Computer Graphics
- Clipping -

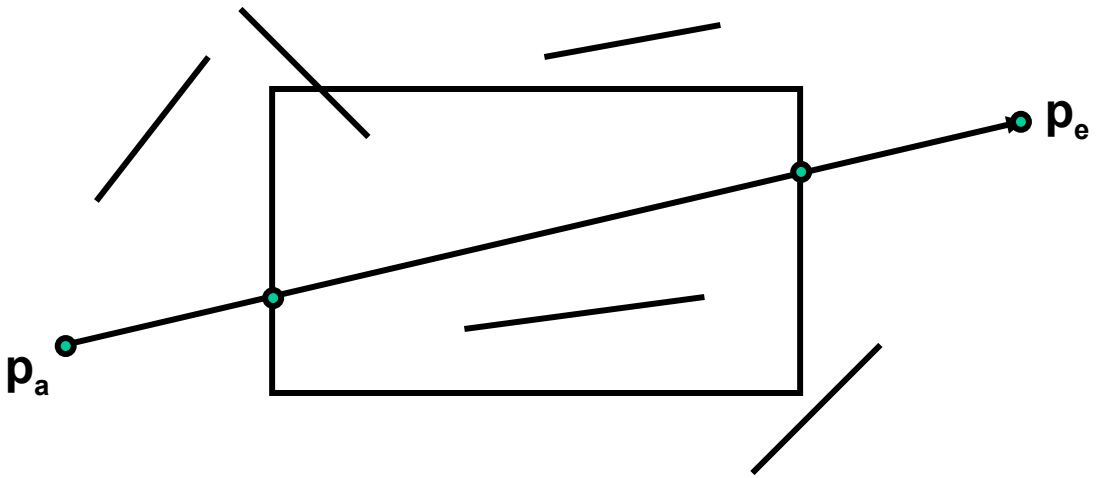
Philipp Slusallek

Motivation

- **Clipping**
  - Happens after transformation from 3D to 2D
  - Many primitives will fall (partially) outside of display window
    - E.g. if standing inside a building
  - Eliminates non-visible geometry early in the pipeline
  - Must cut off parts outside the window
    - Cannot draw outside of window (e.g. plotter)
    - Outside geometry might not be representable (e.g. in fixed point)
  - Must maintain information properly
    - Drawing the clipped geometry should give the correct results
    - Type of geometry might change
      - Cutting off a vertex of a triangle produces a quadrilateral
      - Might need to be split into triangle again
    - Polygons must remain closed after clipping

Line clipping

- **Definition Clipping:**
  - Cut off parts of objects, which lie outside/inside of a defined region.
  - Often: Clipping against a viewport (2D) or a canonical view-volume (3D)

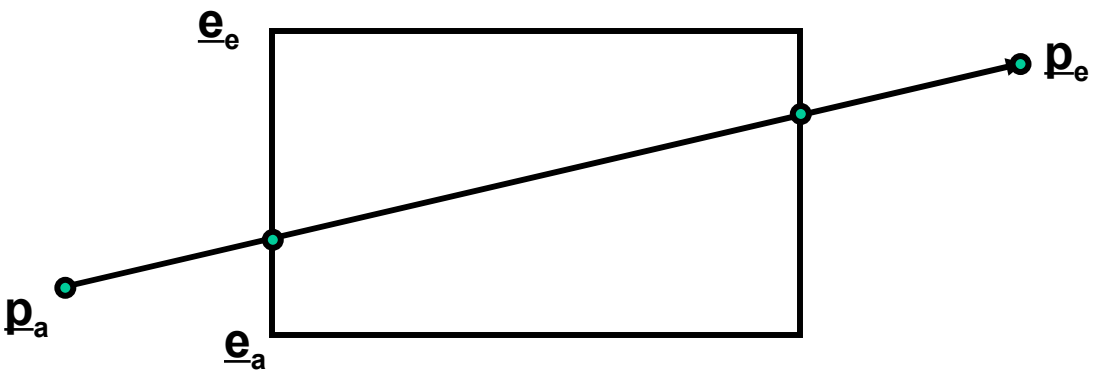


Brute-force method

- **Brute-Force line clipping at the viewport**
  - If both points $\underline{p}_a$ and $\underline{p}_e$ are inside,
    - Accept the whole line
  - Otherwise, clip the line at each edge

