Windows and clipping

Dr Nicolas Holzschuch
University of Cape Town
e-mail: holzschu@cs.uct.ac.za

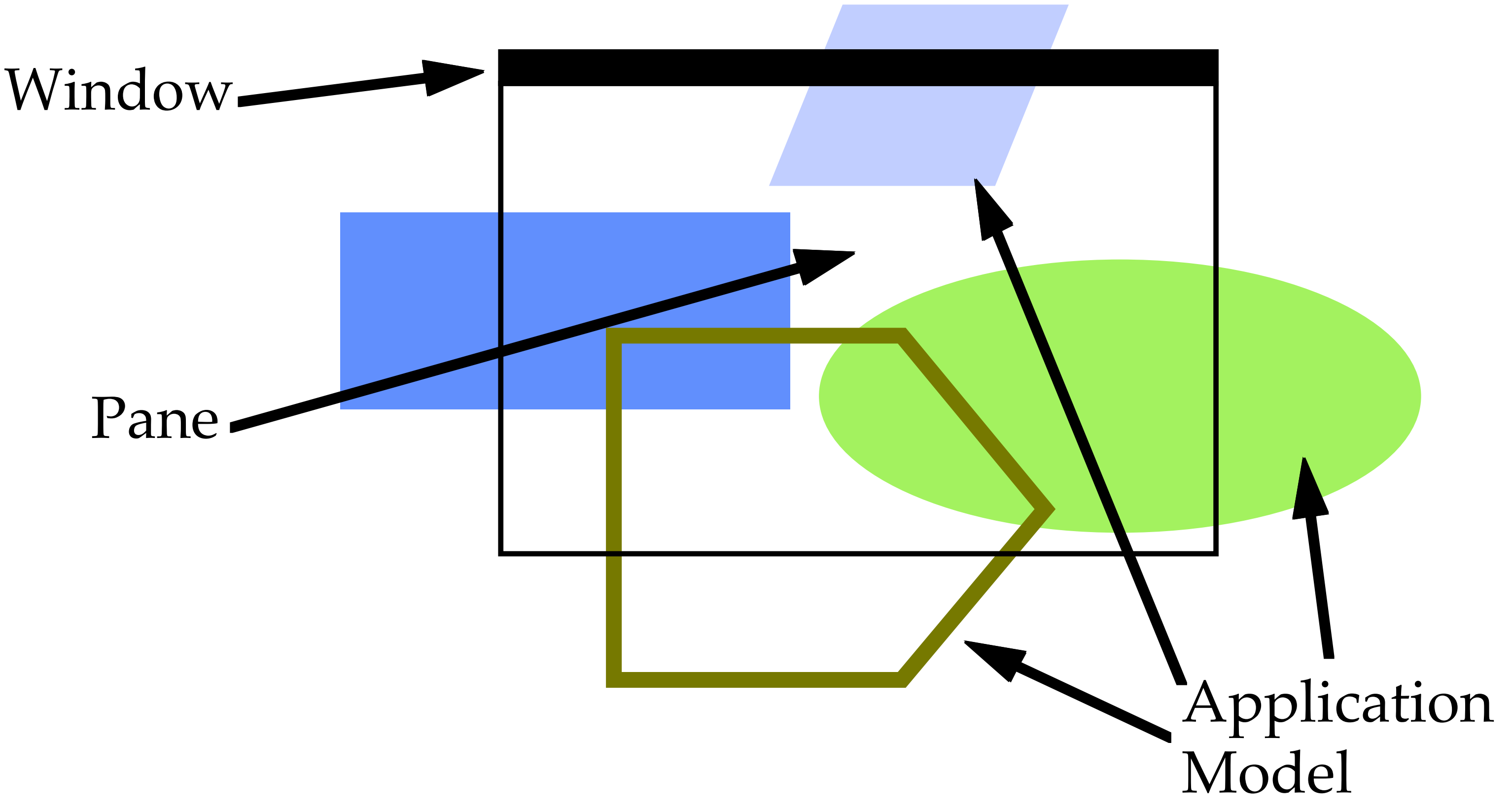
Map of the lecture

- Views, windows, buttons:
  - the need for clipping
- Clipping simple points
- Clipping line segments
  - against one edge of the window
  - against the whole window

Views and windows

- The application models contains the objects adressed by the application
- The *window* is the part of the screen reserved for the application
- Of that window, a part is reserved for drawing (the *pane*)
- A part of the application model is mapped onto the pane: it’s a *view*

Views and windows



Menus and buttons

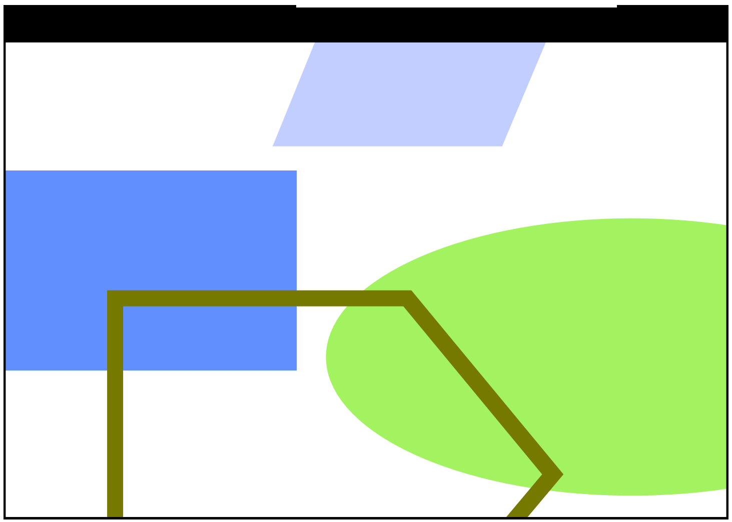
- Apart from the pane, other parts of the window are used by the application:
  - title bar
  - menu bar
  - buttons
  - text areas

