

Computer Graphics

- Clipping -

Philipp Slusallek

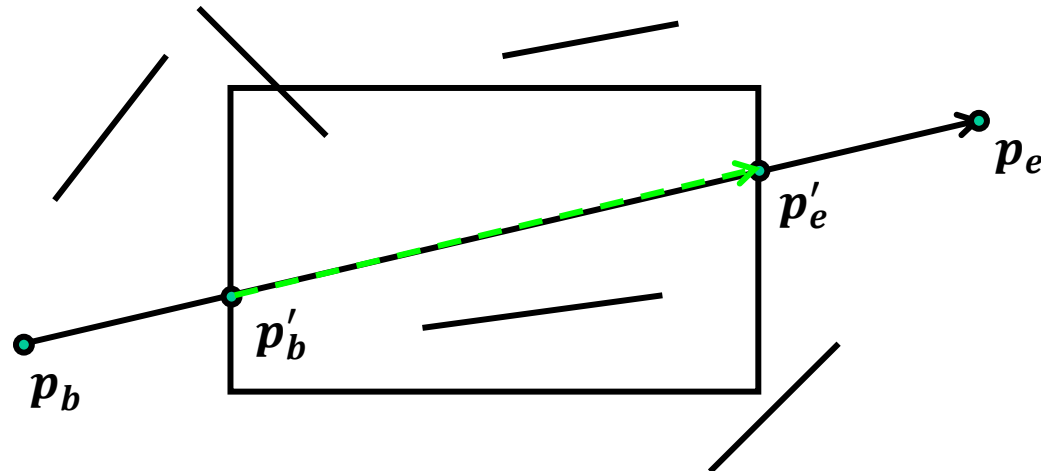
Clipping

- **Motivation**

- Projected primitive might fall (partially) outside of screen window
 - E.g., if standing inside a building
 - Eliminate non-visible geometry early in the pipeline to process visible parts only
 - Happens after 3D to 2D transformation, but still in floating point
 - Must cut off parts outside the window
 - Outside geometry might not be representable (e.g., in fixed point)
 - Cannot draw outside of window (e.g. HW limitation)
 - Must maintain information properly
 - Drawing the clipped geometry should give the correct results:
 - E.g., correct interpolation of colors across triangle even when clipped
 - Type of geometry might change
 - Cutting off a vertex of a triangle produces a quadrilateral (up to hexagon)
 - Might need to be split into triangles again
 - Polygons must remain closed after clipping
-

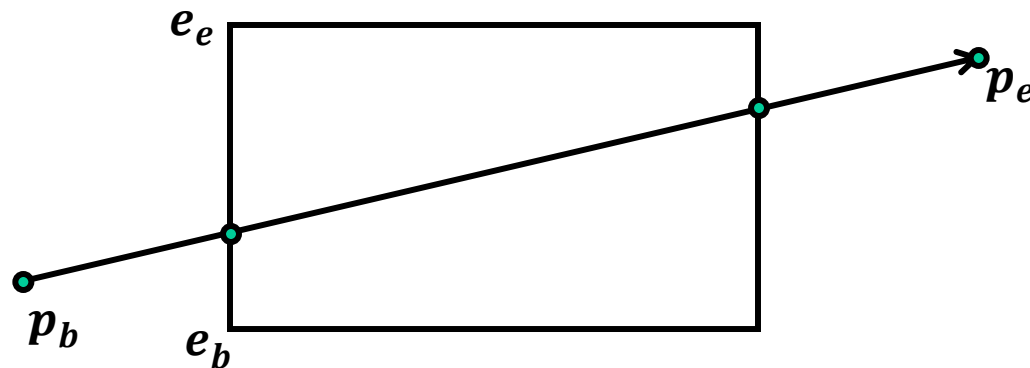
Line Clipping

- **Definition of clipping**
 - Cut off parts of objects which lie outside/inside of a defined region
 - Often clip against viewport (2D) or canonical view-volume (3D)
- **Let's focus first on lines only**



