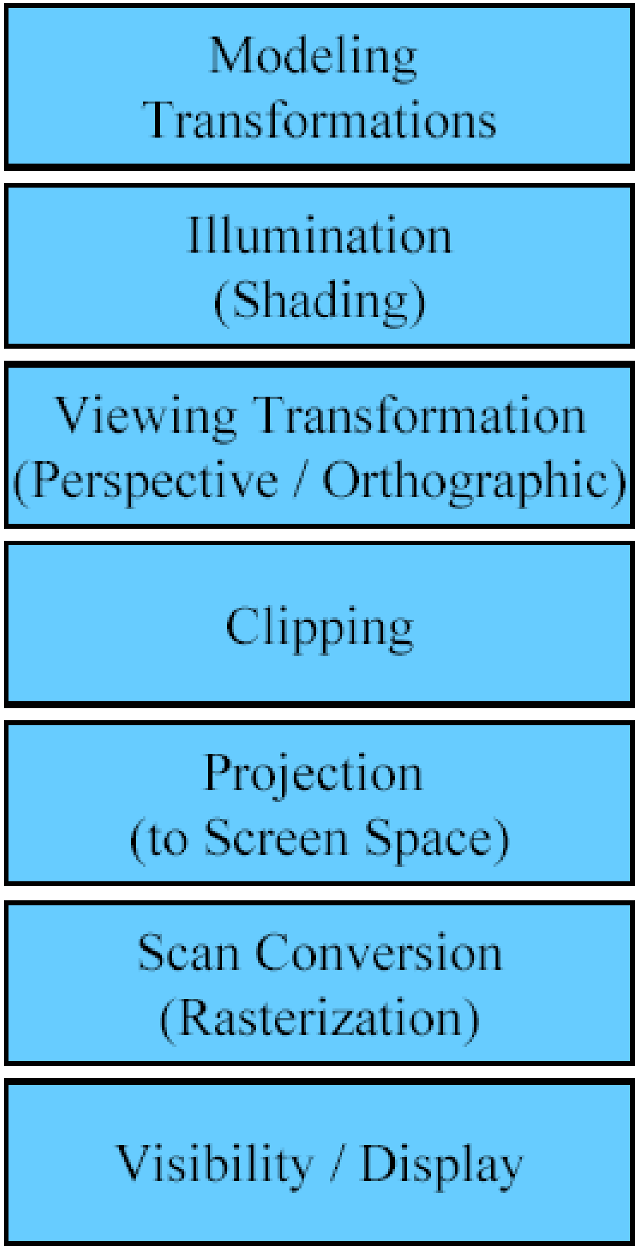
ΗΥ416
ΓΡΑΦΙΚΑ ΥΠΟΛΟΓΙΣΤΩΝ

Αποκοπή

Π. ΤΣΟΜΠΑΝΟΠΟΥΛΟΥ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ & ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Graphics Pipeline



Modeling Transformations

Illumination (Shading)

Viewing Transformation (Perspective / Orthographic)

Clipping

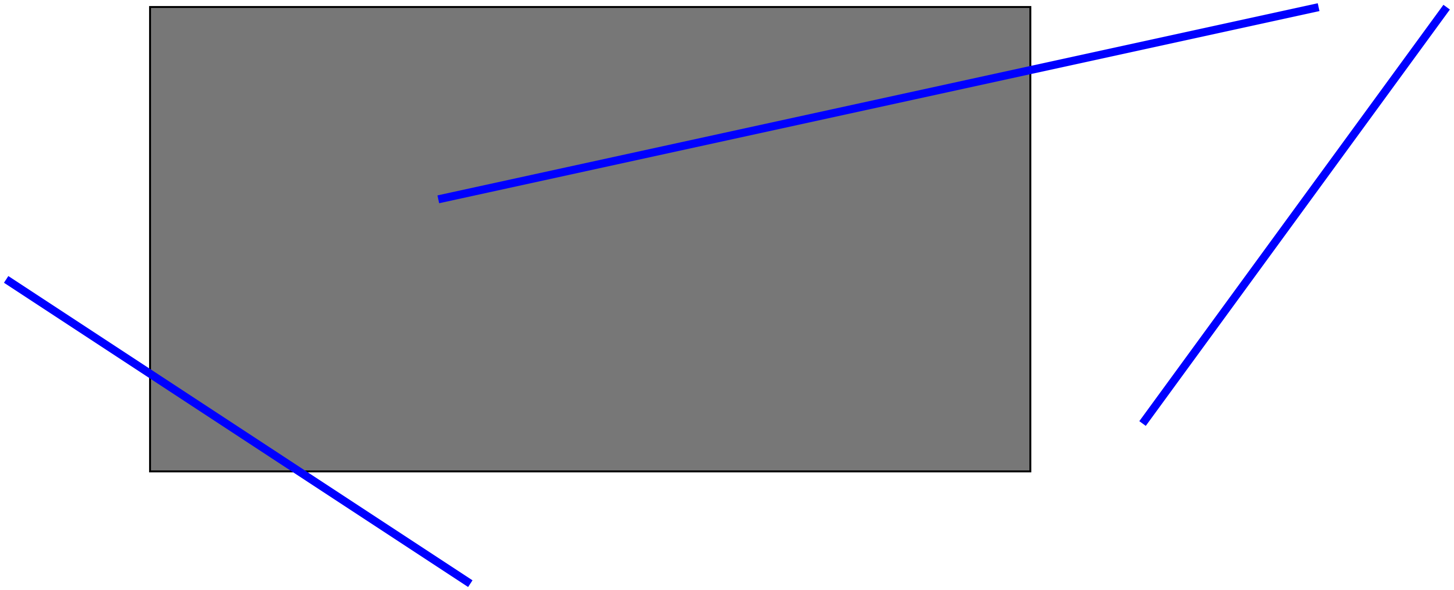
Projection (to Screen Space)

Scan Conversion (Rasterization)

Visibility / Display

Why Clip?

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

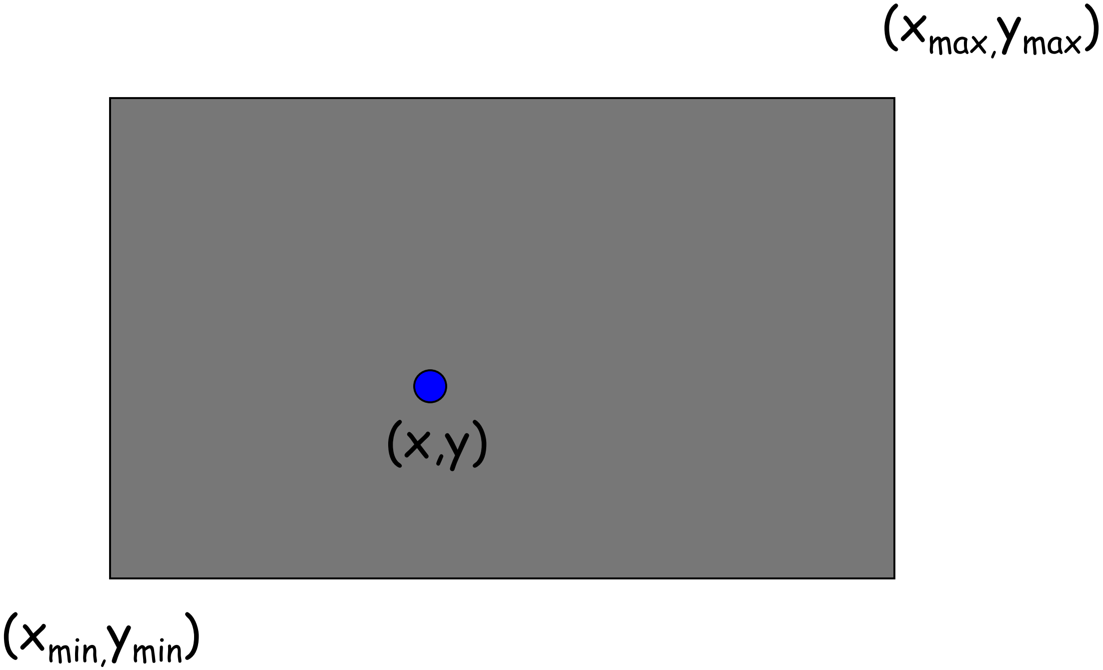


Αποκοπή Σημείων

yota@inf.uth.gr

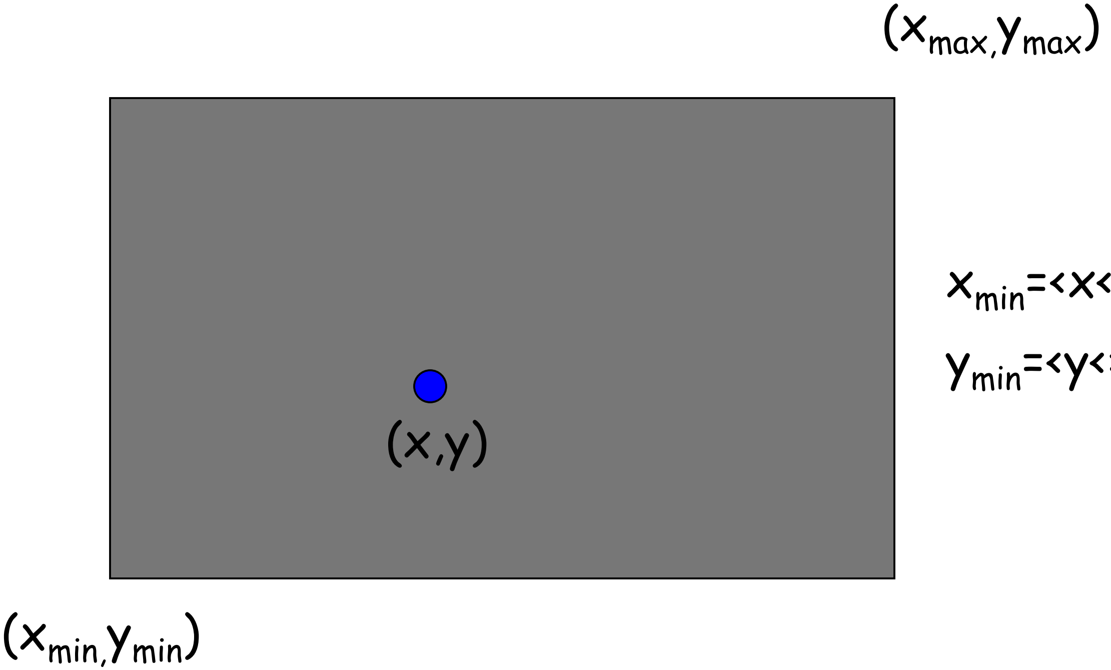
Clipping Points

- 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

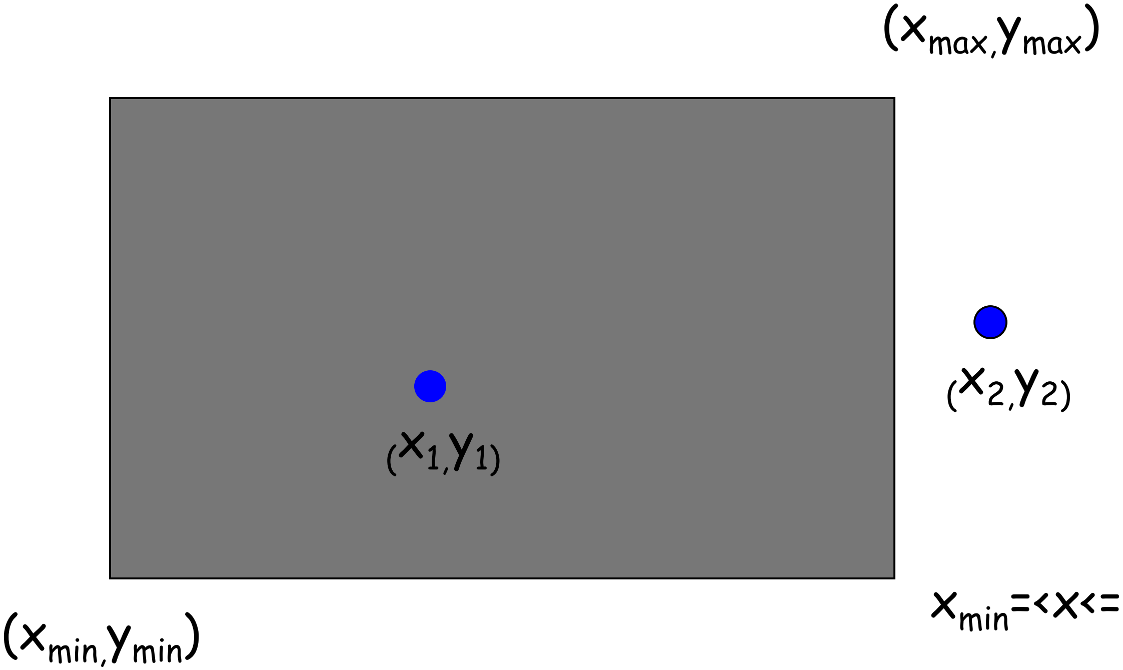


$x_{min} \le x \le x_{max}$?

$y_{min} \le y \le y_{max}$?

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

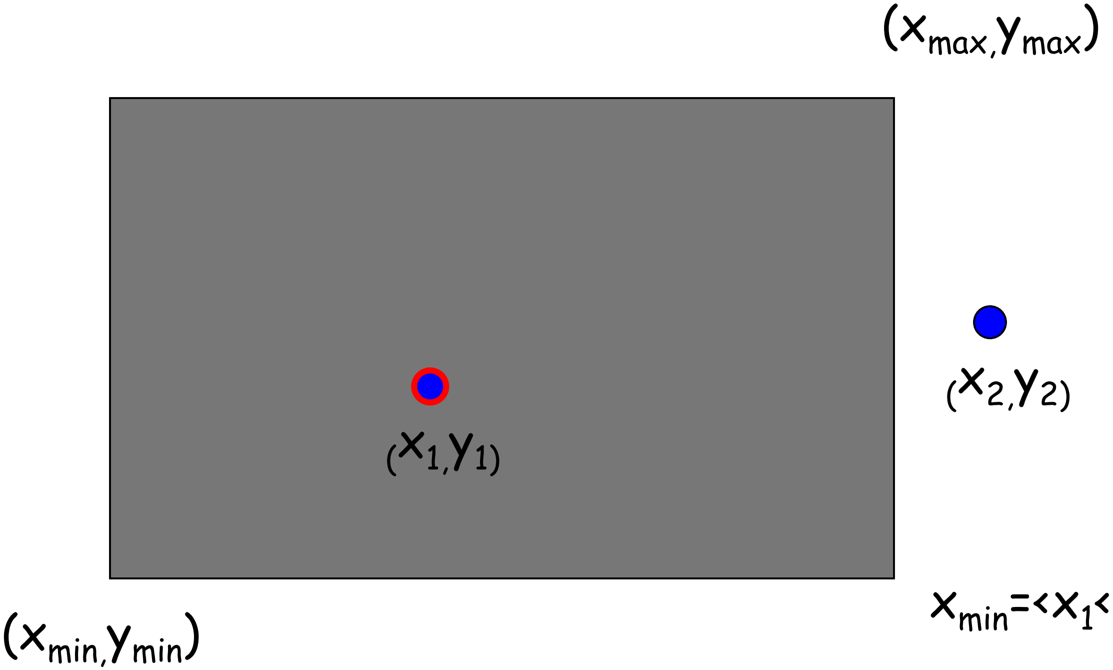


$x_{min} \leq x \leq x_{max}$?

$y_{min} \leq y \leq y_{max}$?

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

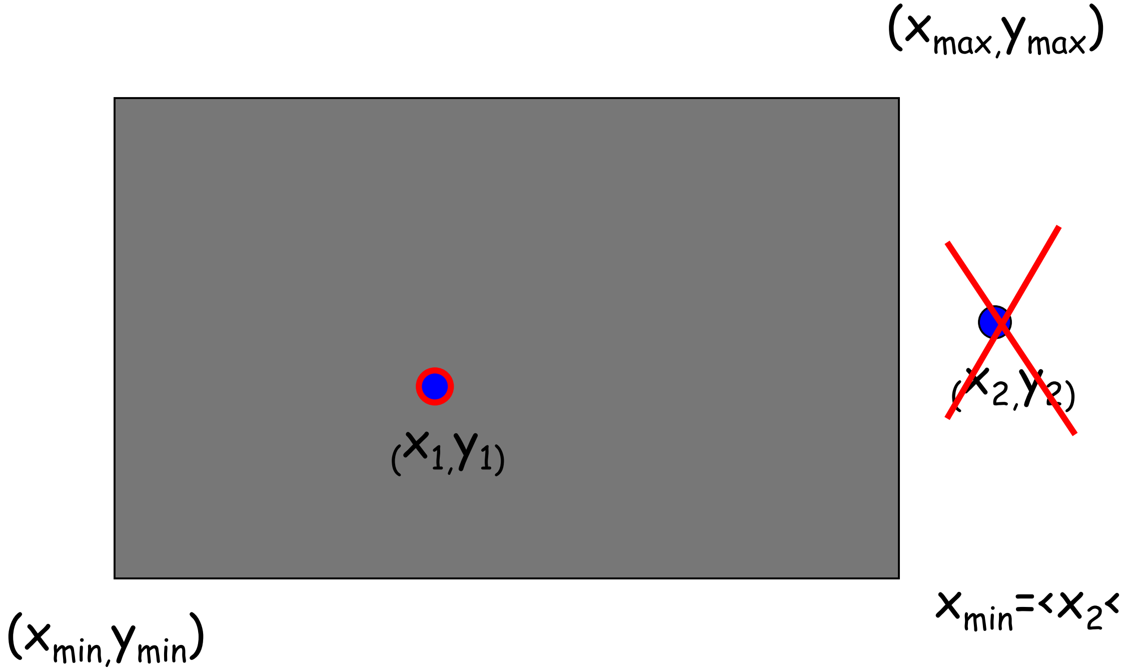


$x_{min} \le x_1 \le x_{max}$ **Yes**

$y_{min} \le y_1 \le y_{max}$ **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