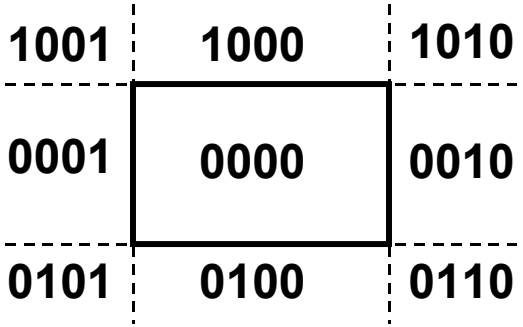
$$\underline{p} = \underline{p}_a + t_{line}(\underline{p}_e - \underline{p}_a) = \underline{e}_a + t_{edge}(\underline{e}_e - \underline{e}_a)$$

- Intersection point, if $0 \le t_{line}, t_{edge} \le 1$
- Pick up suitable end points from the intersection points for the line



Cohen-Sutherland (´74)

- **Advantage: divide and conquer**
  - Efficient trivial accept and trivial reject
  - Non-trivial case: divide and test
- **Outcodes of points:**
  - Bit encoding (**o**ut**c**ode, OC)
    - Each edge defines a half space
    - Set bit, if point is outside
- **Trivial cases**
  - Trivial accept:
    - $(OC(p_a)$ **OR** $OC(p_e)) = 0$
  - Trivial reject:
    - $(OC(p_a)$ **AND** $OC(p_e)) \neq 0$
  - Edges has to be clipped to all edges where bits are set:
    - $OC(p_a)$ **XOR** $OC(p_e)$



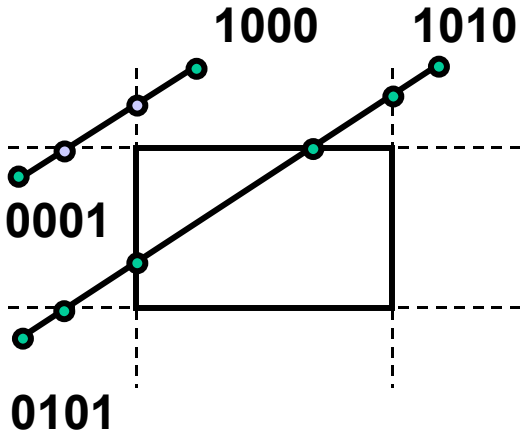
Bit order: **Top, Bottom, Right, Left**

Viewport $(x_{min}, y_{min}, x_{max}, y_{max})$

Cohen-Sutherland

- **Clipping**

```
... // trivial cases
for each vertex p
  oc= OC(p)
  for each edge e
    if (oc[e]) {
      p= cut(p,e);
      oc= OC(p);
    }
  Reject, if point outside
```



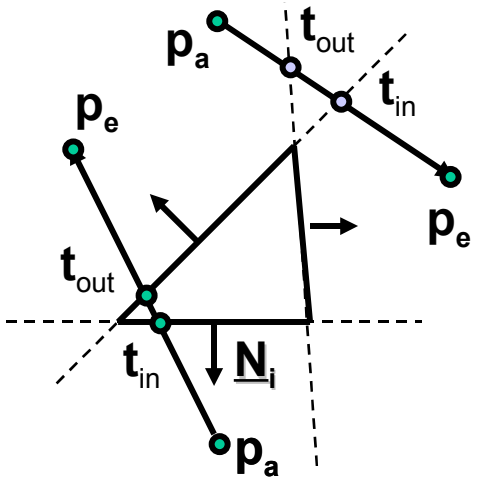
- **Intersection calculation for $x=x_{min}$**

$$\frac{y - y_a}{y_e - y_a} = \frac{x - x_a}{x_e - x_a}$$

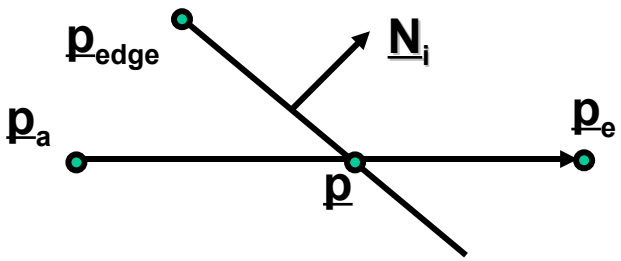
$$y = y_a + (x - x_a)\frac{y_e - y_a}{x_e - x_a}$$