Brute-Force Method

- **Brute-force line clipping at the viewport**
 - If both end points p_b and p_e are inside viewport
 - Accept the whole line
 - Otherwise, clip the line at *each edge*
 - $p_{\text{intersection}} = p_b + t_{\text{line}}(p_e - p_b) = e_b + t_{\text{edge}}(e_e - e_b)$
 - Solve for t_{line} and t_{edge}
 - Intersection within segment iff both $0 \leq t_{\text{line}}, t_{\text{edge}} \leq 1$
 - Replace suitable end point of the line by the intersection point
 - Unnecessarily tests many cases that are irrelevant



Cohen-Sutherland (1974)

- **Advantage: divide and conquer**

- Efficient trivial accept and trivial reject
- Non-trivial case: divide and test

- **Outcodes of points**

- Bit encoding (outcode, OC)
 - Each viewport edge defines a half space
 - Set bit if vertex is outside w.r.t. that edge

- **Trivial cases**

- Trivial accept: both are in viewport
 - $(OC(p_b) \text{ OR } OC(p_e)) == 0$
- Trivial reject: both lie outside w.r.t. *at least one common edge*
 - $(OC(p_b) \text{ AND } OC(p_e)) \neq 0$
- Line has to be clipped to all edges where XOR bits are set, i.e. the points lies on different sides of that edge
 - $OC(p_b) \text{ XOR } OC(p_e)$

1001	1000	1010
0001	0000	0010
0101	0100	0110

Bit order: *top, bottom, right, left*

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

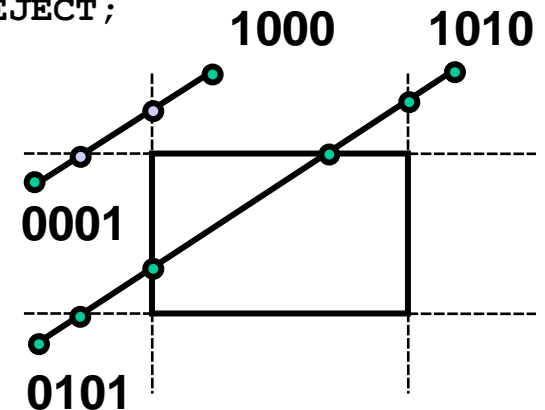
Cohen-Sutherland

- **Clipping of line (p1, p2)**

```
oc1 = OC(p1); oc2 = OC(p2); edge = 0;
do {
  if ((oc1 AND oc2) != 0) // trivial reject of remaining segment
    return REJECT;
  else if ((oc1 OR oc2) == 0) // trivial accept of remaining segment
    return (ACCEPT, p1, p2);
  if ((oc1 XOR oc2)[edge]) {
    if (oc1[edge]) // p1 outside
      {p1 = cut(p1, p2, edge); oc1 = OC(p1);}
    else // p2 outside
      {p2 = cut(p1, p2, edge); oc2 = OC(p2);}
  }
} while (++edge < 4); // Not the most efficient solution
return ((oc1 OR oc2) == 0) ? (ACCEPT, p1, p2) : REJECT;
```

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

$$\frac{y - y_b}{y_e - y_b} = \frac{x_{\min} - x_b}{x_e - x_b}$$
$$y = y_b + (x_{\min} - x_b) \frac{y_e - y_b}{x_e - x_b}$$



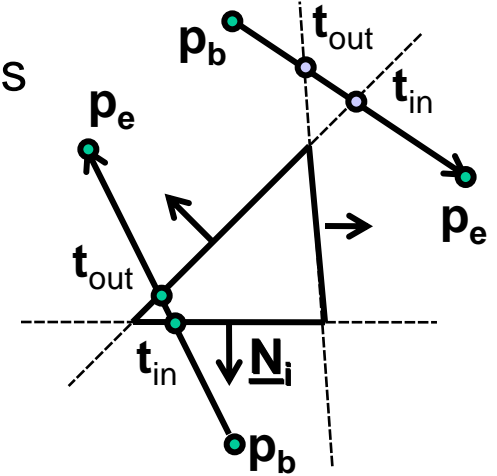
Cyrus-Beck (1978)