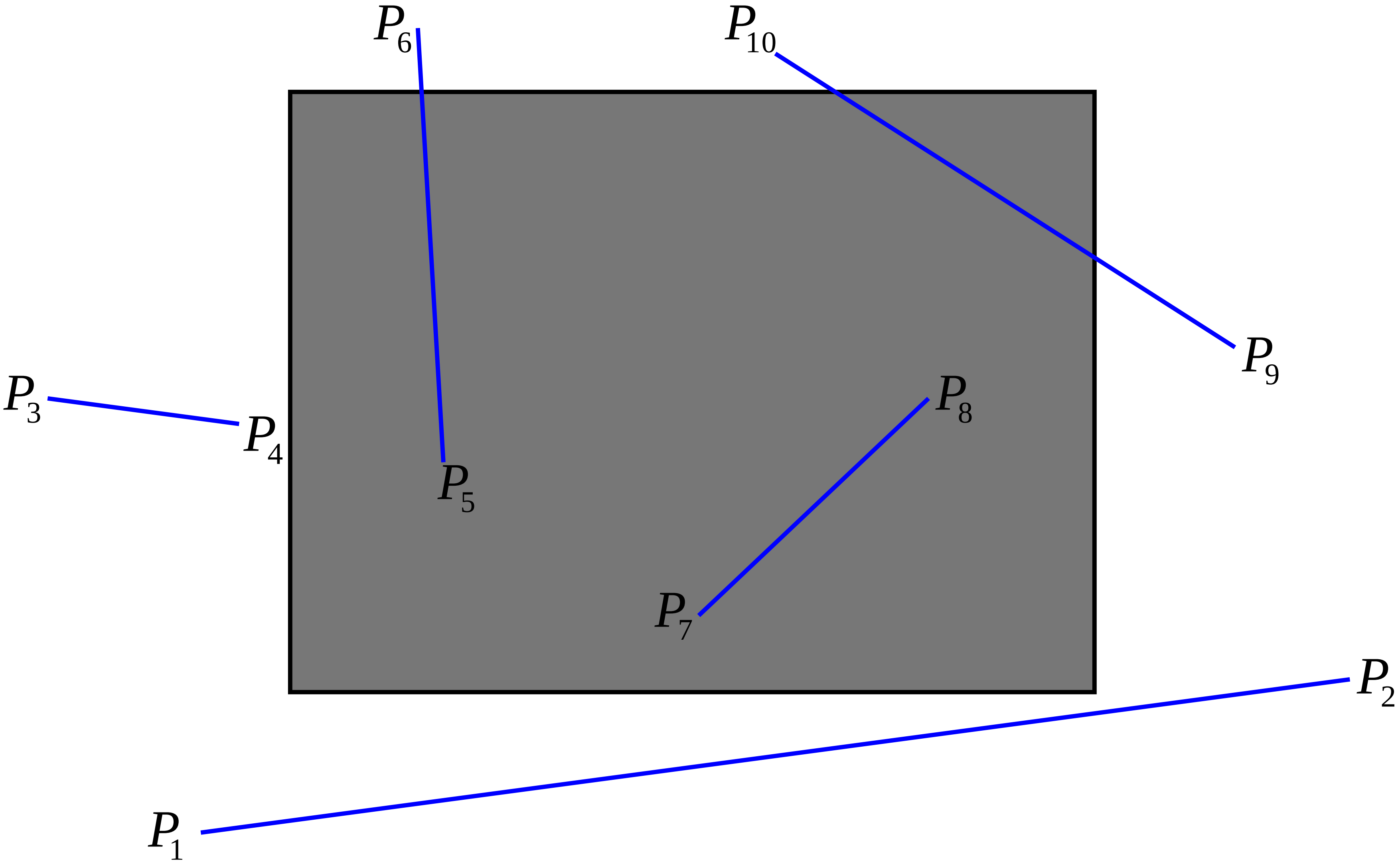


$x_{min} =< x_2 <= x_{max}$ **No**

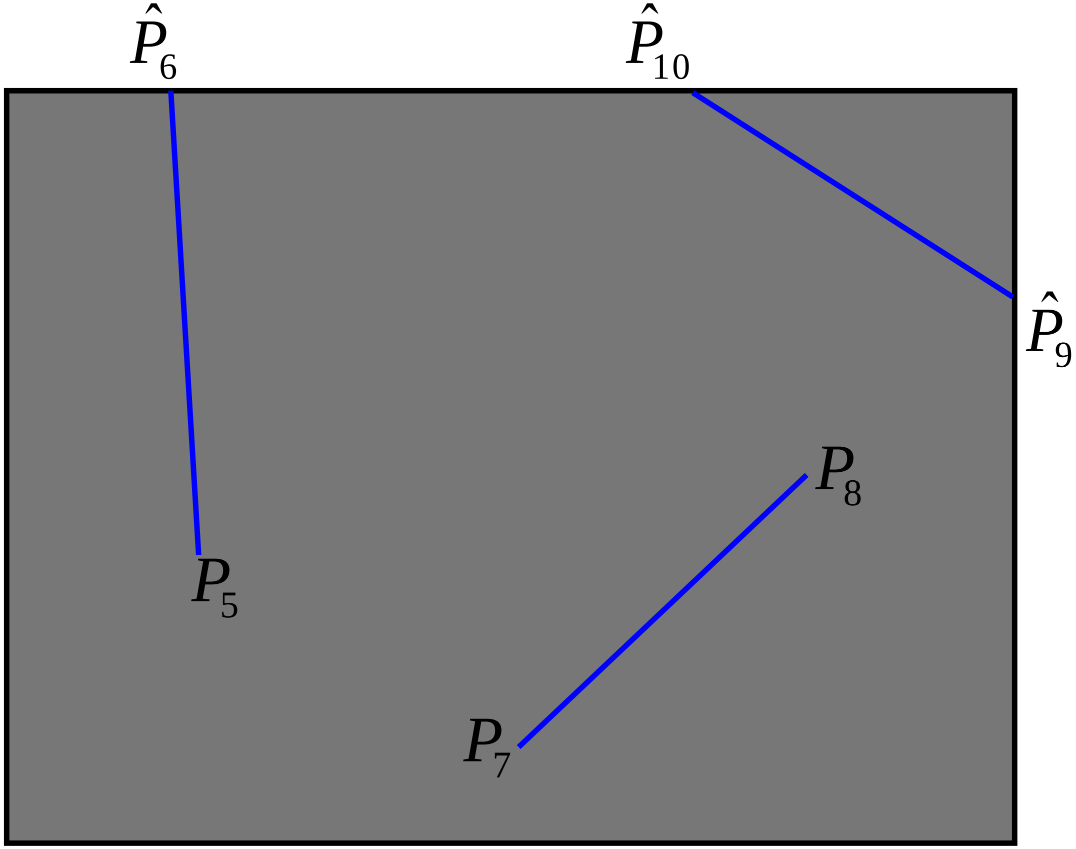
$y_{min} =< y_2 <= y_{max}$ **Yes**

Αποκοπή γραμμών

Clipping Lines

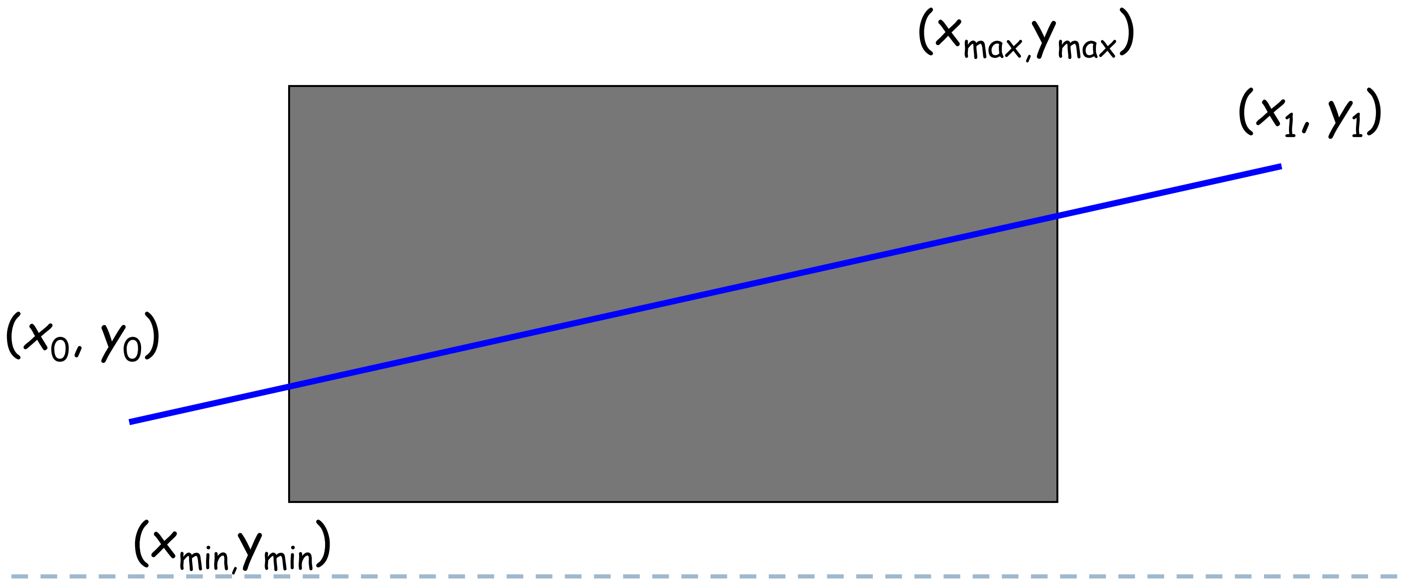


Clipping Lines



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.



Outline

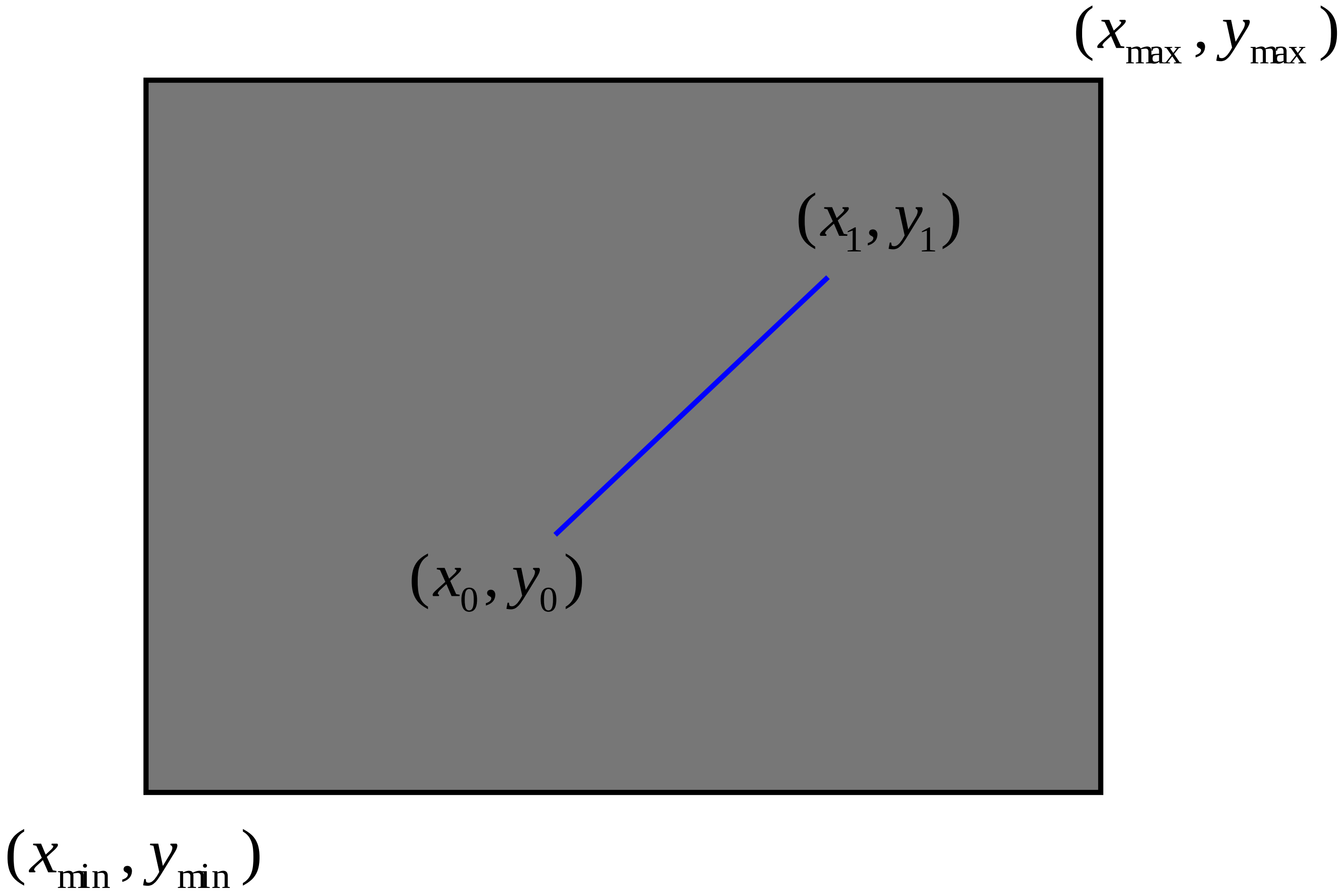
- Simple line clipping algorithm
- Cohen-Sutherland
- Liang-Barsky

Simple line clipping algorithm

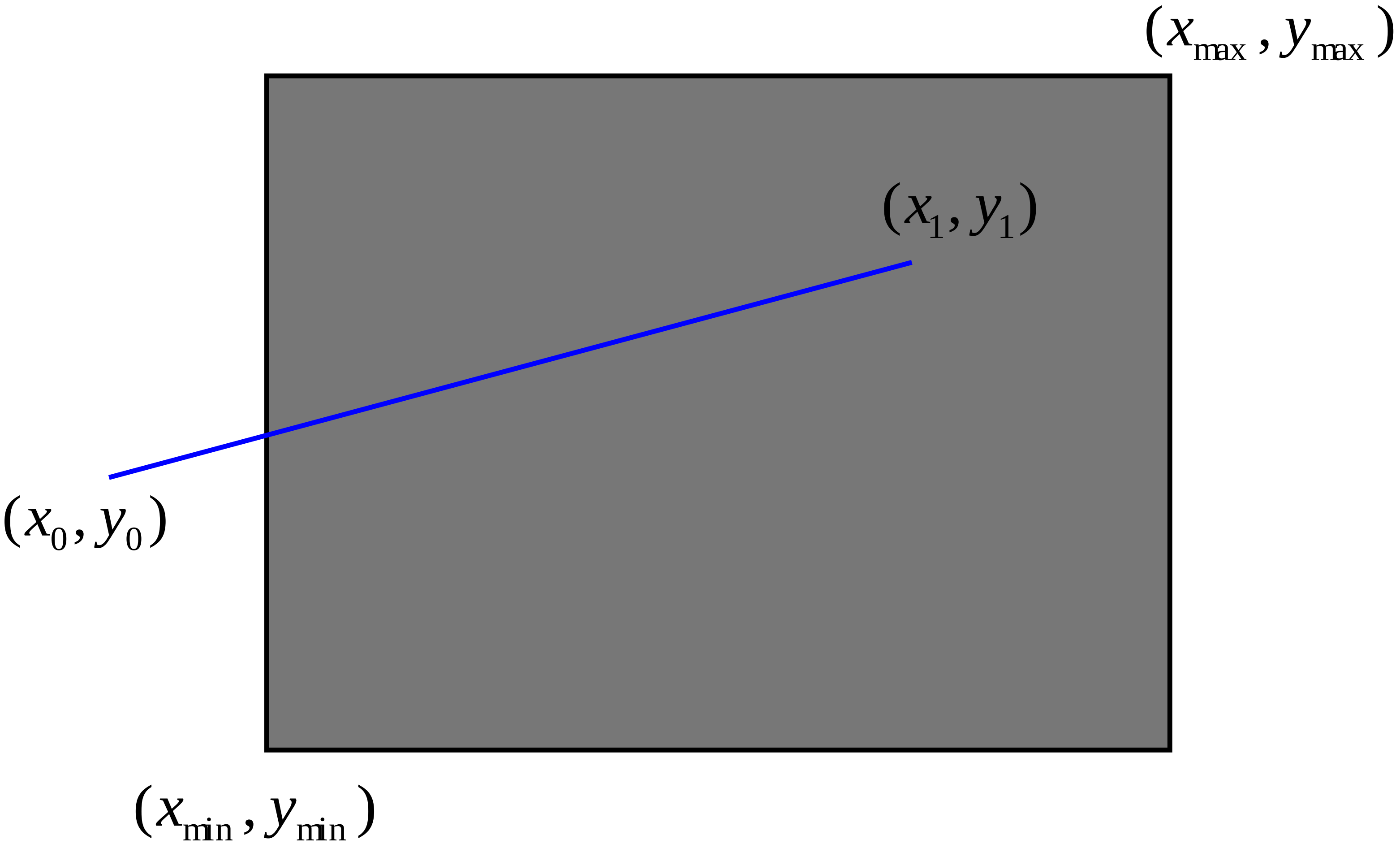
Clipping Lines - Simple Algorithm

- If both end-points inside rectangle, draw line
- If one end-point outside,
  - intersect line with all edges of rectangle
  - clip that point and repeat test

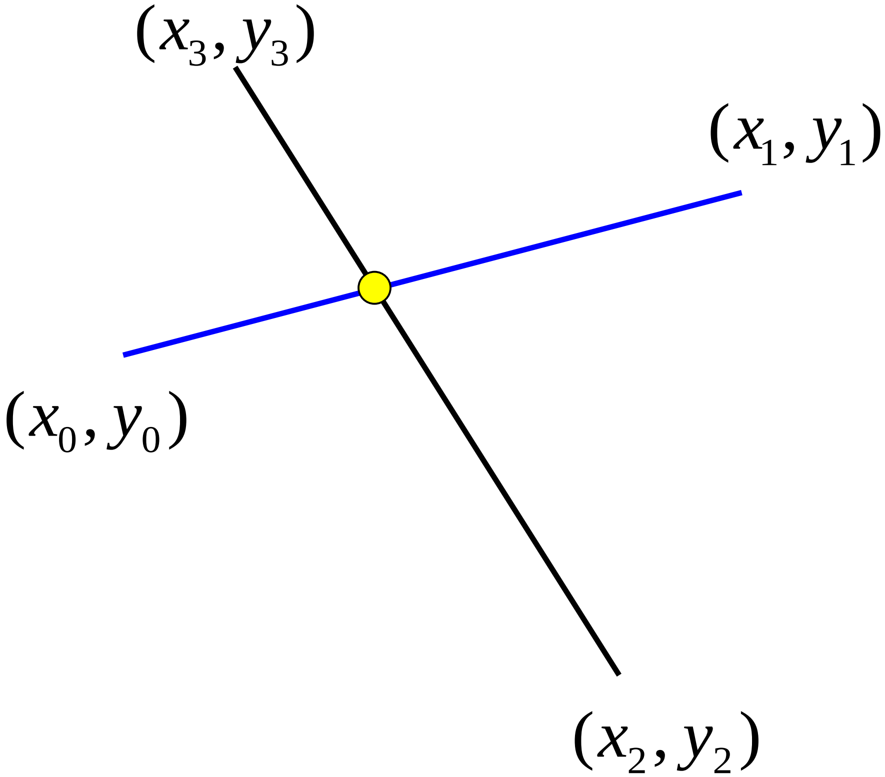
Clipping Lines - Simple Algorithm



Clipping Lines - Simple Algorithm

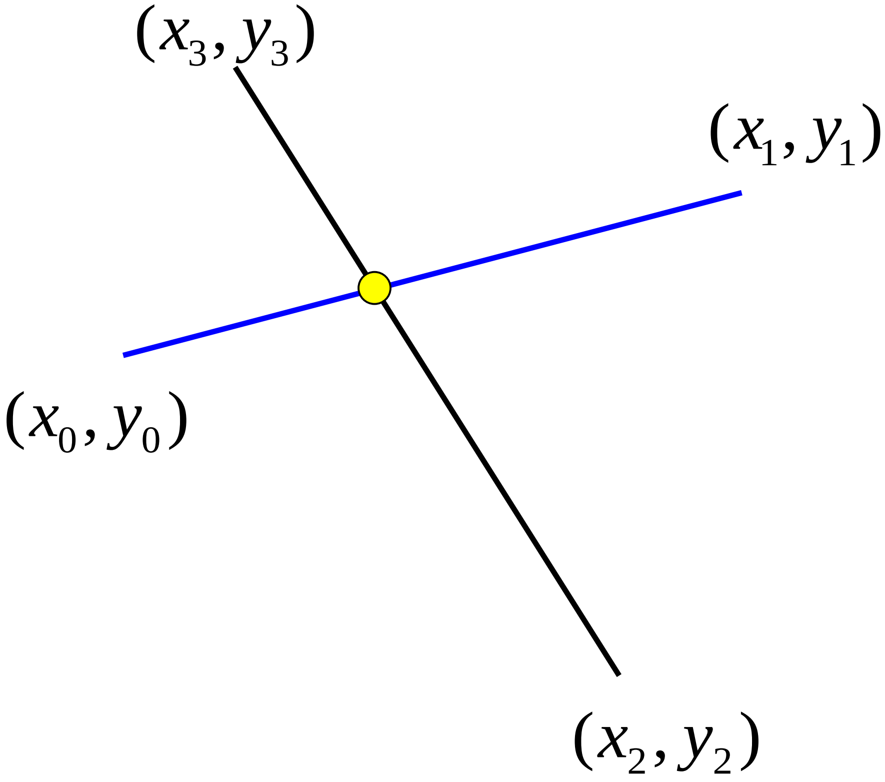


Intersecting Two Lines



Intersecting Two Lines

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

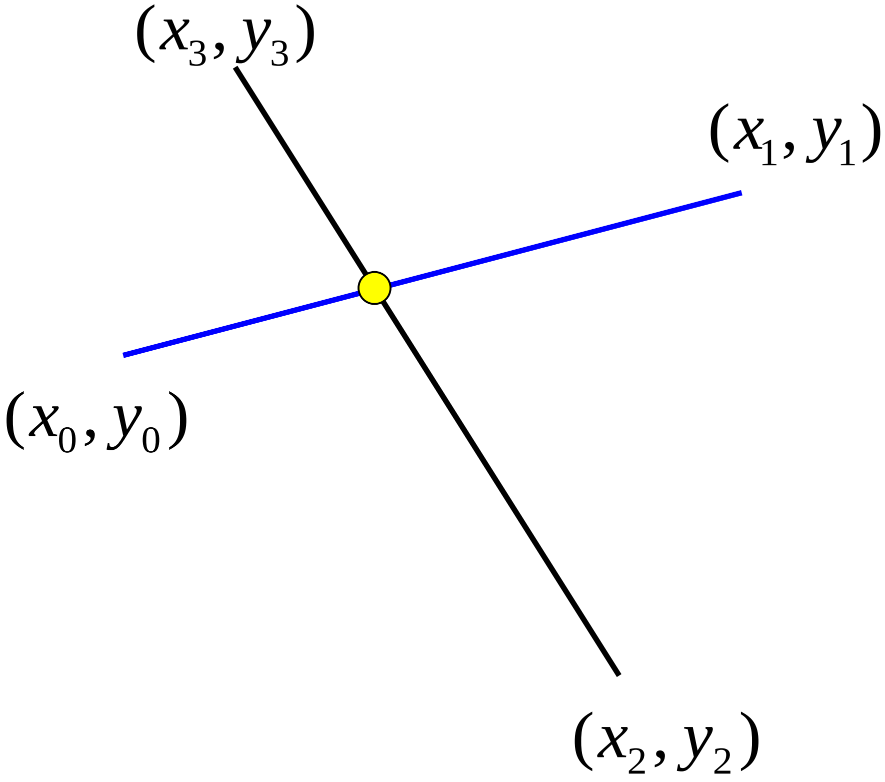


Intersecting Two Lines

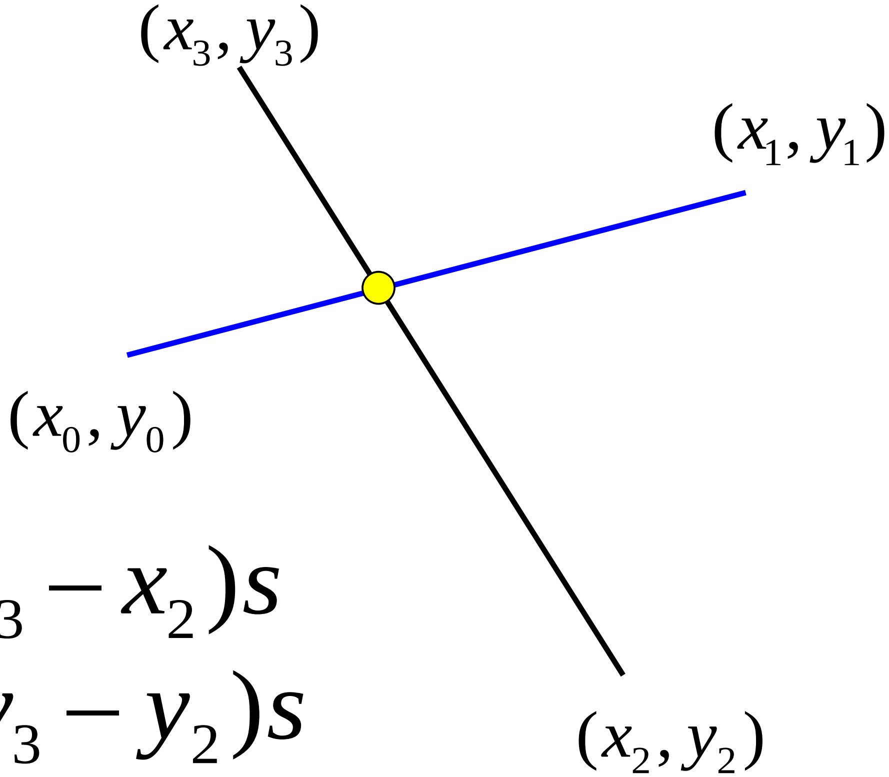
$$x(t) = x_0 + (x_1 - x_0)t$$
$$y(t) = y_0 + (y_1 - y_0)t$$

