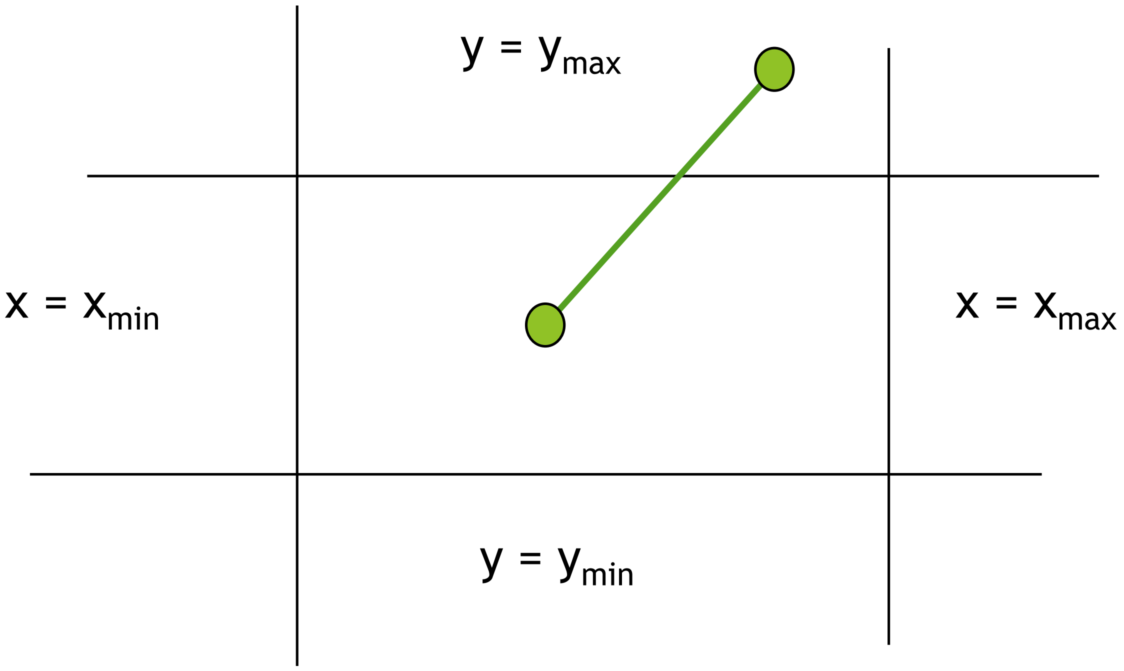
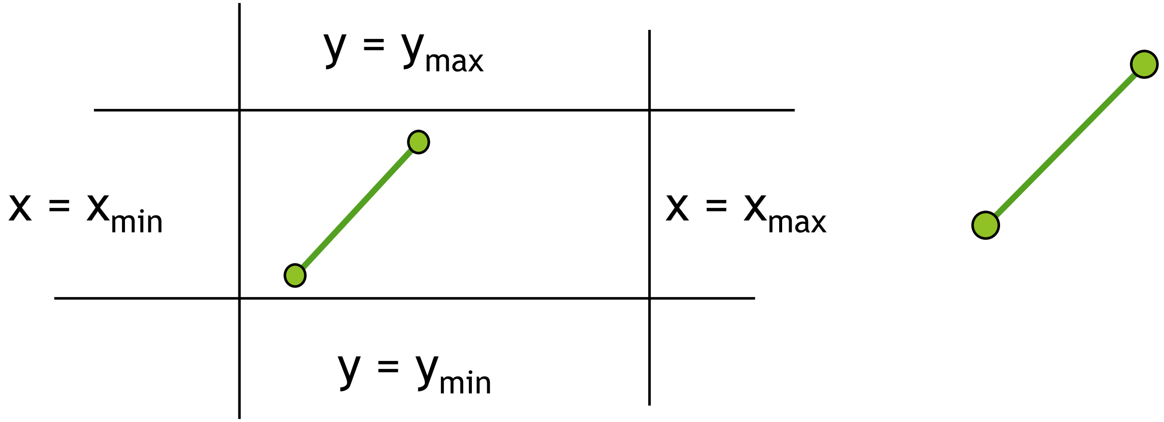
Cohen-Sutherland Algorithm

