Lecture 20:

Clipping

Contents

1. Introduction and Motivation
2. Line Clipping
3. Polygon Clipping

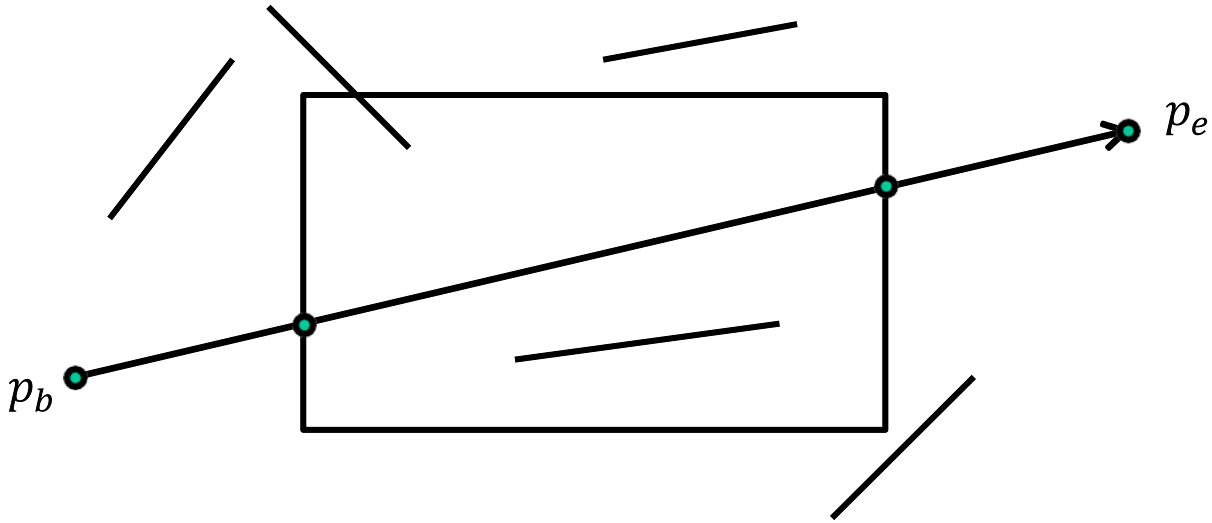
Motivation

- Projected primitive might fall (partially) outside of display area
  - *E.g.* if standing inside a building
- Eliminate non-visible geometry early in the pipeline to process visible parts only
- Happens after transformation from 3D to 2D
- 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: *e.g.* correct interpolation of colors at triangle vertices when one is clipped
  - 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

Definition of clipping

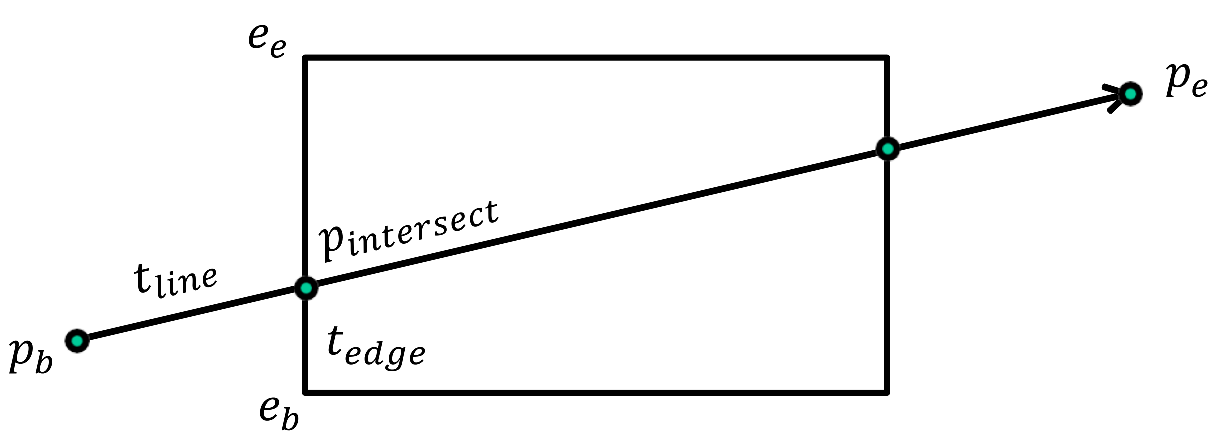
- Cut off parts of objects which lie outside / inside of a defined region
- Often clip against viewport (2D) or canonical view-volume (3D)