Clipping: the basic problem

- The view is smaller than the application model
- Need to select the part of the application model to display
- Ensure that there is no overlap
- Graphics have to be *clipped*

Clipping for window systems

- No drawing outside the window:



- Ensures visual impression of “window”

Requirements

- For window systems: draw only:
  - the primitives that are inside the window
  - the parts of the primitive that are inside the window
- For menus, buttons, text areas:
  - parts of the application that must stay untouched
- An essential part of all GUI libraries

The difficulties of clipping

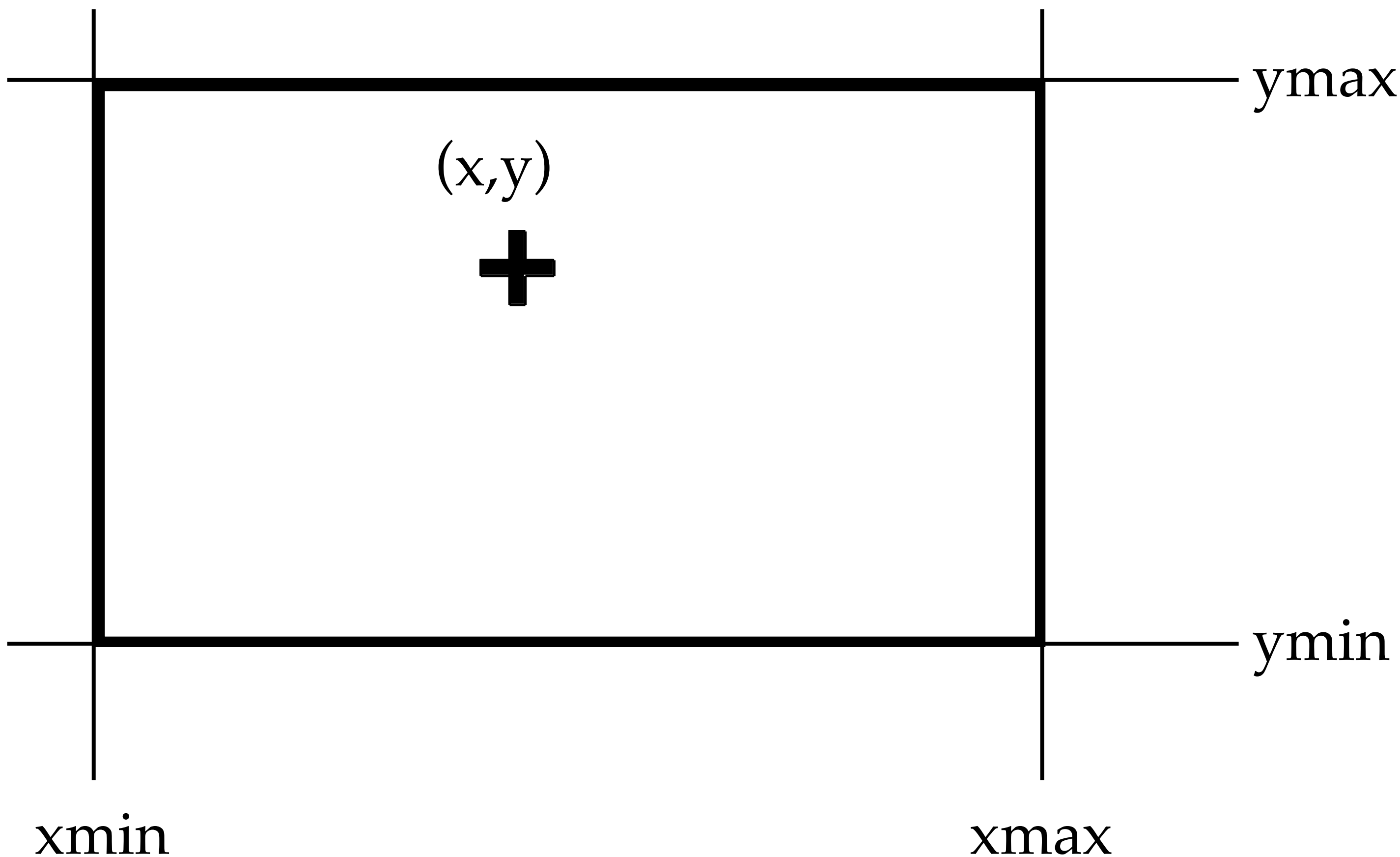
- Parts of the primitives are outside the window:
  - I must find the new shape
  - implies new vertices, new edges, etc.
- Will be done quite often:
  - must be a simple, non-costly, algorithm
  - preferably, clipping before rasterizing

Clipping in the application model

- Rasterizing:
  - low level algorithm
  - done by the graphics library
- Clipping:
  - higher level algorithm
  - sometimes done by the window system
  - sometimes you have to do it
  - do it in the application model

Clipping simple points

Draw the point iff: ((xmin < x < xmax) && (ymin < y < ymax))