- Idea: eliminate as many cases as possible without computing intersections
- Start with four lines that determine the sides of the clipping window



The Cases

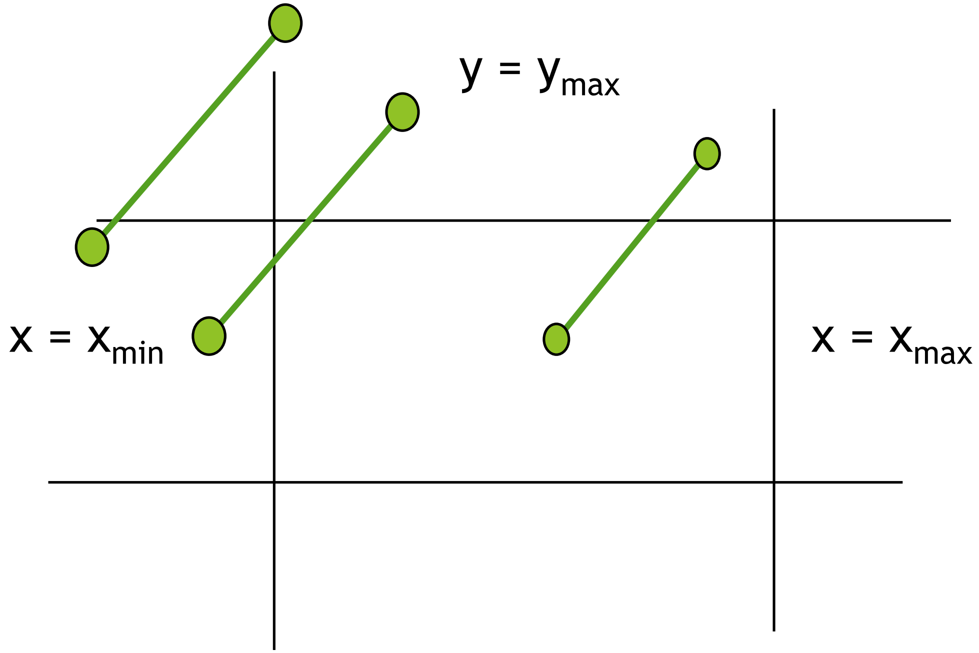
- Case 1: both endpoints of line segment inside all four lines
  - Draw (accept) line segment as is



- Case 2: both endpoints outside all lines and on same side of a line
  - Discard (reject) the line segment

The Cases

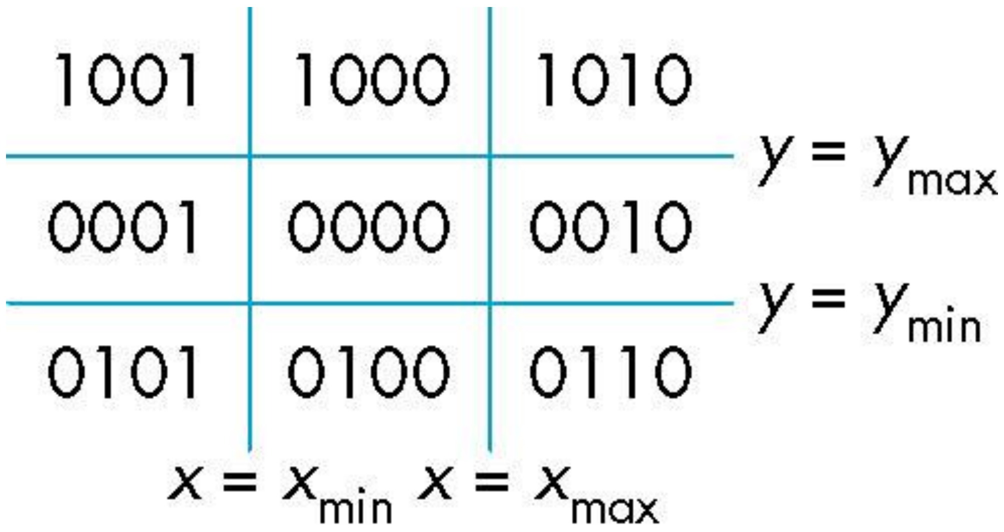
- Case 3: One endpoint inside, one outside
  - Must do at least one intersection
- Case 4: Both outside
  - May have part inside
  - Must do at least one intersection