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

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

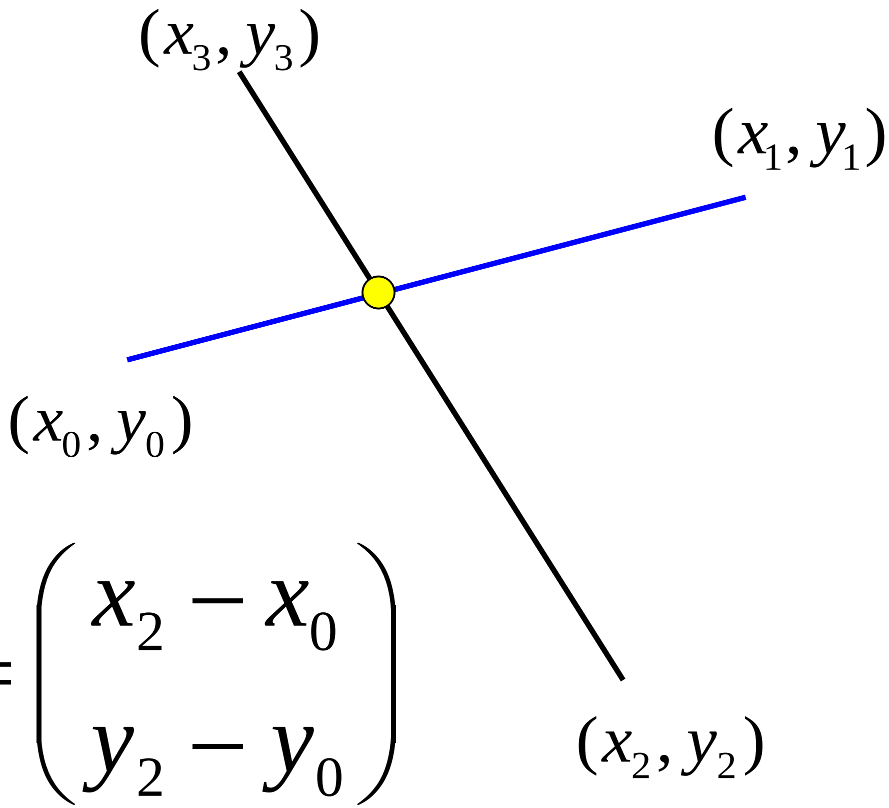


Intersecting Two Lines



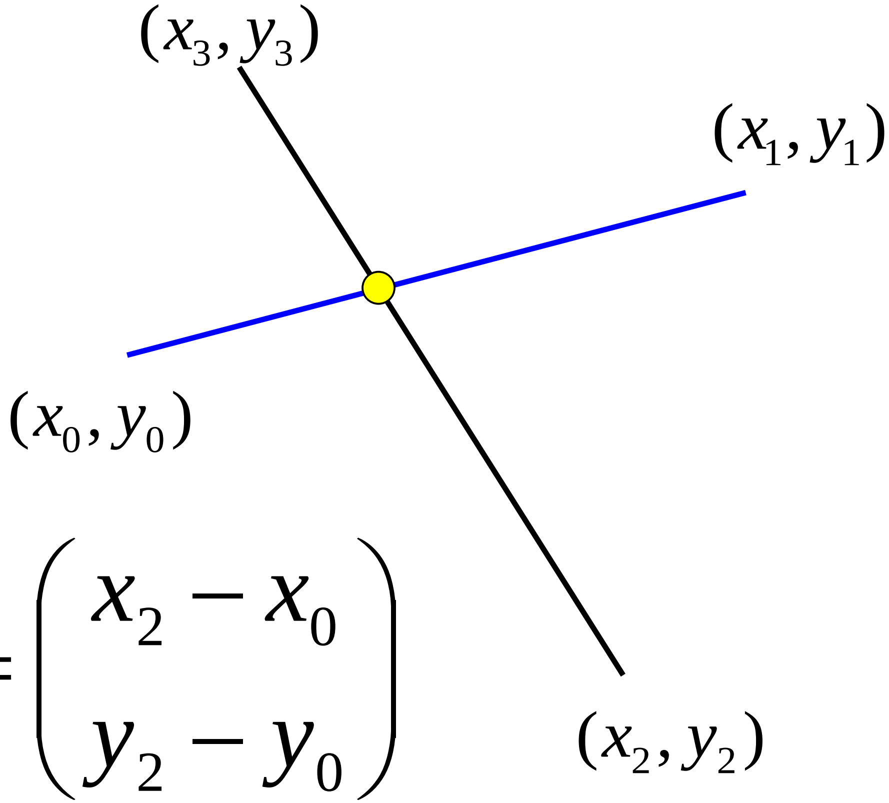
$$x_0 + (x_1 - x_0)t = x_2 + (x_3 - x_2)s$$
$$y_0 + (y_1 - y_0)t = y_2 + (y_3 - y_2)s$$

Intersecting Two Lines



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

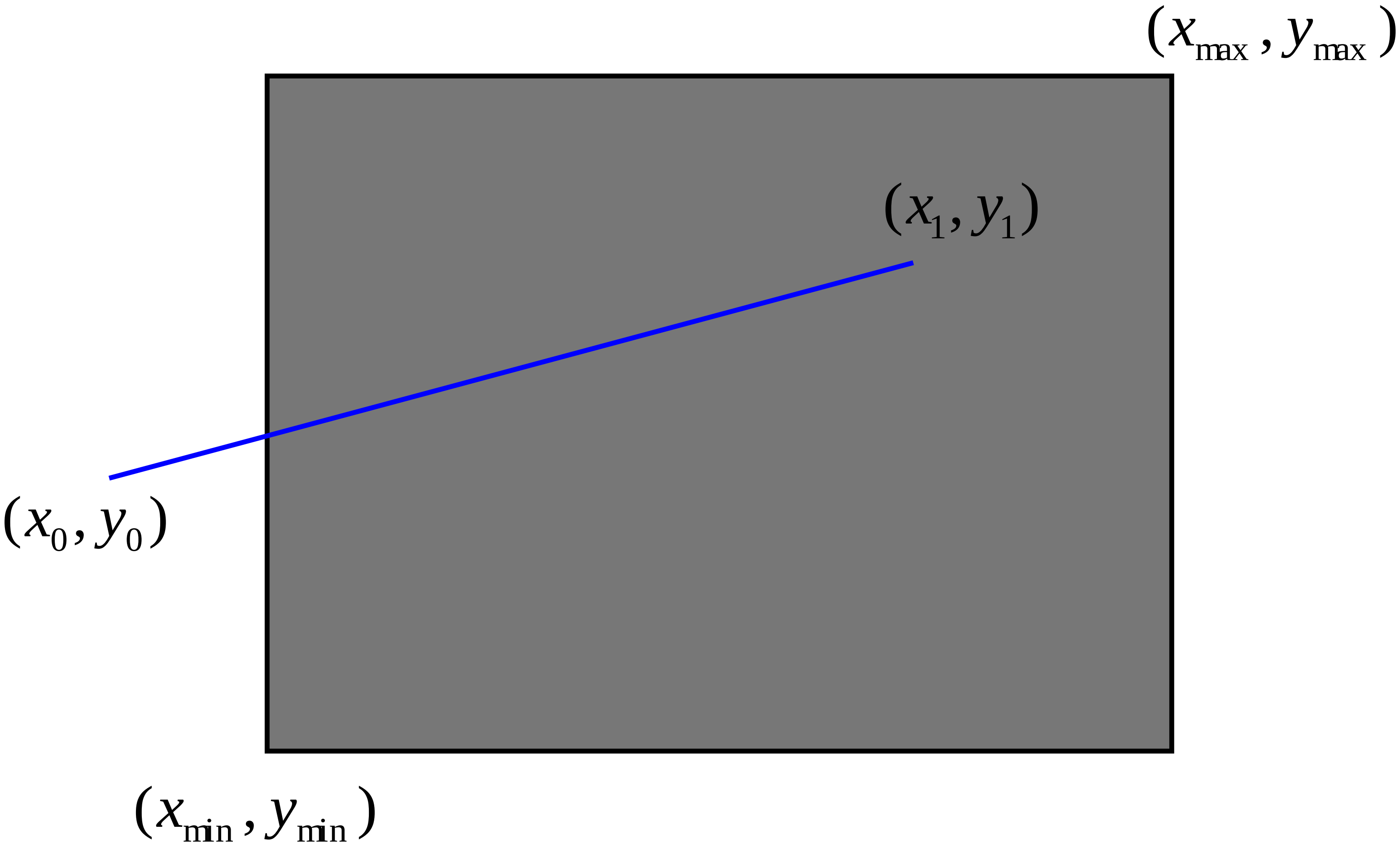
Intersecting Two Lines



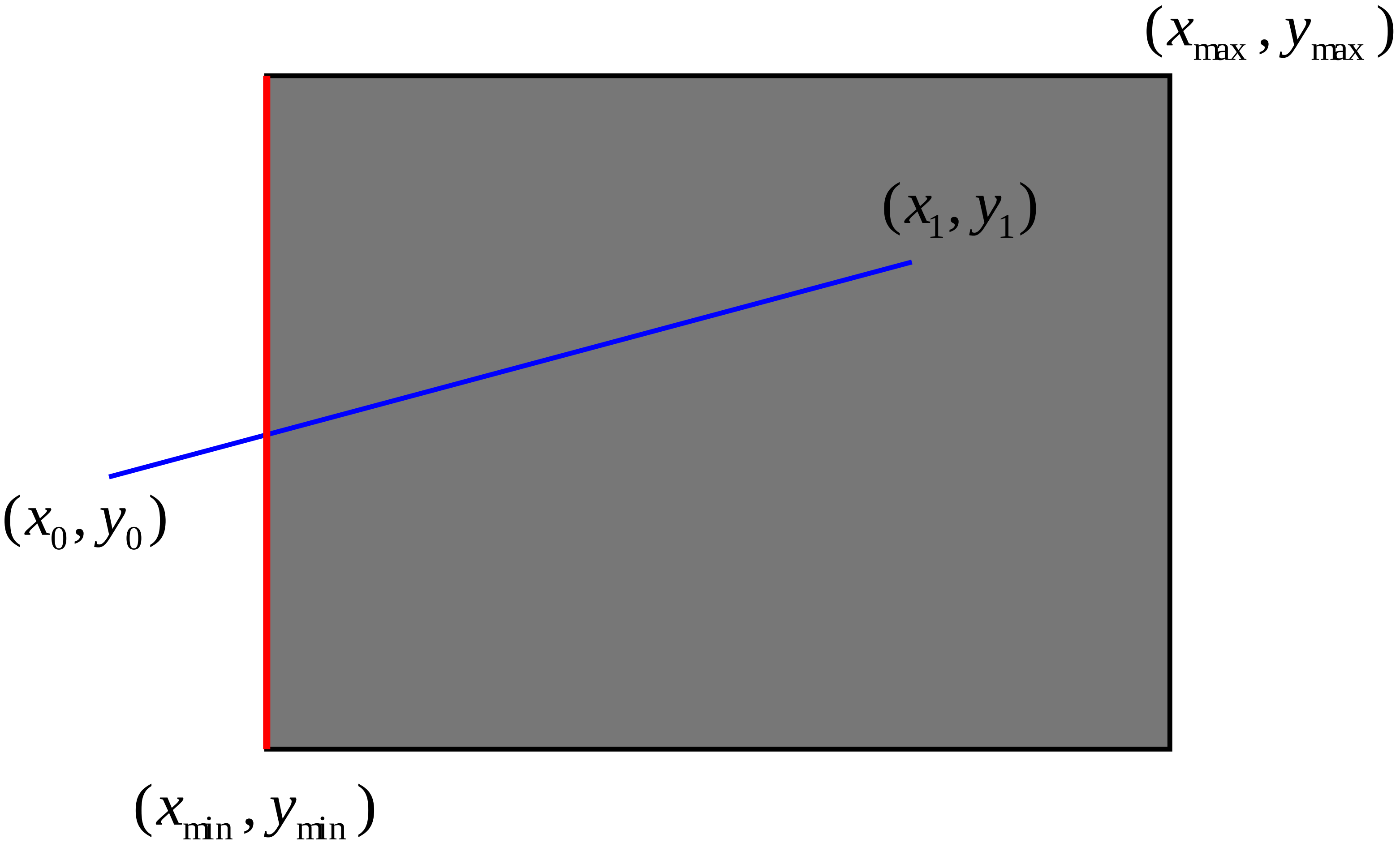
$$\begin{pmatrix} x_1 - x_0 & x_2 - x_3 \\ y_1 - y_0 & y_2 - y_3 \end{pmatrix} \begin{pmatrix} t \\ s \end{pmatrix} = \begin{pmatrix} x_2 - x_0 \\ y_2 - y_0 \end{pmatrix}$$

