The University of New Mexico

Implementation I

Ed Angel

Professor of Computer Science, Electrical and Computer Engineering, and Media Arts

University of New Mexico

Objectives

- Introduce basic implementation strategies
- Clipping
- Scan conversion

Overview

- At end of the geometric pipeline, vertices have been assembled into primitives
- Must clip out primitives that are outside the view frustum
  - Algorithms based on representing primitives by lists of vertices
- Must find which pixels can be affected by each primitive
  - Fragment generation
  - Rasterization or scan conversion

Required Tasks

- Clipping
- Rasterization or scan conversion
- Transformations
- Some tasks deferred until fragement processing
  - Hidden surface removal
  - Antialiasing

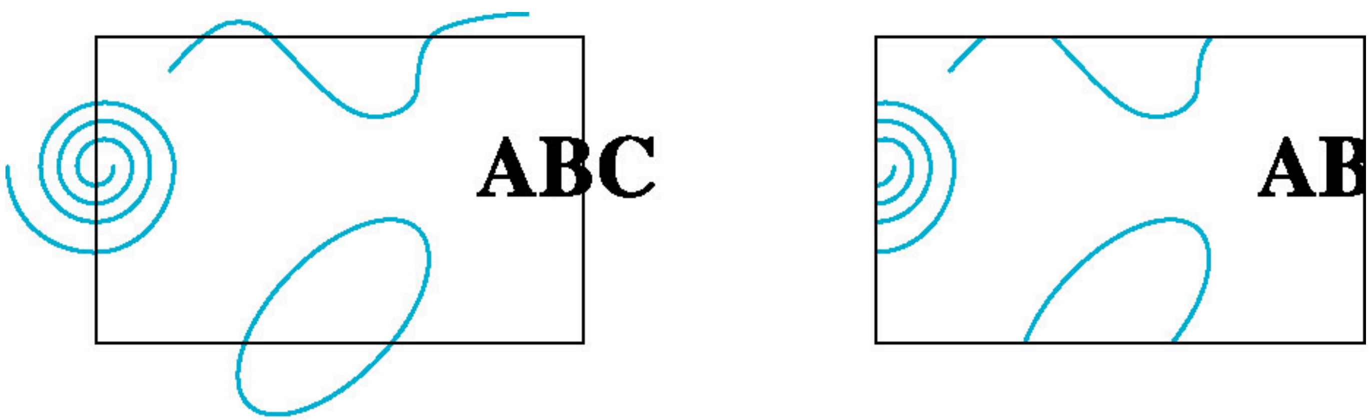
[redacted]

Rasterization Meta Algorithms

- Consider two approaches to rendering a scene with opaque objects
- For every pixel, determine which object that projects on the pixel is closest to the viewer and compute the shade of this pixel
  - Ray tracing paradigm
- For every object, determine which pixels it covers and shade these pixels
  - Pipeline approach
  - Must keep track of depths

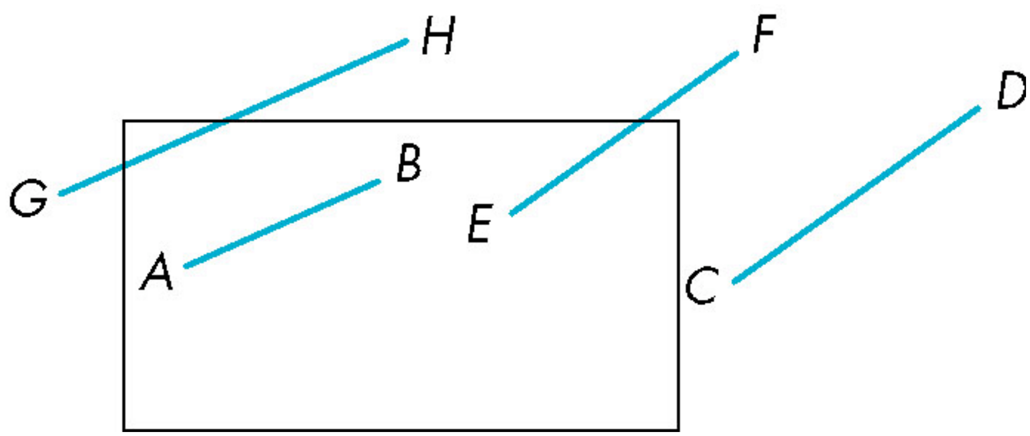
Clipping

- 2D against clipping window
- 3D against clipping volume
- Easy for line segments polygons
- Hard for curves and text
    - Convert to lines and polygons first