Cyrus-Beck (´78)

- **Parametric line-clipping algorithm**
  - Only convex polygons: max. 2 intersection points
  - Use edge orientation
- **Idea:**
  - Clipping line $\underline{p}_a + t_i(\underline{p}_e - \underline{p}_a)$ with each edge
  - Intersection points sorted by parameter t_i
  - Select
    - t_{in}: entry point $((\underline{p}_e - \underline{p}_a)\cdot\underline{N}_i < 0)$ with largest t_i and
    - t_{out}: exit point $((\underline{p}_e - \underline{p}_a)\cdot\underline{N}_i > 0)$ with smallest t_i
  - If $t_{out} < t_{in}$, line lies completely outside



- **Intersection calculation:**



$$(p - p_{edge})\cdot N_i = 0$$

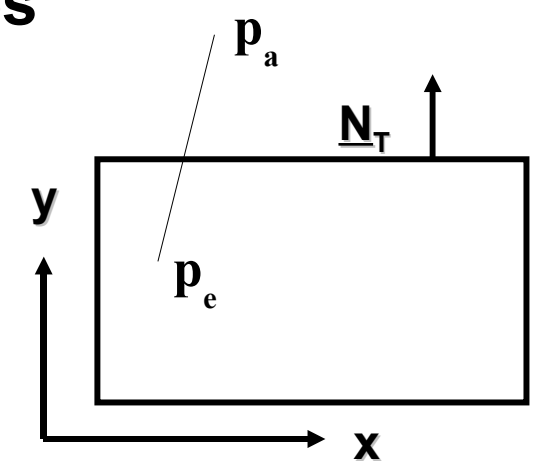
$$t_i(p_e - p_a)\cdot N_i + (p_a - p_{edge})\cdot N_i = 0$$

$$t_i = \frac{(p_{edge} - p_a)\cdot N_i}{(p_e - p_a)\cdot N_i}$$

Liang-Barsky (´84)

- **Cyrus-Beck for axis-parallel rectangles**
  - Using Window-Edge-Coordinates (with respect to an edge T)

$$WEC_T(p) = (p - p_T) \cdot N_T$$



- **Example: top (y= y_{max})**

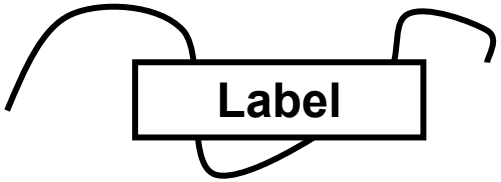
$$N_T = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \; p_a - p_T = \begin{pmatrix} x_a - x_{max} \\ y_a - y_{max} \end{pmatrix}$$

$$t_T = \frac{(p_a - p_T) \cdot N_T}{(p_a - p_e) \cdot N_T} = \frac{WEC_T(p_a)}{WEC_T(p_a) - WEC_T(p_e)} = \frac{y_a - y_{max}}{y_a - y_e}$$

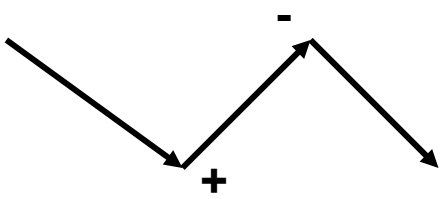
  - Window-Edge-Coordinate (WEC): Decision function for an edge
    - Directed distance to edge
      - Only sign matters, similar to Cohen-Sutherland opcode
    - Sign of the dot product determines whether the point is in or out
    - Normalization unimportant

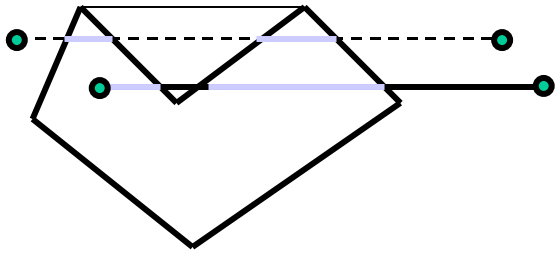
Enhancements CB-Clipping

- **Outer clipping (exterior clipping, covering)**
  - Inversion of clipping: max. 2 segments per line
  - Examples: window systems, labels, ...



