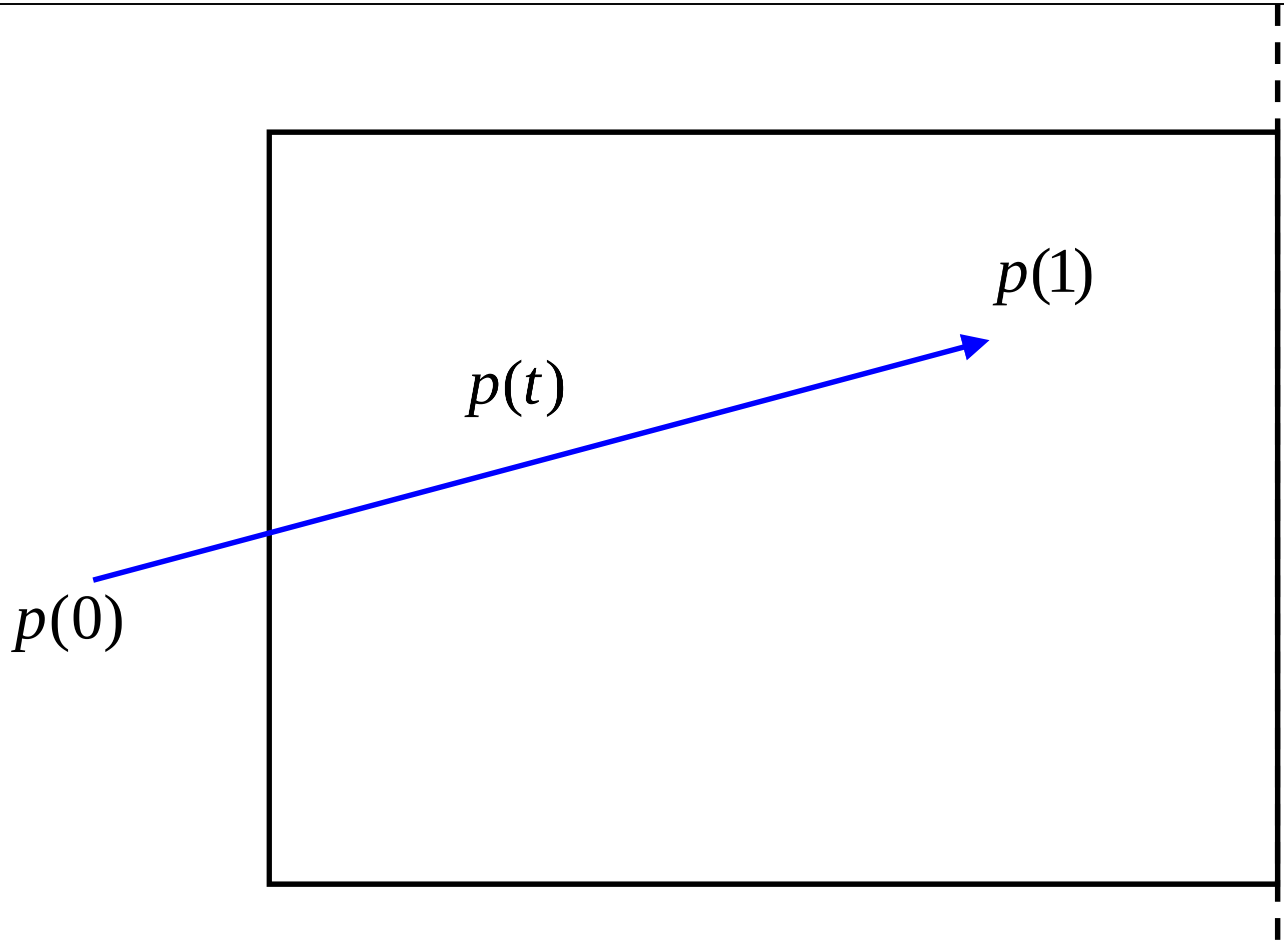
$x_0 < x_1$ moving in

Liang-Barsky: Classifying Lines



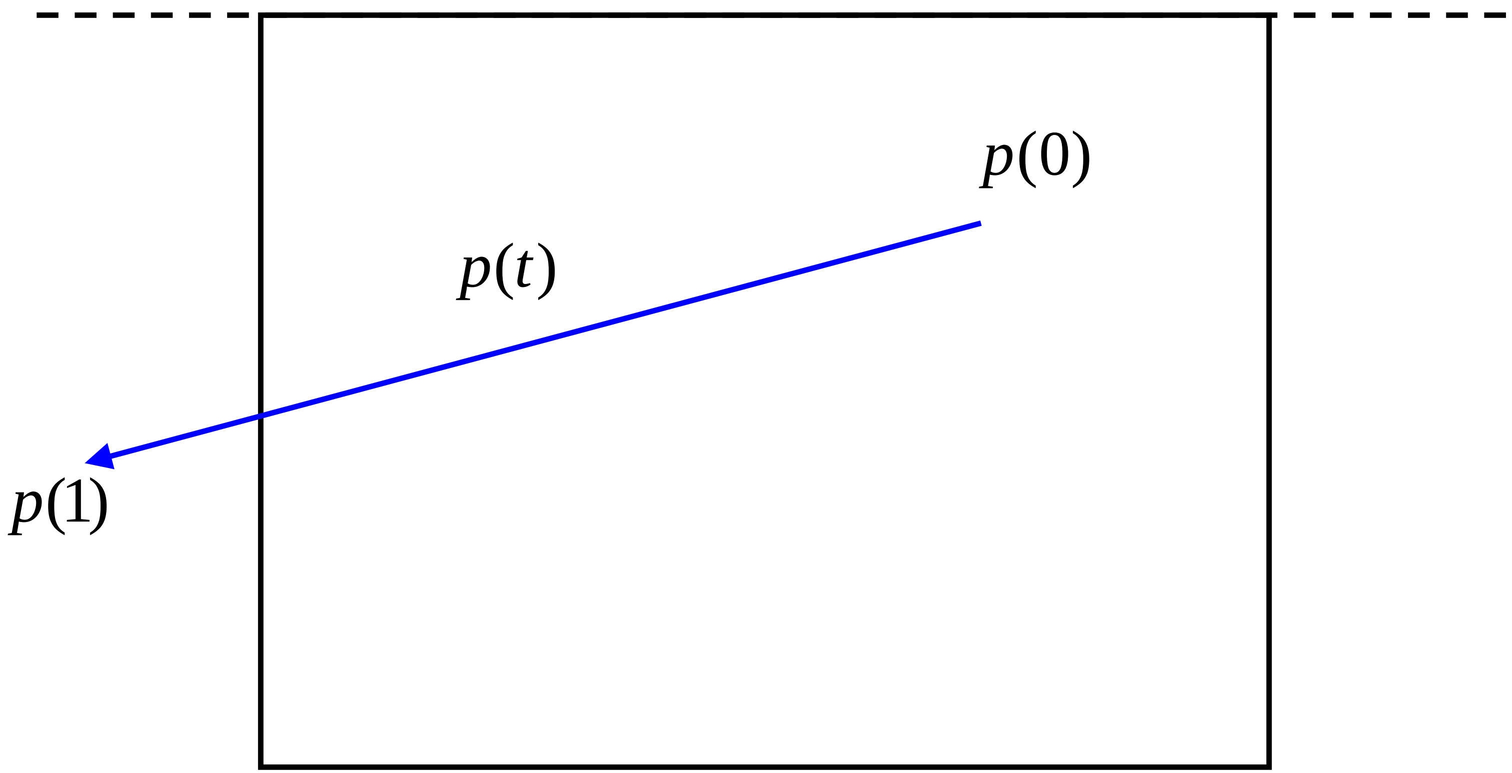
$x_1 < x_0$ moving in

Liang-Barsky: Classifying Lines



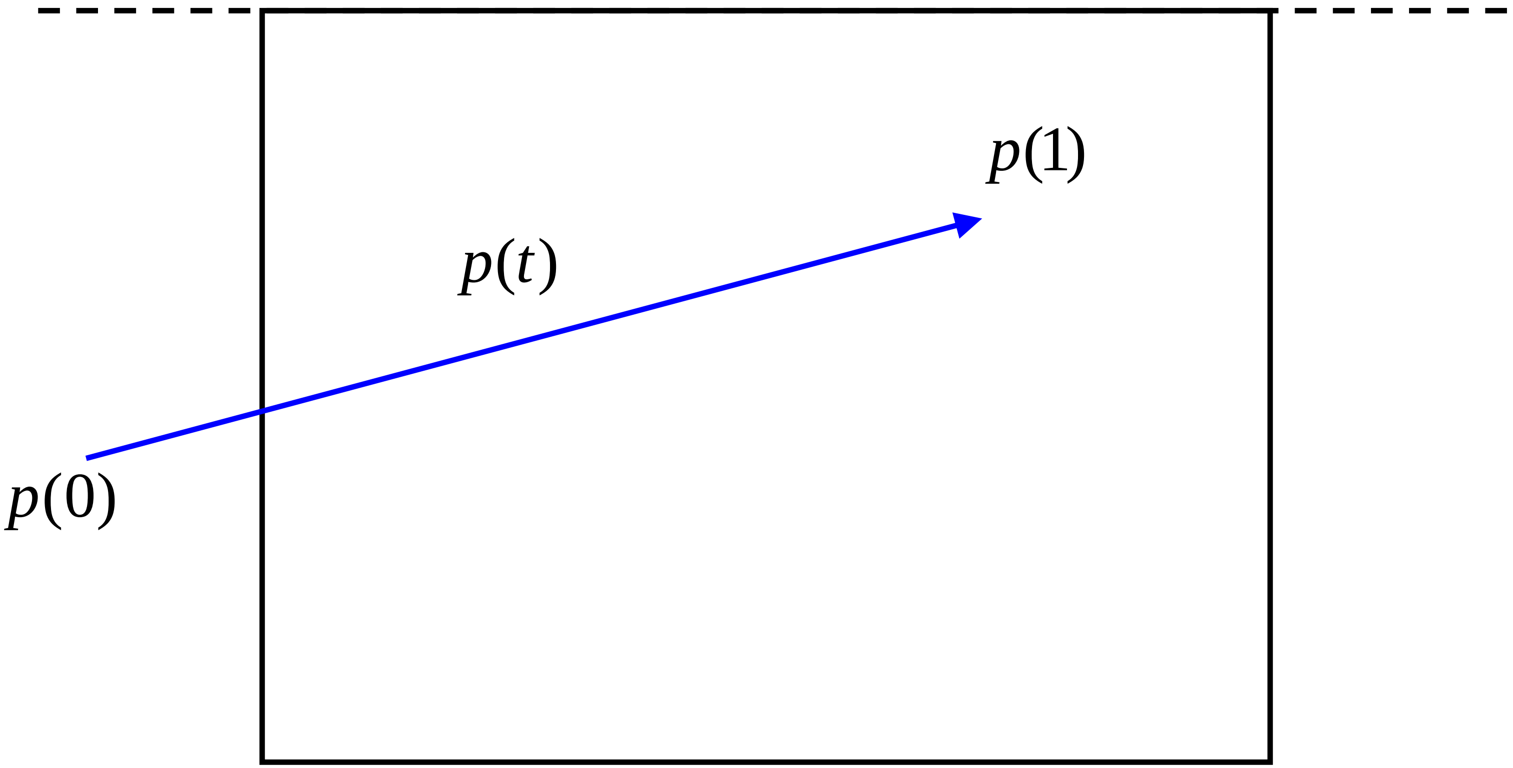
$x_0 < x_1$ moving out

Liang-Barsky: Classifying Lines



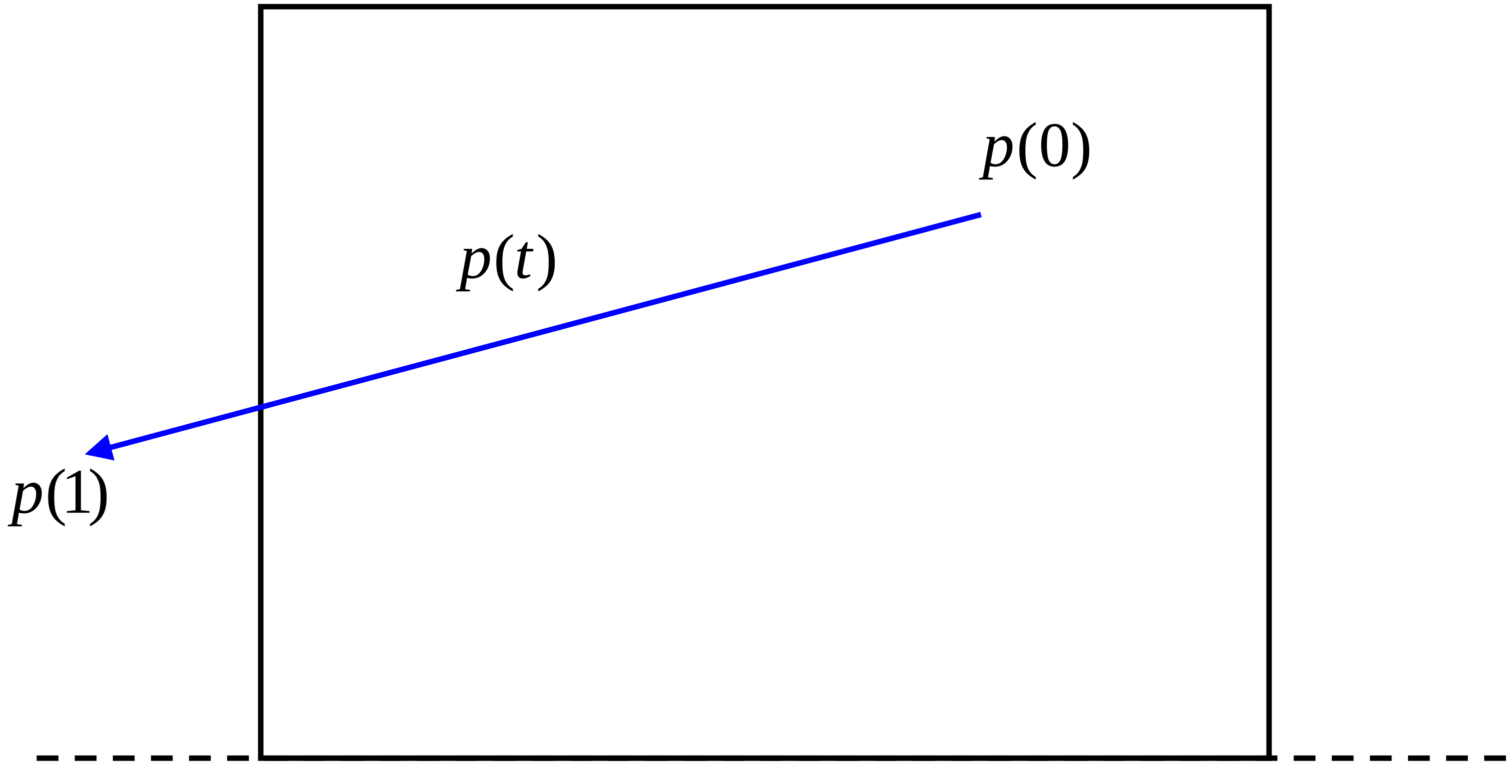