Defining Outcodes

- For each endpoint, define an outcode $b_0b_1b_2b_3$

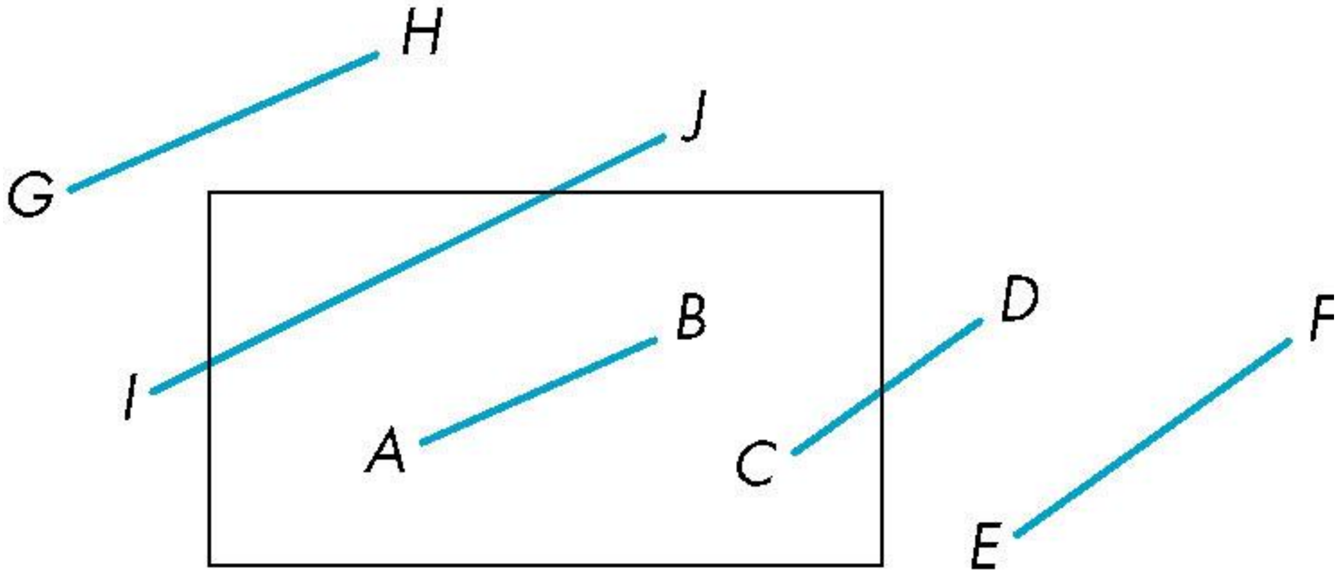
$b_0 = 1$ if $y > y_{max}$, 0 otherwise
$b_1 = 1$ if $y < y_{min}$, 0 otherwise
$b_2 = 1$ if $x > x_{max}$, 0 otherwise
$b_3 = 1$ if $x < x_{min}$, 0 otherwise



- Outcodes divide space into 9 regions
- Computation of outcode requires at most 4 subtractions