- **Parametric line-clipping algorithm**

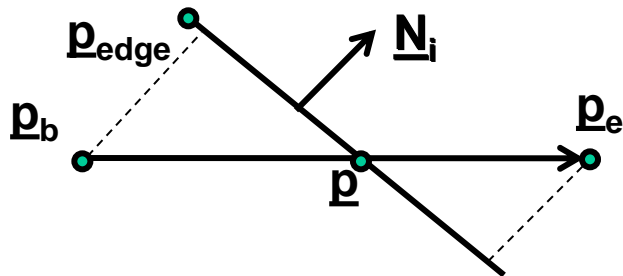
- Only convex polygons: max. 2 intersection points
- Use edge orientation, via „normals“ pointing out

- **Idea: clipping against polygons**

- Clip line $p = p_b + t_i(p_e - p_b)$ against each edge
- In the loop, select
 - t_{in} : entry point $((p_e - p_b) \cdot N_i < 0)$ with largest t_i
 - t_{out} : exit point $((p_e - p_b) \cdot N_i > 0)$ with smallest t_i
 - Keep largest entry and smallest exit point
- If $t_{out} < t_{in}$, line lies completely outside (akin to ray-box intersect.)



- **Intersection calculation**



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

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

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

Liang-Barsky (1984)

- **Cyrus-Beck for axis-aligned rectangles**

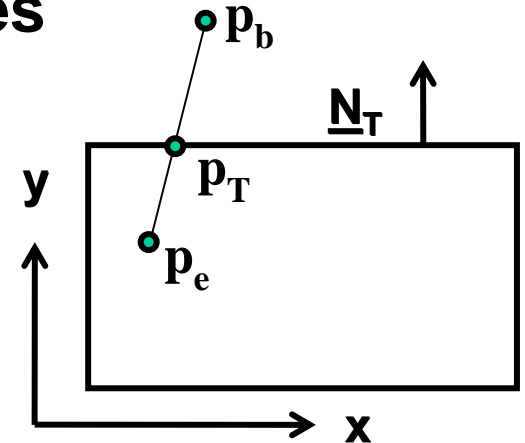
- Using window-edge coordinates (with respect to an edge T)

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

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

$$N_T = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \quad p_b - p_T = \begin{pmatrix} x_b - x_{\max} \\ y_b - y_{\max} \end{pmatrix}$$

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



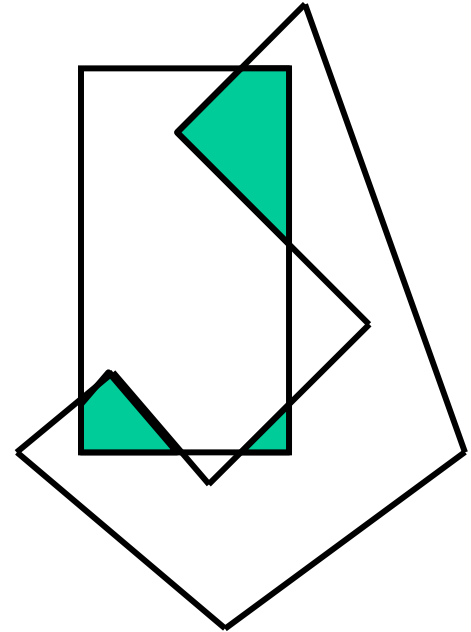
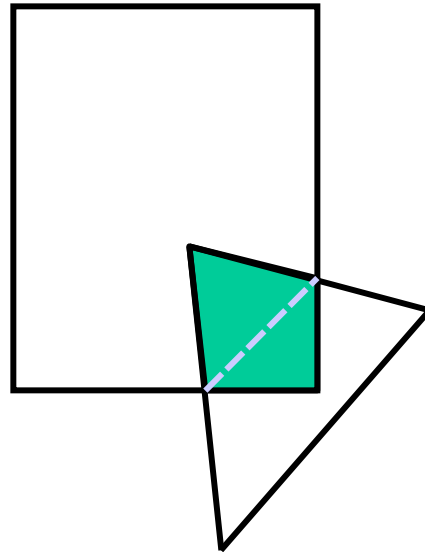
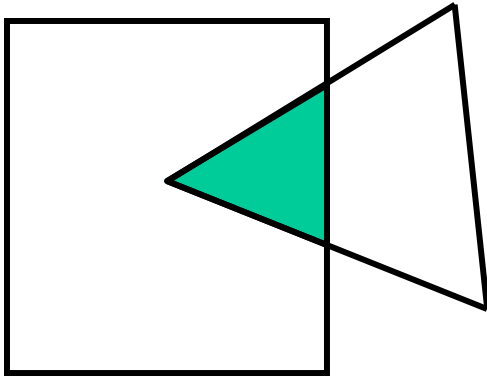
- Window-edge coordinate (WEC): decision function for an edge
 - Directed distance to edge
 - Only sign matters, similar to Cohen-Sutherland opcode
 - Sign of the dot product determines whether the point is in or out
 - Normalization unimportant
 - We will see similar approaches again, soon
-