$y_1 < y_0$ moving in

Liang-Barsky: Classifying Lines



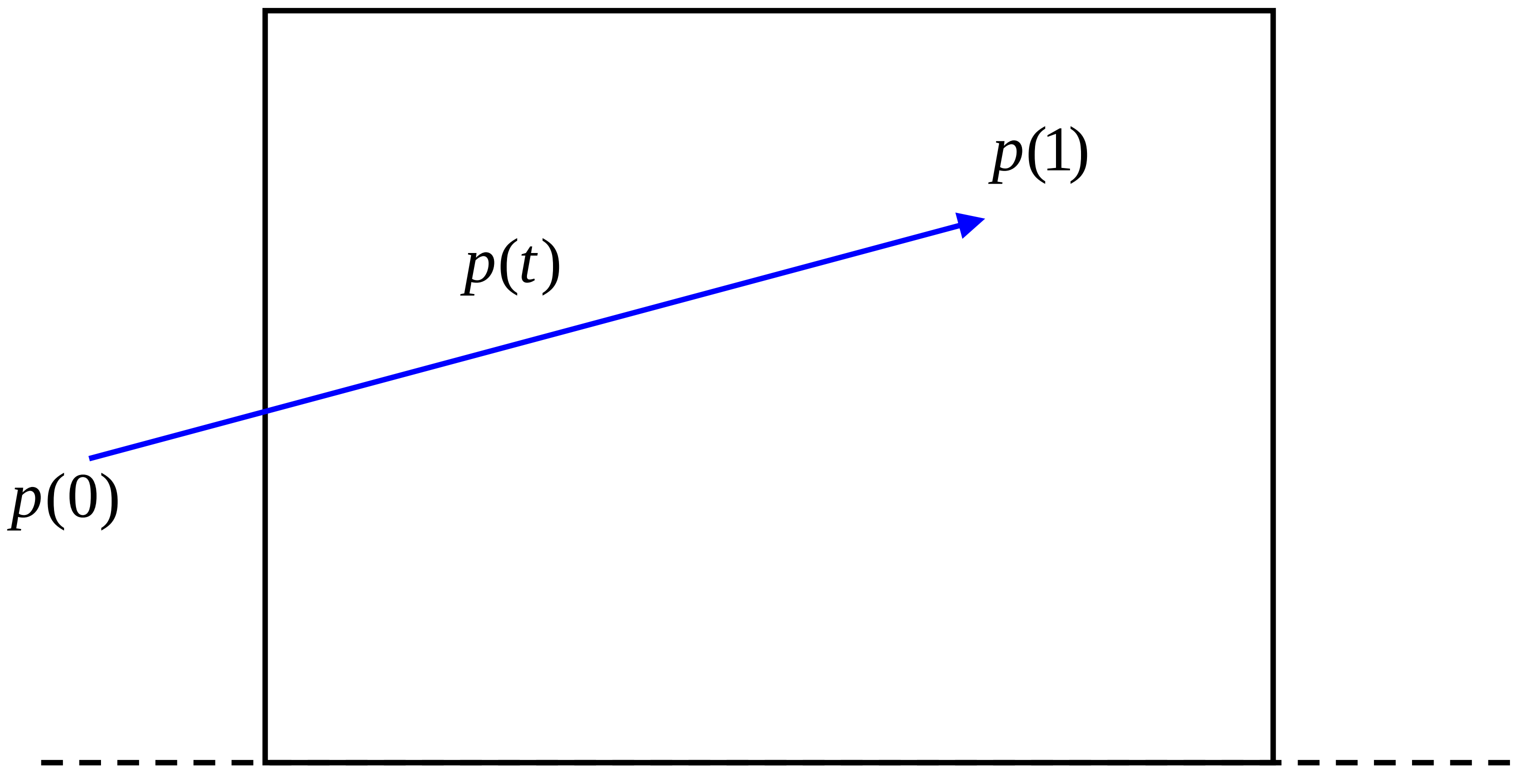
$y_0 < y_1$ moving out

Liang-Barsky: Classifying Lines



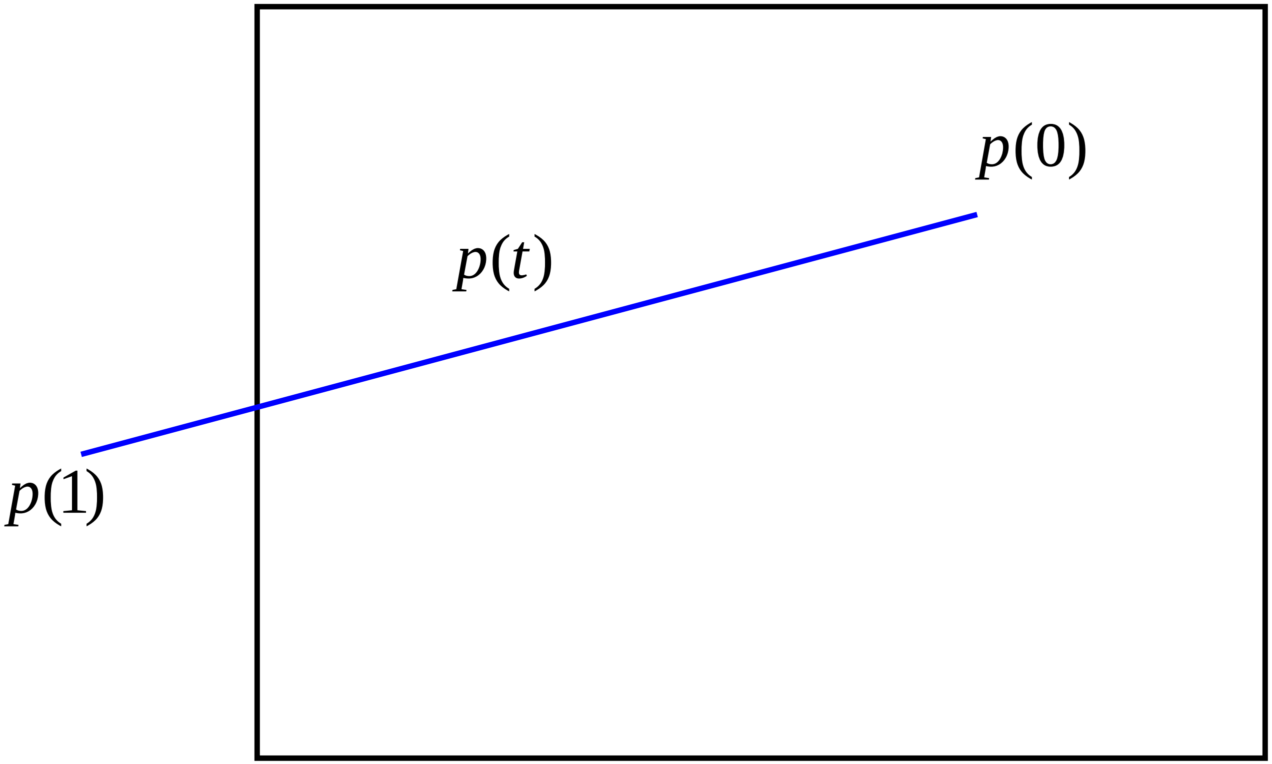
$y_1 < y_0$ moving out

Liang-Barsky: Classifying Lines

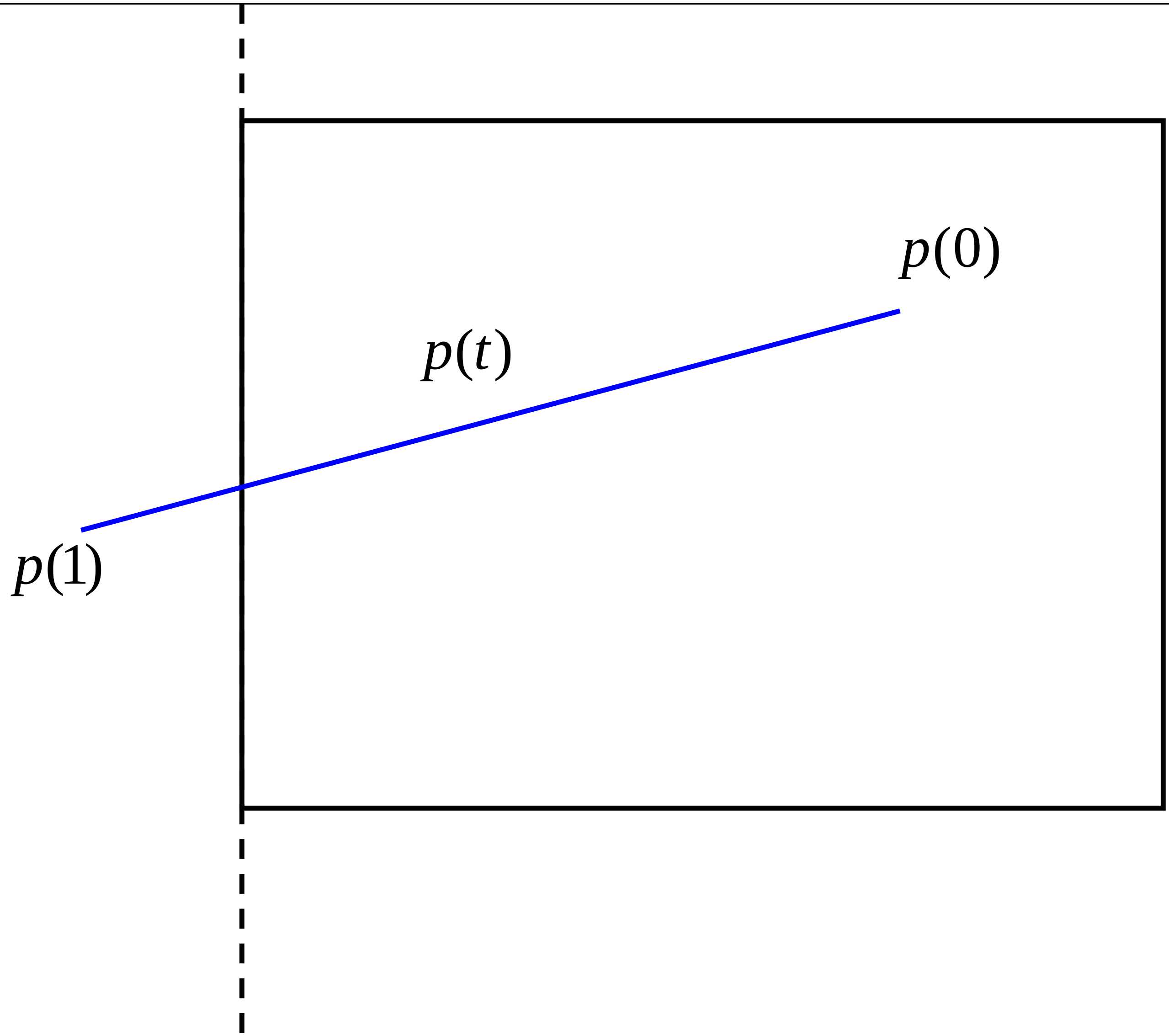


$y_0 < y_1$ moving in

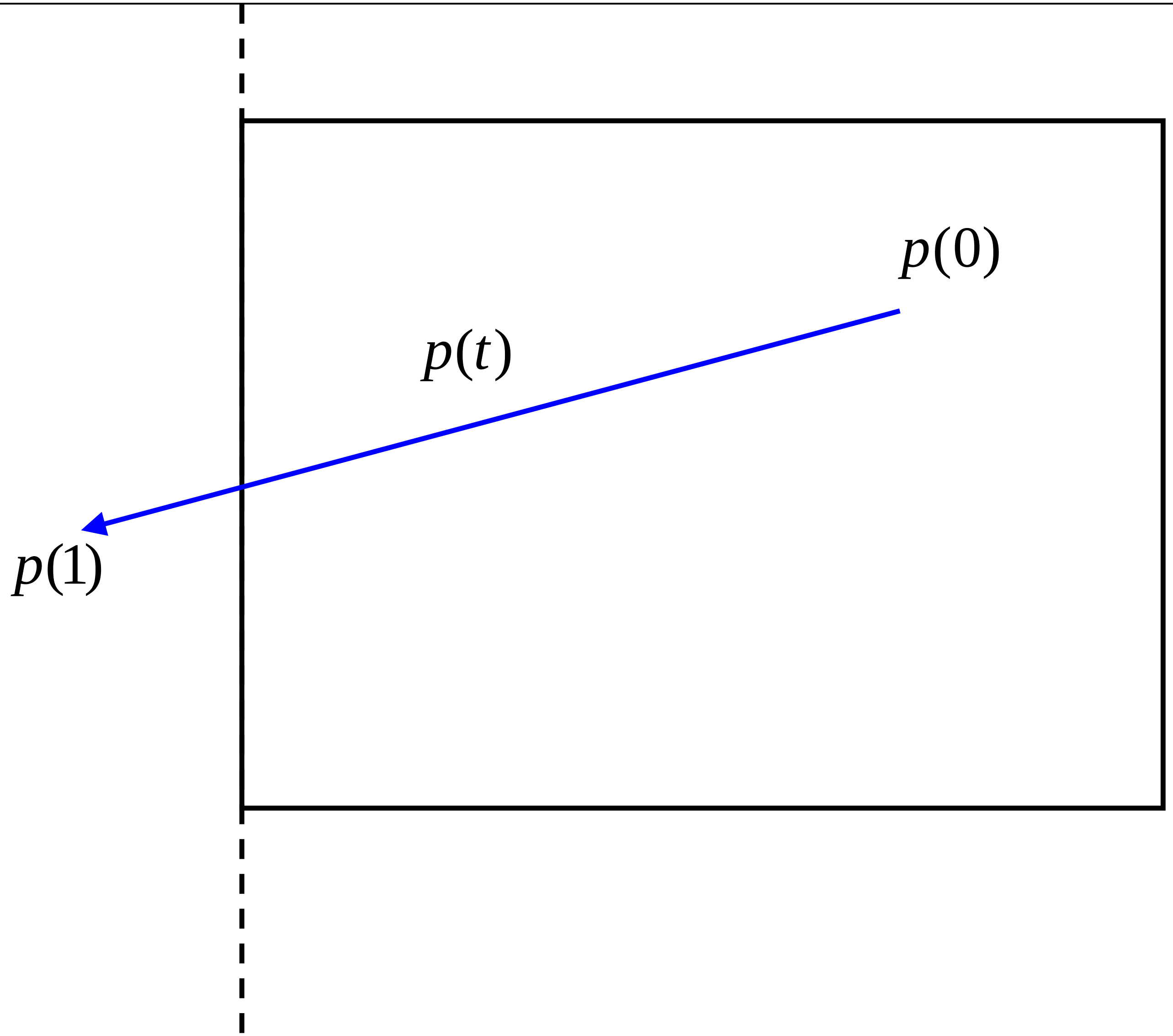