Clipping 2D Line Segments

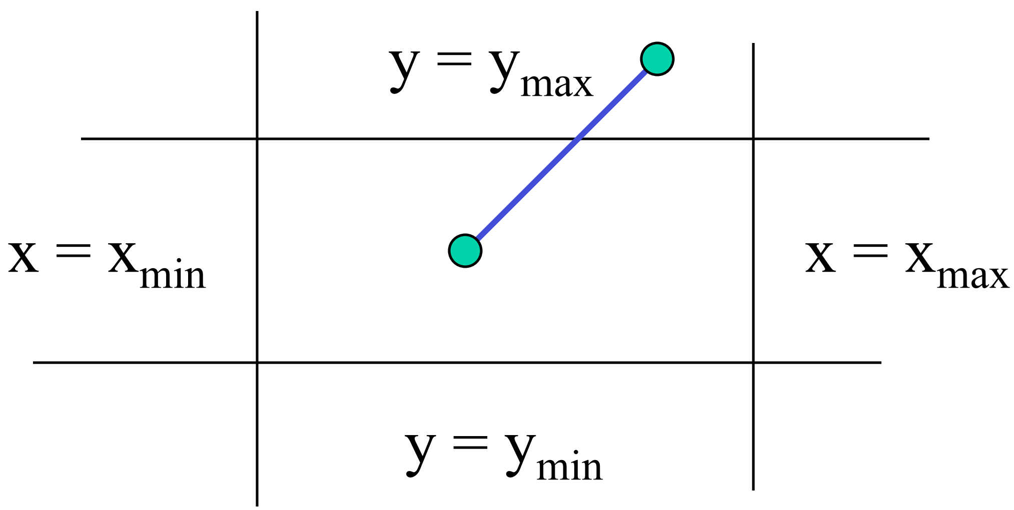
- Brute force approach: compute intersections with all sides of clipping window
  - Inefficient: one division per intersection