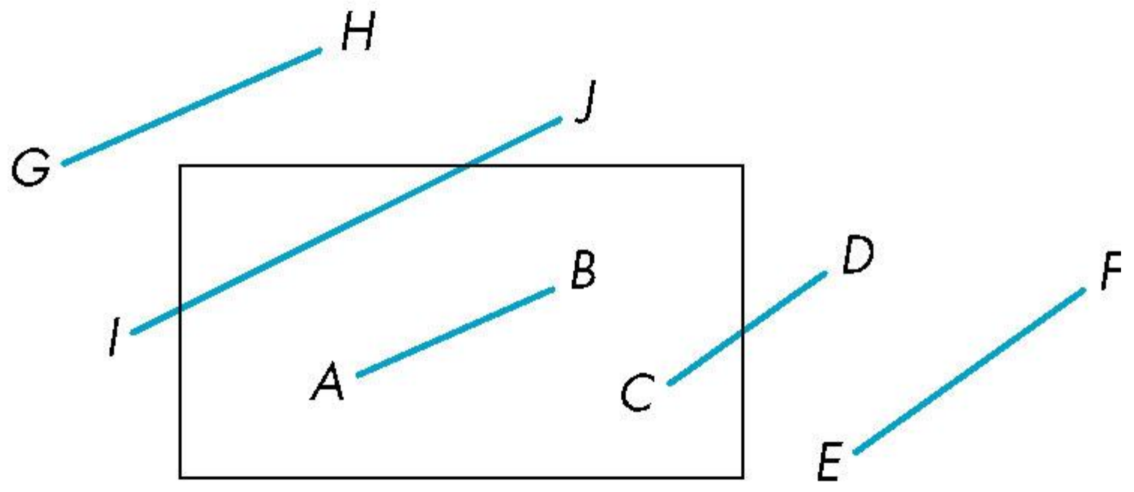
Using Outcodes

- Consider the 5 cases below
- AB: outcode(A) = outcode(B) = 0
  - Accept line segment



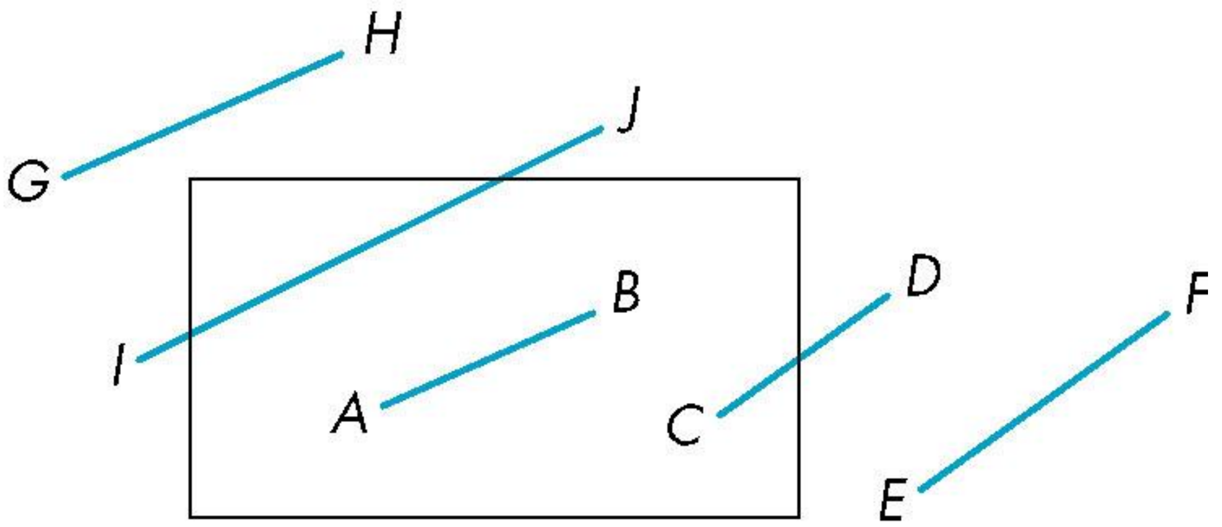
Using Outcodes

- CD: outcode (C) = 0, outcode(D) $\neq$ 0
  - Compute intersection
  - Location of 1 in outcode(D) determines which edge to intersect with
  - Note if there were a segment from A to a point in a region with 2 ones in outcode, we might have to do two intersections