Liang-Barsky Algorithm



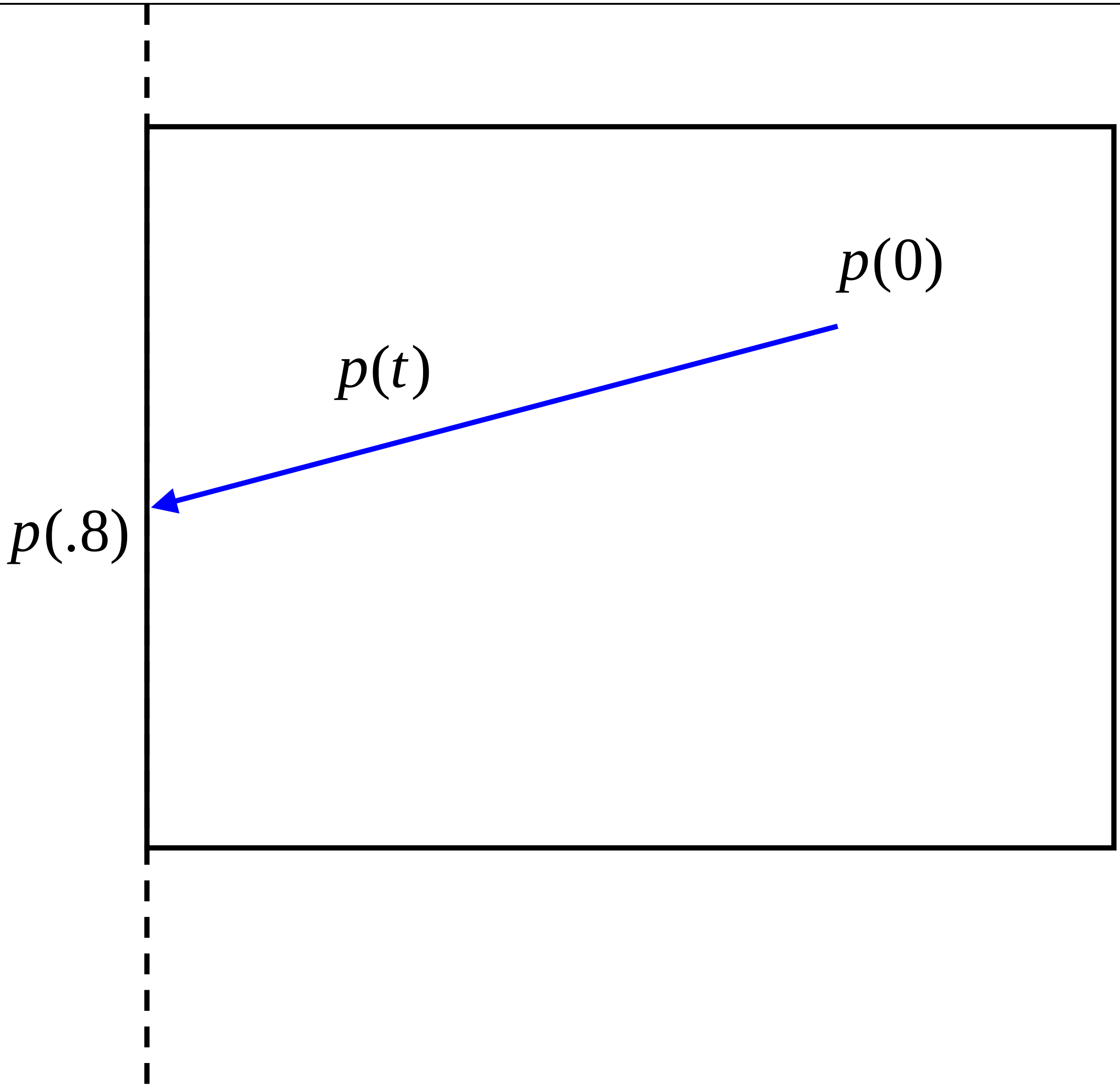
Liang-Barsky Algorithm



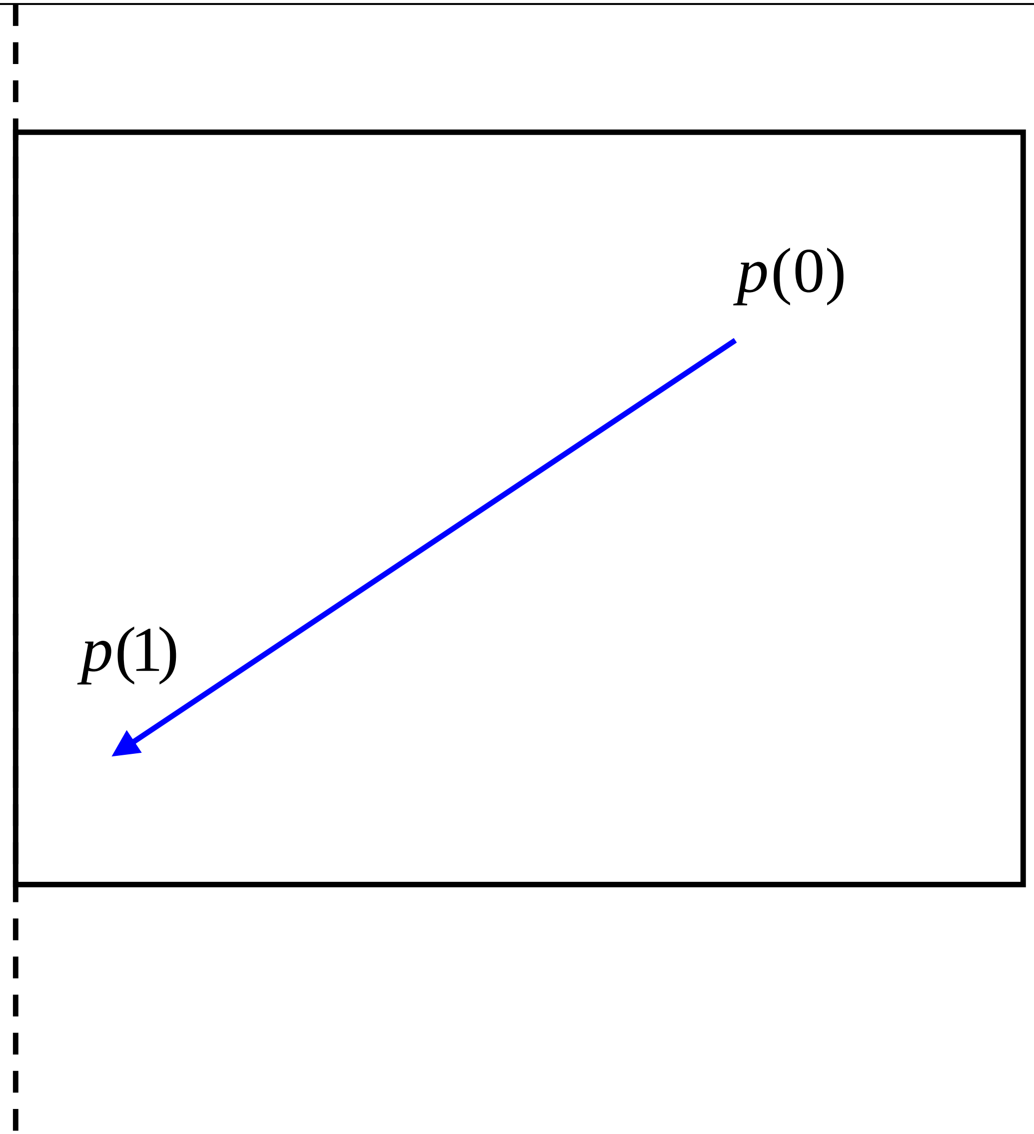
Liang-Barsky Algorithm



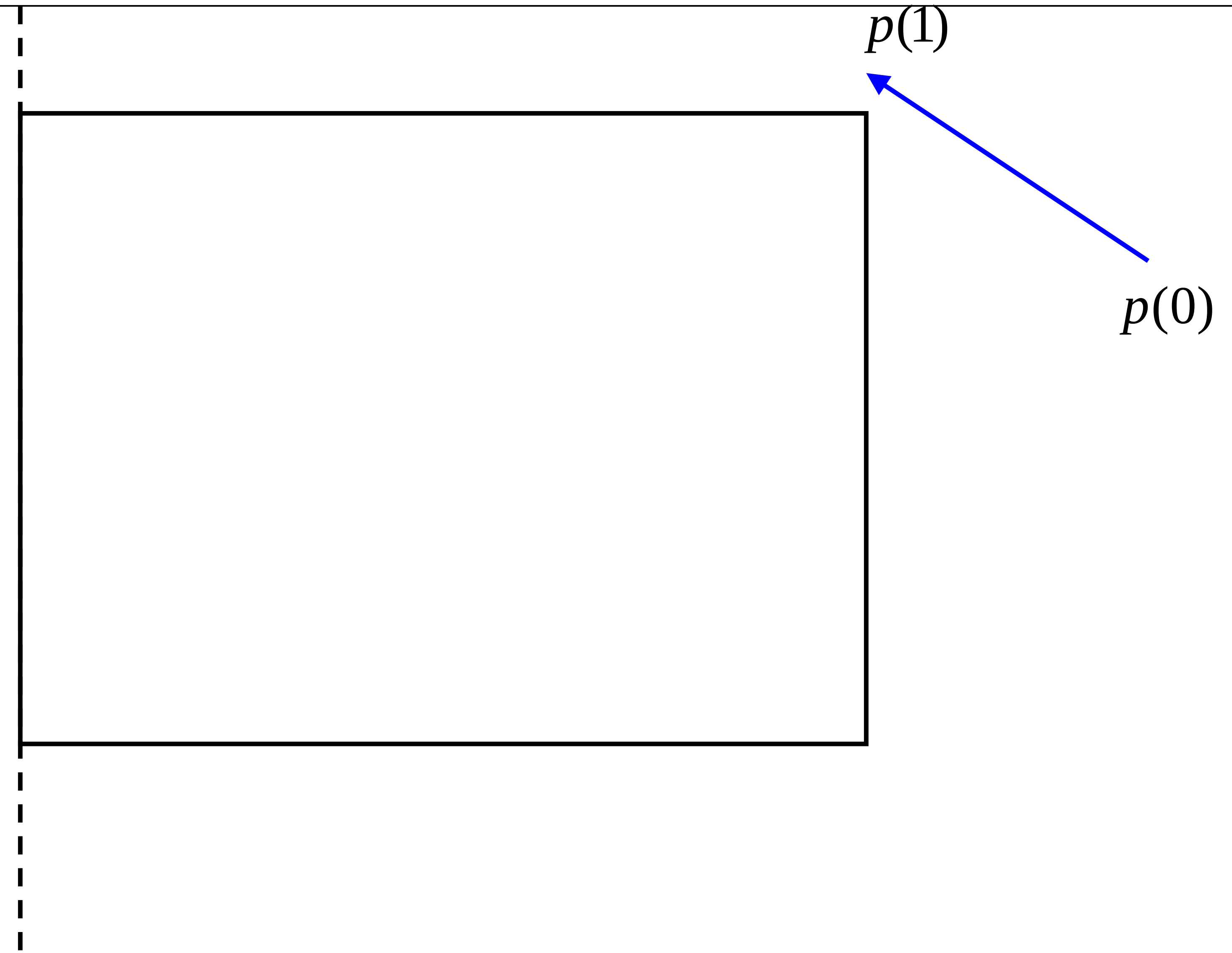
Liang-Barsky Algorithm



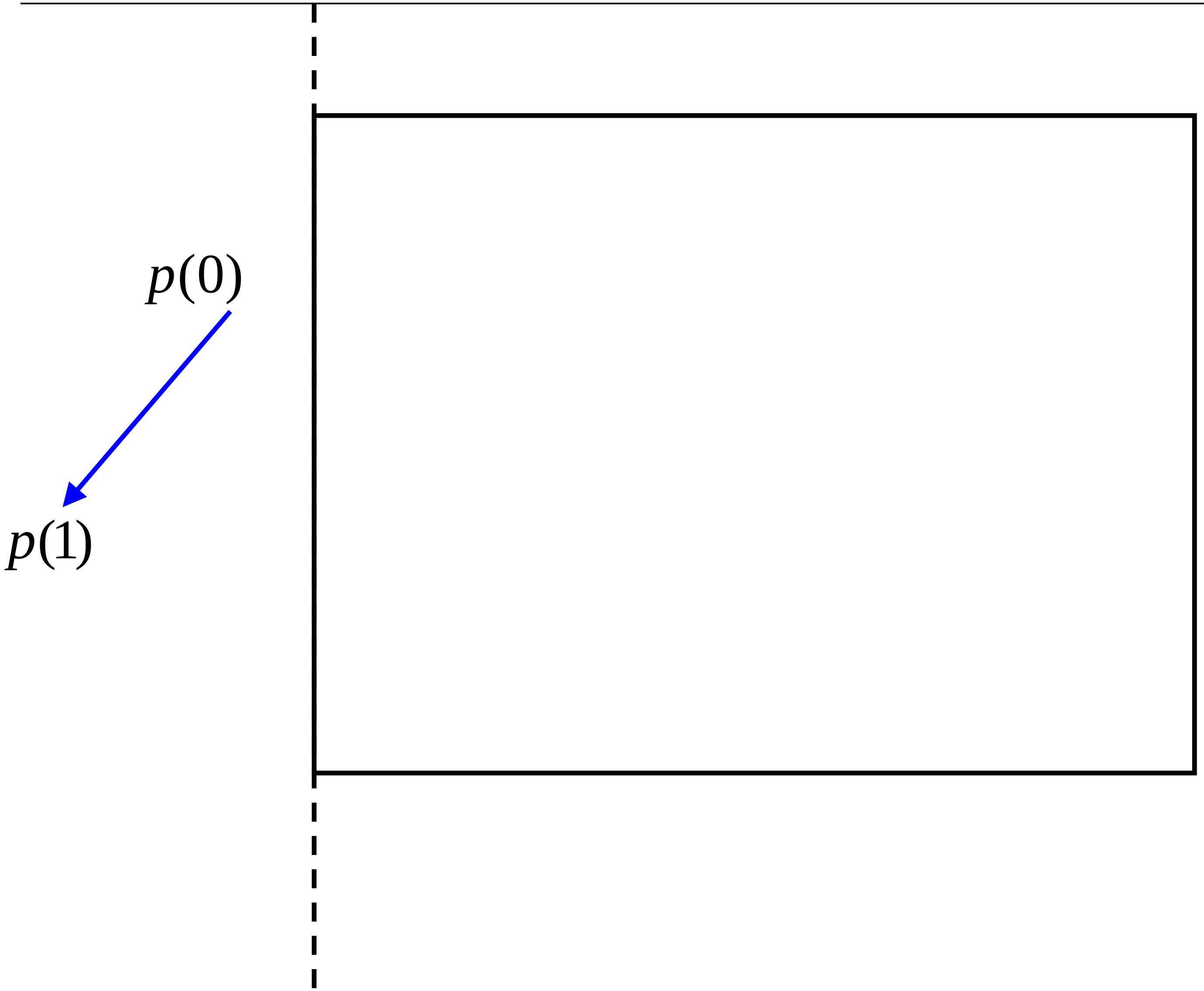
Liang-Barsky Algorithm



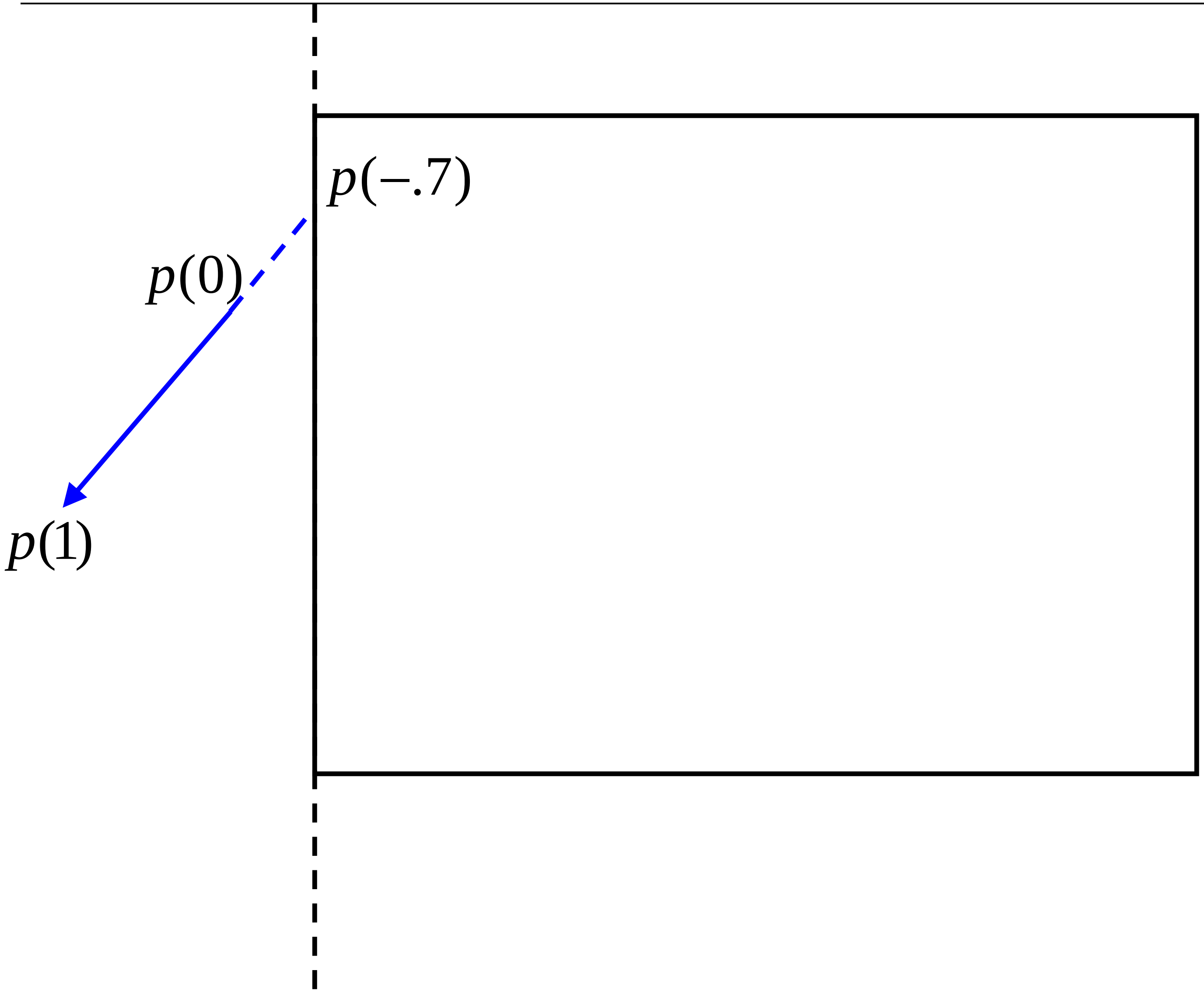
Liang-Barsky Algorithm