Substitute *t* or *s* back into equation to find intersection

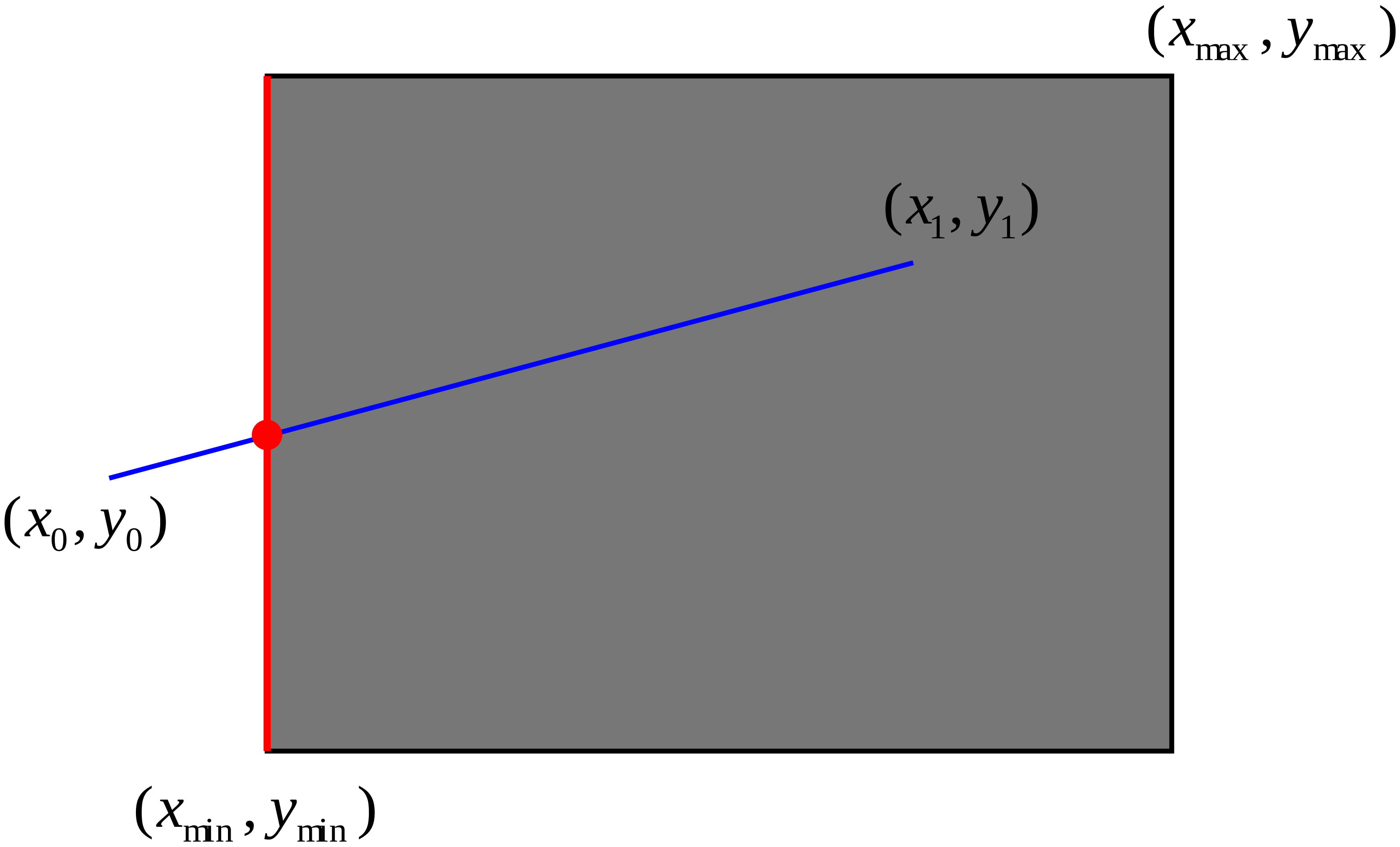
Clipping Lines - Simple Algorithm



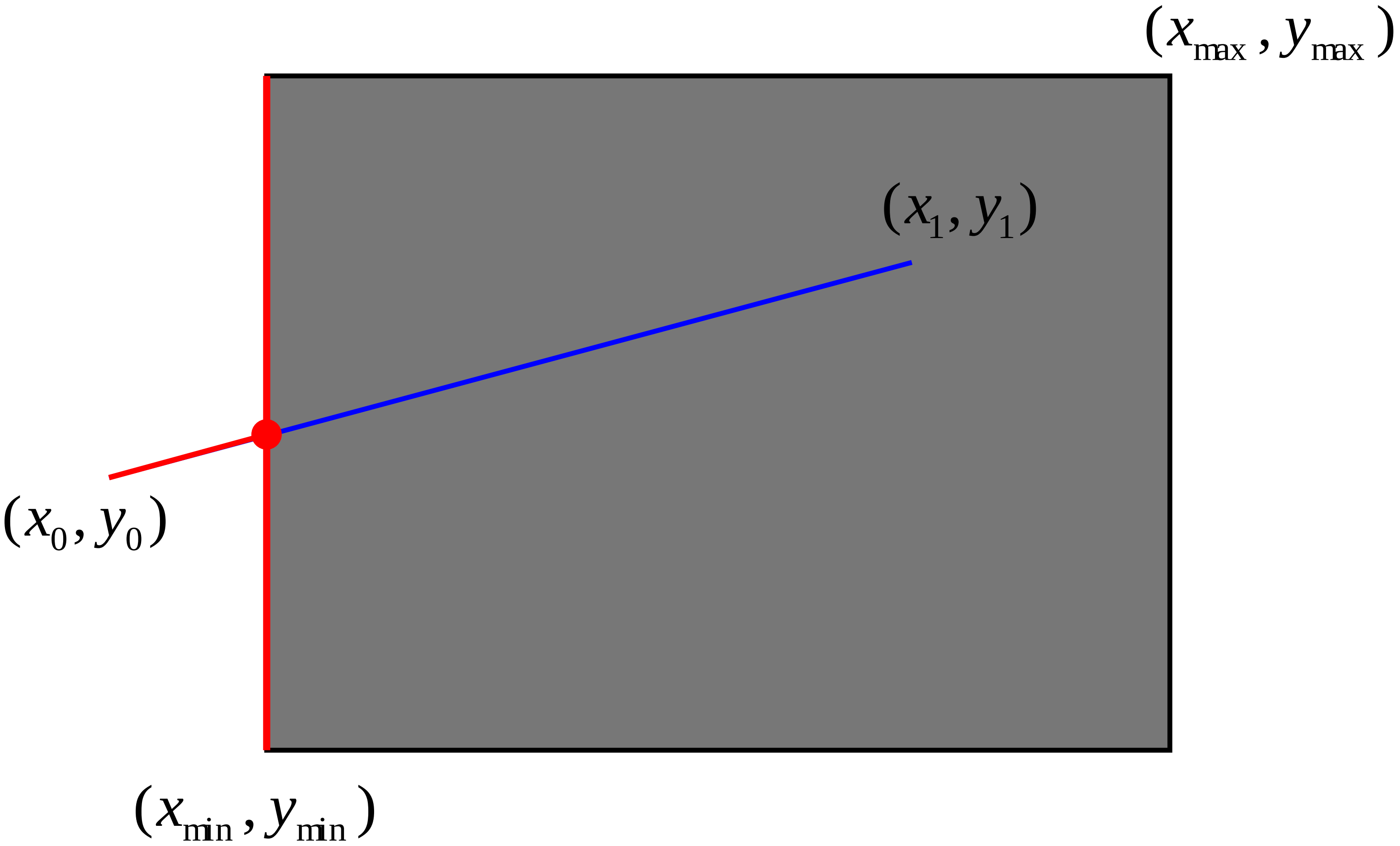
Clipping Lines - Simple Algorithm



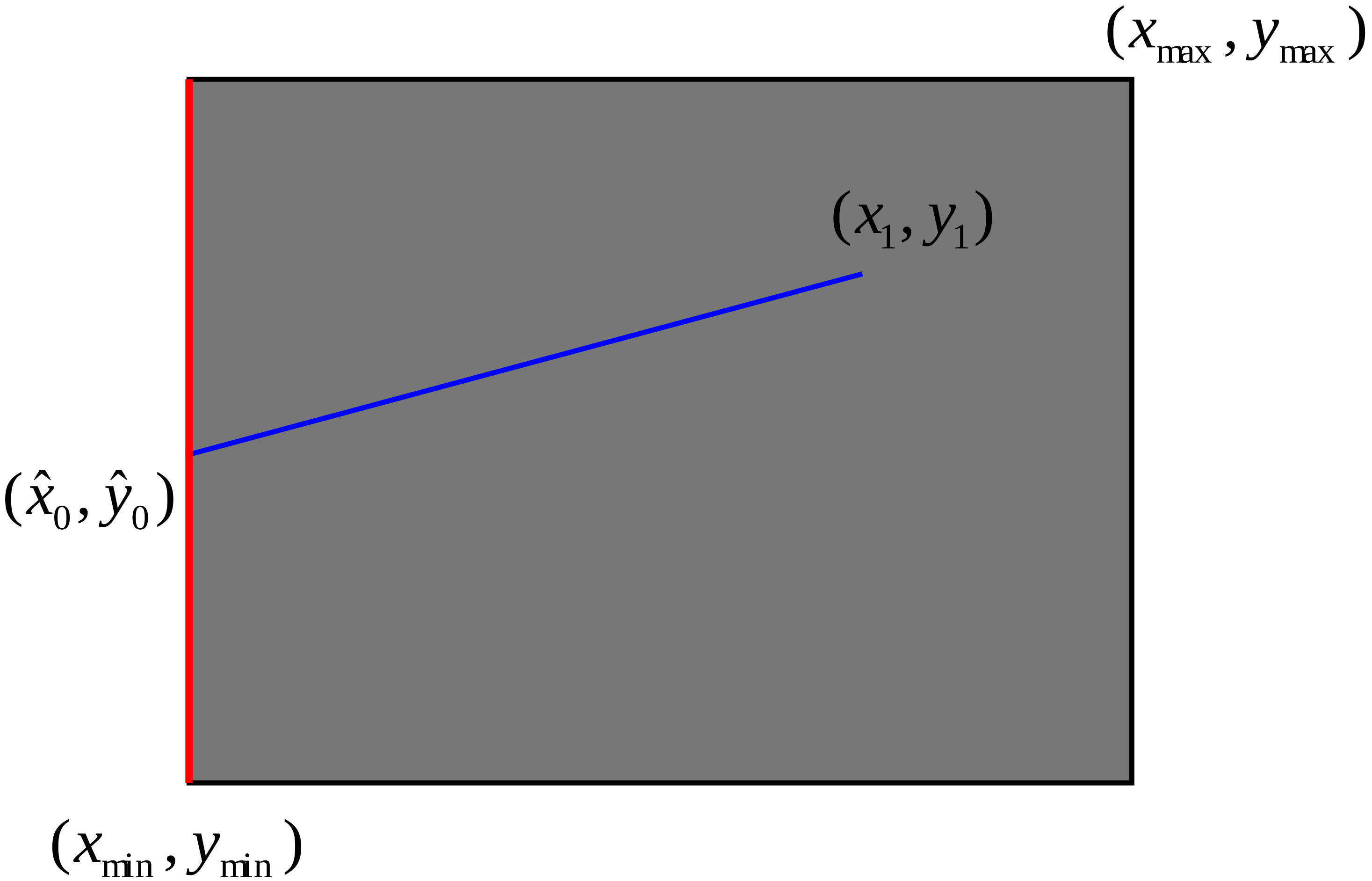
Clipping Lines - Simple Algorithm



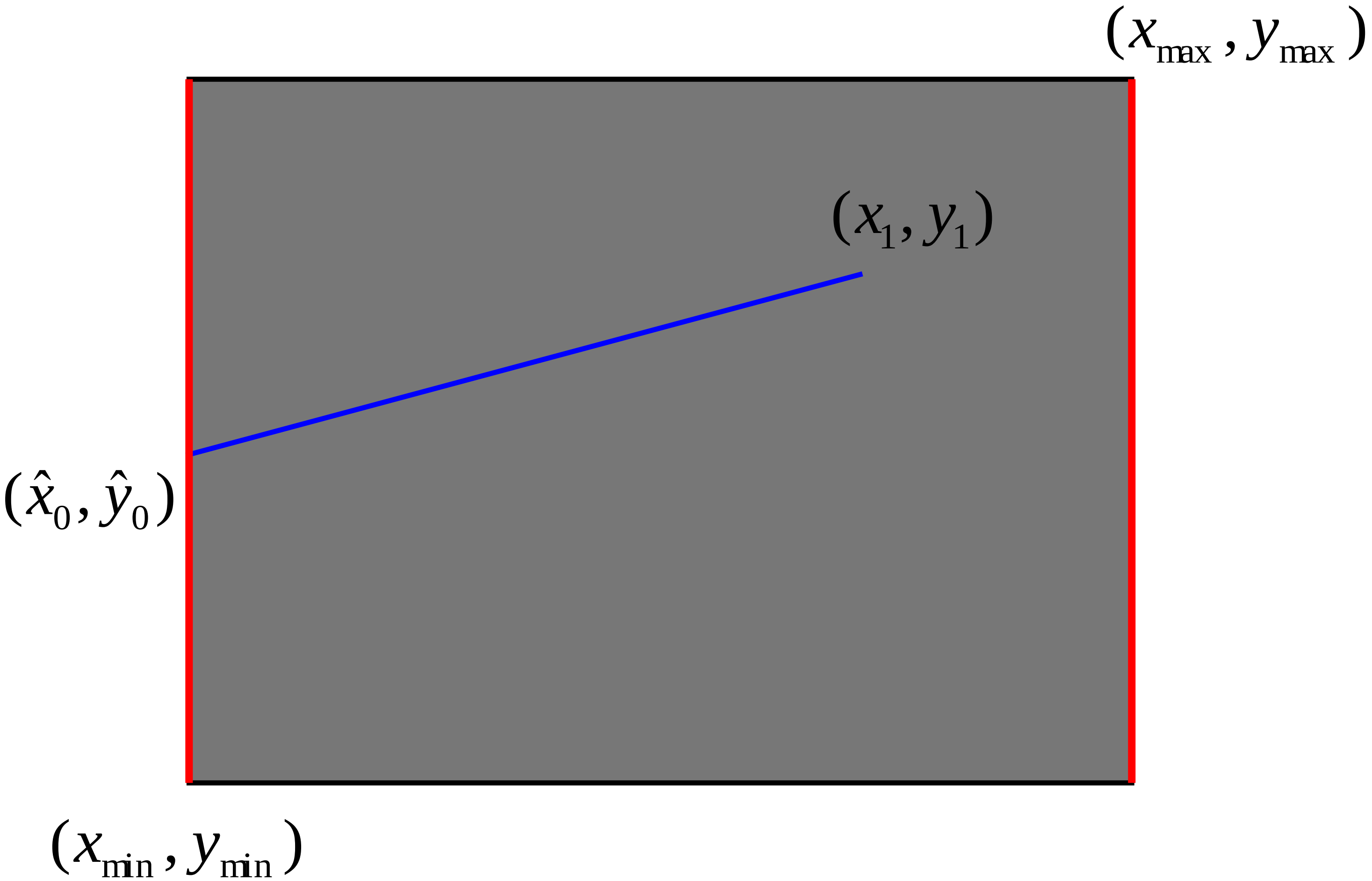
Clipping Lines - Simple Algorithm



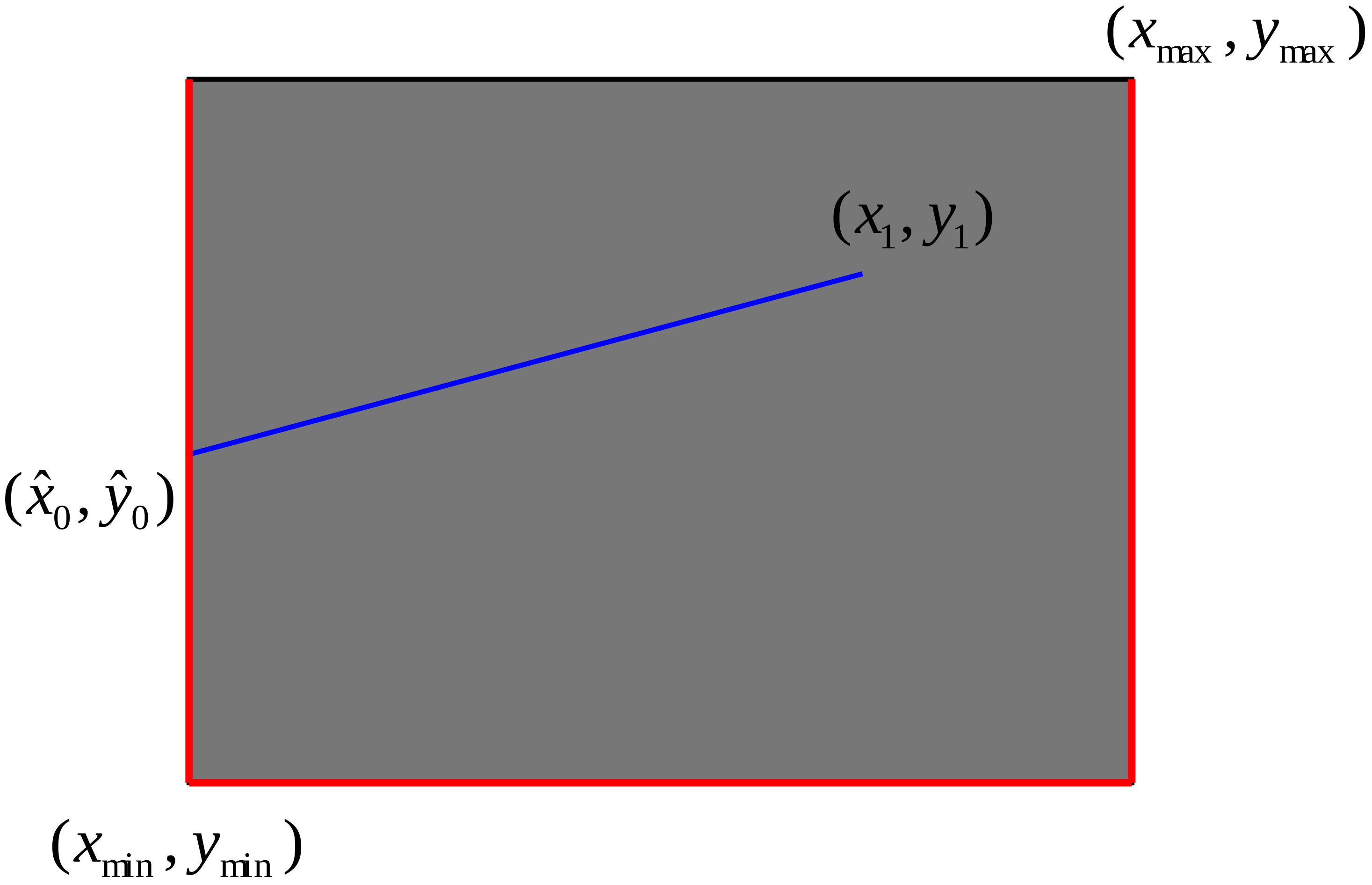
Clipping Lines - Simple Algorithm



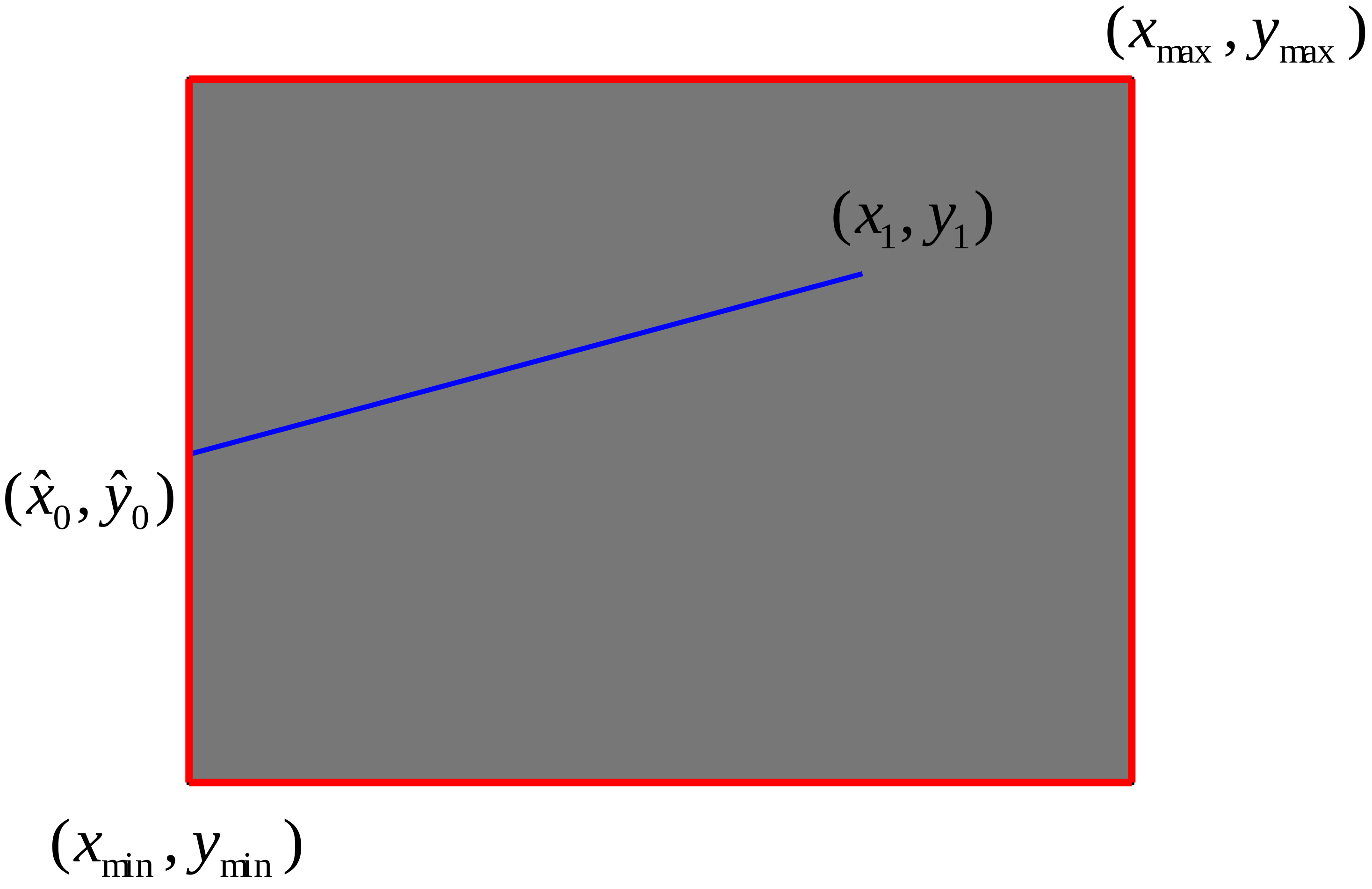
Clipping Lines - Simple Algorithm



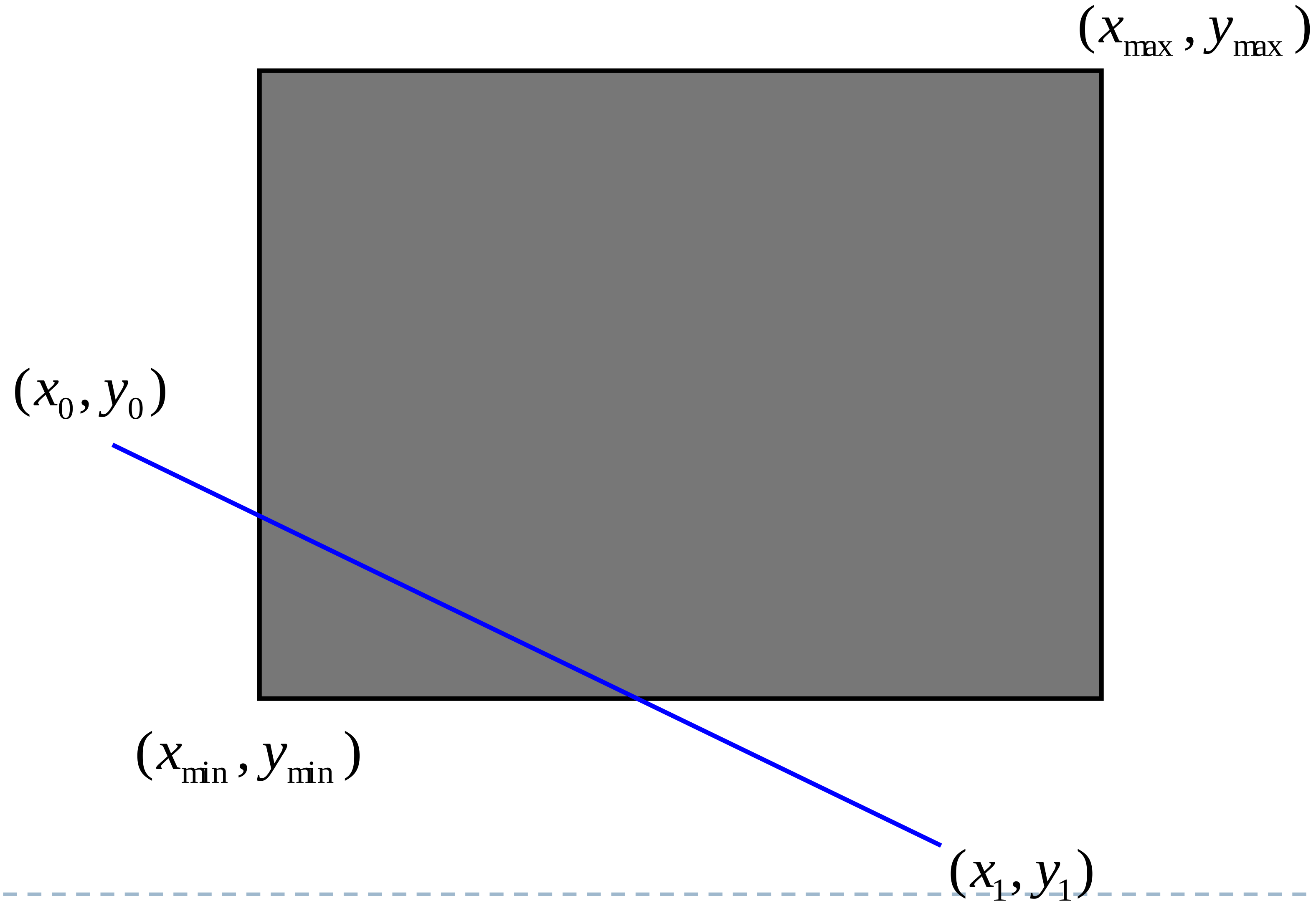
Clipping Lines - Simple Algorithm



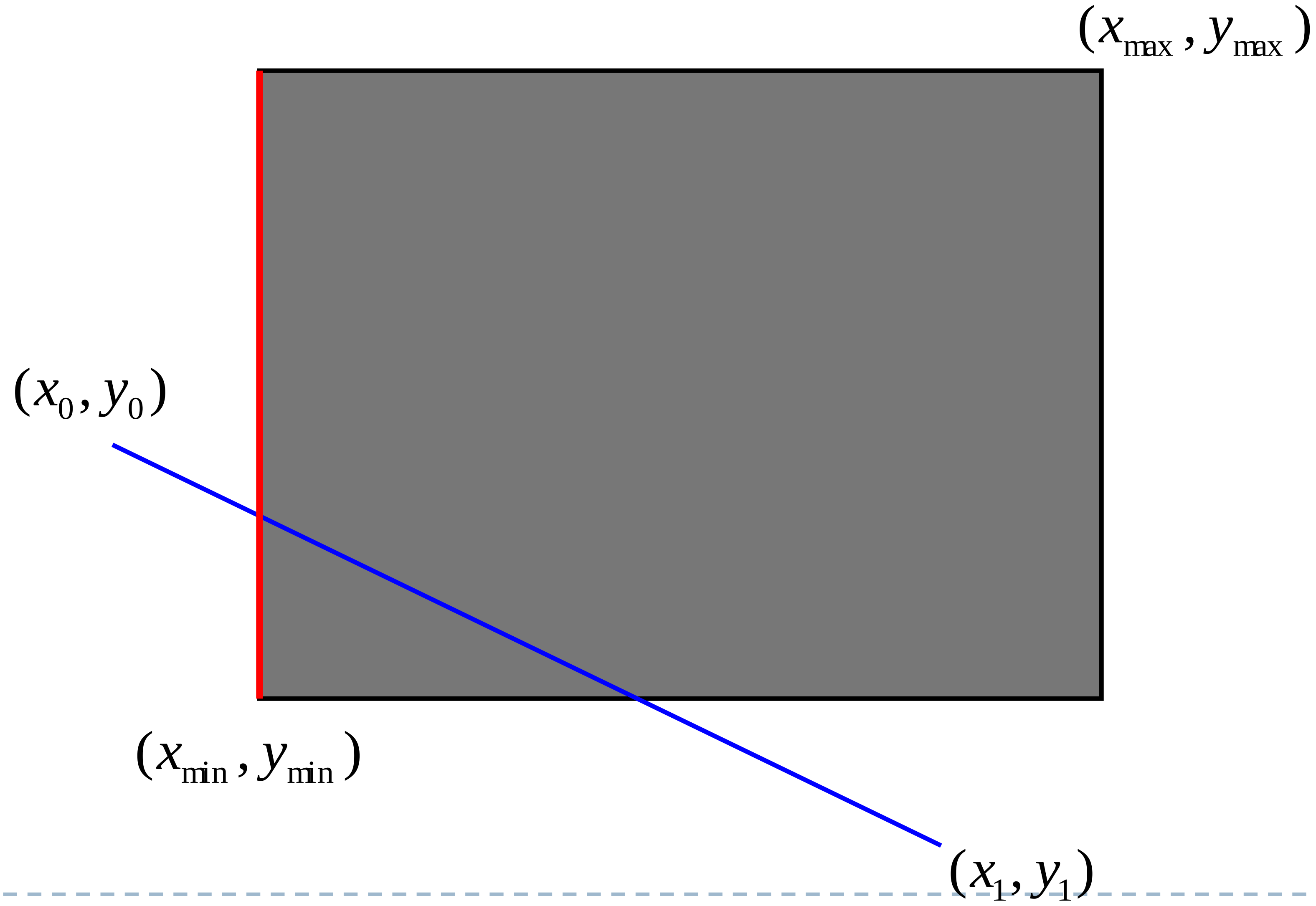
Clipping Lines - Simple Algorithm



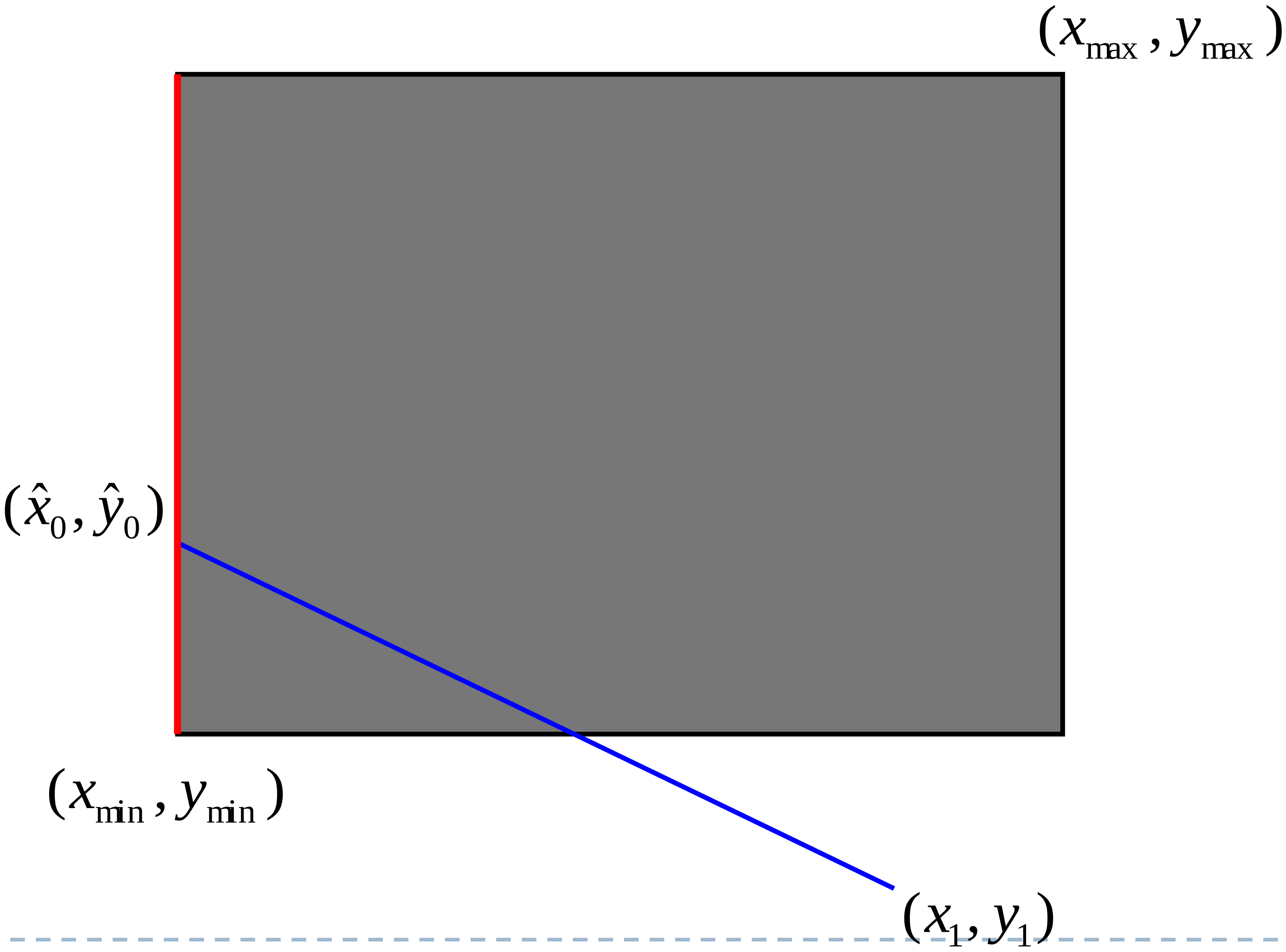
Clipping Lines - Simple Algorithm



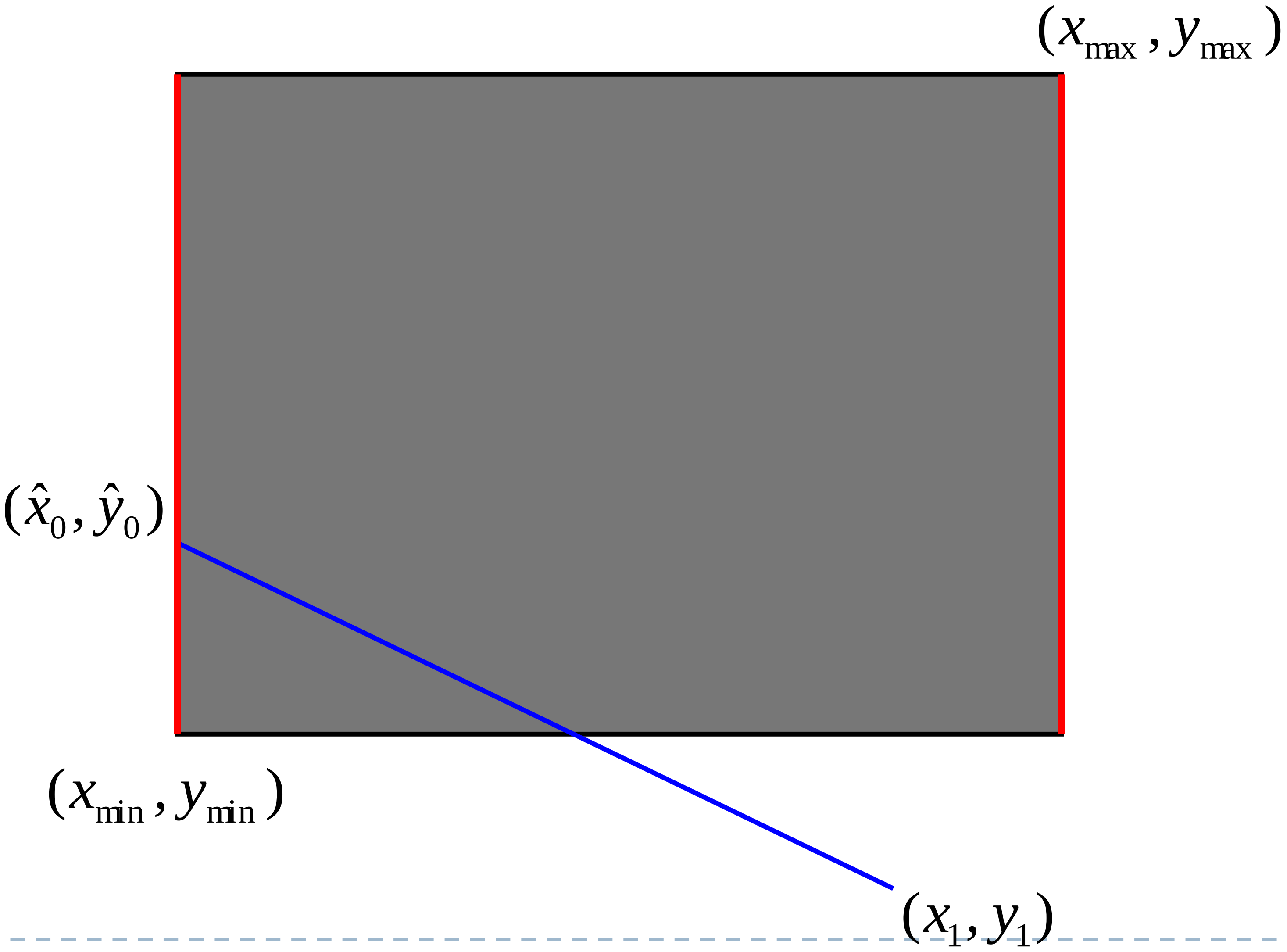
Clipping Lines - Simple Algorithm



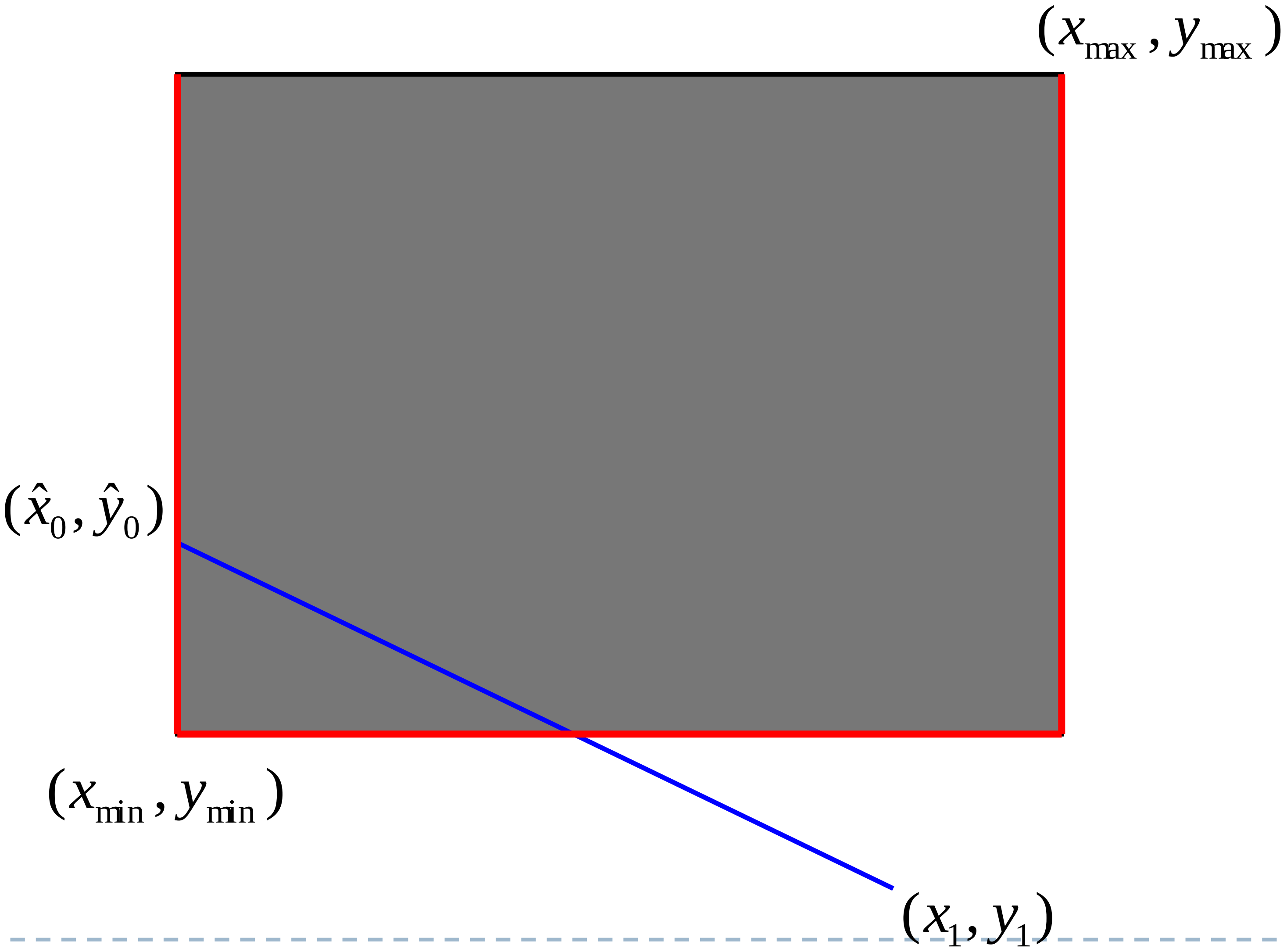
Clipping Lines - Simple Algorithm



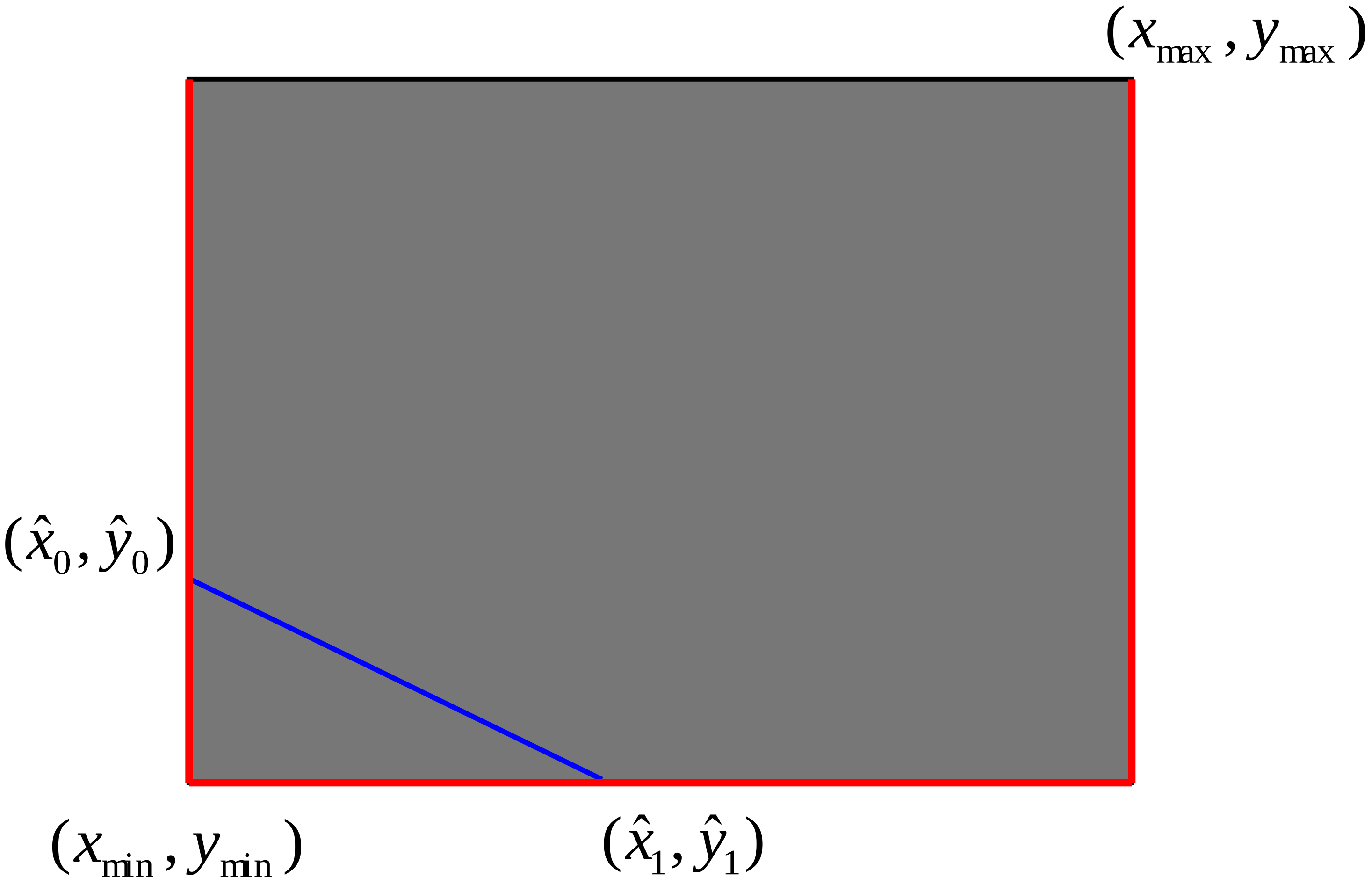
Clipping Lines - Simple Algorithm



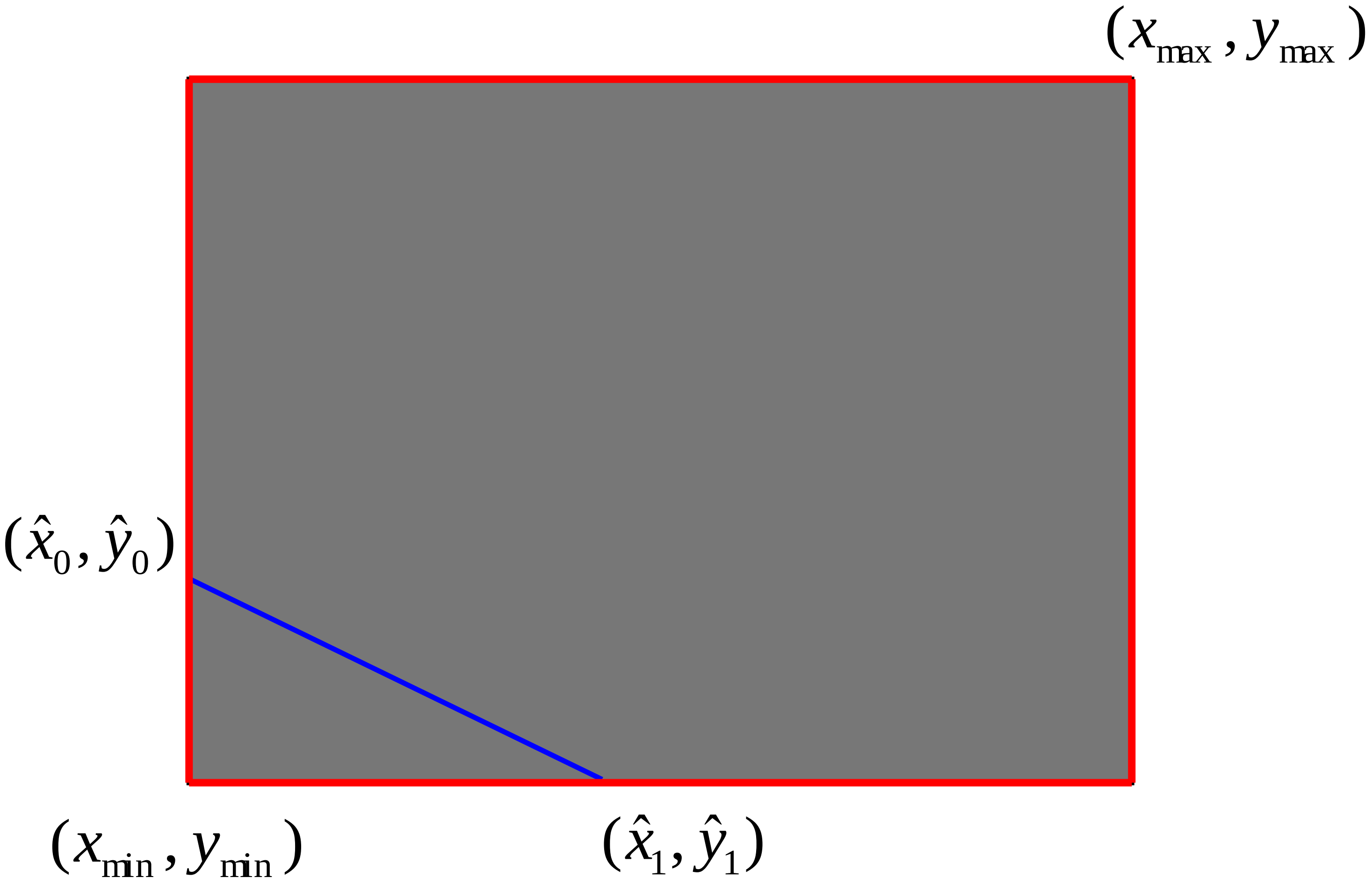