Cohen-Sutherland Algorithm

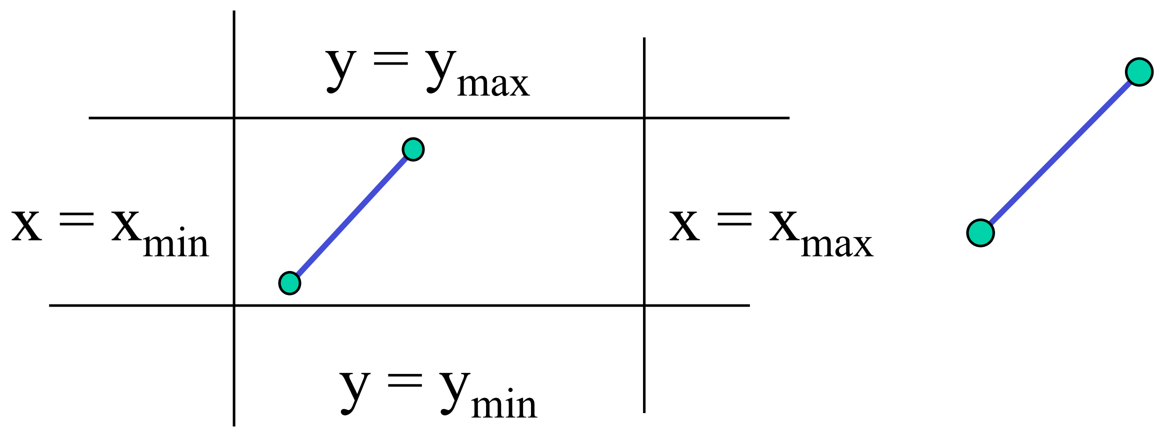
The University of New Mexico

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



The Cases

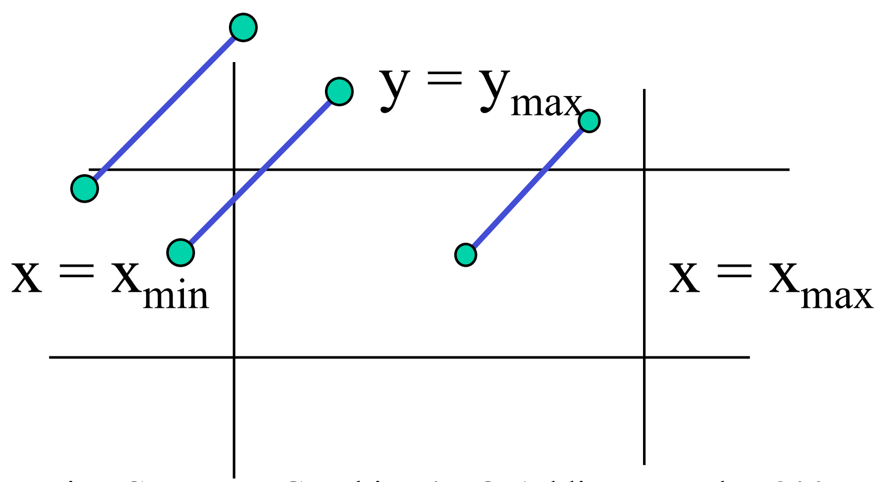
- Case 1: both endpoints of line segment inside all four lines
  - Draw (accept) line segment as is



- 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

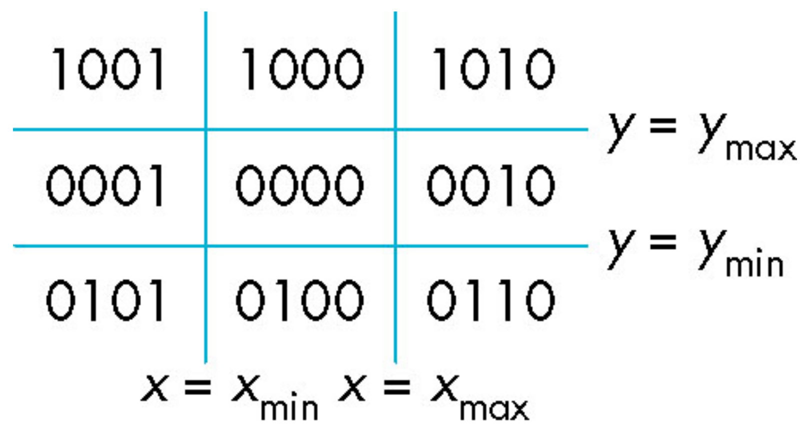


Defining Outcodes

- For each endpoint, define an outcode

$$b_0 b_1 b_2 b_3$$

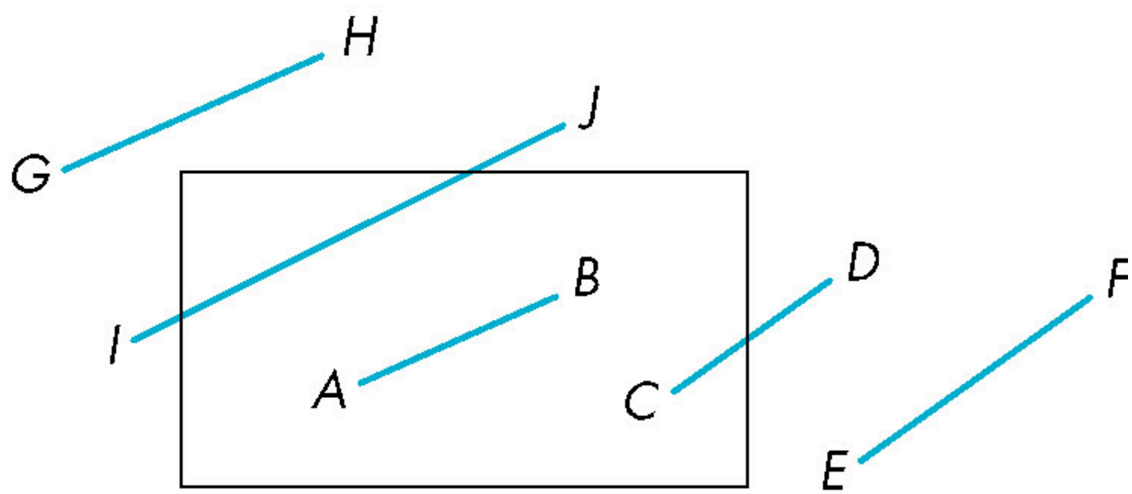
$b_0 = 1$ if $y > y_{max}$, 0 otherwise
$b_1 = 1$ if $y < y_{min}$, 0 otherwise
$b_2 = 1$ if $x > x_{max}$, 0 otherwise
$b_3 = 1$ if $x < x_{min}$, 0 otherwise