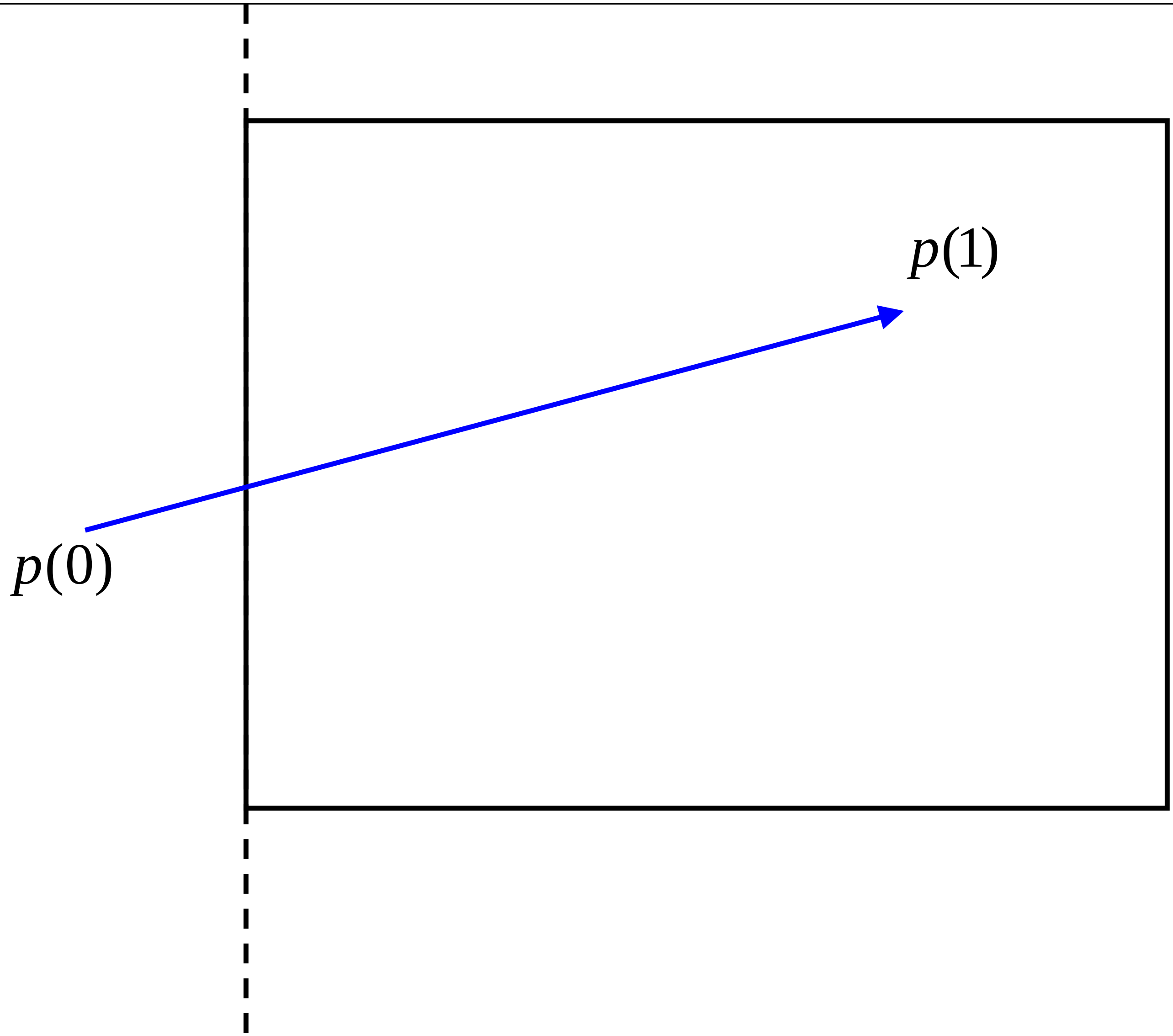
Liang-Barsky Algorithm



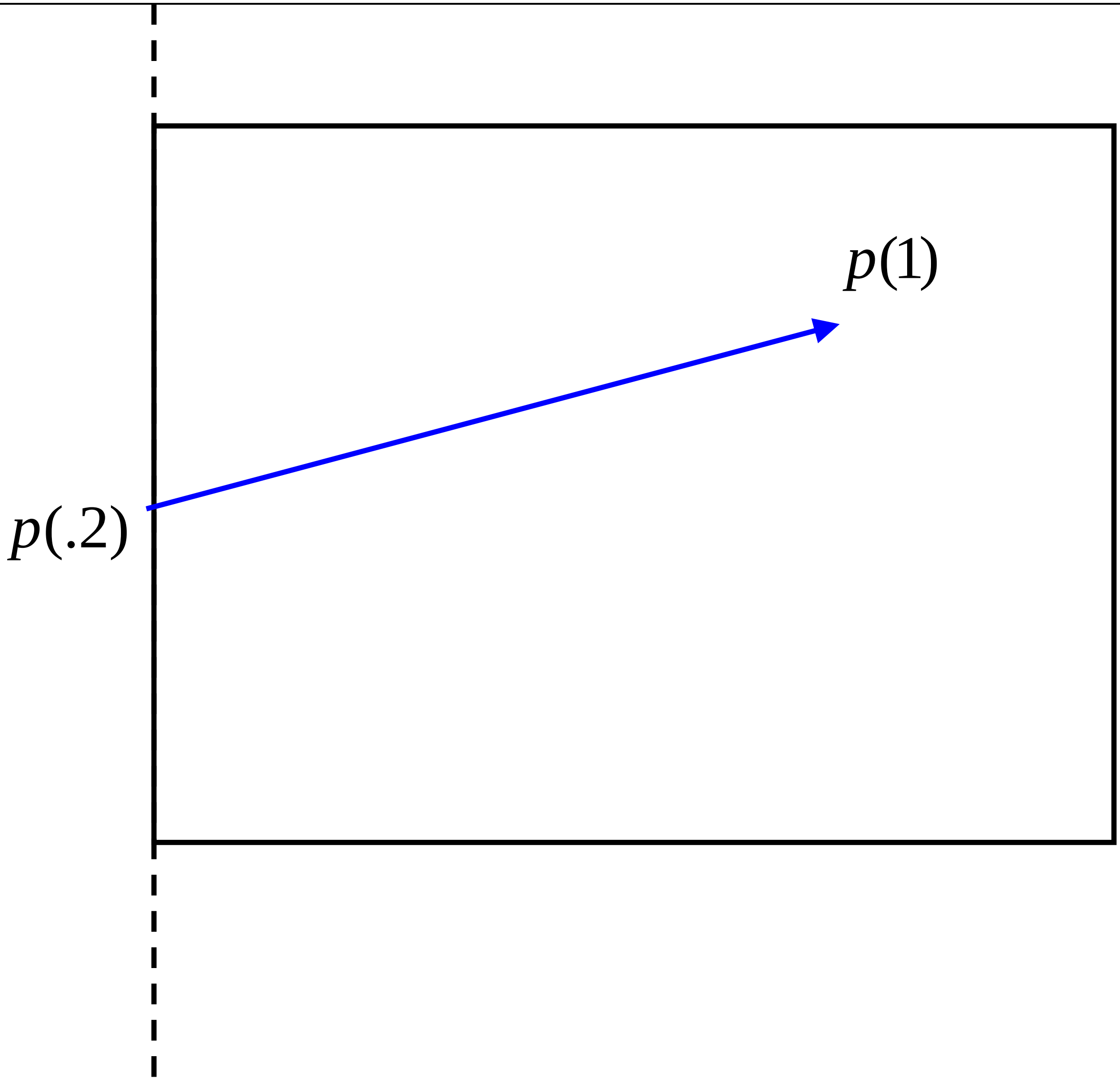
Liang-Barsky Algorithm



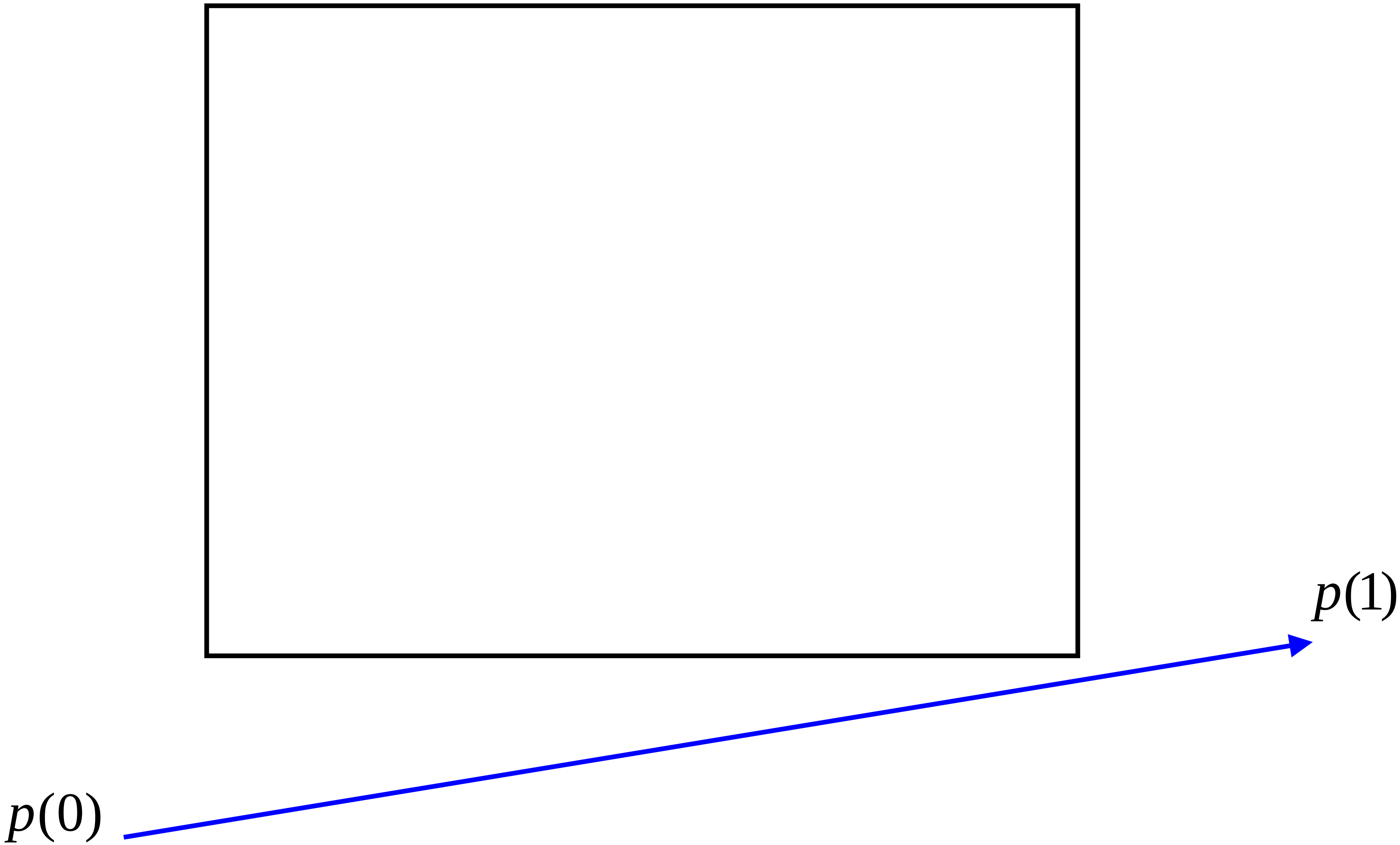
Liang-Barsky Algorithm



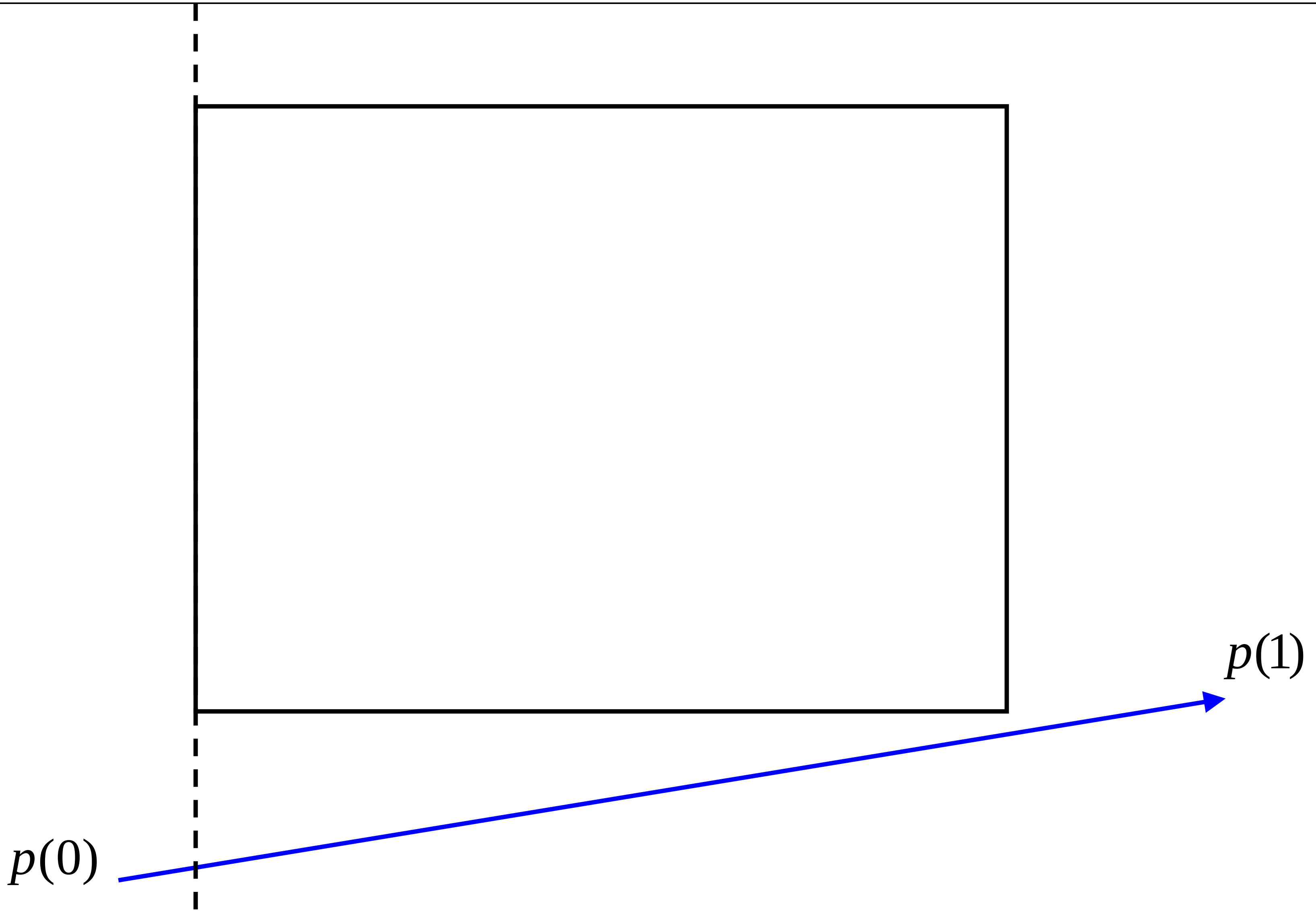
Liang-Barsky Algorithm



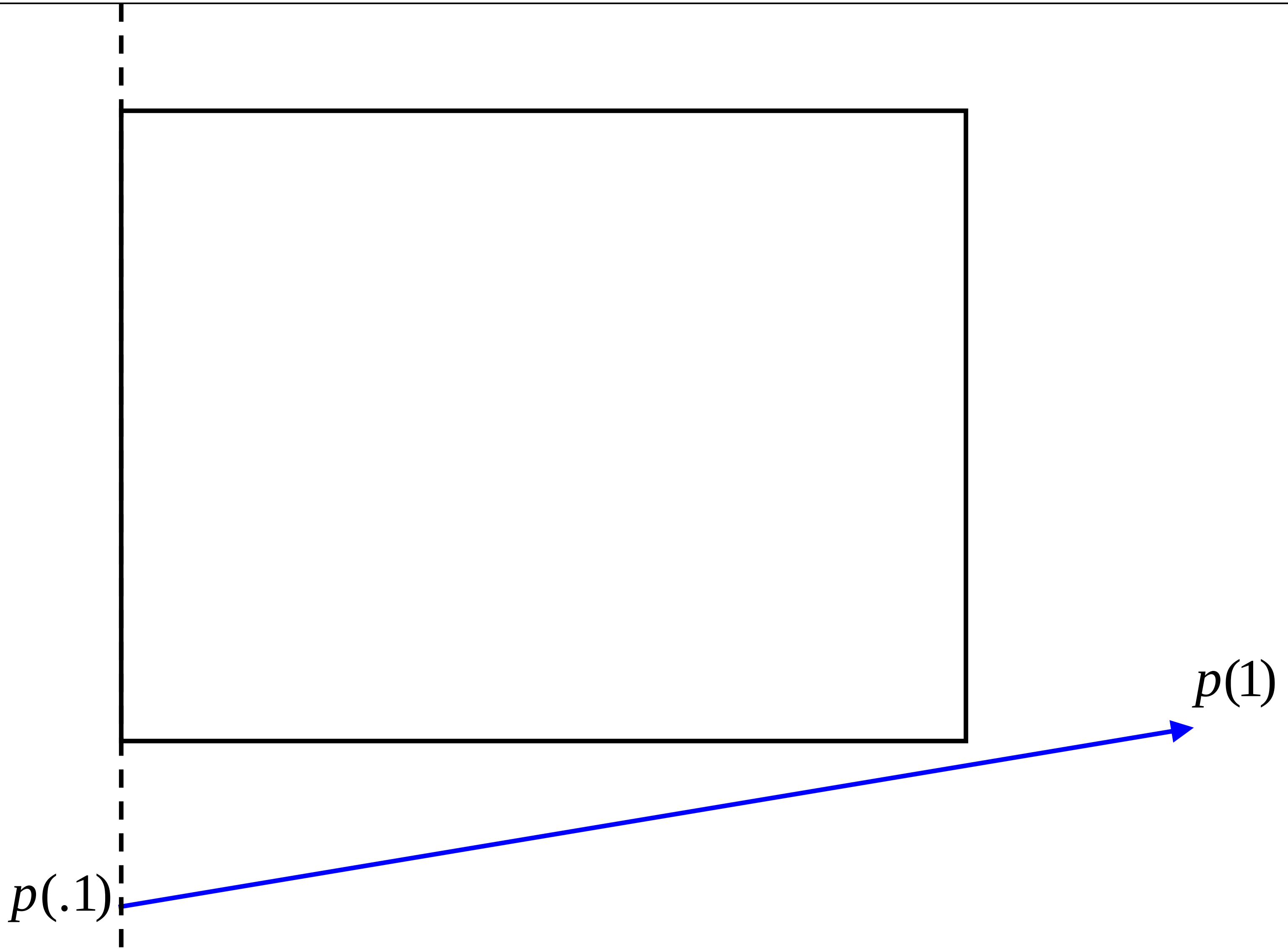
Liang-Barsky Algorithm