- **Concave region clipping**
  - Detection of convexity
    - Cross product of adjacent (non-colinear) segments changes sign
  - Procedure
    - Add regions to make it convex
      - inner clipping on the whole
      - outer clipping on additions
    - Convex region gap
      - combine segments
    - Jordan curve theorem
      - determines of start point is in
      - sorting all intersection points
      - generate consecutive segments



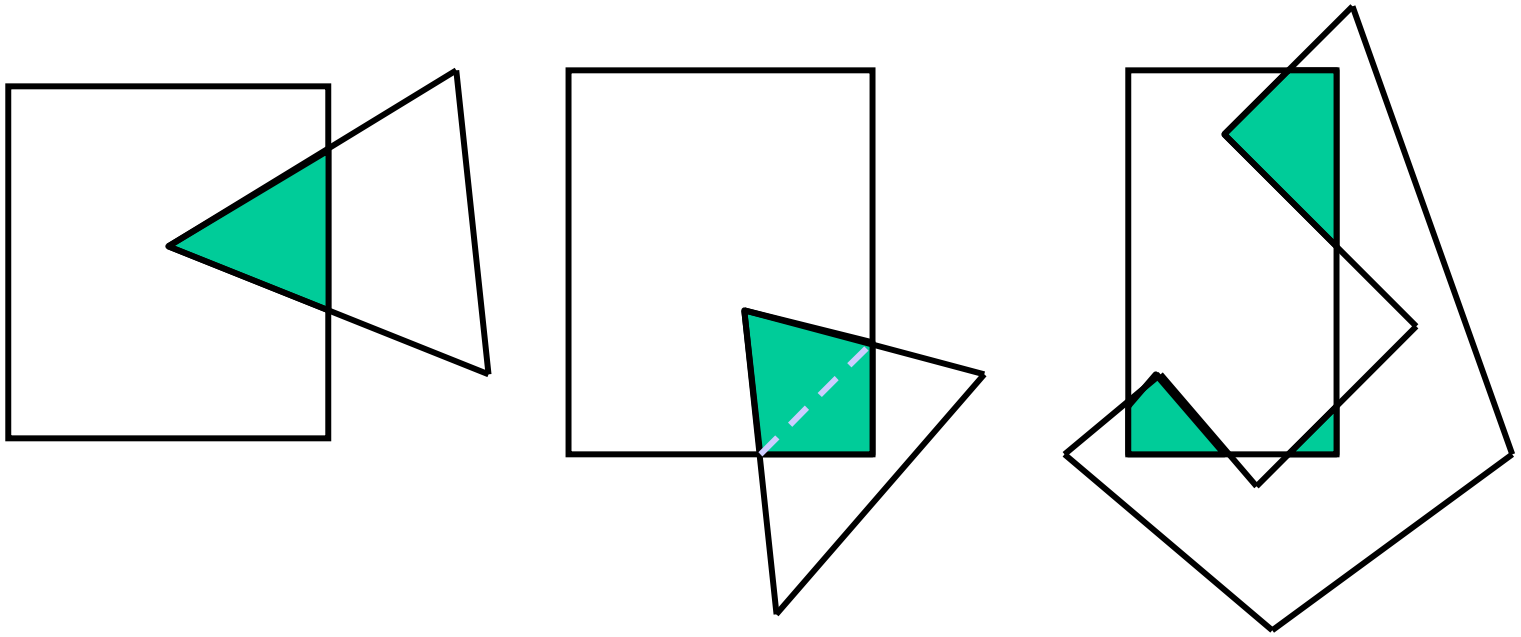


Line clipping - Summary

- **Cohen-Sutherland, Cyrus-Beck, and Liang-Barsky algorithms readily extend to 3D**
- **Cohen-Sutherland algorithm**
  - \+ Efficient when a majority of lines can trivially accepted or rejected
    - Very large clip rectangles: almost all lines inside
    - Very small clip rectangles: almost all lines outside
  - – Repeated clipping for remaining lines
  - – Testing for 2D/3D point coordinates
- **Cyrus-Beck (Liang-Barsky) algorithms**
  - \+ Efficient when many lines must be clipped
  - \+ Testing for 1D parameter values
  - – Testing intersections always for all clipping edges (in the Liang-Barsky trivial rejection testing possible)