Clipping Lines - Simple Algorithm



Clipping Lines - Simple Algorithm



Clipping Lines - Simple Algorithm



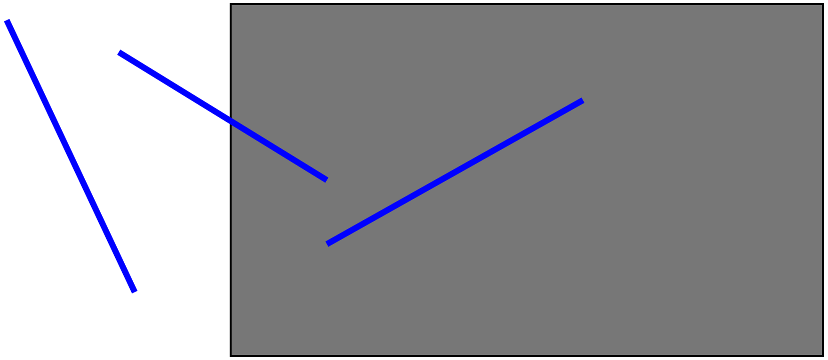
Clipping Lines - Simple Algorithm

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

- Is there a better way?

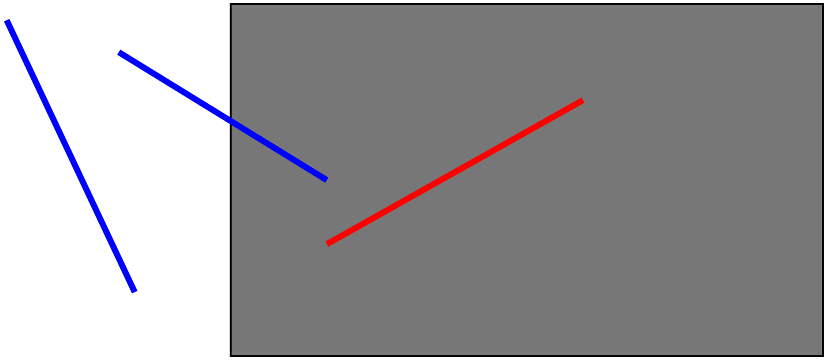
Trivial Accepts

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



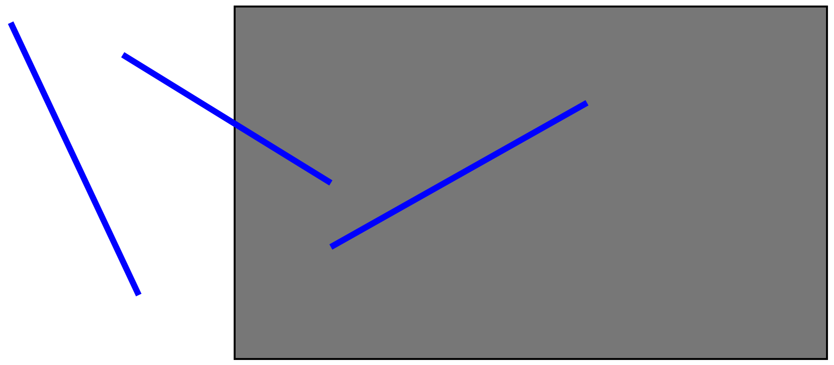
Trivial Accepts

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



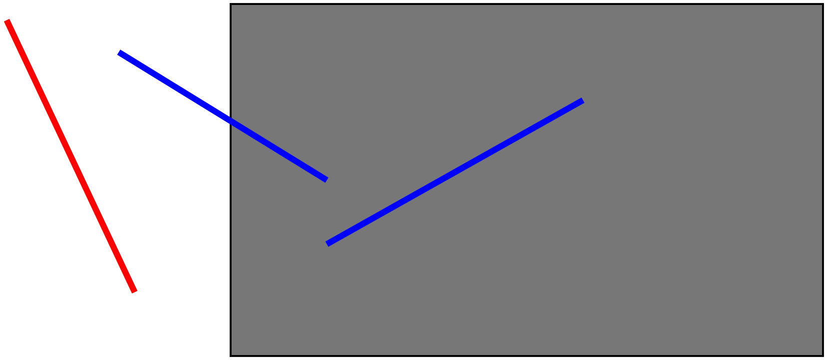
Trivial Rejects

- 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

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

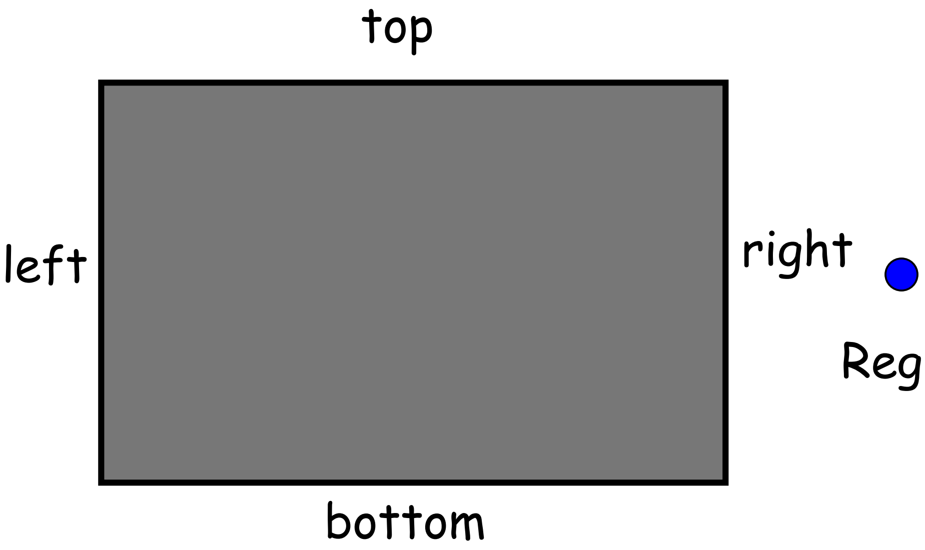
Cohen-Sutherland Algorithm

- 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 edges

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

Cohen-Sutherland Algorithm

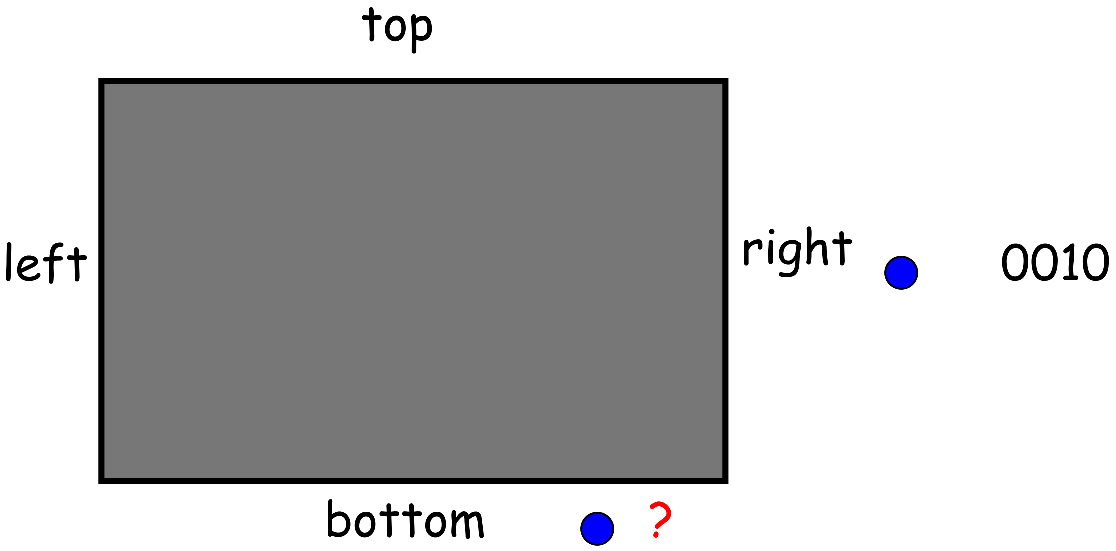
bit 4	bit 3	bit 2	bit 1
top	bottom	right	left



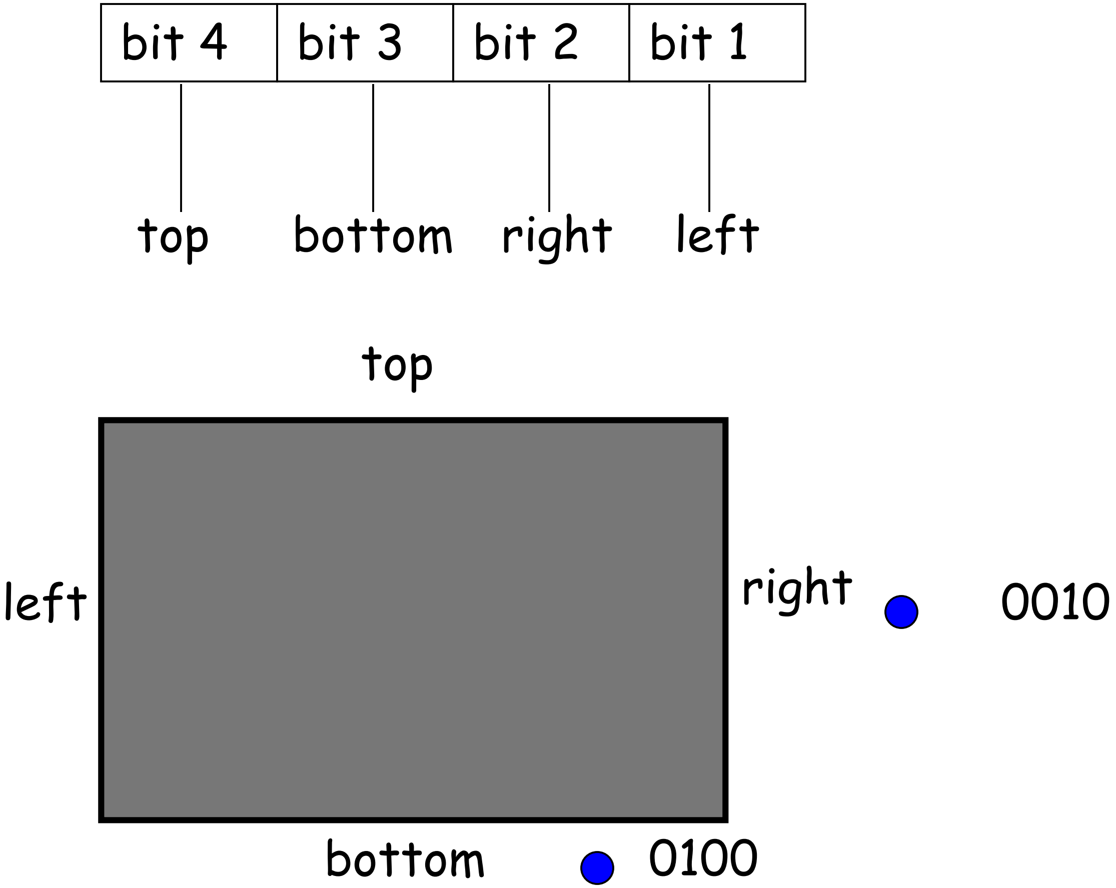
Region code?