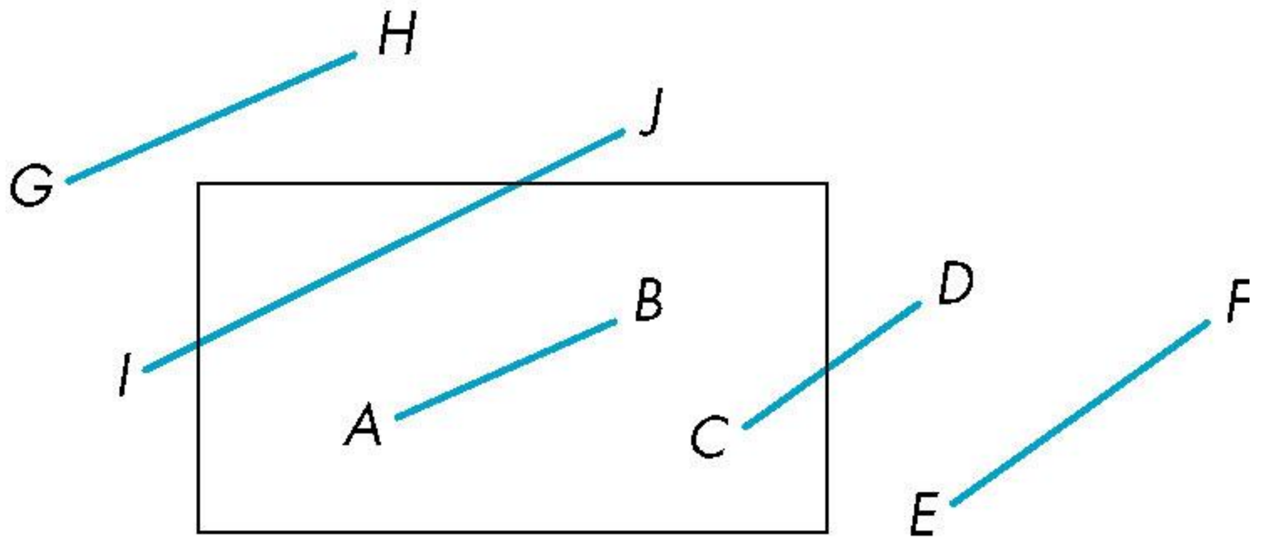
Using Outcodes

- EF: outcode(E) logically ANDed with outcode(F) (bitwise) $\neq 0$
  - Both outcodes have a 1 bit in the same place
  - Line segment is outside of corresponding side of clipping window
  - reject



Using Outcodes

- GH and IJ: same outcodes, neither zero but logical AND yields zero
- Shorten line segment by intersecting with one of sides of window
- Compute outcode of intersection (new endpoint of shortened line segment)
- Reexecute algorithm