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

Using Outcodes

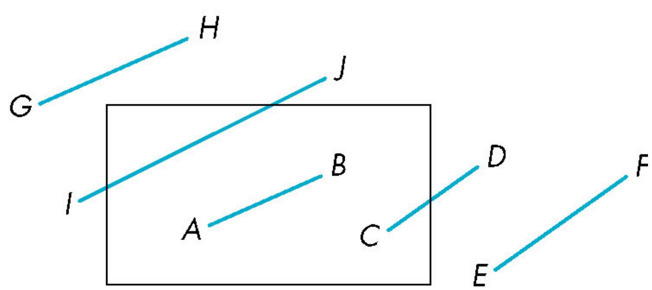
- Consider the 5 cases below
- AB: outcode(A) = outcode(B) = 0
  - Accept line segment



The University of New Mexico

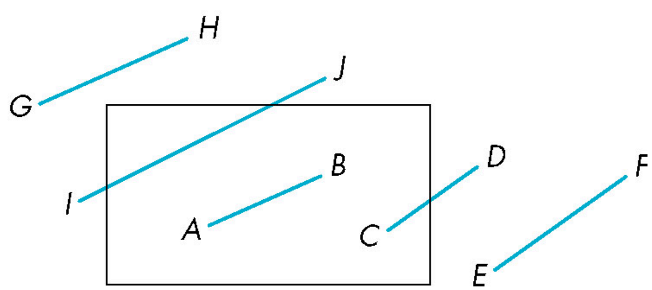
Using Outcodes

- CD: outcode (C) = 0, outcode(D) ≠ 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 interesections