Cohen-Sutherland Algorithm

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



Cohen-Sutherland Algorithm



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

top

left

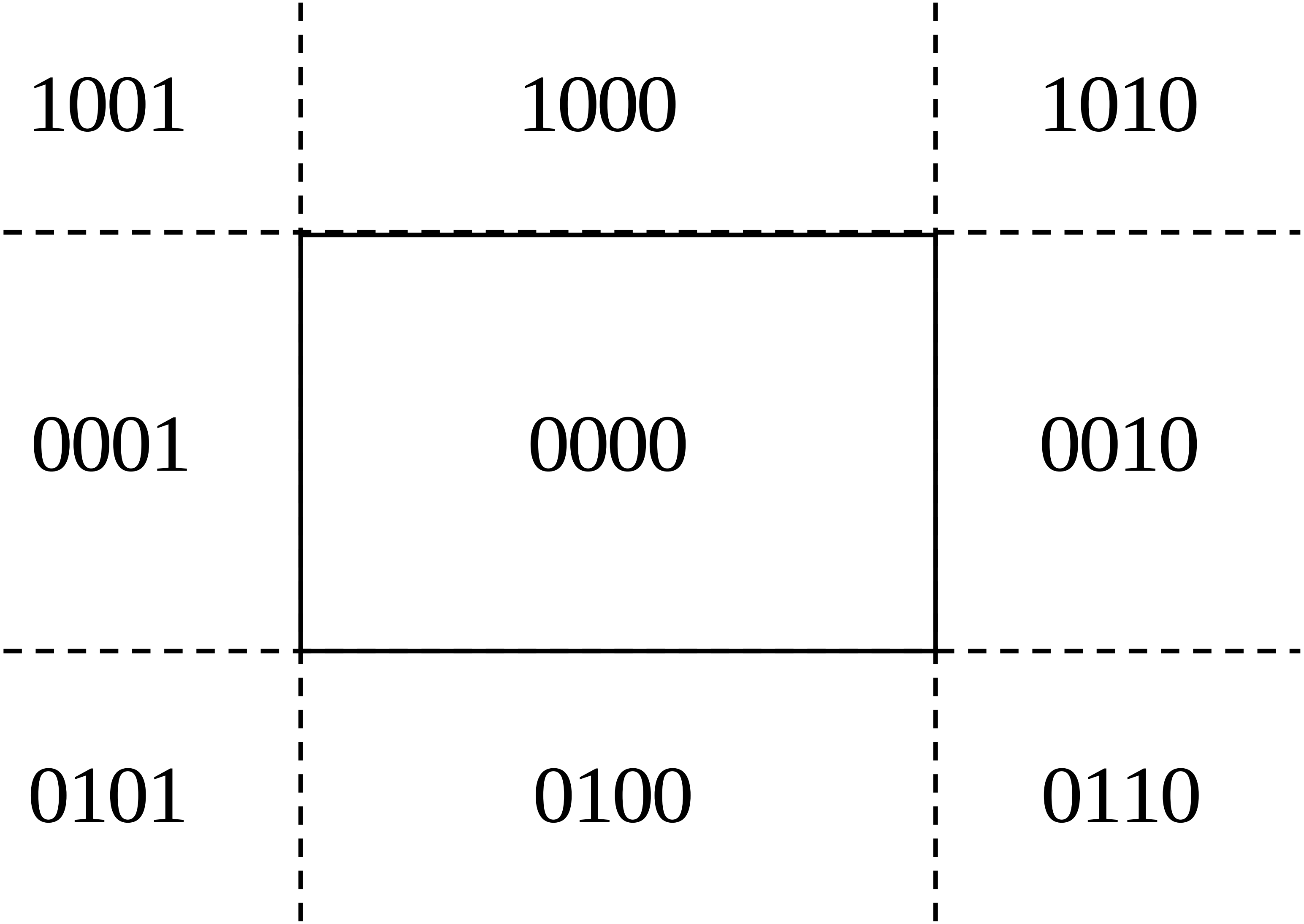
right

0010

bottom

0100

Cohen-Sutherland Algorithm



1001	1000	1010
0001	0000	0010
0101	0100	0110

Cohen-Sutherland Algorithm

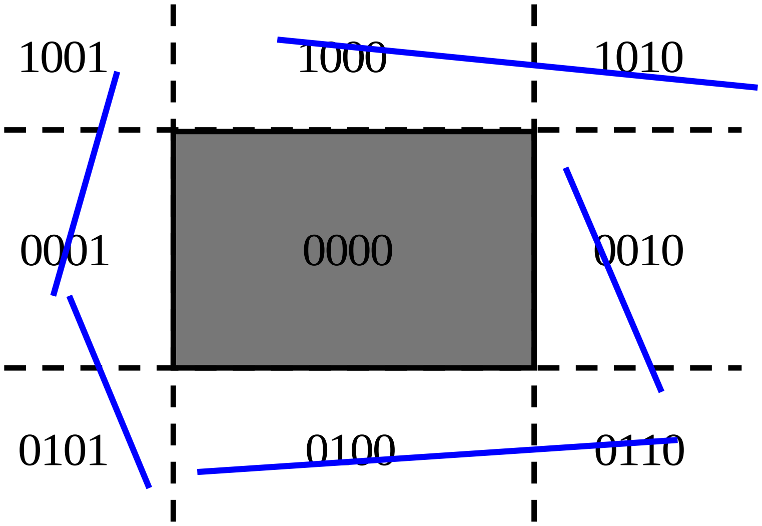
- Classify p_0, p_1 using region codes c_0, c_1
- If $c_0 \wedge c_1 \neq 0$, trivially reject
- If $c_0 \vee c_1 = 0$, trivially accept
- Otherwise reduce to trivial cases by splitting into two segments

Cohen-Sutherland Algorithm

- Classify p_0, p_1 using region codes c_0, c_1
- 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

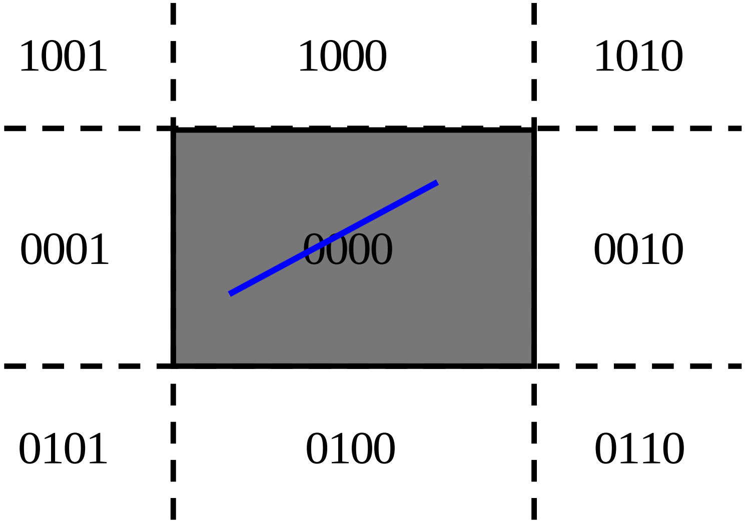
- Classify p_0, p_1 using region codes c_0, c_1
- If $c_0 \wedge c_1 \neq 0$, trivially reject
- If $c_0 \vee c_1 = 0$, trivially accept
- Otherwise reduce to trivial cases by splitting into two segments



Line is outside the window! reject

Cohen-Sutherland Algorithm

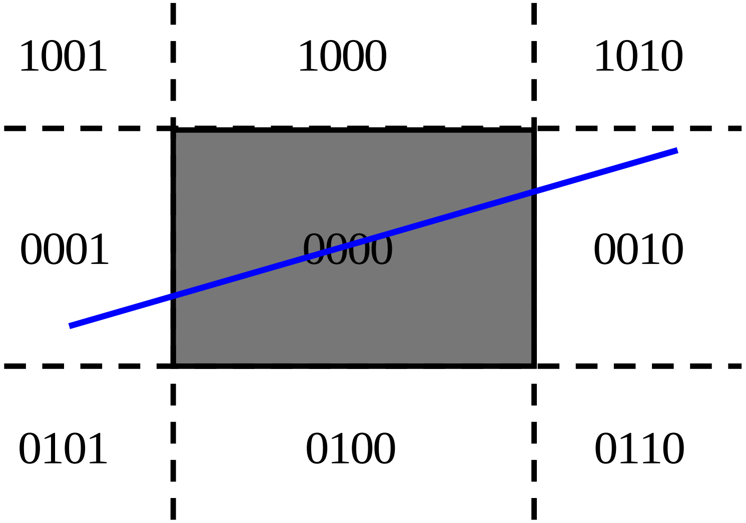
- Classify p_0, p_1 using region codes c_0, c_1
- If $c_0 \wedge c_1 \neq 0$, trivially reject
- If $c_0 \vee c_1 = 0$, trivially accept
- Otherwise reduce to trivial cases by splitting into two segments



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



Window Intersection

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

- $y_{intersect} = y_1 + m(x_{left} - x1)$
  - ☐ where $m=(y_2-y_1)/(x_2-x_1)$

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

- $x_{intersect} = x_1 + (y_{top} - y1)/m$
  - ☐ where $m=(y_2-y_1)/(x_2-x_1)$

Window Intersection

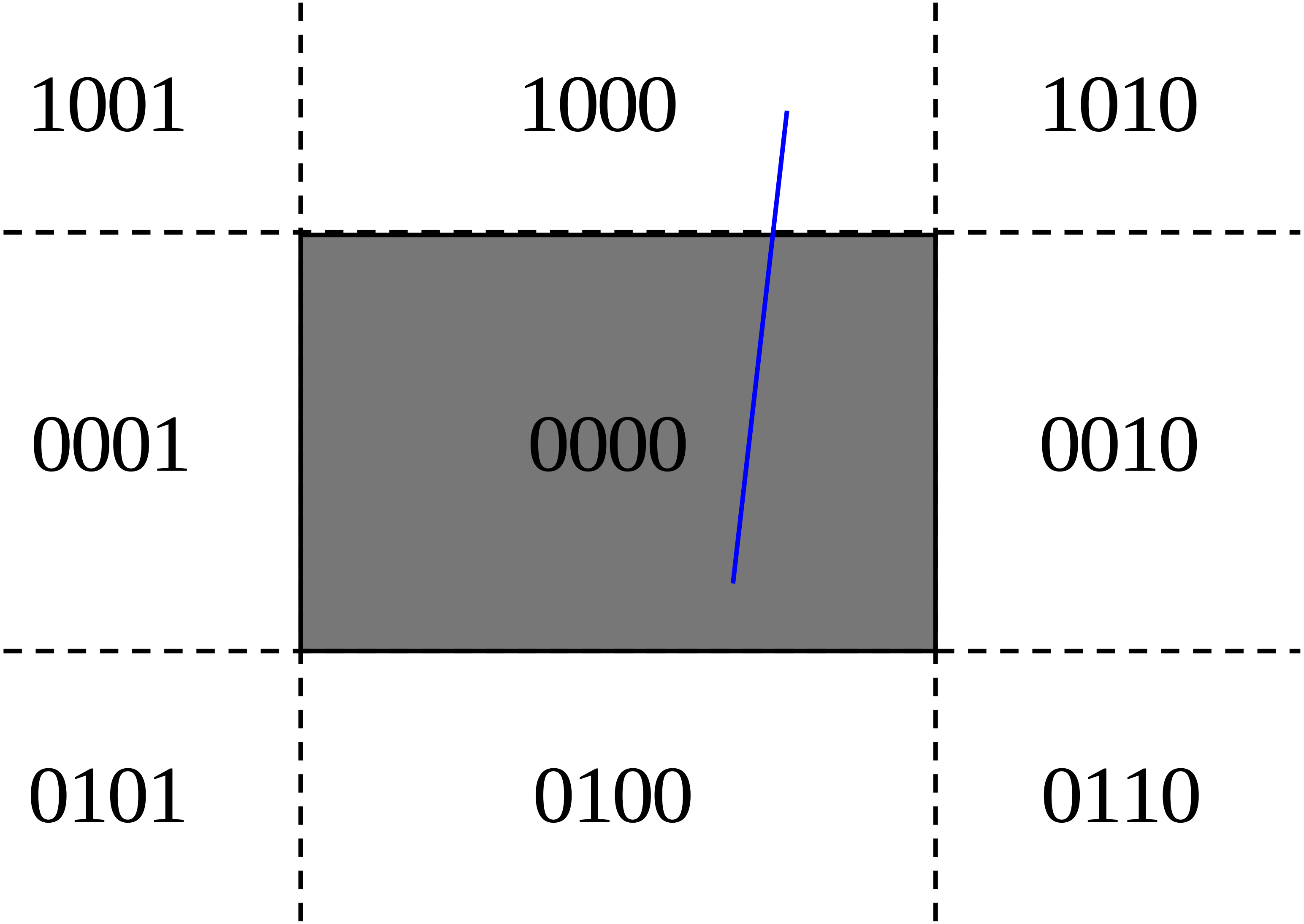
$(x_1, y_1), (x_2, y_2)$ intersect with vertical edge at x_{right}

- $y_{intersect} = y_1 + m(x_{right} - x1)$
  - ☐ where $m=(y_2-y_1)/(x_2-x_1)$

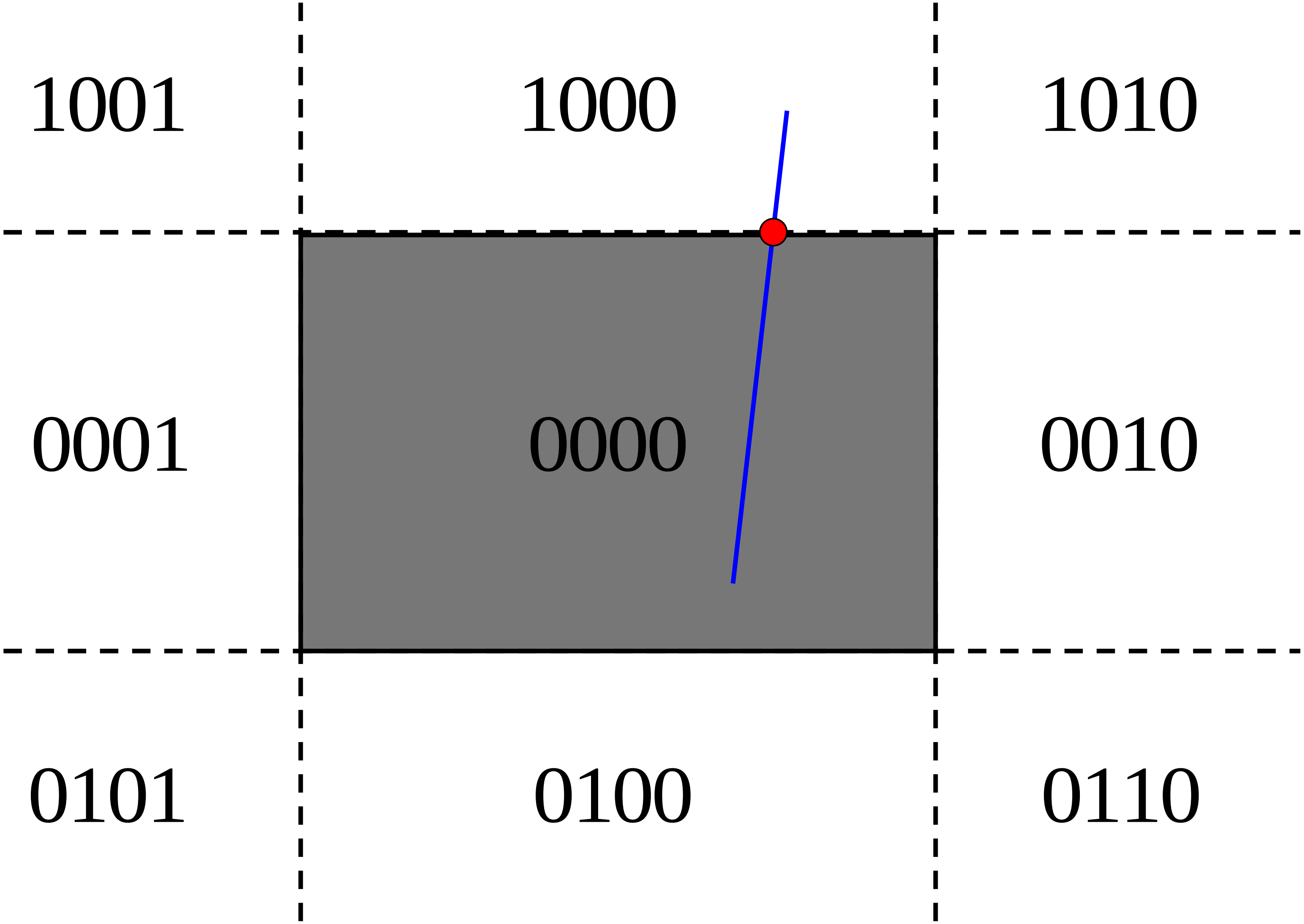
$(x_1, y_1), (x_2, y_2)$ intersect with horizontal edge at y_{bottom}

- $x_{intersect} = x_1 + (y_{bottom} - y1)/m$
  - ☐ where $m=(y_2-y_1)/(x_2-x_1)$