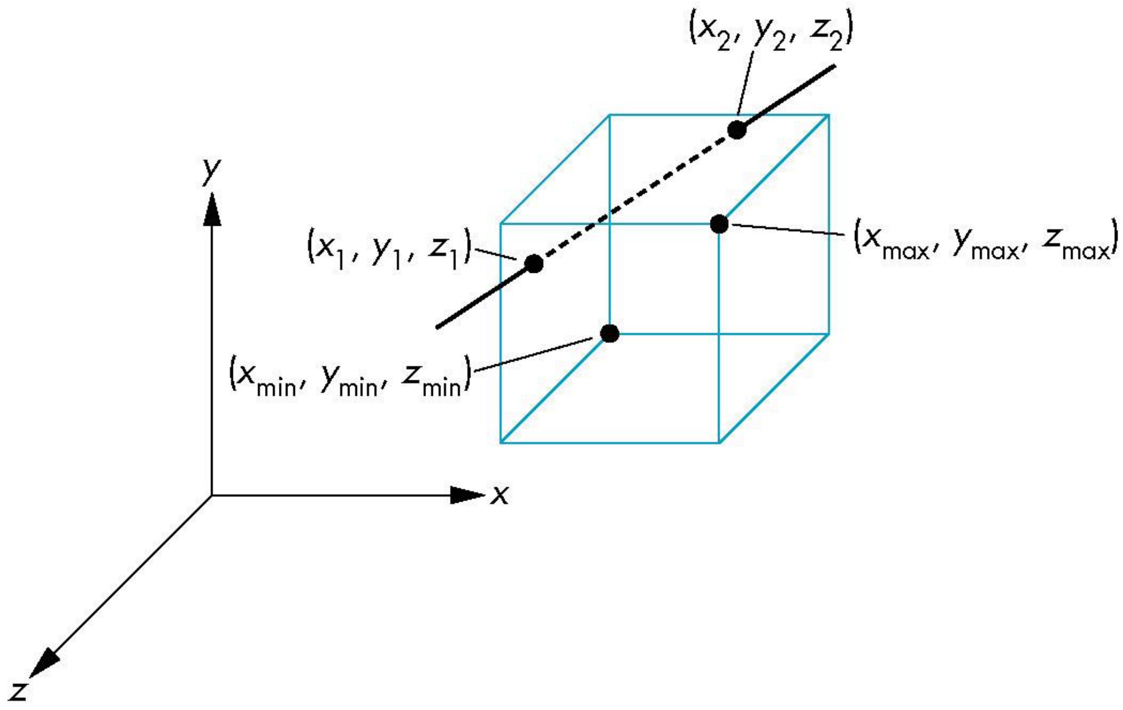


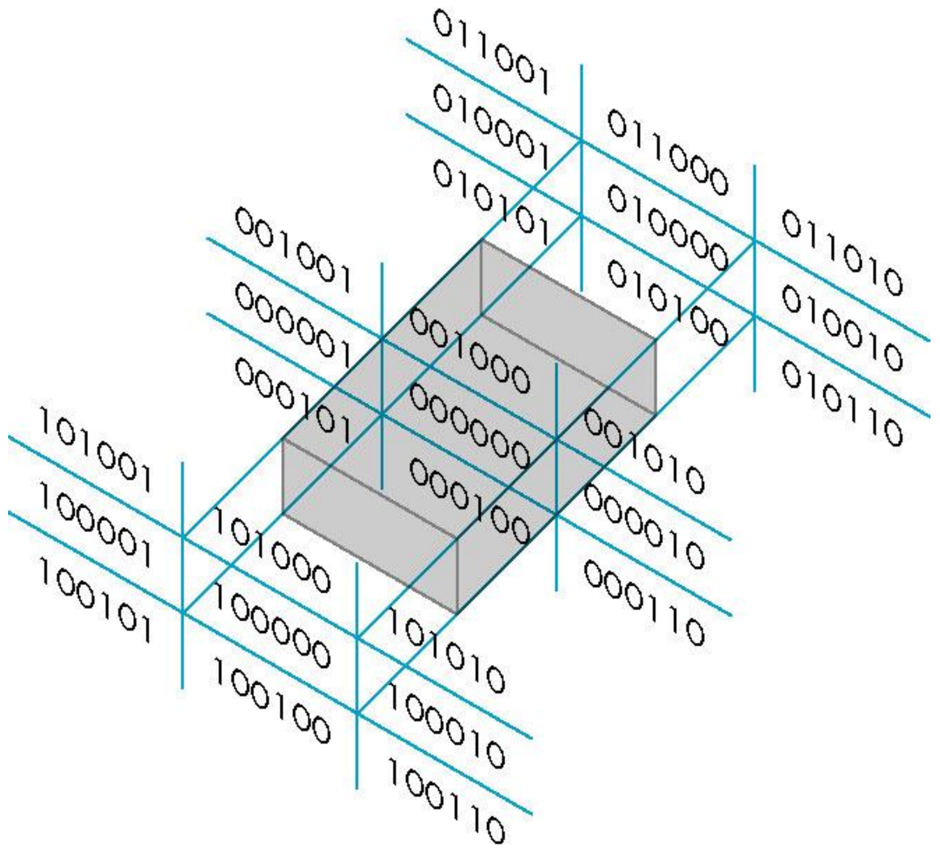
Efficiency

- In many applications, the clipping window is small relative to the size of the whole data base
  - Most line segments are outside one or more side of the window and can be eliminated based on their outcodes
- Inefficiency when code has to be reexecuted for line segments that must be shortened in more than one step