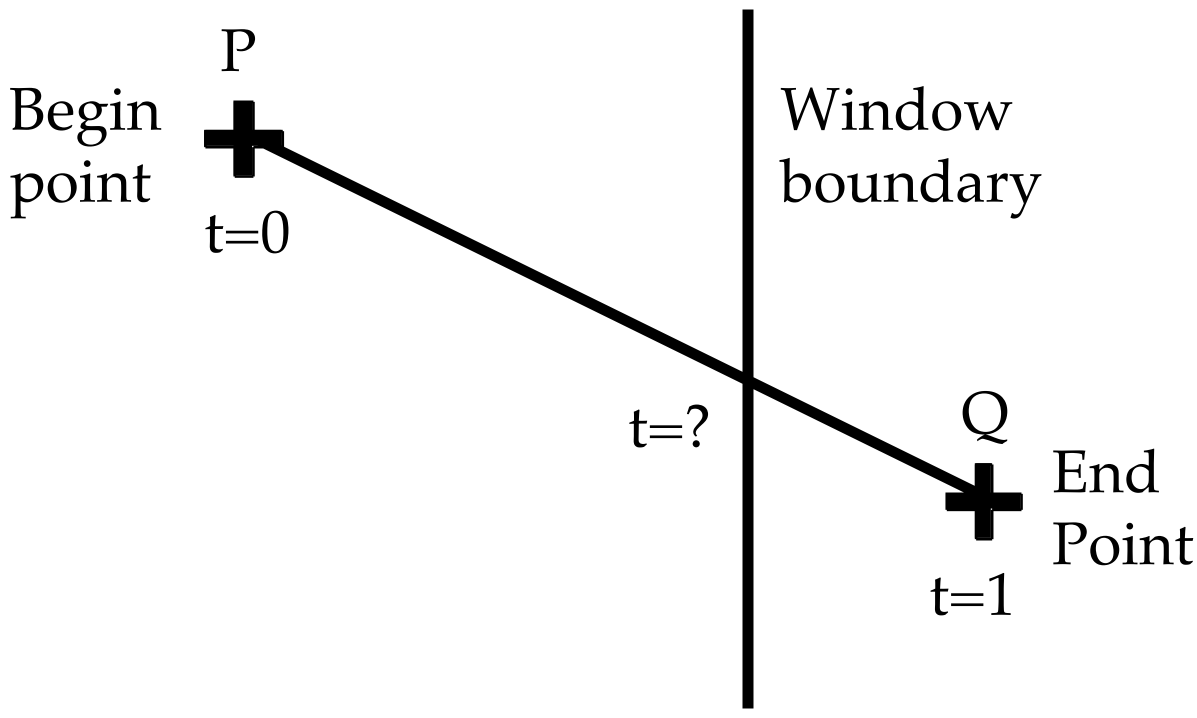


Clipping line segments

- Start by clipping against one edge of the window
- Several definition for line segments:
  - $ax+by+c = 0$
  - $y = mx+d$
  - $M(t) = P + t\,\mathbf{u}$…
  - same for the edge
- Which one is best for clipping?