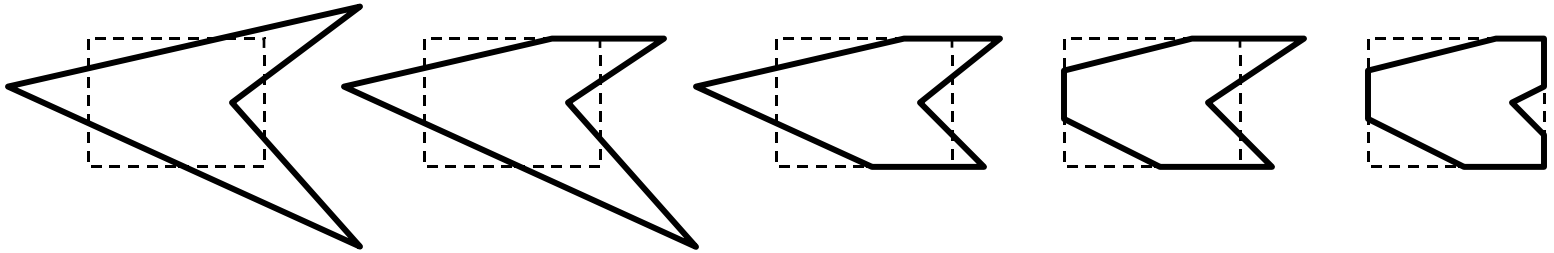
Polygon Clipping

- **Extending line clipping**
  - Polygons have to remain closed
    - Filling, hatching, shading, ...

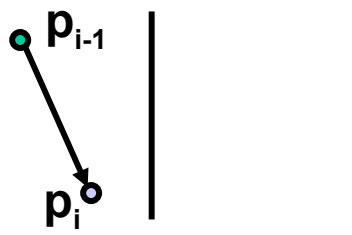


Sutherland-Hodgeman (´74)

- **Idea:**
  - Iterative clipping against each clipping line

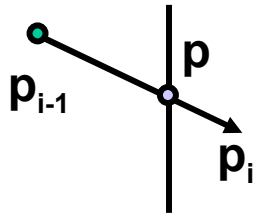


  - Local operations on p_{i-1} and p_i



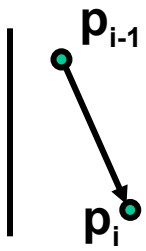
inside outside

output: p_i



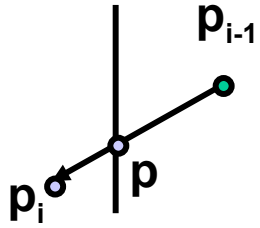
inside outside

output : p



inside outside

output : -

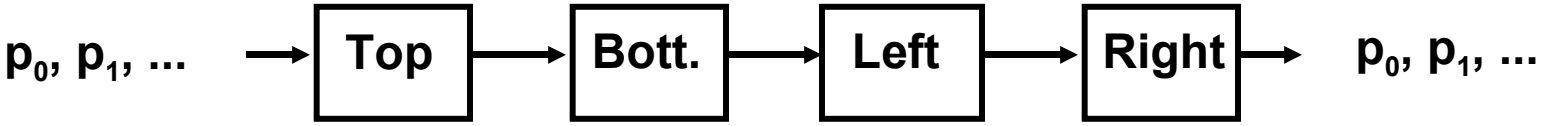


inside outside

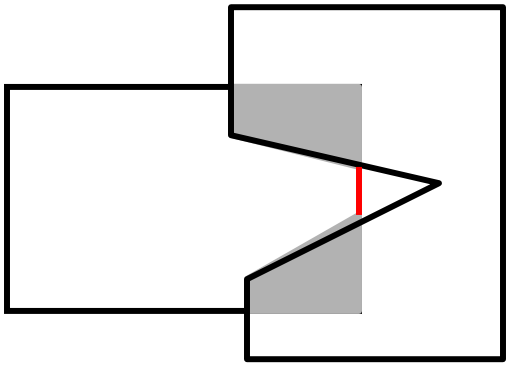
first output : p and
second output: p_i

Enhancements

- **Recursive polygon clipping**
  - Pipelined Sutherland-Hodgeman

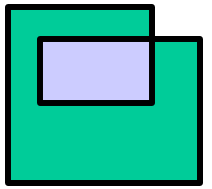


- **Problems**
  - Degenerated polygons/edges
    - Elimination by post-processing, if necessary