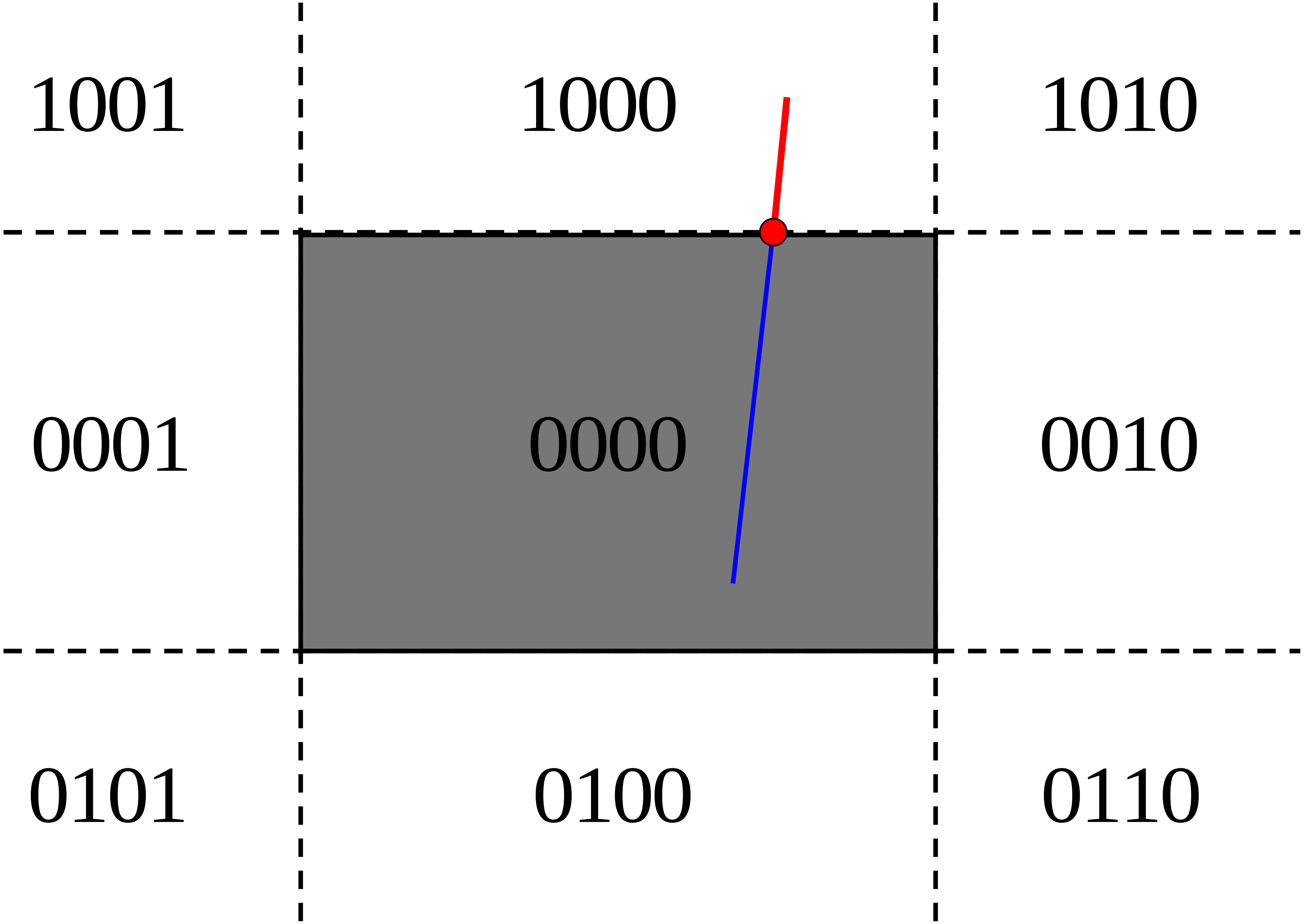
Example 1



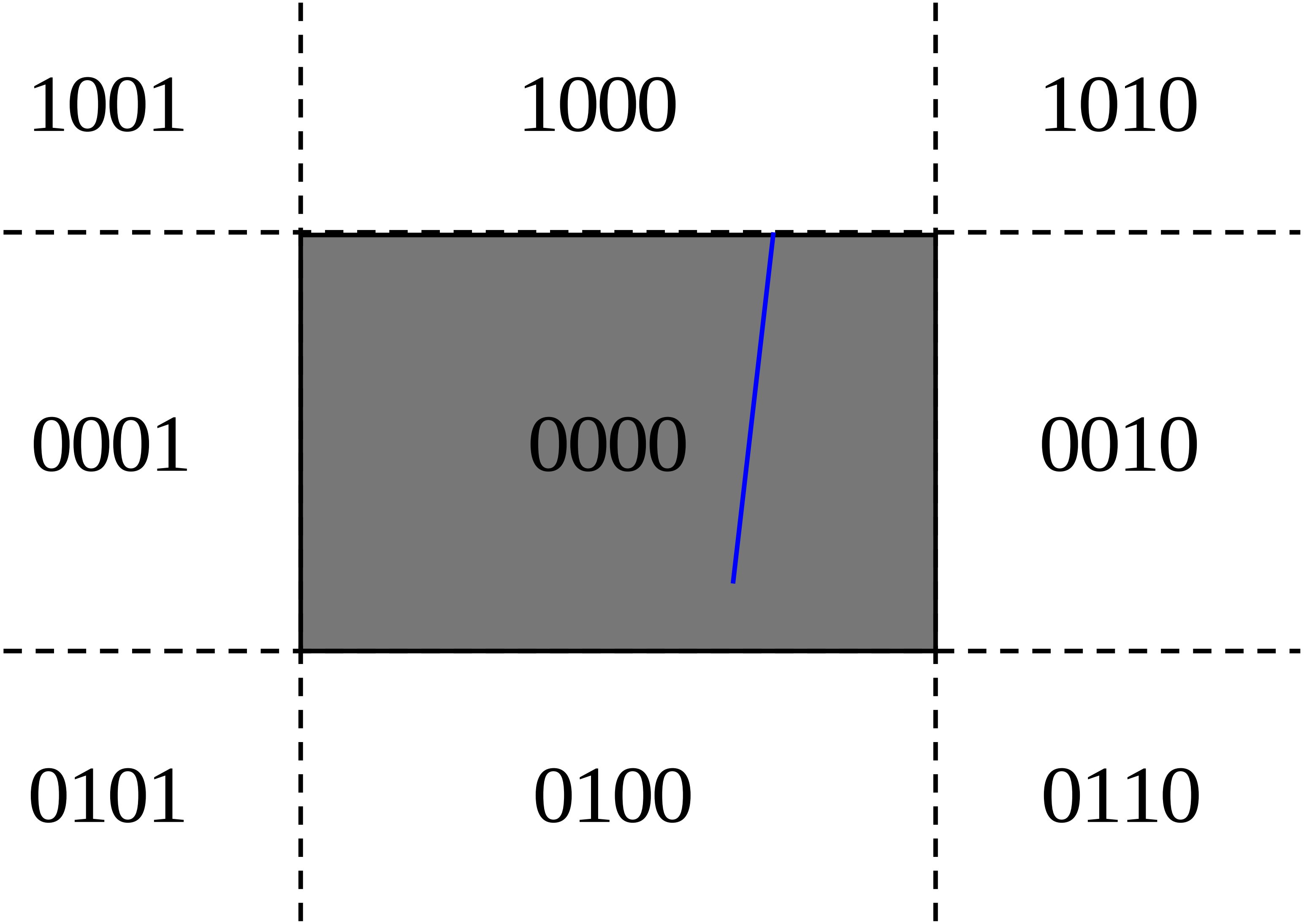
Example 1



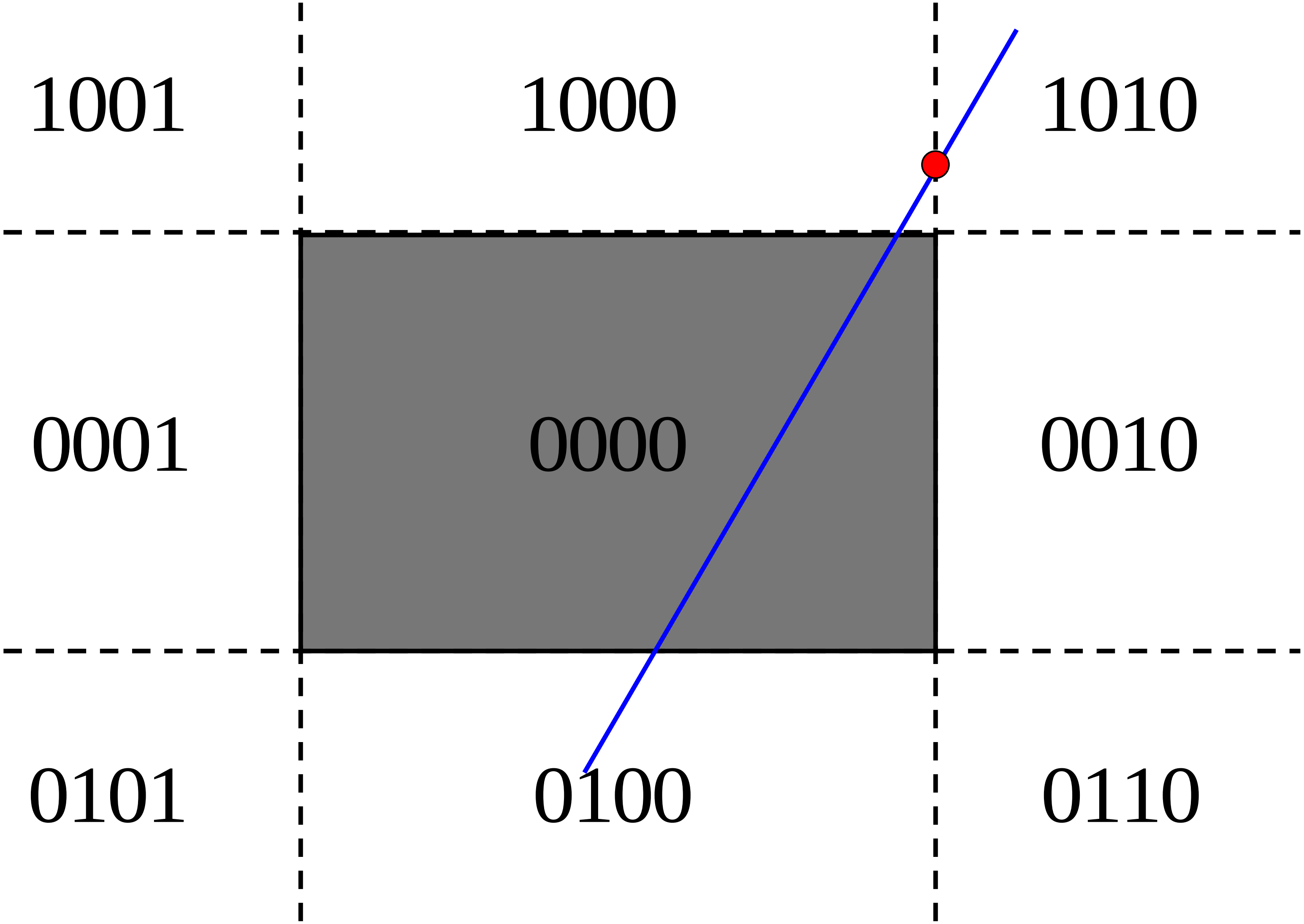
Example 1



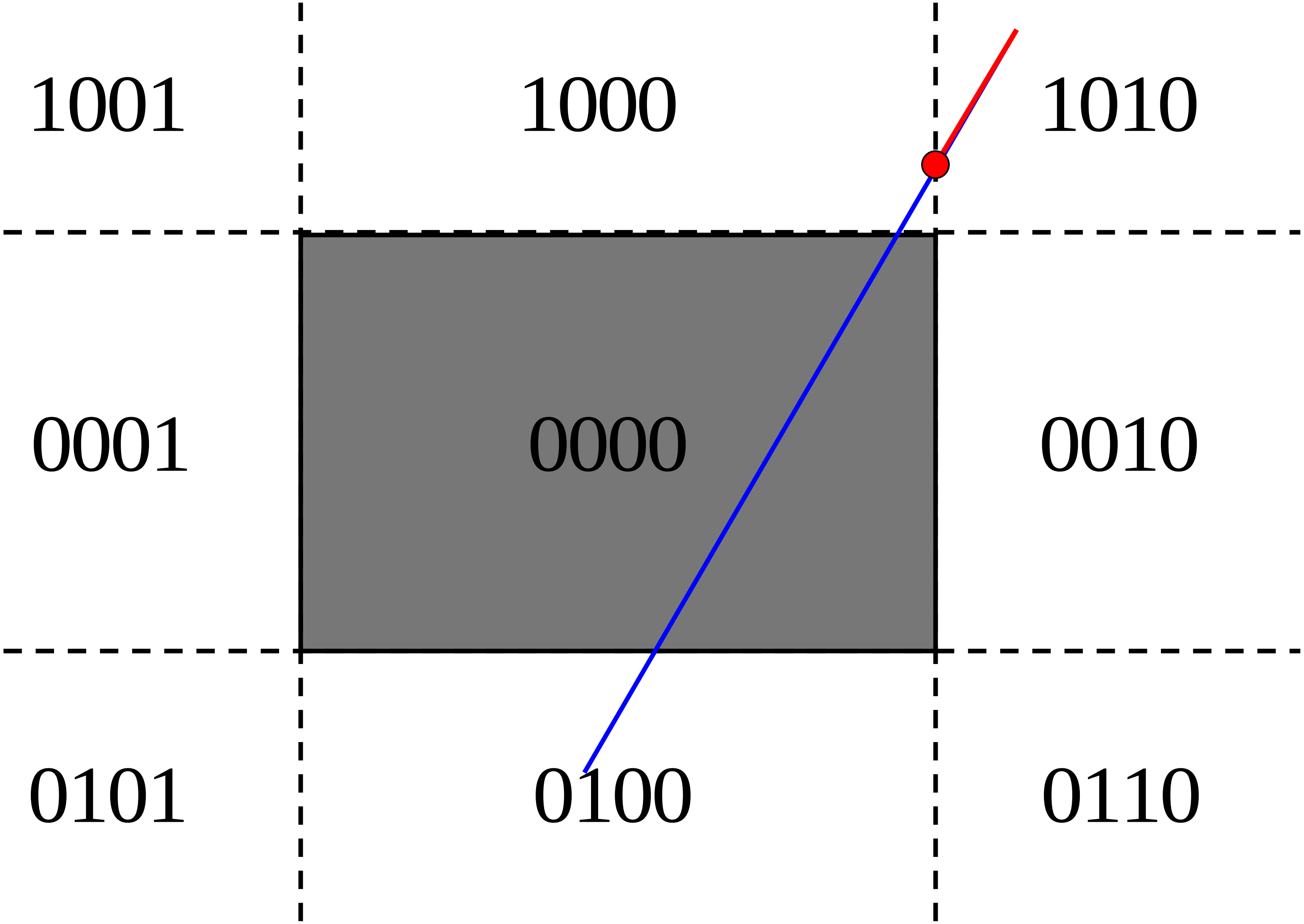
Example 1



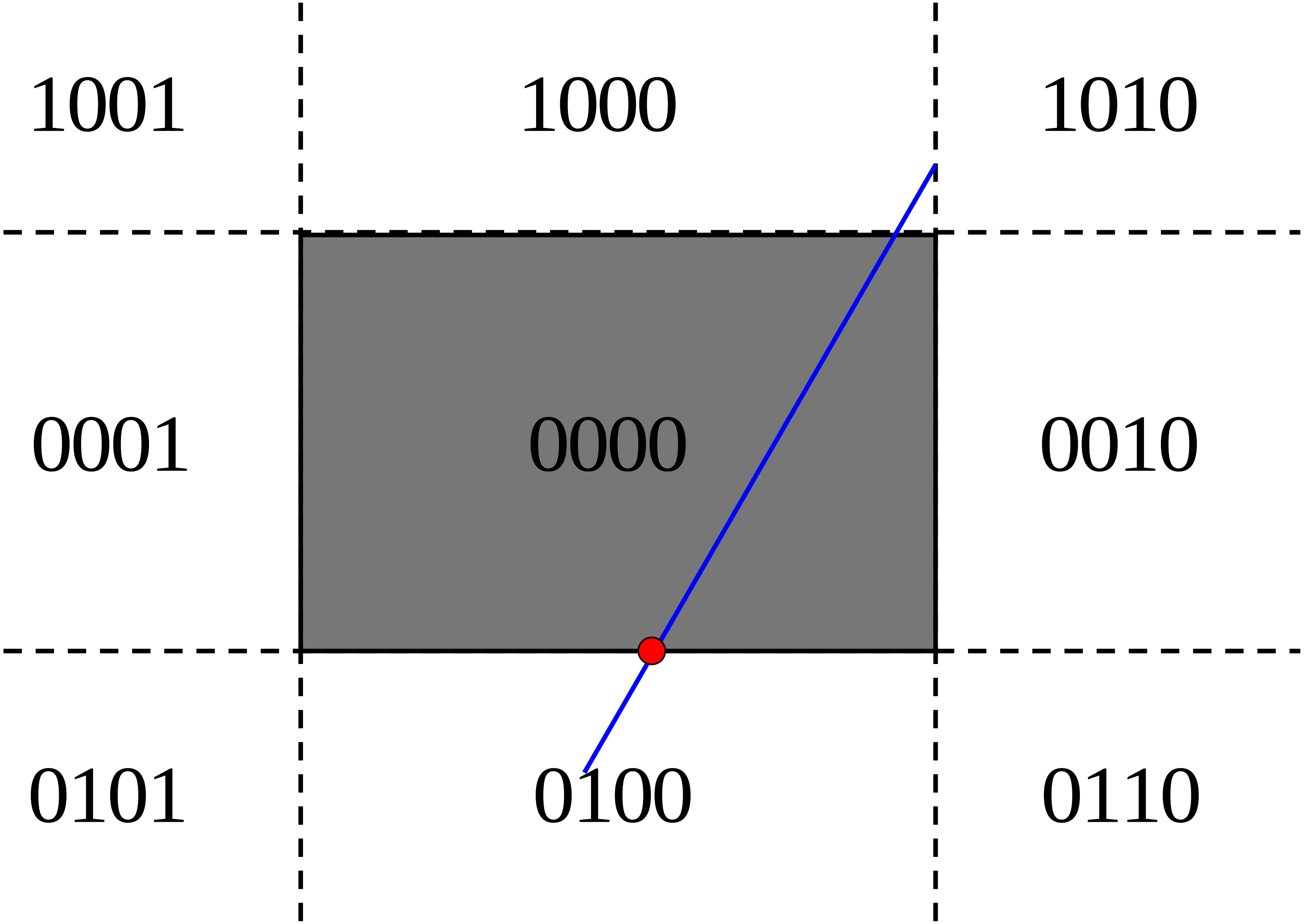
Example 2



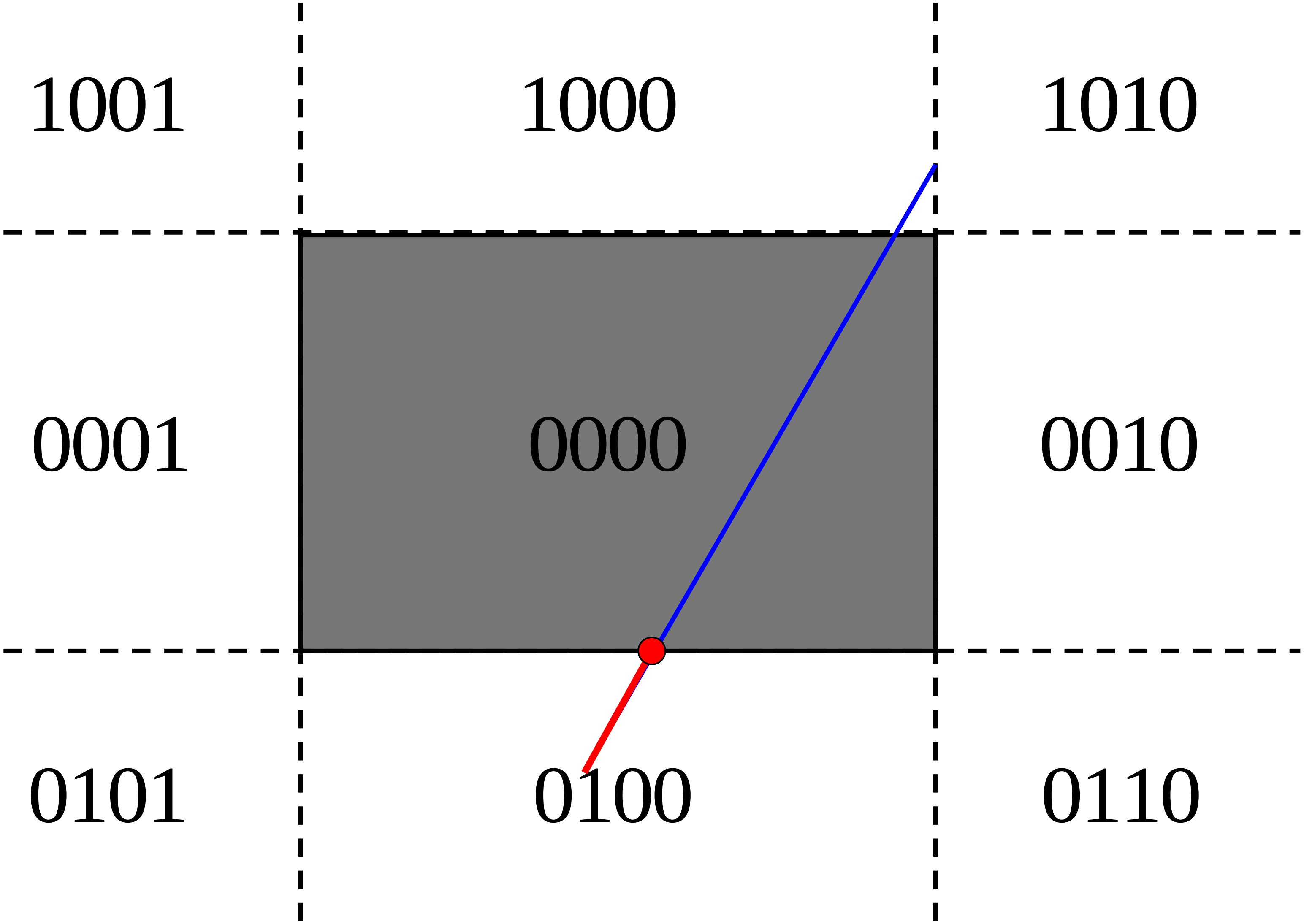
Example 2



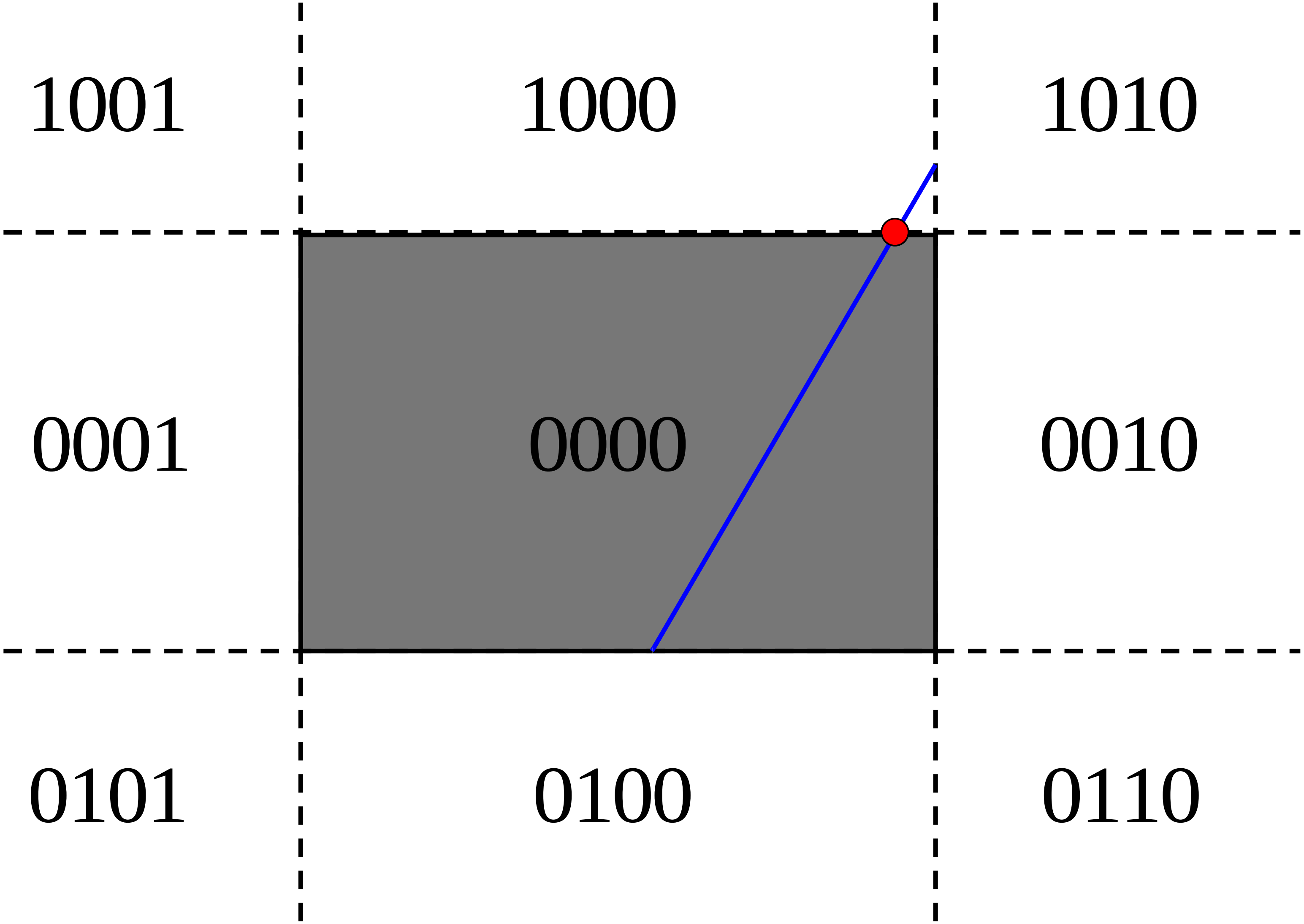
Example 2



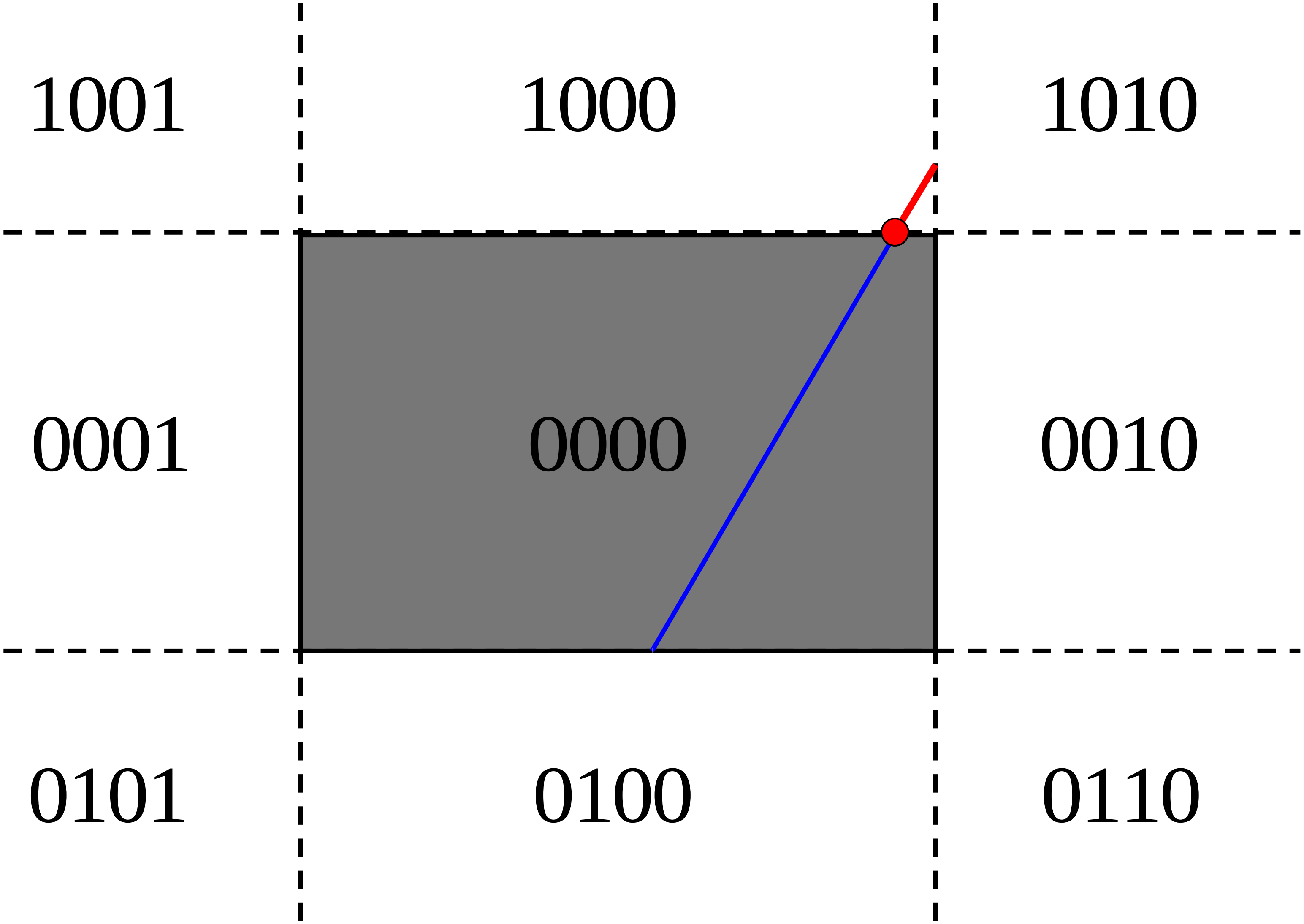
Example 2



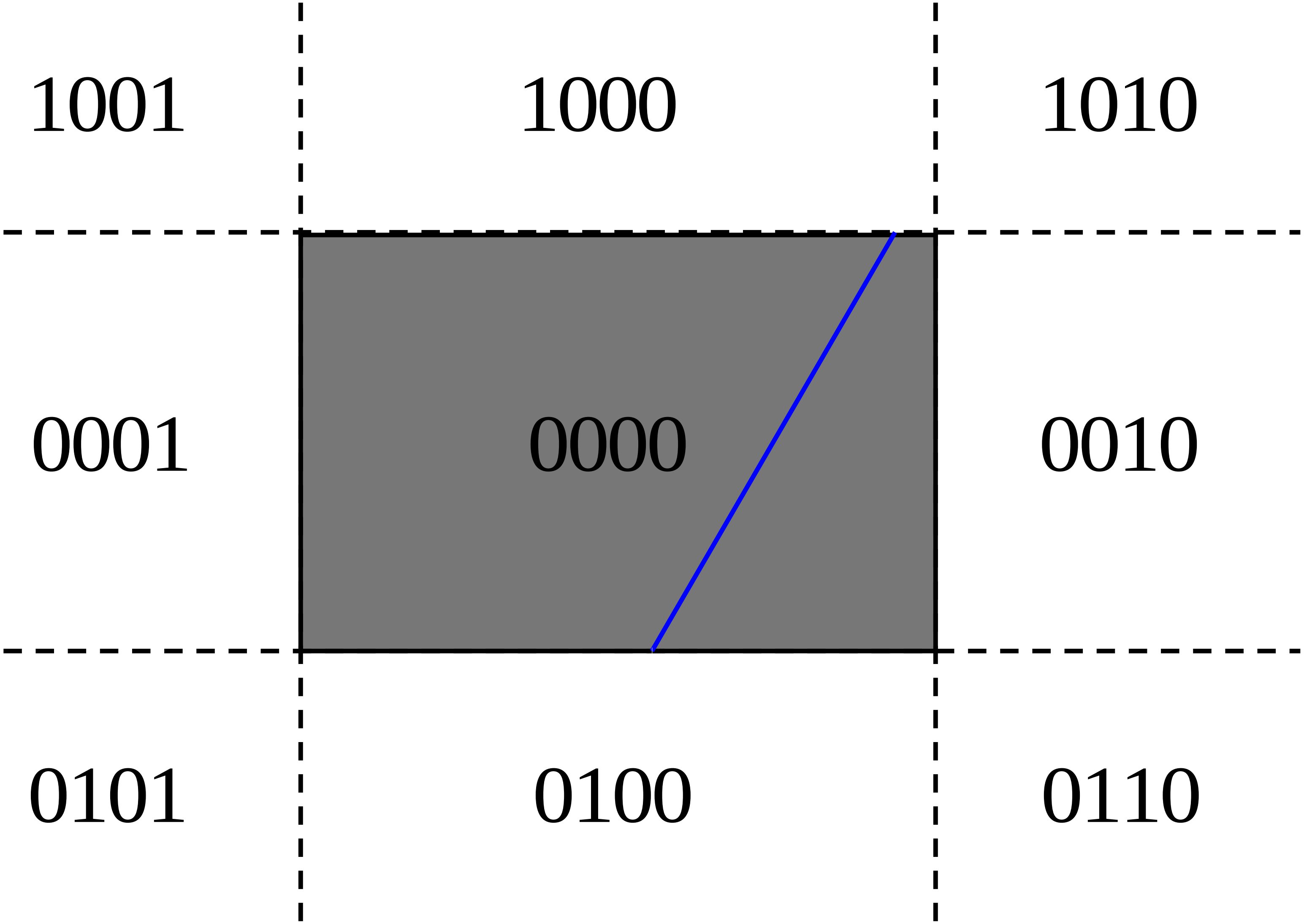
Example 2



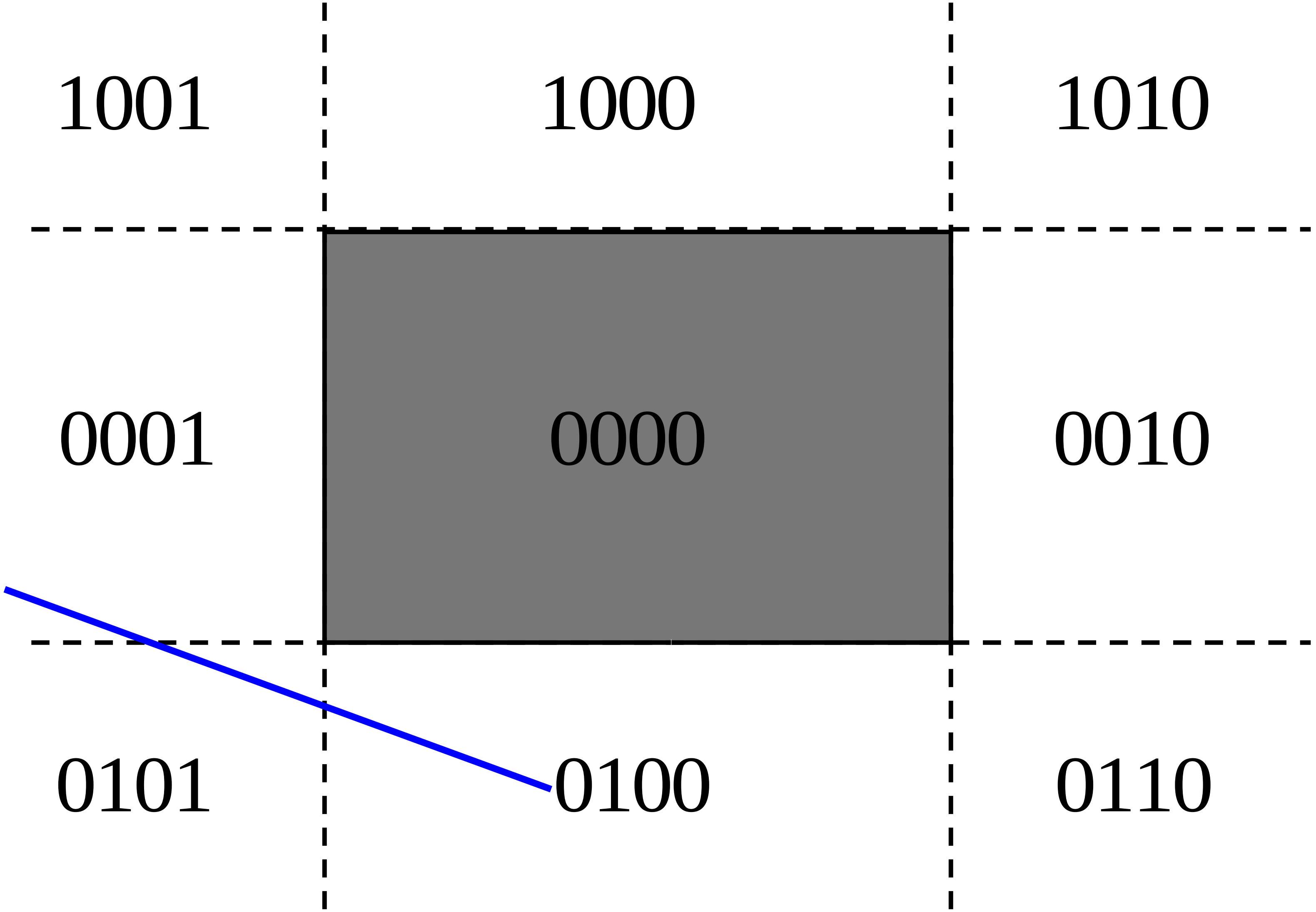
Example 2



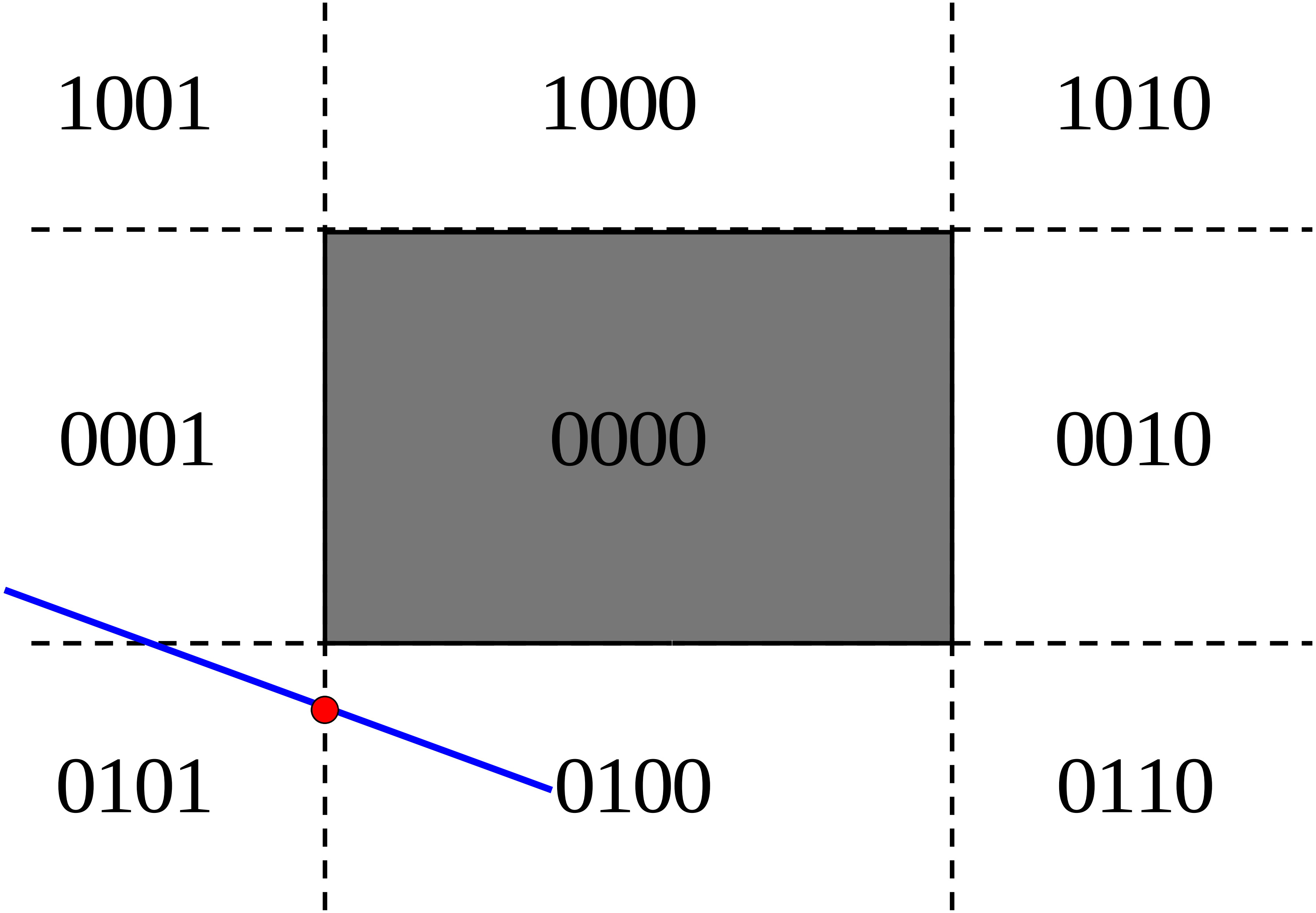
Example 2



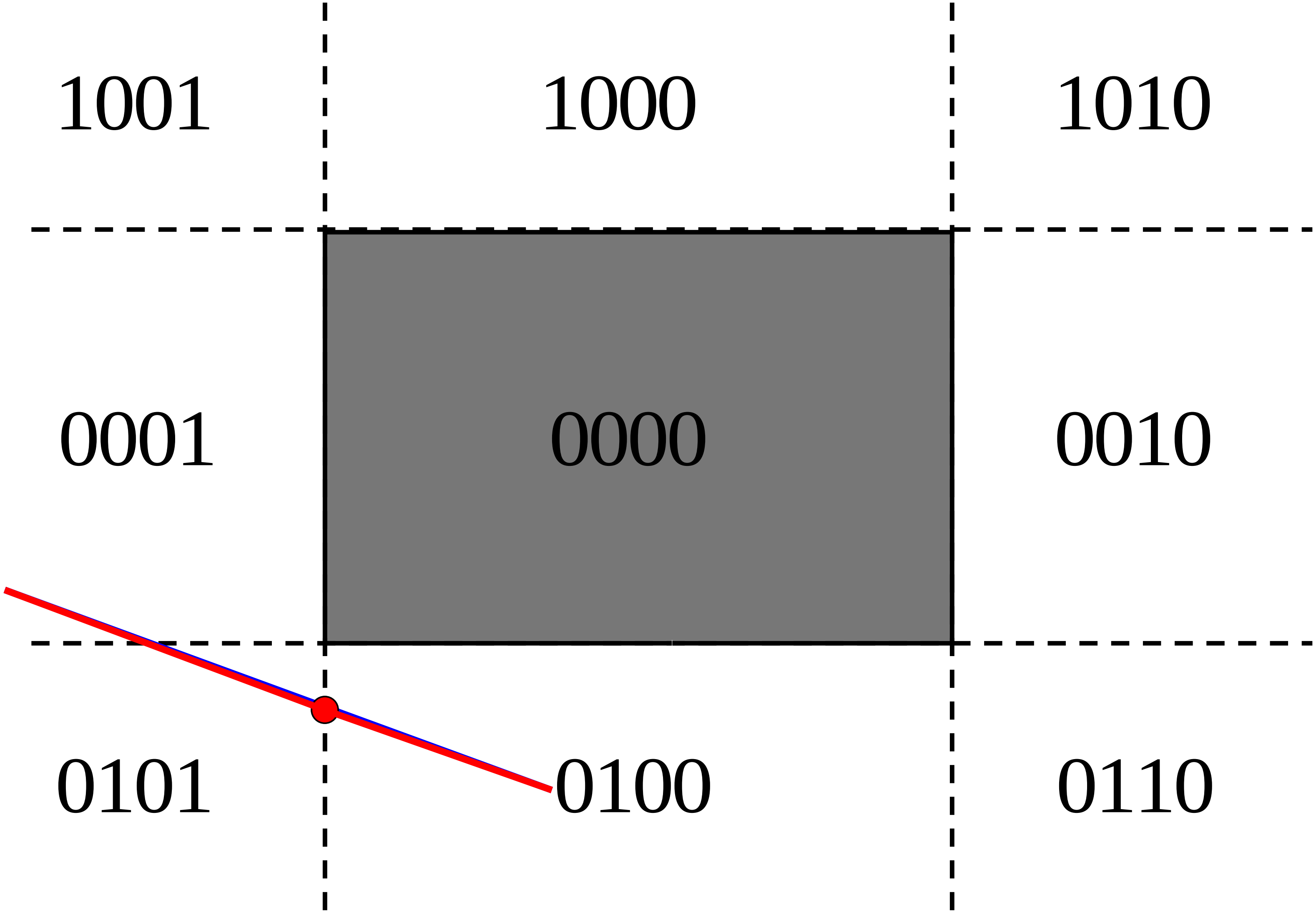
Example 3



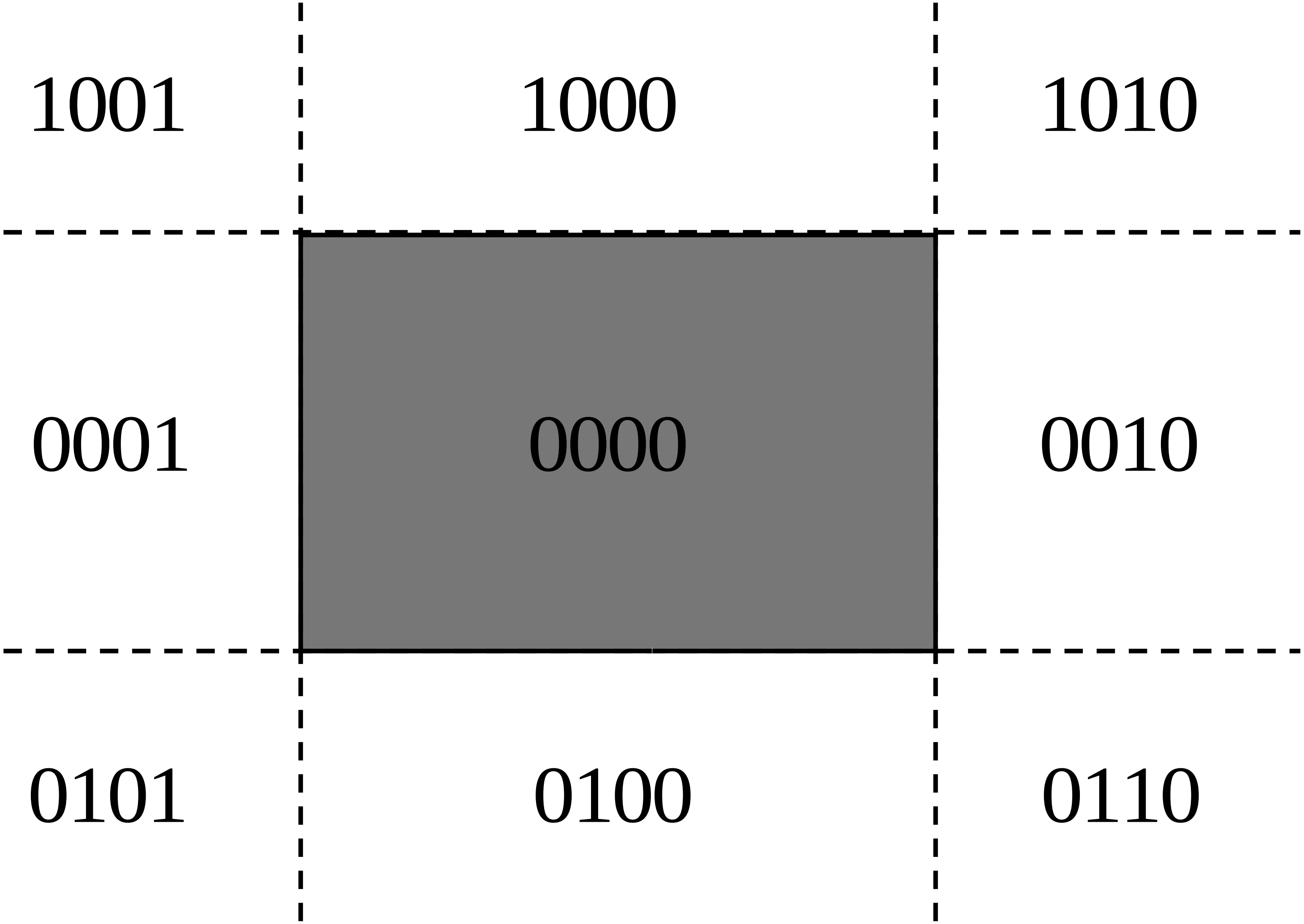
Example 3



Example 3



Example 3



1001	1000	1010
0001	0000	0010
0101	0100	0110

Cohen-Sutherland Algorithm

```
ComputeOutCode(x0, y0, outcode0)
ComputeOutCode(x1, y1, outcode1)
repeat
    check for trivial reject or trivial accept
    pick the point that is outside the clip rectangle
    if TOP then
        x = x0 + (x1 – x0) * (ymax – y0)/(y1 – y0); y = ymax;
    else if BOTTOM then
        x = x0 + (x1 – x0) * (ymin – y0)/(y1 – y0); y = ymin;
    else if RIGHT then
        y = y0 + (y1 – y0) * (xmax – x0)/(x1 – x0); x = xmax;
    else if LEFT then
        y = y0 + (y1 – y0) * (xmin – x0)/(x1 – x0); x = xmin;
    end {calculate the line segment}
    if (x0, y0 is the outer point) then
        x0 = x; y0 = y; ComputeOutCode(x0, y0, outcode0)
    else
        x1 = x; y1 = y; ComputeOutCode(x1, y1, outcode1)
    end {Subdivide}
until done
```

Cohen-Sutherland Algorithm

- Extends easily to 3D line clipping
  - 27 regions
  - 6 bits

Cohen-Sutherland Algorithm

- Use region codes to quickly eliminate/include lines
  - Best algorithm when trivial accepts/rejects are common
- Must compute viewing window clipping of remaining lines
  - Non-trivial clipping cost
  - Redundant clipping of some lines
- More efficient algorithms exist