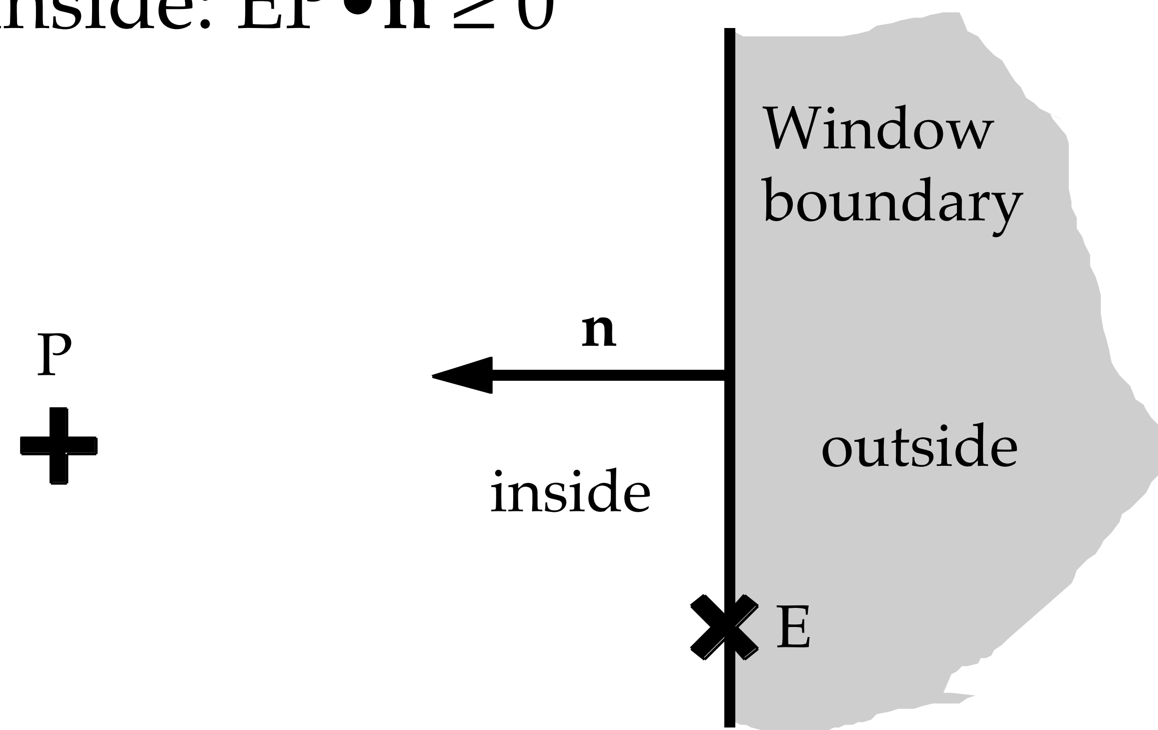
Use parametric representation

- Line defined as $M(t) = P + t\,\mathbf{u}$
- Find t

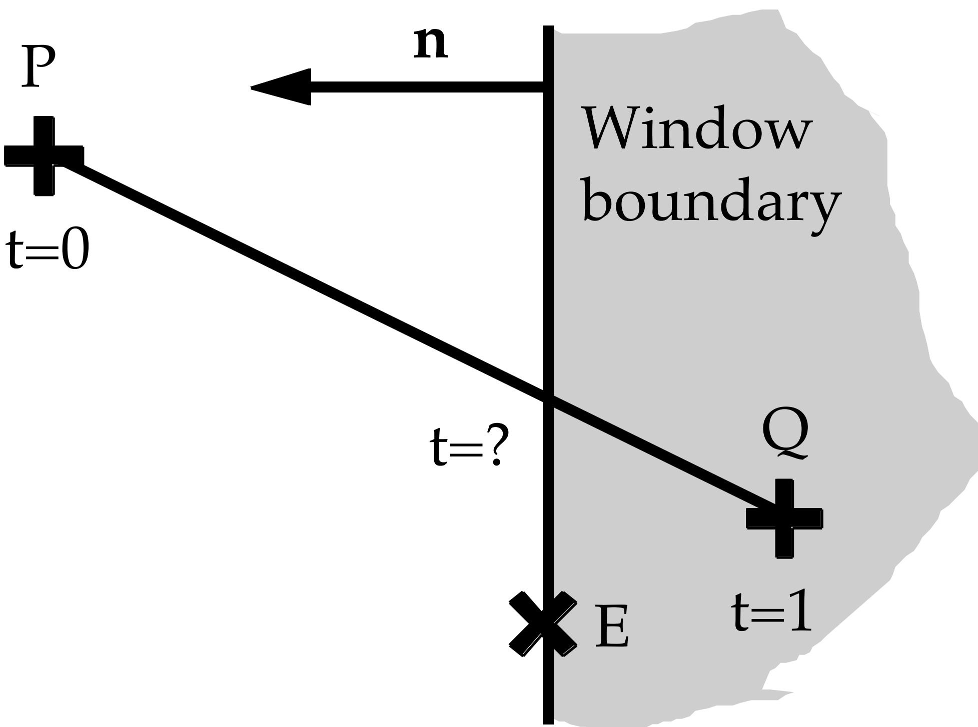


Define the boundary

- Boundary defined by point and normal
- A point is inside: EP•**n** ≥ 0



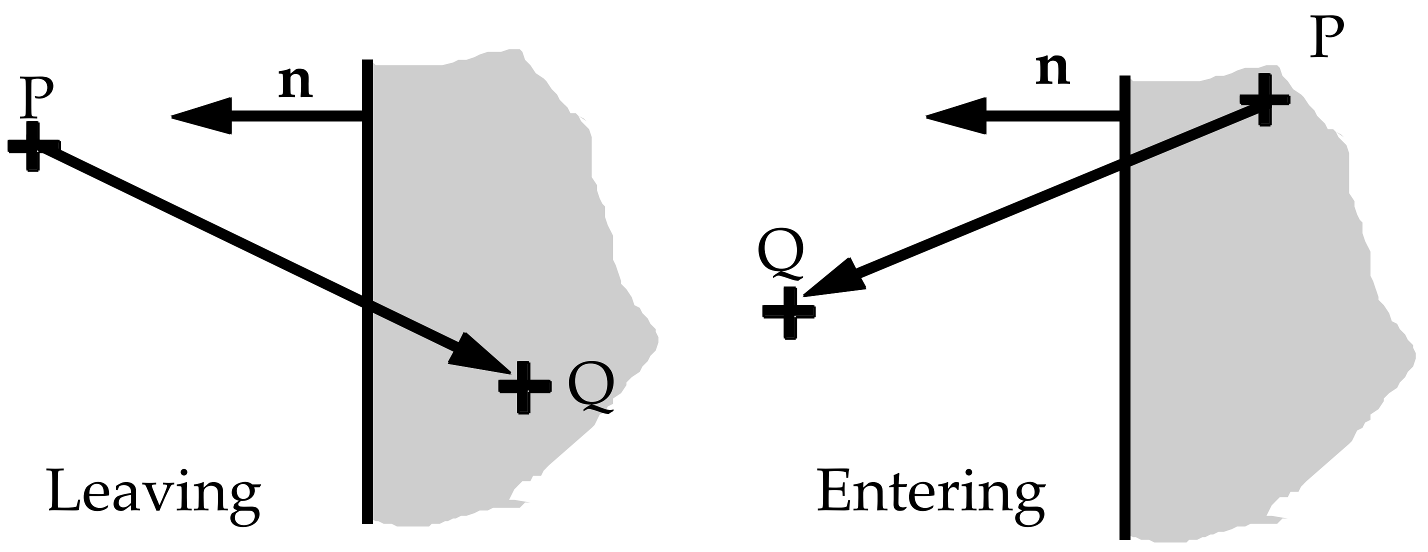
Finding the new vertex



$$t = -\frac{EP \bullet \mathbf{n}}{PQ \bullet \mathbf{n}}$$

Where is the new line segment?