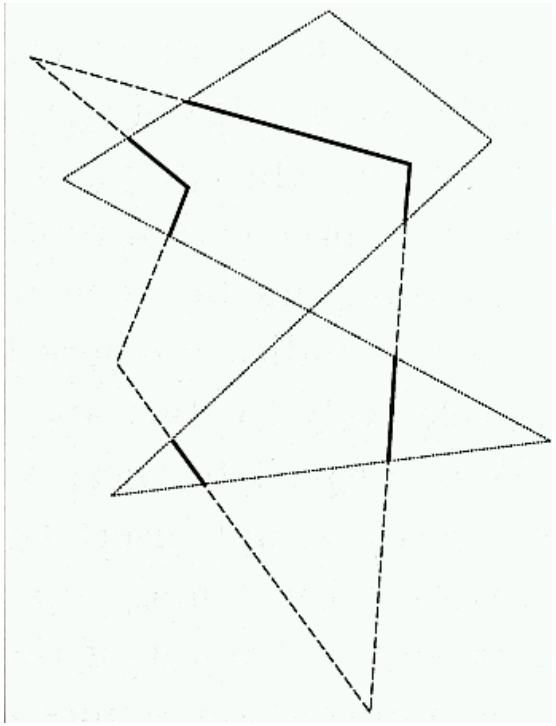
Other clipping algorithms

- **Weiler & Atherton (´77)**
  - Arbitrary concave polygons with holes against each other
- **Vatti (´92)**
  - Also with self-overlap
- **Greiner & Hormann (TOG ´98)**
  - Simpler and faster as Vatti
  - Also supports boolean operations
  - Idea:
    - Odd winding number rule
      - Intersection with the polygon leads to a winding number ± 1
    - Walk along both polygons
    - Alternate winding number
    - Mark point of entry and point of exit
    - Combine results

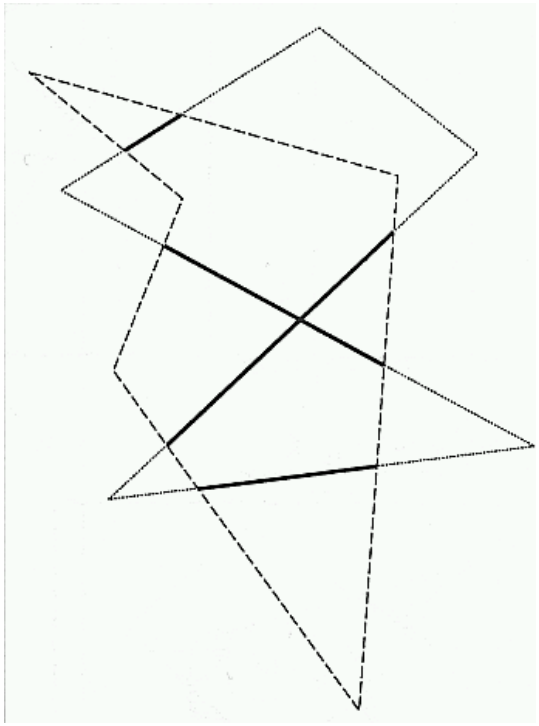


Non-zero WN: In
Odd WN: Out

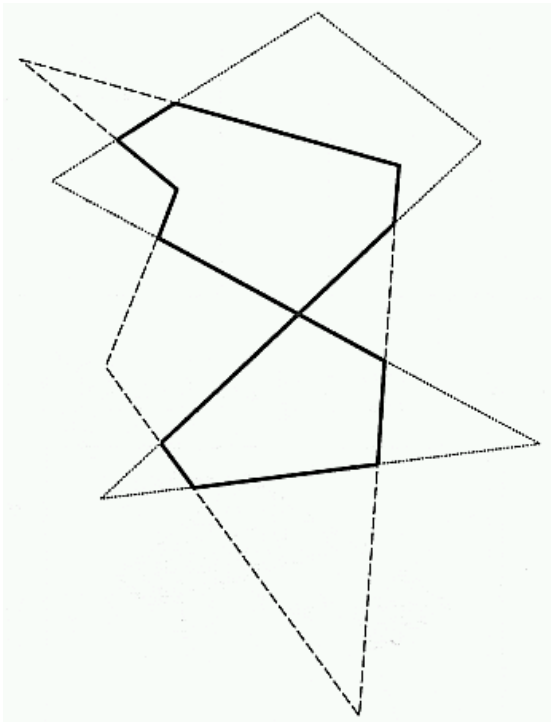
Greiner & Hormann



A in B



B in A



(A in B) ∪ (B in A)