Let`s focus first on lines only



Brute-force 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
  - $p_{intersect} = p_{\mathrm{b}} + t_{line}(p_e - p_{\mathrm{b}}) = e_{\mathrm{b}} + t_{edge}(e_e - e_{\mathrm{b}})$
  - Solve for t_{line} and t_{edge}
    - Intersection within segment if both $0 \leq t_{line}$ and $t_{edge} \leq 1$
  - Replace suitable end points for the line by the intersection point
- Unnecessarily test 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 with respect to that edge

1001	1000	1010
0001	0000	0010
0101	0100	0110

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

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

Trivial cases

- **Trivial accept**: both are in viewport
  - `(OC(p_b) OR OC(p_e)) == 0`
- **Trivial reject**: both lie outside with respect to at least one common edge
  - `(OC(p_b) AND OC(p_e)) != 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) XOR OC(p_e)`

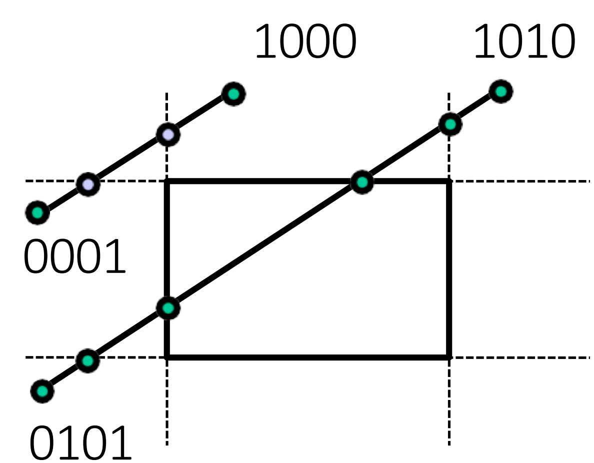
Clipping of line (p1, p2)

```
oc1 = OC(p1); oc2 = OC(p2); edge = 0;
do {
    if ((oc1 & oc2) != 0)            // trivial reject of remaining segment
        return REJECT;
    else if ((oc1 | oc2) == 0)       // trivial accept of remaining segment
        return (ACCEPT, p1, p2);
    if ((oc1 ^ 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 | oc2) == 0) ? (ACCEPT, p1, p2) : REJECT;
```

Intersection calculation for $x = x_{boundary}$

$$\frac{y - y_b}{y_e - y_b} = \frac{x_{boundary} - x_b}{x_e - x_b}$$

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

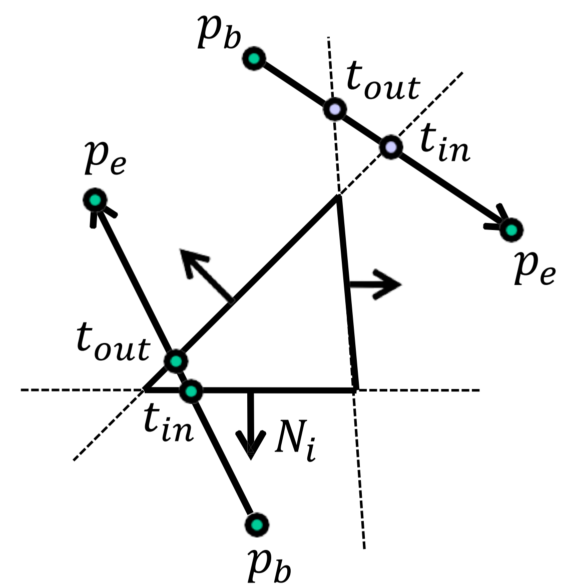


Cyrus-Beck (1978)