Line Clipping - Summary

- **Cohen-Sutherland, Cyrus-Beck, and Liang-Barsky algorithms readily extend to 3D**
 - **Cohen-Sutherland algorithm**
 - + Efficient when majority of lines can be trivially accepted / 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

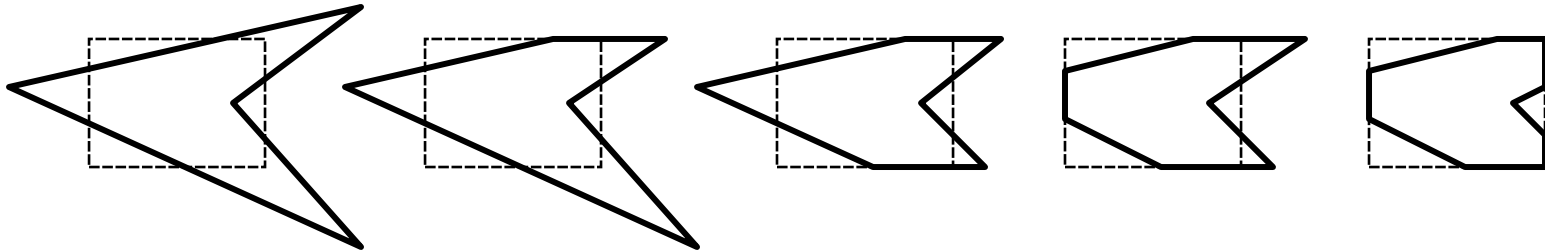
- **Extended version of line clipping**
 - Condition: polygons have to remain closed
 - Filling, hatching, shading, ...