3D clipping against view volume

- **Requirements**
  - Avoid unnecessary rasterization
  - Avoid overflow on transformation at fixed point !
- **Clipping against viewing frustum**
  - Enhanced Cohen-Sutherland with 6-bit outcode
  - After perspective division
    - $-1 < y < 1$
    - $-1 < x < 1$
    - $-1 < z < 0$
  - Clip against side planes of the viewing frustum
  - Works analogous with Liang-Barsky or Sutherland-Hodgeman

3D clipping against view volume

- **Clipping in homogeneous coordinates**
  - Avoid division by w
  - Inside test with a linear distance function (WEC)
    - Left: X/W > -1 ➔ $W+X = WEC_L(\underline{p}) > 0$
    - Top: Y/W < 1 ➔ $W - Y = WEC_T(\underline{p}) > 0$
    - Back: Z/W > -1 ➔ $W+Z = WEC_B(\underline{p}) > 0$
    - ...
  - Intersection point calculation
    - Test: $WEC_L(\underline{p}_a) > 0$ and $WEC_L(\underline{p}_e) < 0$
    - Calculation:

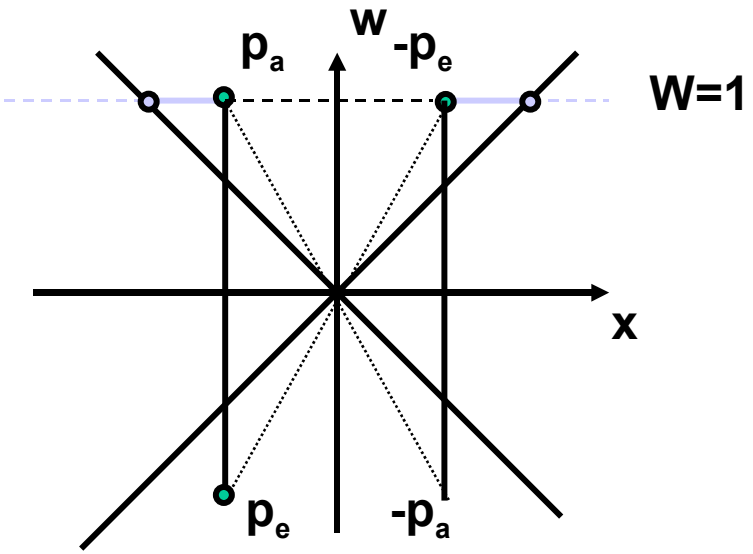
$$WEC(\underline{p}_a + t(\underline{p}_e - \underline{p}_a)) = 0$$

$$W_a + t(W_e - W_a) + X_a + t(X_e - X_a) = 0$$

$$t = \frac{W_a + X_a}{(W_a + X_a) - (W_e + X_e)} = \frac{WEC_L(\underline{p}_a)}{WEC_L(\underline{p}_a) - WEC_L(\underline{p}_e)}$$

Problems with Homog. Coord.

- **Negative w**
  - Points with $w < 0$ or lines with $w_a < 0$ and $w_e < 0$
    - Negate and continue
  - Lines with $w_a \cdot w_e < 0$ (NURBS)
    - Line moves through infinity
      - External line
    - Clipping two times
      - Original Line
      - Negated line
    - Generates up to two segments



Practical implementations

- **Combining clipping and scissoring**
  - Clipping is expensive and should be avoided
    - Intersection calculation
    - Variable number of new points
  - Enlargement of clipping region
    - Larger than viewport, but
    - Still avoiding overflow due to fixed-point representation
  - Result
    - Less clipping
    - Applications should avoid drawing objects which are lying outside of the viewing frustum
    - Objects which are lying partially outside will be clipped implicitly during rasterization.