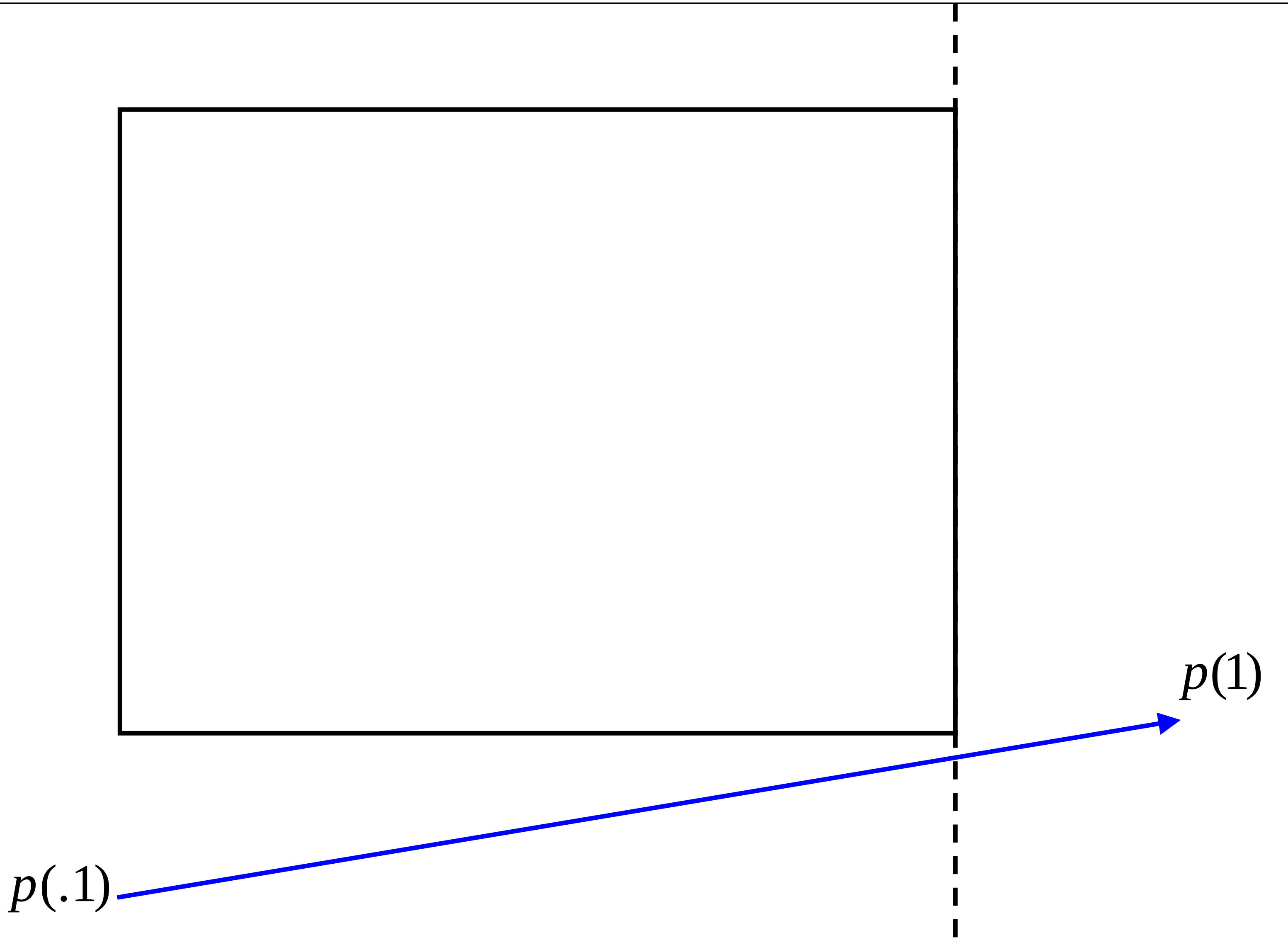
Liang-Barsky Algorithm



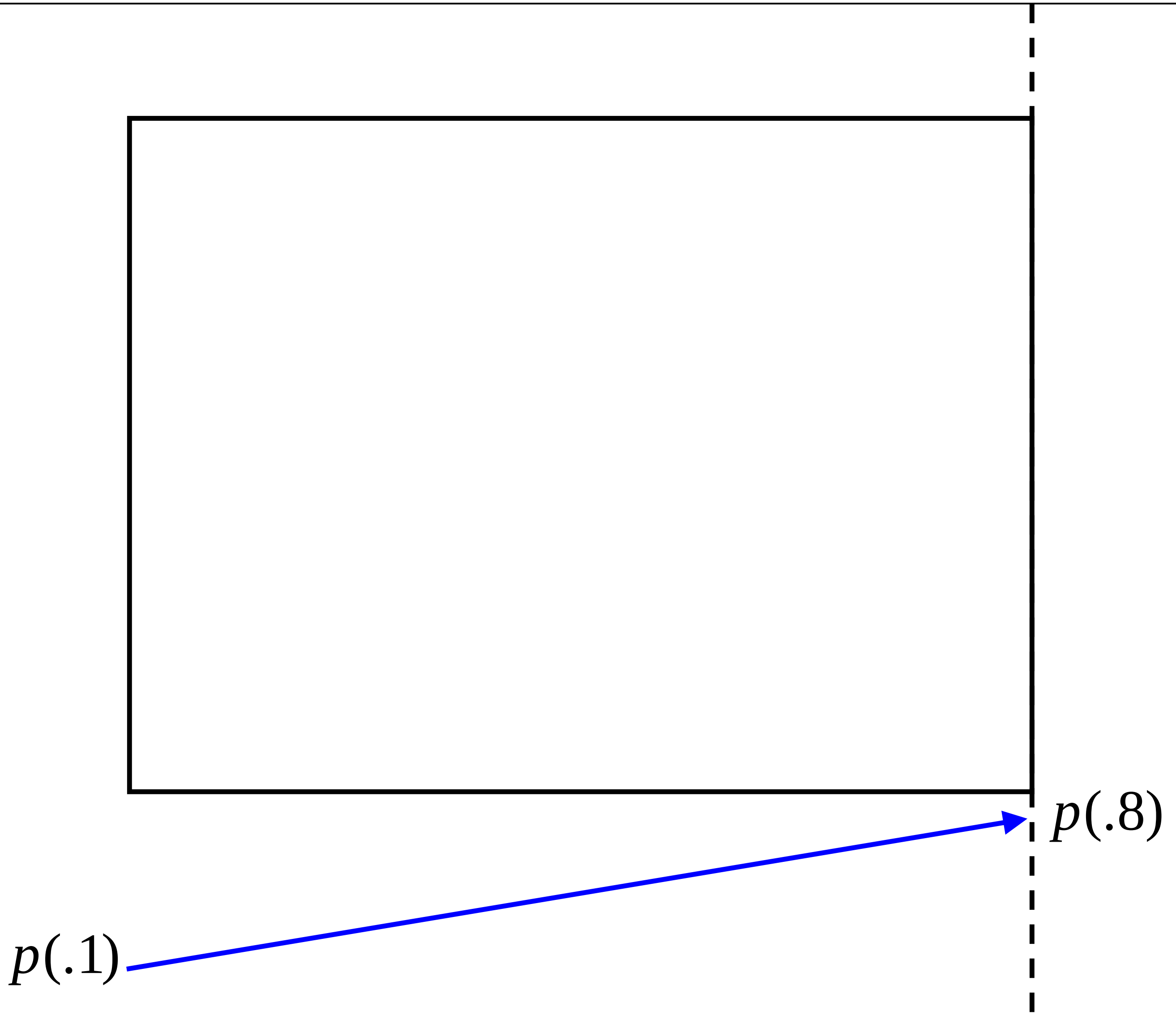
Liang-Barsky Algorithm



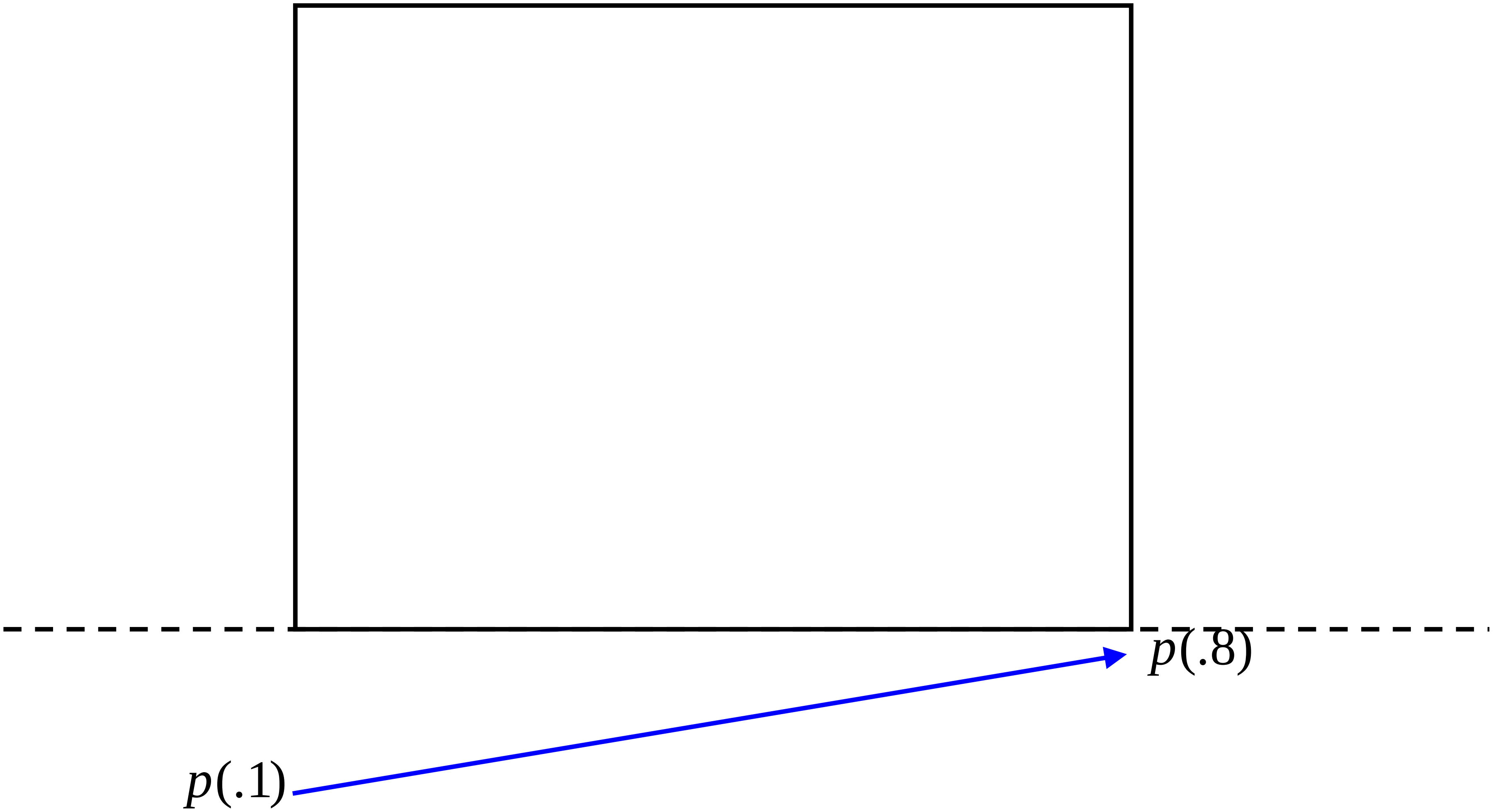
Liang-Barsky Algorithm



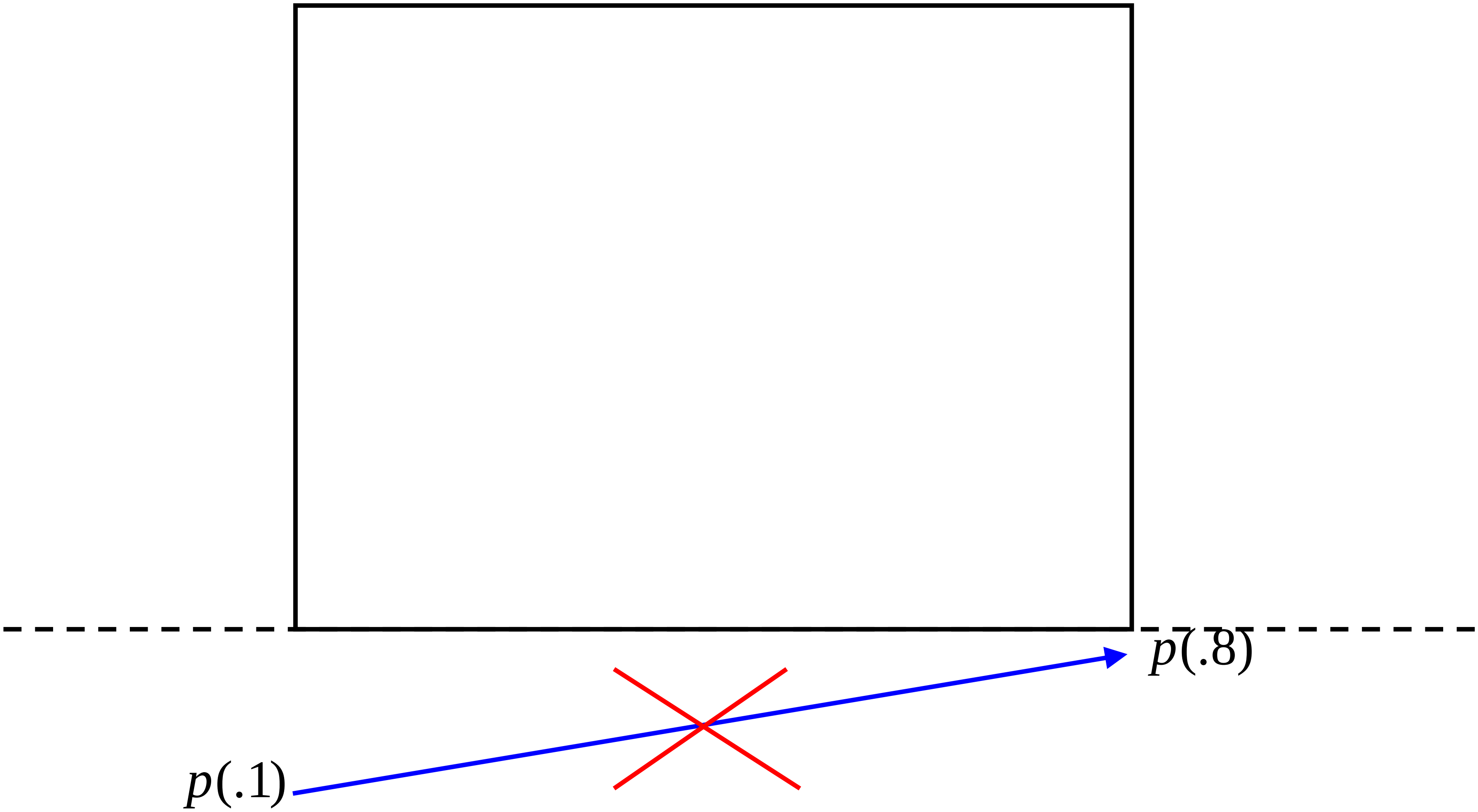
Liang-Barsky Algorithm



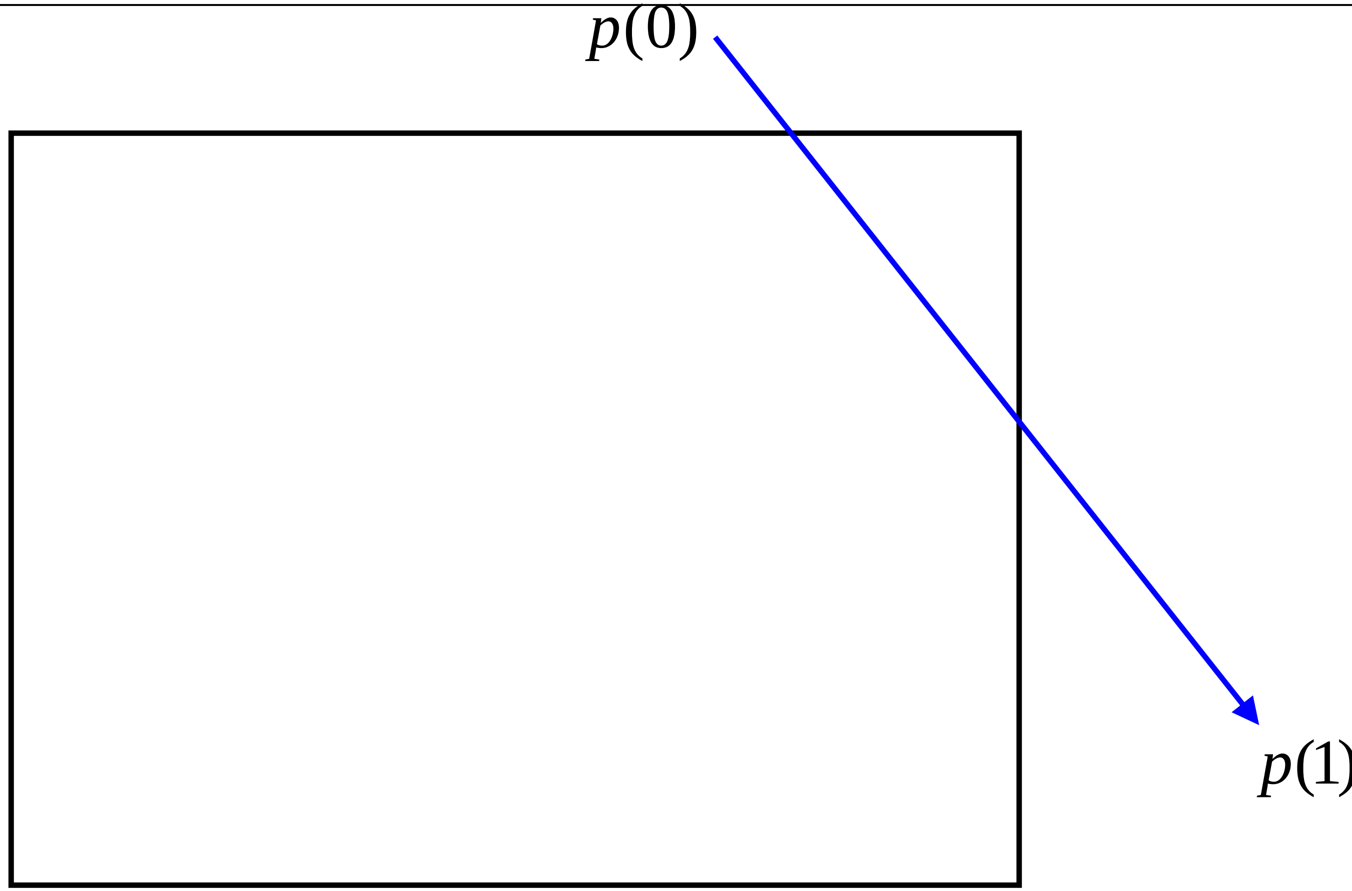
Liang-Barsky Algorithm