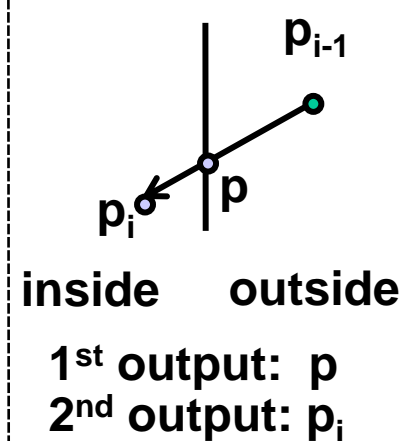
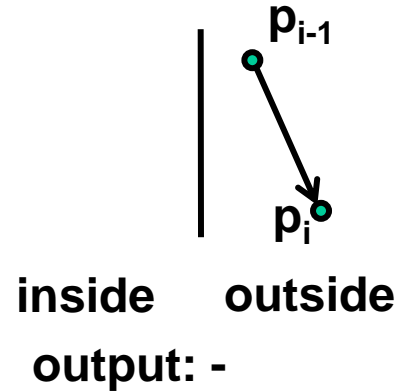
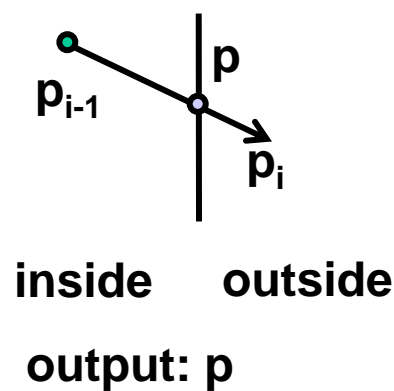
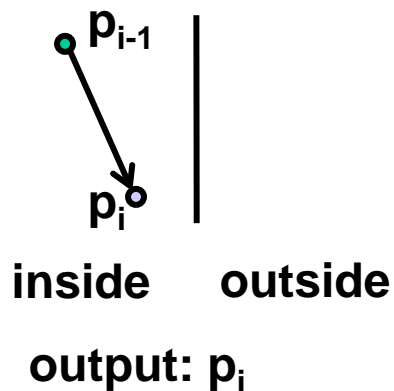
Sutherland-Hodgeman (1974)