- Should I draw from 0 to *t*, or from *t* to 1?
- Depends if the line was entering, or leaving:

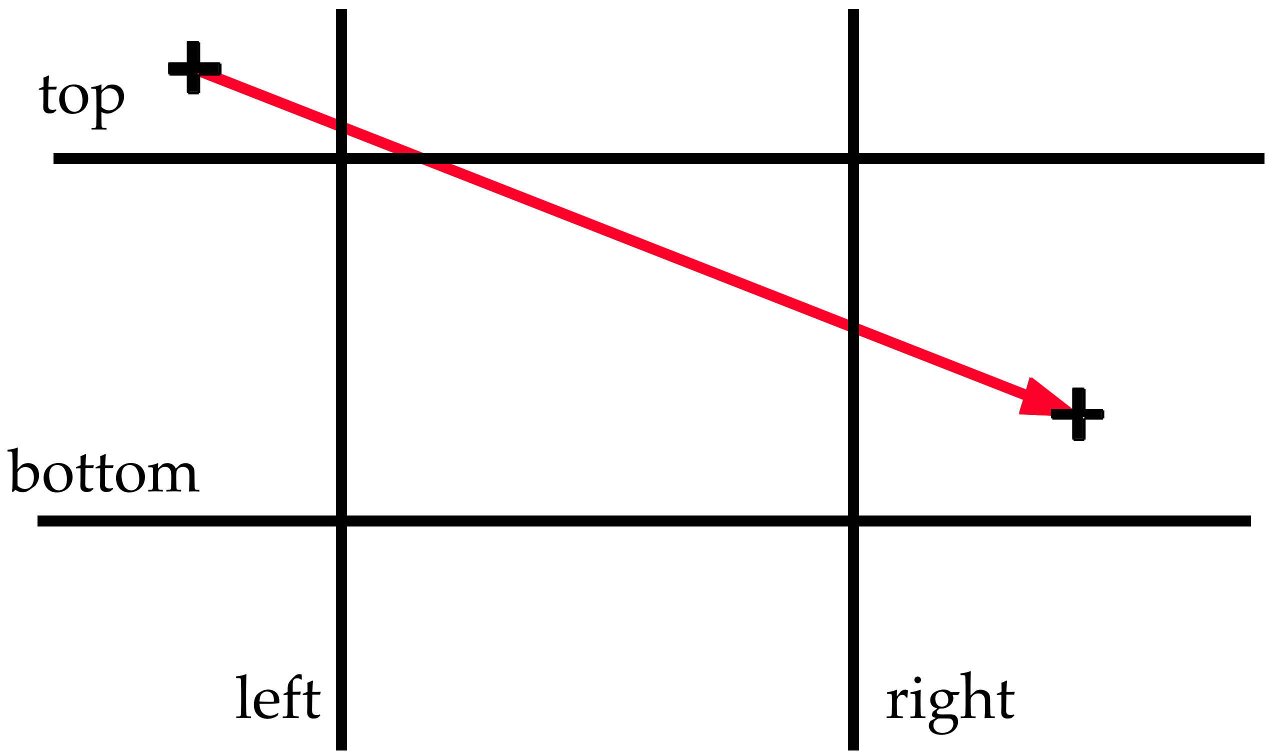


Entering or leaving?

- From the sign of $PQ \bullet \mathbf{n}$:
  - positive:
    - the line is entering,
    - draw from t to 1
  - negative:
    - the line is leaving,
    - draw from 0 to t

Clipping a line against a window

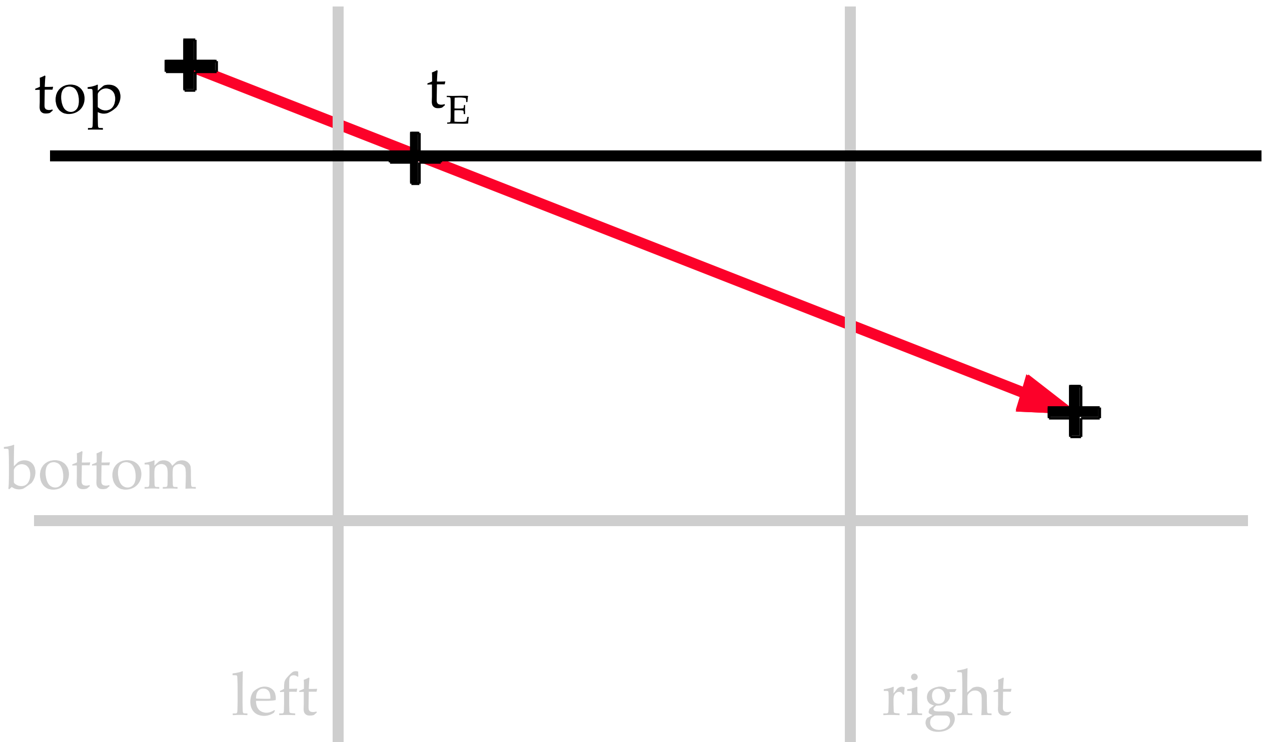
- Clip the line against each boundary:



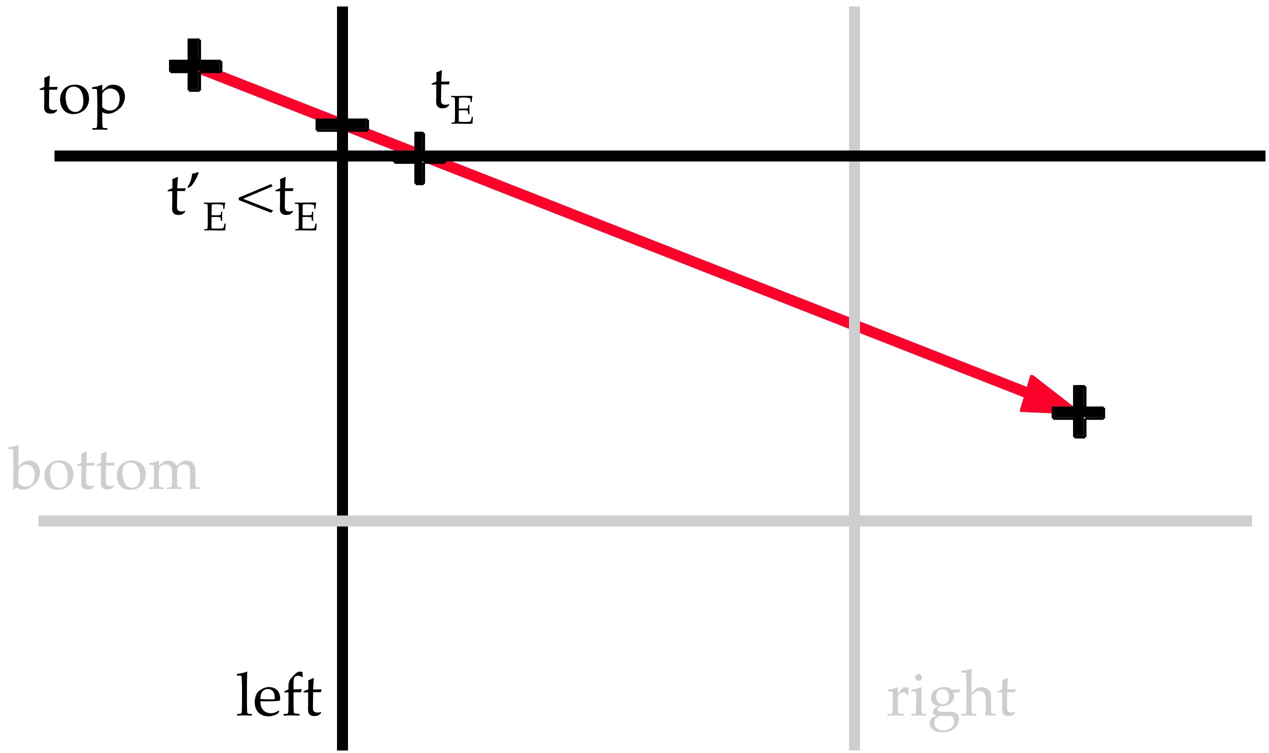
Clipping against a window

- Clip against each boundary in turn
- For each boundary:
  - compute t
  - status: entering/leaving
- Keep greater t_entering and smaller t_leaving
- If t_entering ≥ t_leaving, nothing to draw
- Else, draw from t_entering to t_leaving