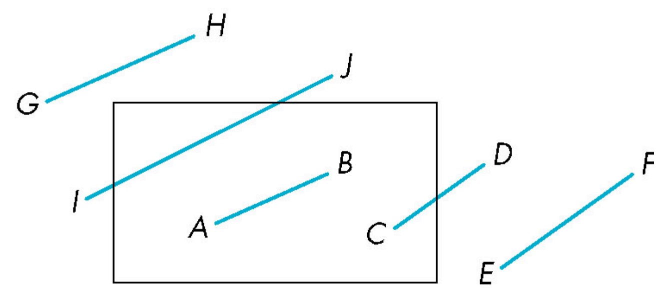
Using Outcodes

- 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

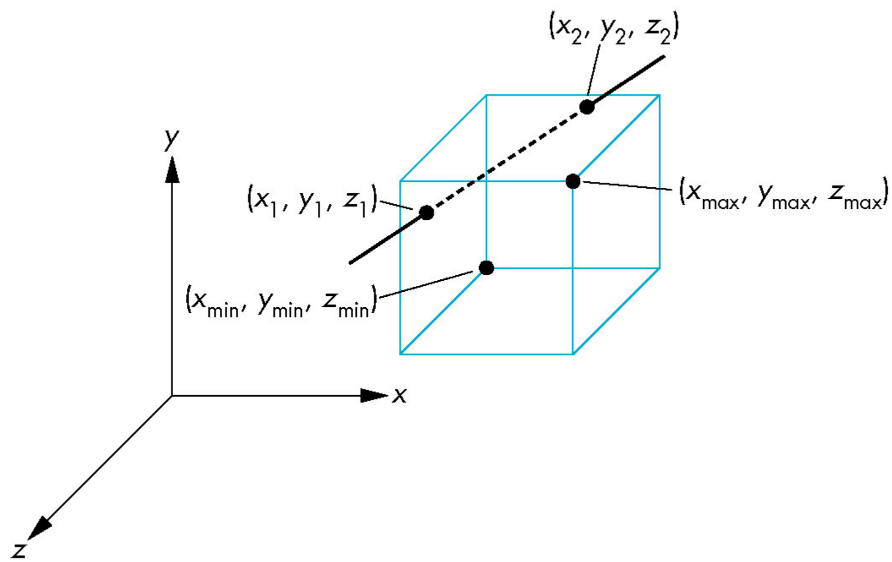


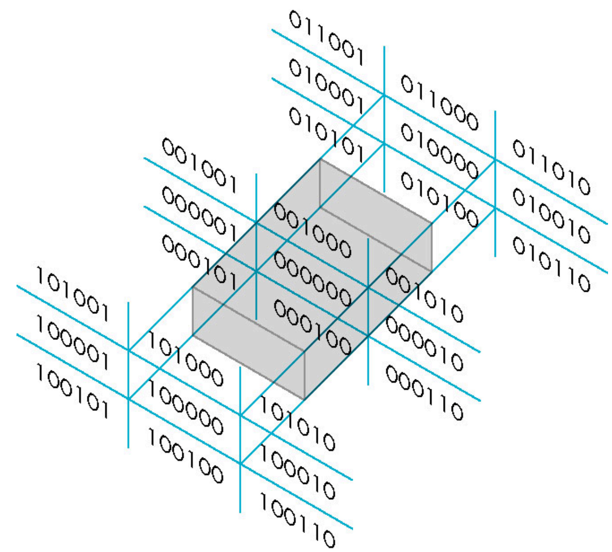
Efficiency

- In many applications, the clipping window is small relative to the size of the entire 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



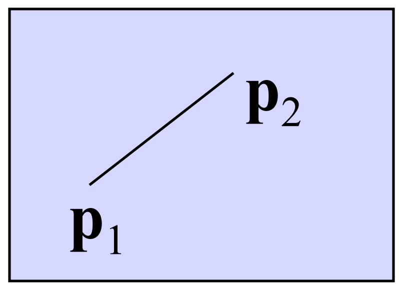


Liang-Barsky Clipping

The University of New Mexico

- Consider the parametric form of a line segment

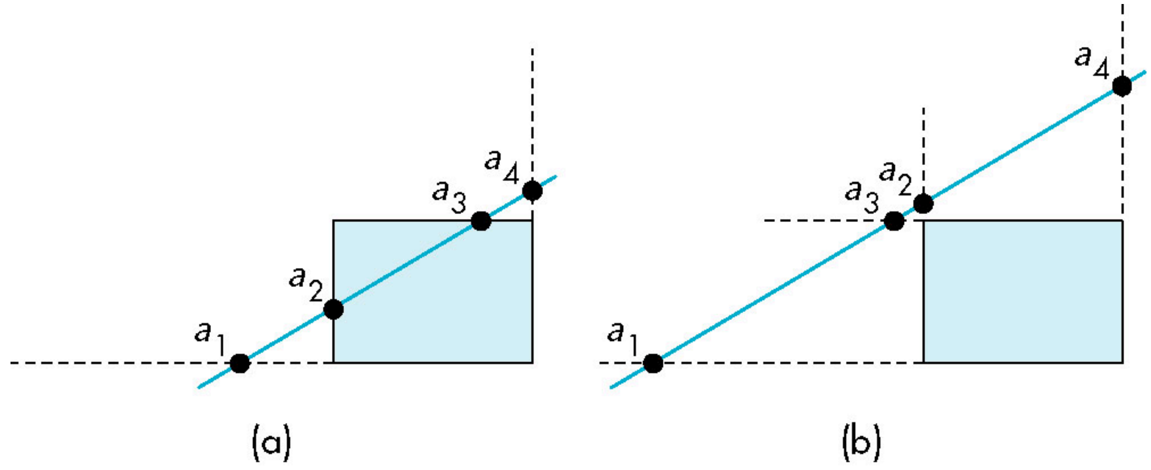
$$\mathbf{p}(\alpha) = (1-\alpha)\mathbf{p}_1 + \alpha\mathbf{p}_2 \quad 1 \ge \alpha \ge 0$$