- **Idea**

- Iterative clipping against each edge in sequence

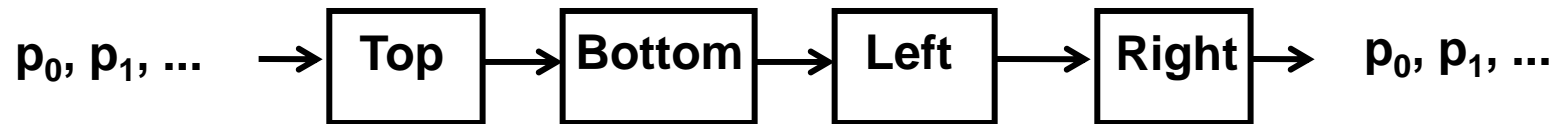


- Four different local operations based on sides of p_{i-1} and p_i



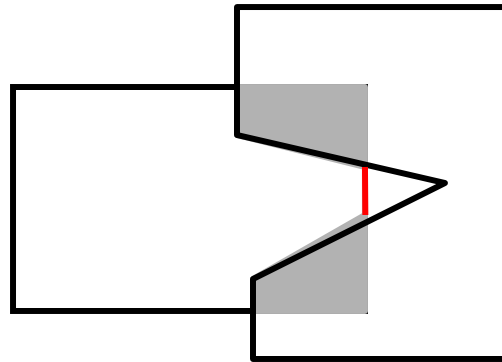
Enhancements

- **Pipelined polygon clipping**



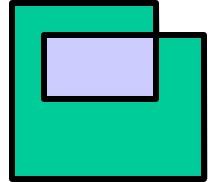
- **Problems**

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

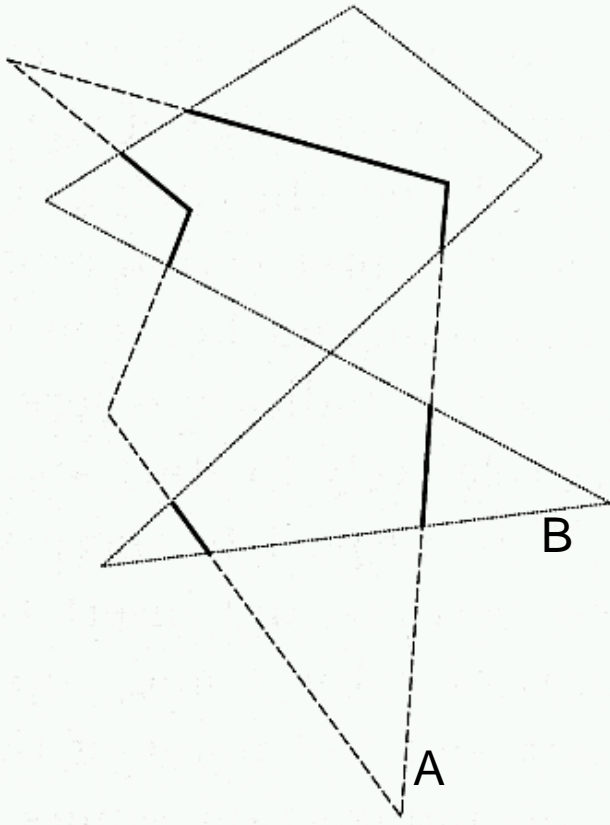


Other Clipping Algorithms

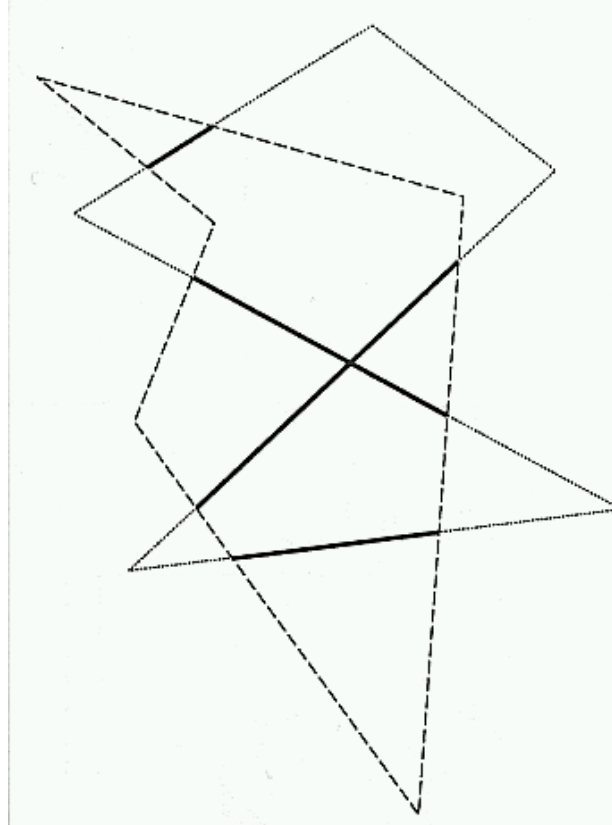
- **Weiler & Atherton ('77)**
 - Arbitrary concave polygons with holes against each other
- **Vatti ('92)**
 - Also with self-overlap
- **Greiner & Hormann (TOG '98)**
 - Simpler and faster as Vatti
 - Also supports Boolean operations
 - Idea:
 - Winding number (WN)
 - Intersection with the polygon leads to a change in winding number of ± 1
 - Walk along both polygons
 - Alternate winding number value, depending on going in/out
 - Mark point of entry and point of exit
 - Combine results