Cohen Sutherland in 3D

- Use 6-bit outcodes
- When needed, clip line segment against planes

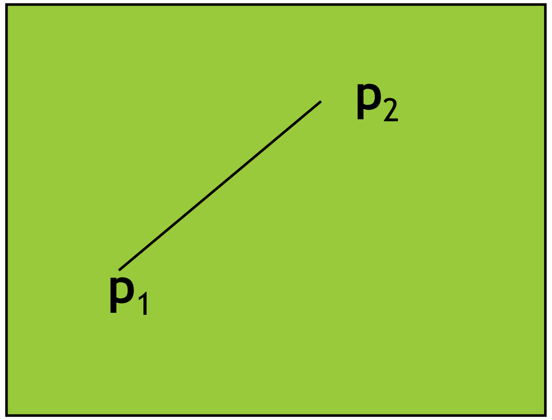




Liang-Barsky Clipping

$$p(t) = (1-t)p_1 + tp_2 \quad 1 \geq t \geq 0$$

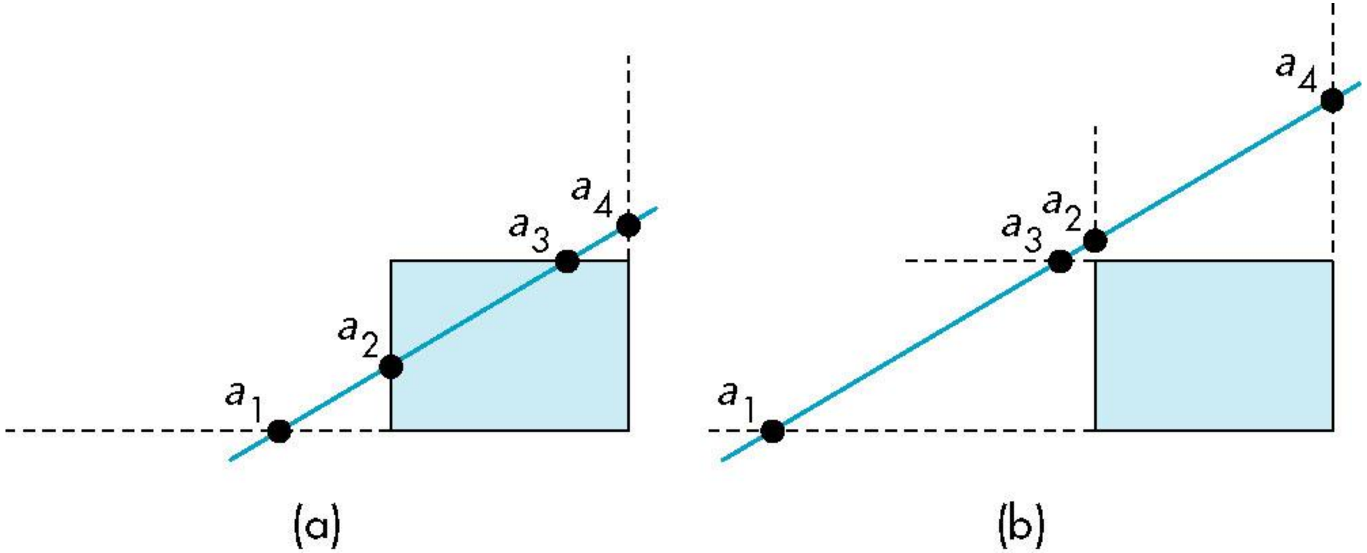
- Consider the parametric form of a line segment



- We can distinguish between the cases by looking at the ordering of the values of t where the line determined by the line segment crosses the lines that determine the window

Liang-Barsky Clipping

- In (a): $\alpha_4 > \alpha_3 > \alpha_2 > \alpha_1$
  - Intersect right, top, left, bottom: shorten
- In (b): $\alpha_4 > \alpha_2 > \alpha_3 > \alpha_1$
  - Intersect right, left, top, bottom: reject



Advantages

- Can accept/reject as easily as with Cohen-Sutherland
- Using values of α, we do not have to use algorithm recursively as with C-S
- Extends to 3D