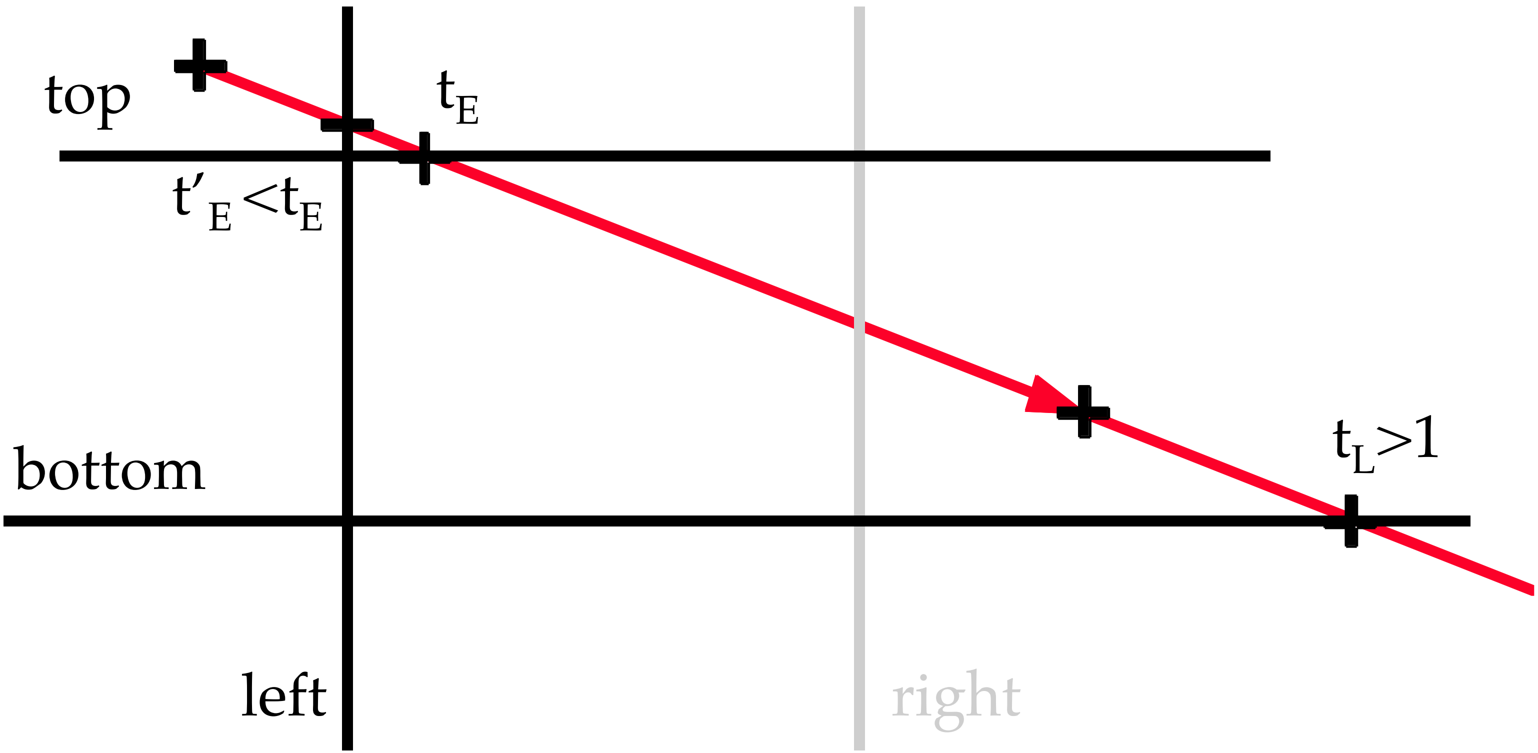
Clipping a line against a window



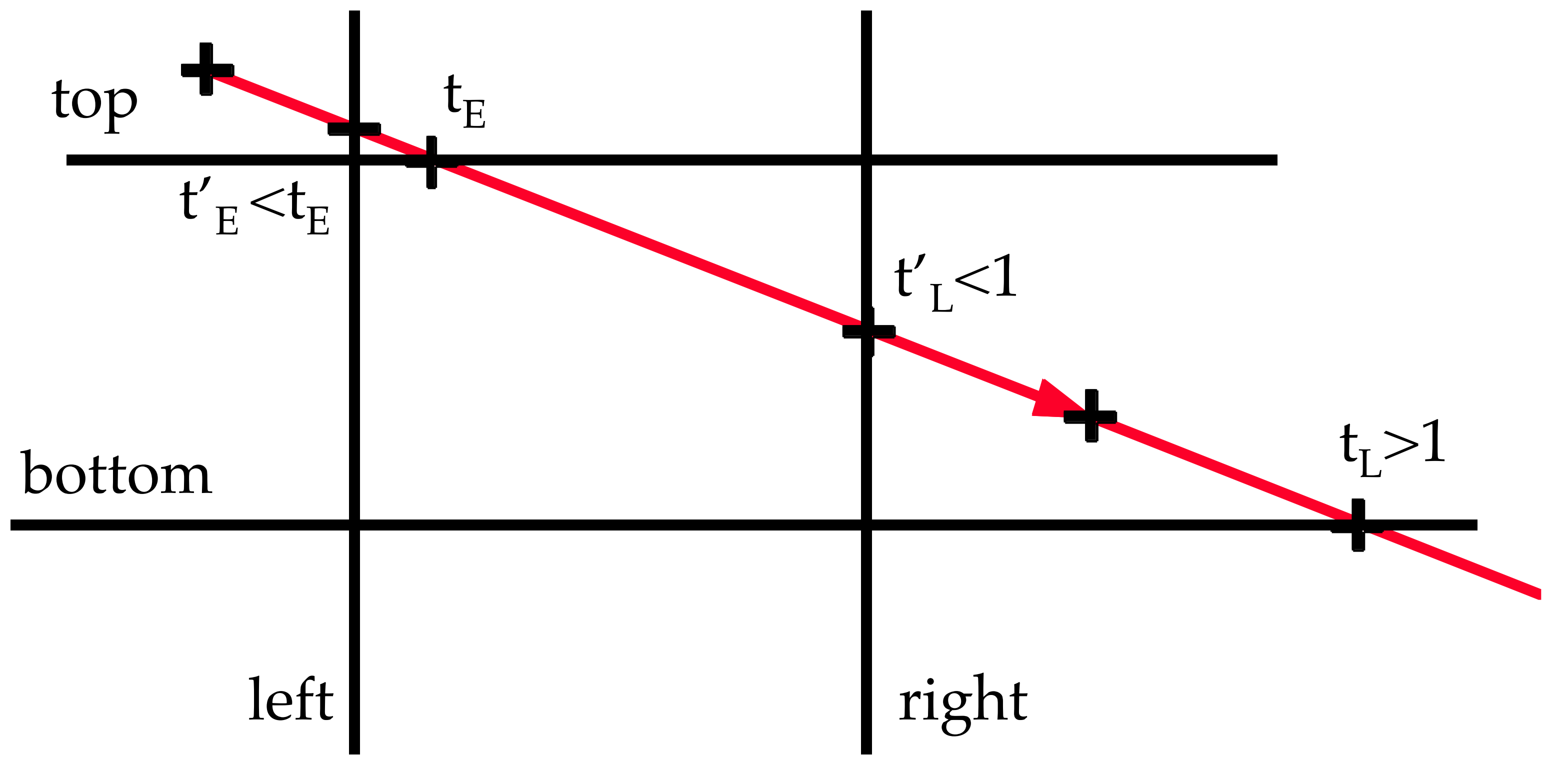
Clipping a line against a window



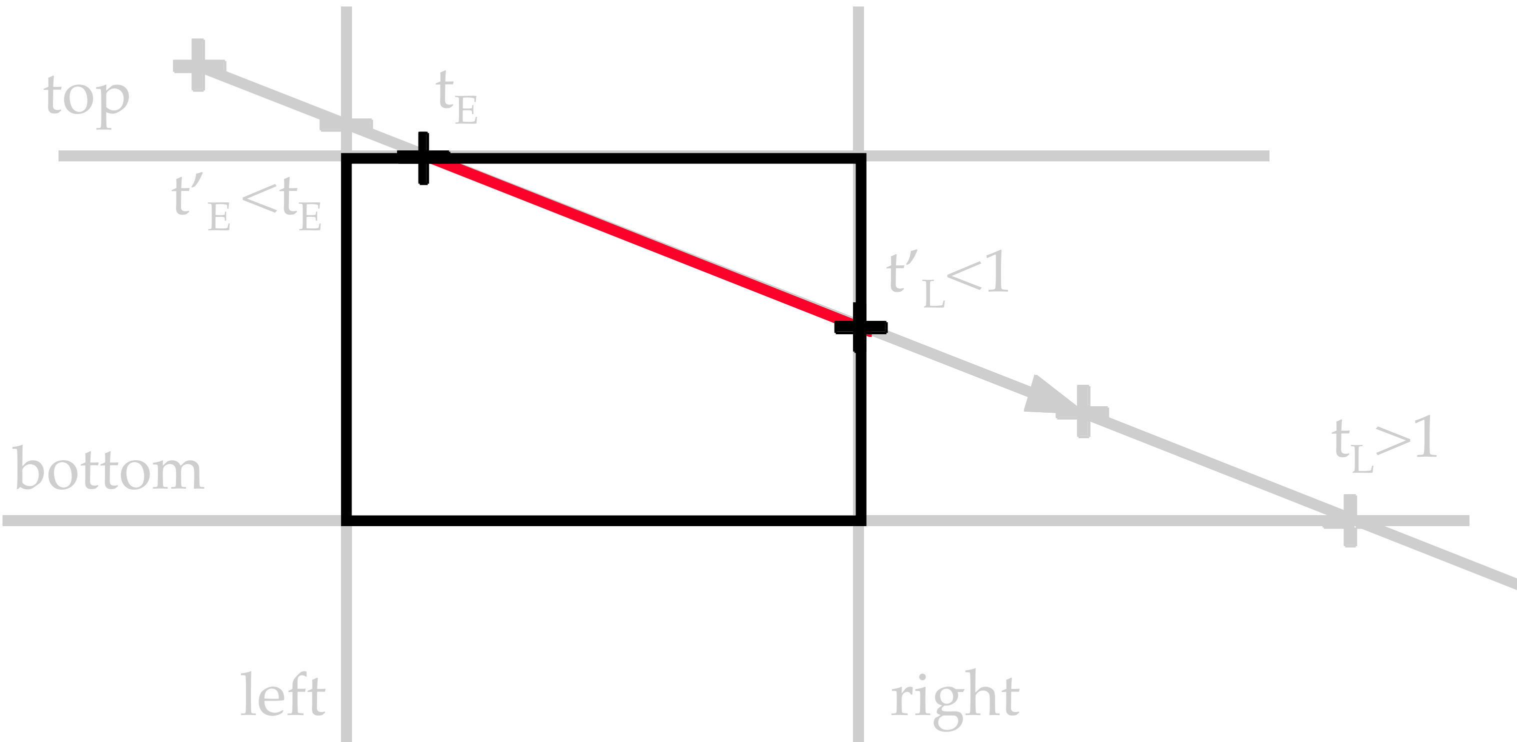
Clipping a line against a window



Clipping a line against a window



Clipping a line against a window



Clipping a line: conclusion

- A simple algorithm
- Requires only standard operations
  - dot products, divisions
  - even faster if you use horizontal/vertical boundaries
- Easy to implement using standard libraries:
  - try it in Java