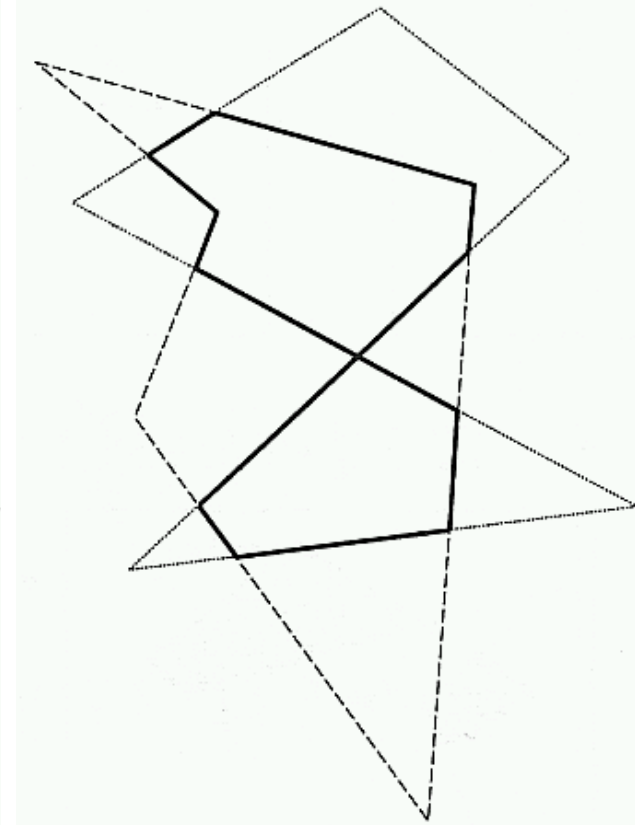
Greiner & Hormann



A in B



B in A



(A in B) U (B in A)

3D Clipping agst. 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 canonical viewing frustum
 - Works analogously with Liang-Barsky or Sutherland-Hodgeman
-

3D Clipping agst. View Volume

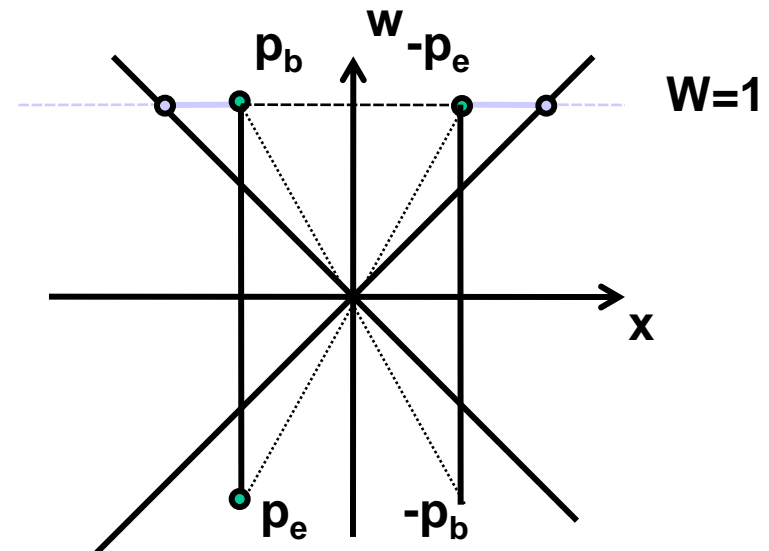
- **Clipping in homogeneous coordinates**
 - Use canonical view frustum, but avoid costly division by W
 - Inside test with a linear distance function (WEC)
 - Left: $X / W > -1 \quad \rightarrow \quad W + X = WEC_L(\underline{p}) > 0$
 - Top: $Y / W < 1 \quad \rightarrow \quad W - Y = WEC_T(\underline{p}) > 0$
 - Back: $Z / W > -1 \quad \rightarrow \quad W + Z = WEC_B(\underline{p}) > 0$
 - ...
 - Intersection point calculation (before homogenizing)
 - Test: $WEC_L(\underline{p}_b) > 0$ and $WEC_L(\underline{p}_e) < 0$
 - Calculation:

$$\begin{aligned} WEC(p_b + t(p_e - p_b)) &= 0 \\ W_b + t(W_e - W_b) + X_b + t(X_e - X_b) &= 0 \\ t &= \frac{W_b + X_b}{(W_b + X_b) - (W_e + X_e)} = \frac{WEC_L(p_b)}{WEC_L(p_b) - WEC_L(p_e)} \end{aligned}$$

Problems with Homogen. Coord.

- **Negative w**

- Points with $w < 0$ or lines with $w_b < 0$ and $w_e < 0$
 - Negate and continue
- Lines with $w_b \cdot w_e < 0$ (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, new triangles
- Enlargement of clipping region
 - (Much) larger than viewport, but
 - Still avoiding overflow due to fixed-point representation with a few more bits
- Result
 - Less clipping
 - Applications should avoid drawing objects that are outside of the viewport/viewing frustum
 - Objects that are still partially outside will be implicitly clipped during rasterization (see later)
 - Slight penalty because they will still be processed (triangle setup)