- We can distinguish between the cases by looking at the ordering of the values of α 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

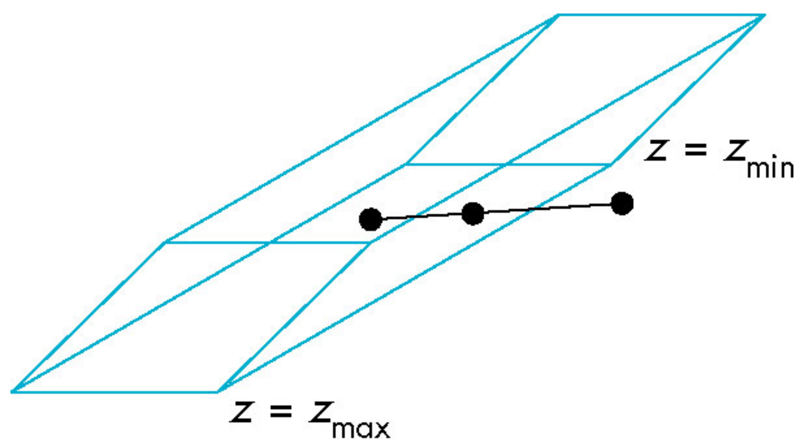


Advantages

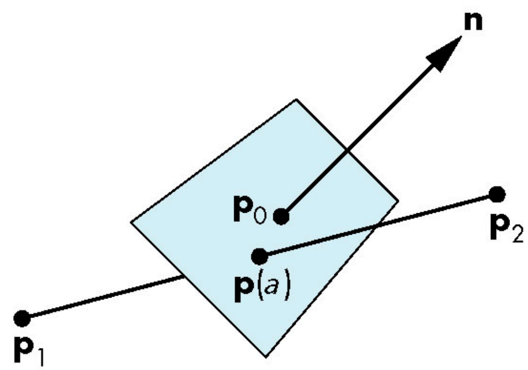
- Can accept/reject as easily as with Cohen-Sutherland
- Using values of α, we do not have to use algorithm recursively as with C-S
- Extends to 3D

Clipping and Normalization

- General clipping in 3D requires intersection of line segments against arbitrary plane
- Example: oblique view



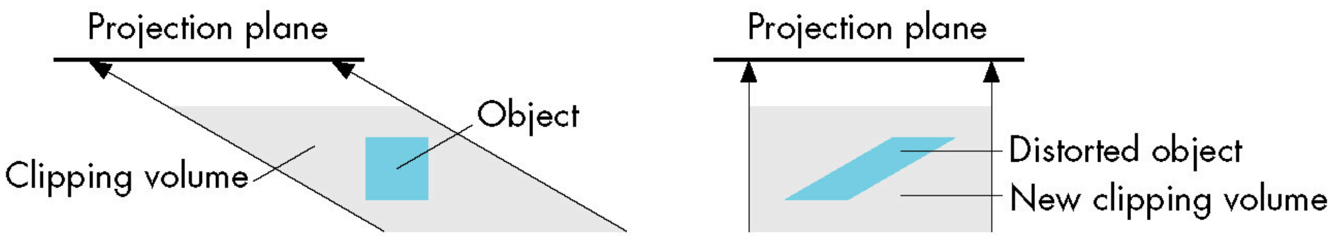
Plane-Line Intersections



$$a = \frac{n \bullet (p_o - p_1)}{n \bullet (p_2 - p_1)}$$

Normalized Form

top view



before normalization after normalization

Normalization is part of viewing (pre clipping) but after normalization, we clip against sides of right parallelepiped

Typical intersection calculation now requires only a floating point subtraction, e.g. is $x > x_{max}$?