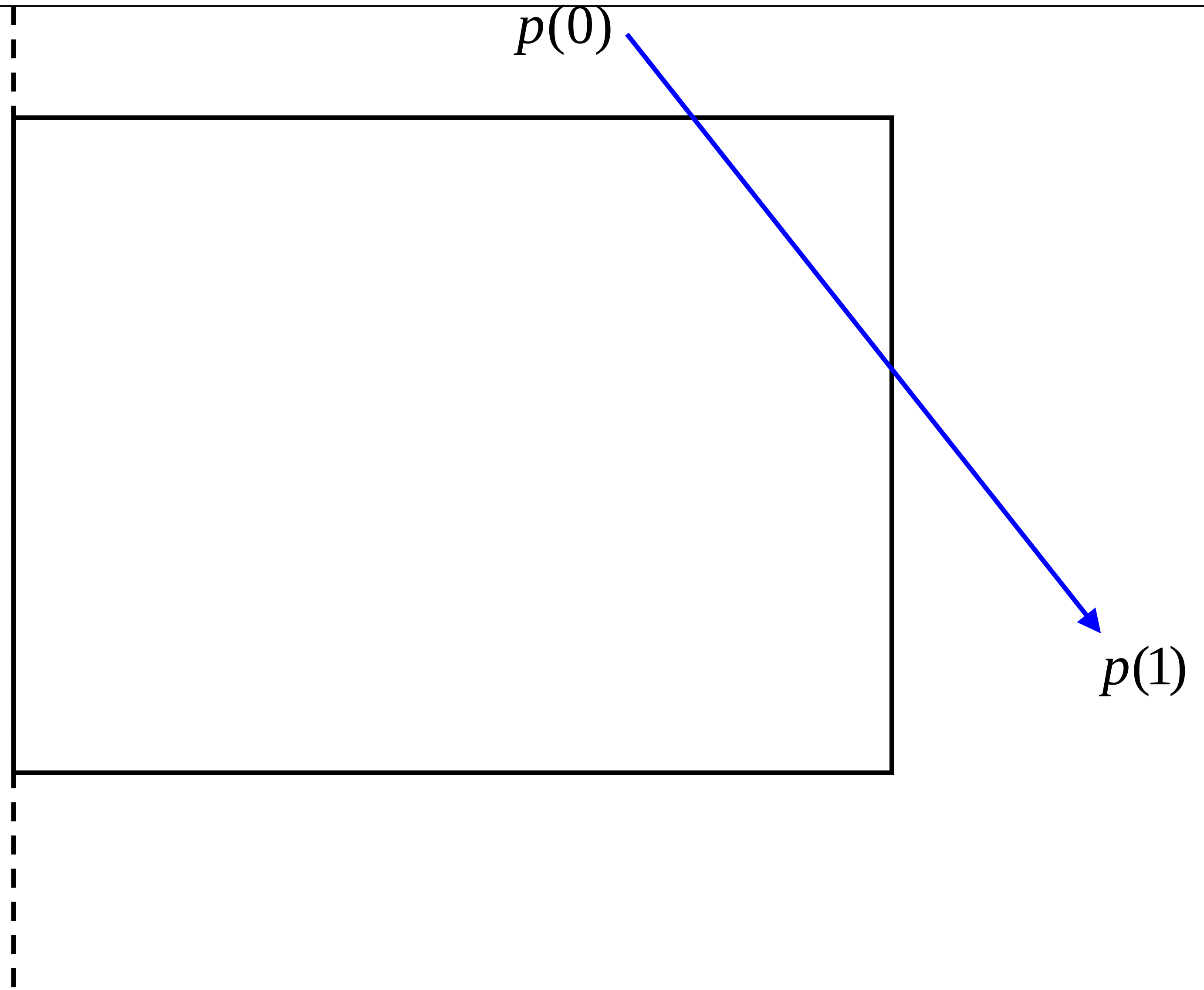
Liang-Barsky Algorithm



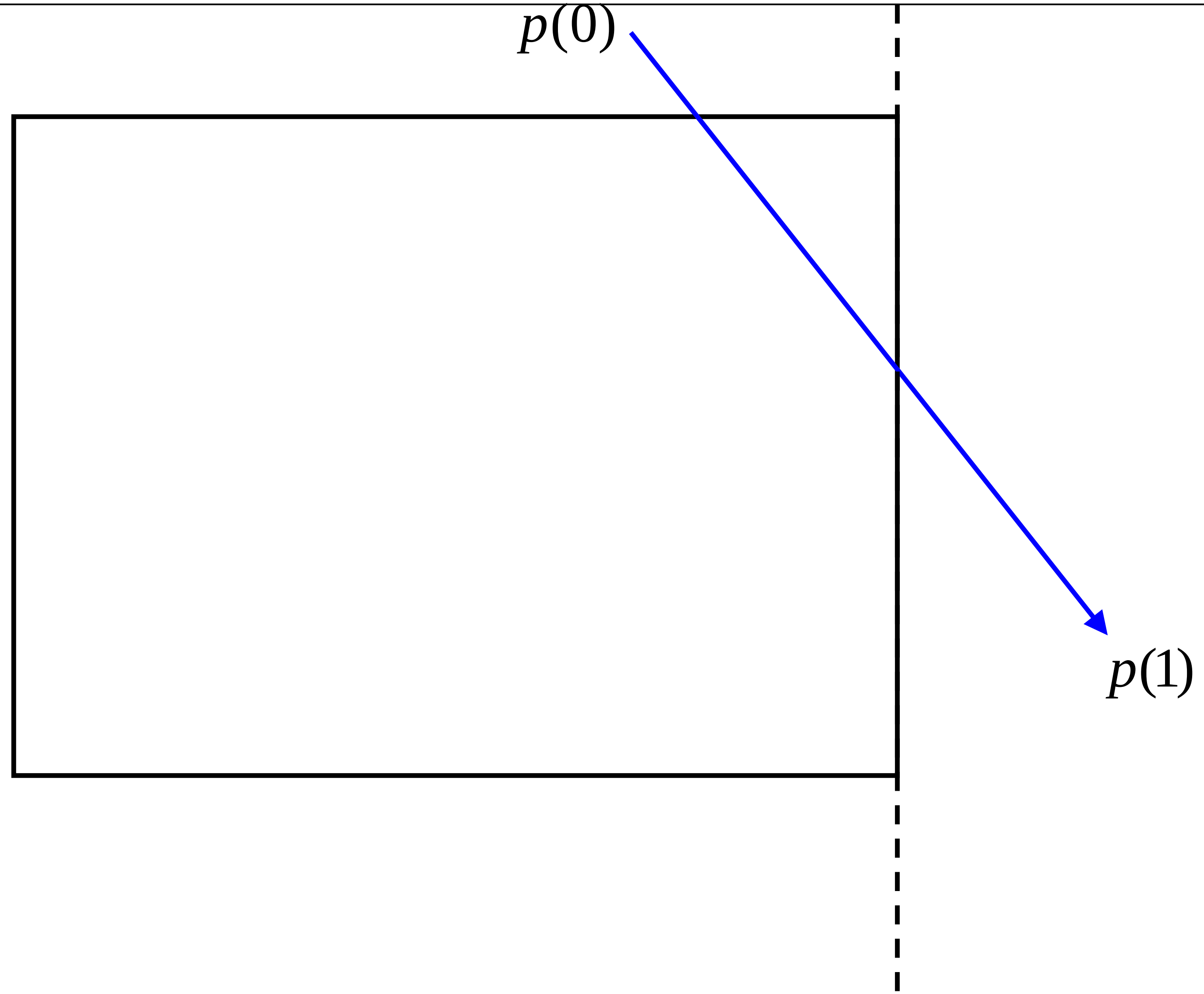
Liang-Barsky Algorithm



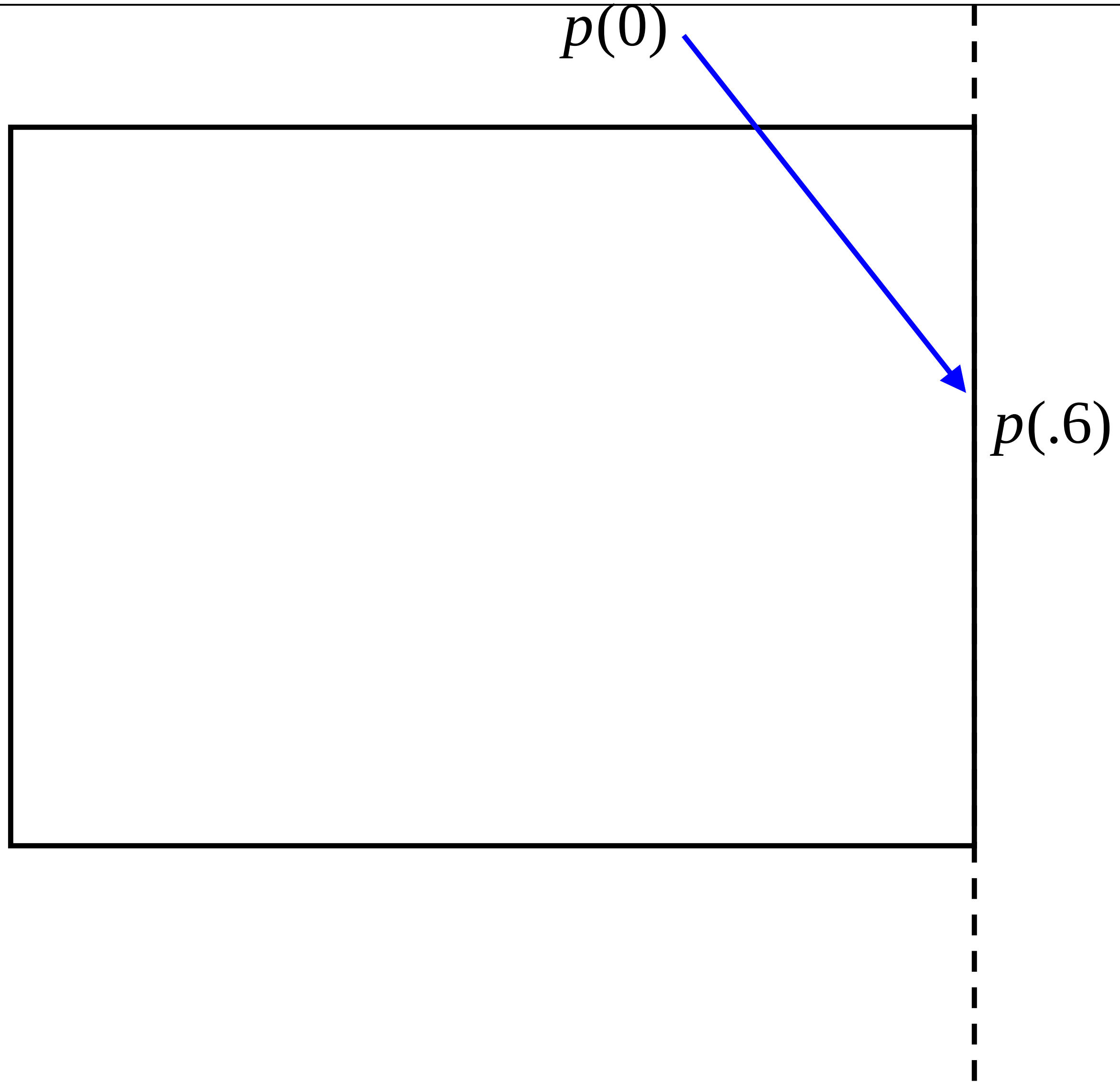
Liang-Barsky Algorithm



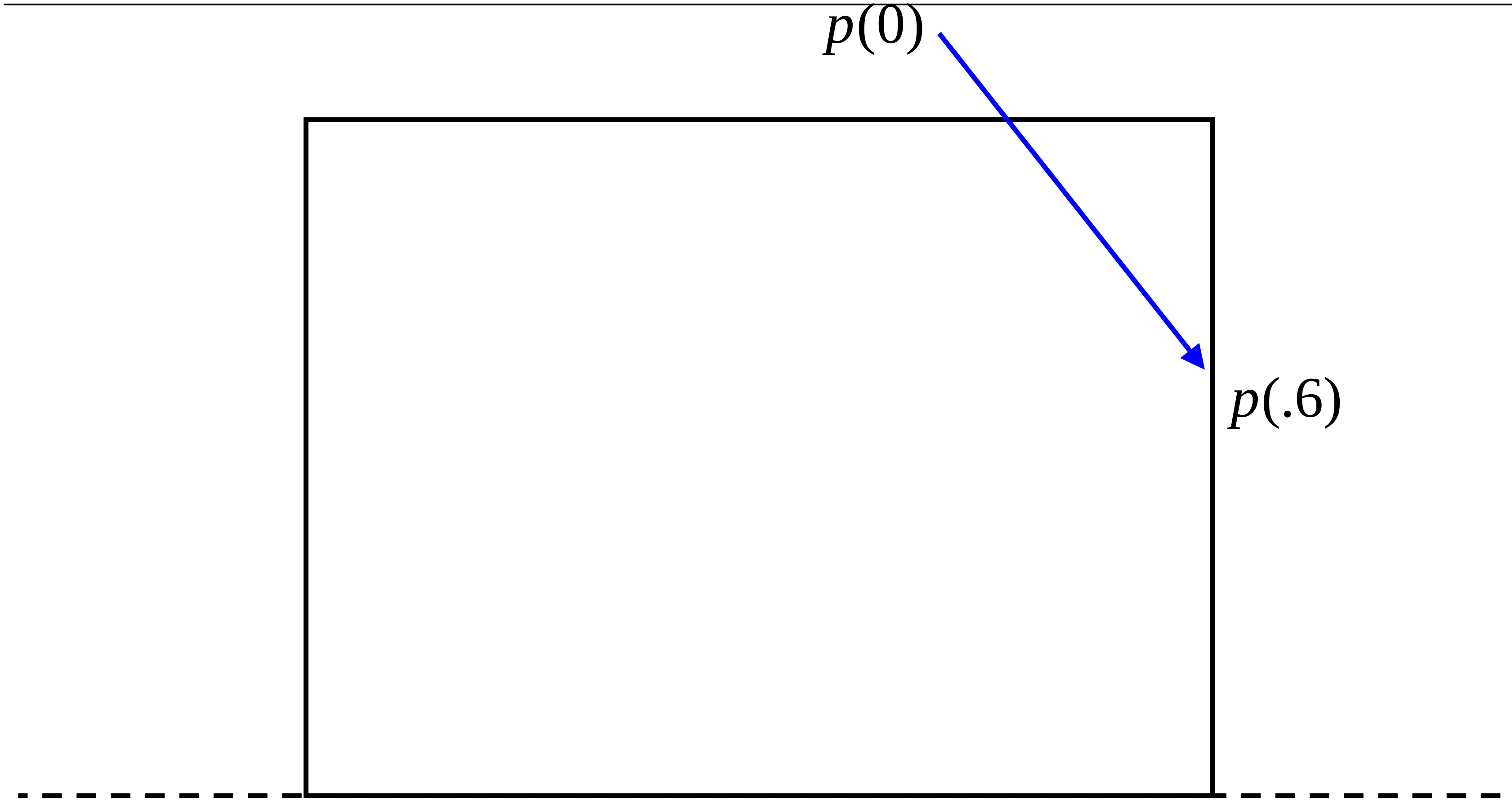
Liang-Barsky Algorithm



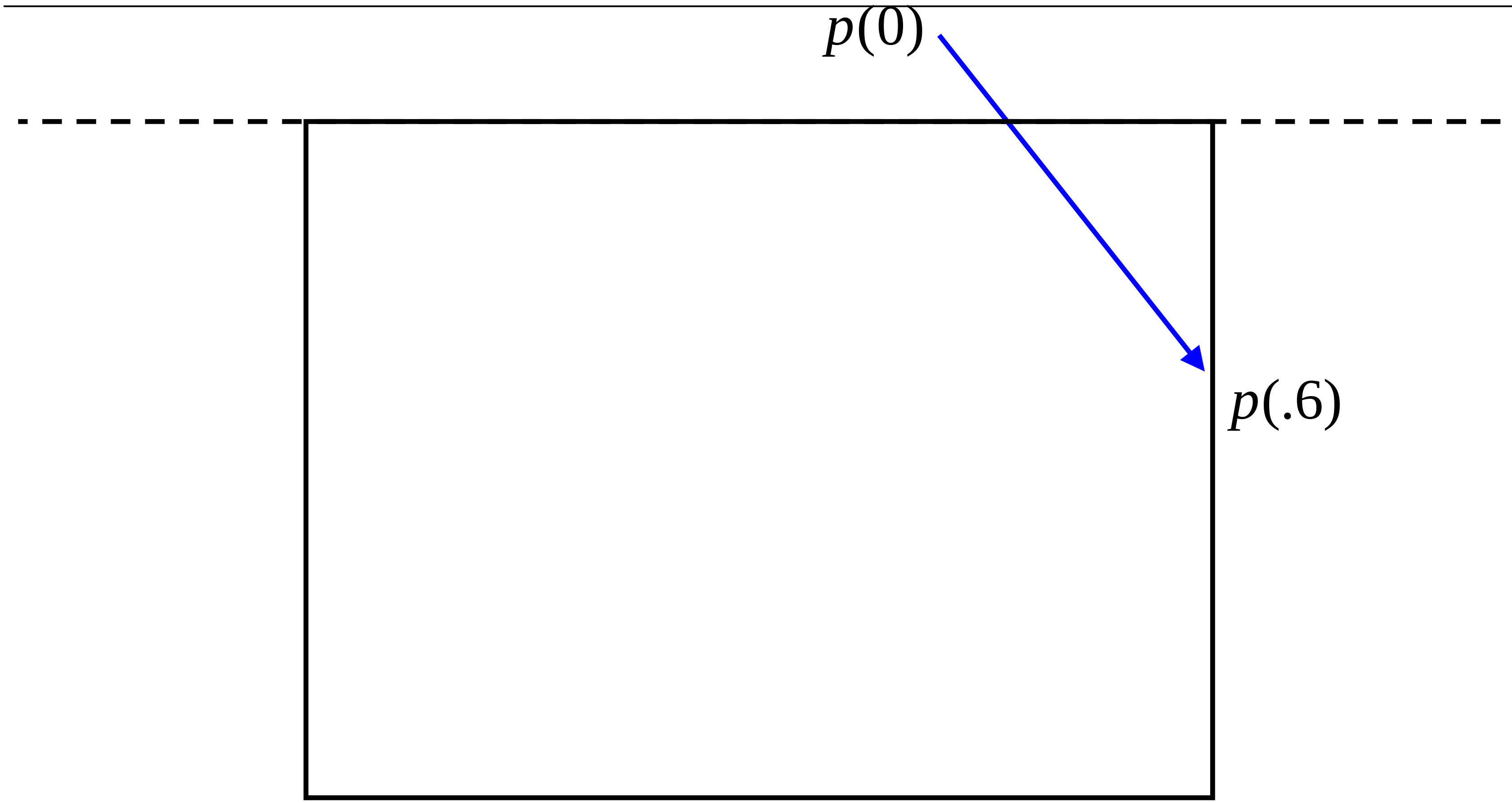
Liang-Barsky Algorithm



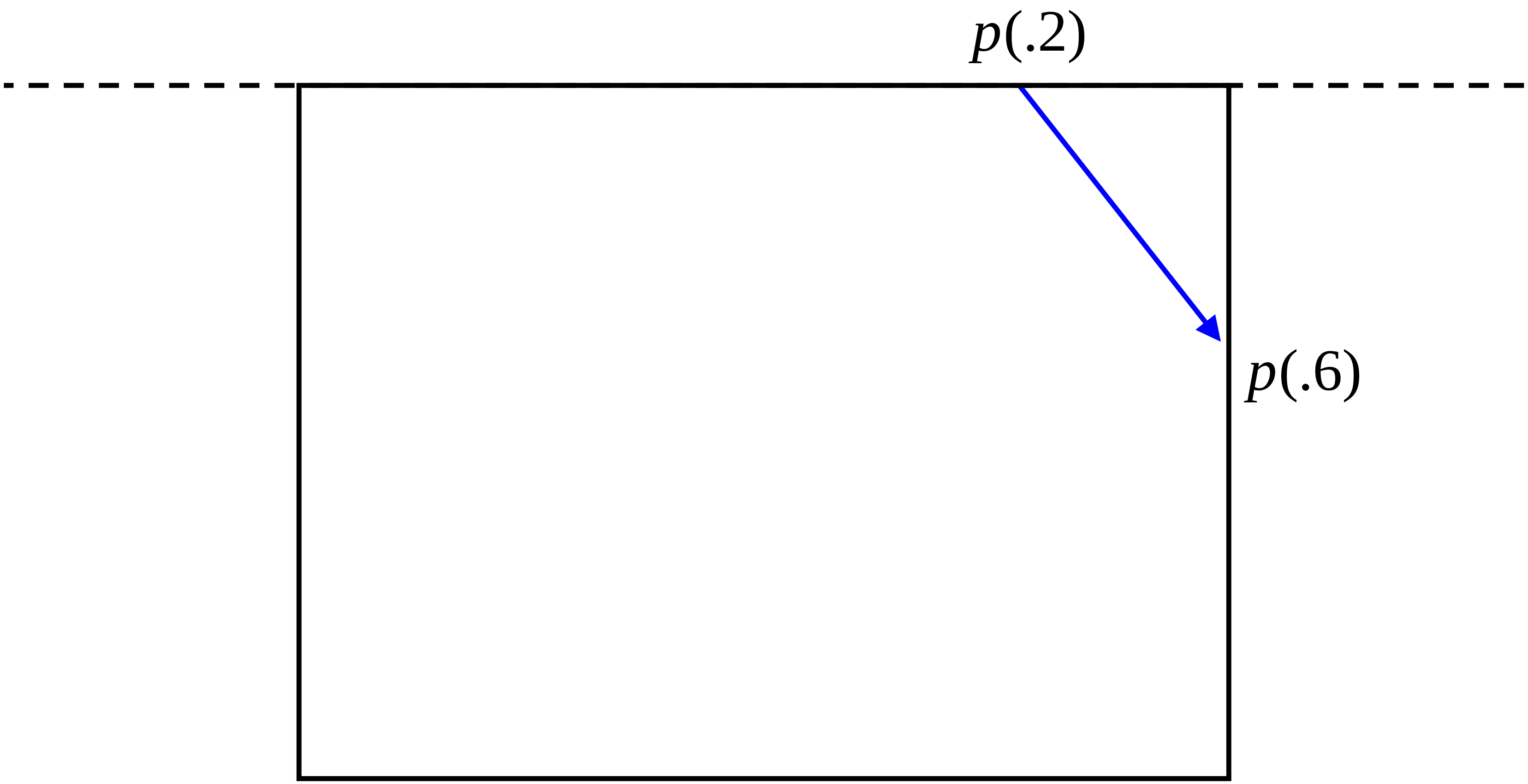
Liang-Barsky Algorithm



Liang-Barsky Algorithm



Liang-Barsky Algorithm



Comparison

- Cohen-Sutherland
  - Repeated clipping is expensive
  - Best used when trivial acceptance and rejection is possible for most lines
- Liang-Barsky
  - Computation of t-intersections is cheap (only one division)
  - Computation of (x,y) clip points is only done once
  - Algorithm doesn’t consider trivial accepts/rejects
  - Best when many lines must be clipped

Line Clipping – Considerations

- Just clipping end-points does not produce the correct results inside the window
- Must also update *sum* in midpoint algorithm

- Clipping against non-rectangular polygons is also possible but seldom used