Parametric line-clipping algorithm

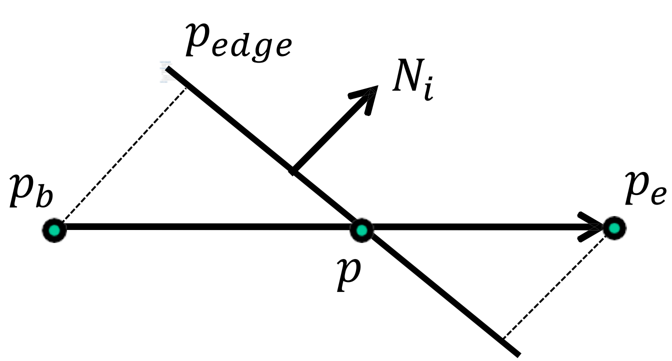
- Only convex polygons: max 2 intersection points
- Use edge orientation

Idea: clipping against polygons



- Clip line $p = p_b + t_i(p_e - p_b)$ with each edge
- Intersection points sorted by parameter t_i
- Select
  - t_{in}: entry point $\big((p_e - p_b) \cdot N_i < 0\big)$ with largest t_i
  - t_{out}: exit point $\big((p_e - p_b) \cdot N_i > 0\big)$ with smallest t_i
- If $t_{out} < t_{in}$, line lies completely outside (akin to ray-box intersect.)

Intersection calculation



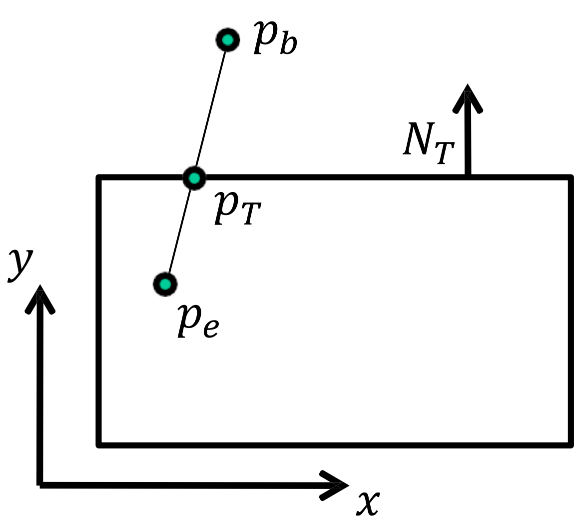
$$\left(p - p_{edge}\right) \cdot N_i = 0$$

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

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

Liang-Barsky intersection algorithm between the line and the clip window

- Significantly more efficient than Cohen–Sutherland
  - By doing as much testing as possible before computing line intersections.
- Consider the parametric definition of a line:
  - $x = x_b + t(x_e - x_b)$
  - $y = y_b + t(y_e - y_b)$
- What if we could find the range for t in which both x and y are inside the viewport?
  - $x_{min} \leq x_b + t(x_e - x_b) \leq x_{max}$
  - $y_{min} \leq y_b + t(y_e - y_b) \leq y_{max}$
- Rearranging, we get
  - $-t(x_e - x_b) \leq (x_b - x_{min})$ (left)
  - $t(x_e - x_b) \leq (x_{max} - x_b)$ (right)
  - $-t(y_e - y_b) \leq (y_b - y_{min})$ (bottom)
  - $t(y_e - y_b) \leq (y_{max} - y_b)$ (top)
- In general:
  - $tp_i \leq q_i, \quad i = 1,2,3,4$



Cases:

- $p_i = 0$
  - Line is parallel to a clipping window edge
    - if for the same i, $q_i < 0$, line is completely outside ⇒ **reject**
    - else, **accept**
- $u = {}^{q_i}/_{p_i}$ gives the intersection point
- $p_i < 0$
  - Line starts outside the clip window and goes inside
    - $u_1 = \max(0, {}^{q_i}/_{p_i})$
- $p_i > 0$
  - Line starts inside the clip window and goes outside
    - $u_2 = \min(1, {}^{q_i}/_{p_i})$
- If $u_1 > u_2$, line is completely outside ⇒ **reject**
- else, **accept**

Cohen-Sutherland, Cyrus-Beck, and Liang-Barsky algorithms readily extend to 3D

Cohen-Sutherland algorithm

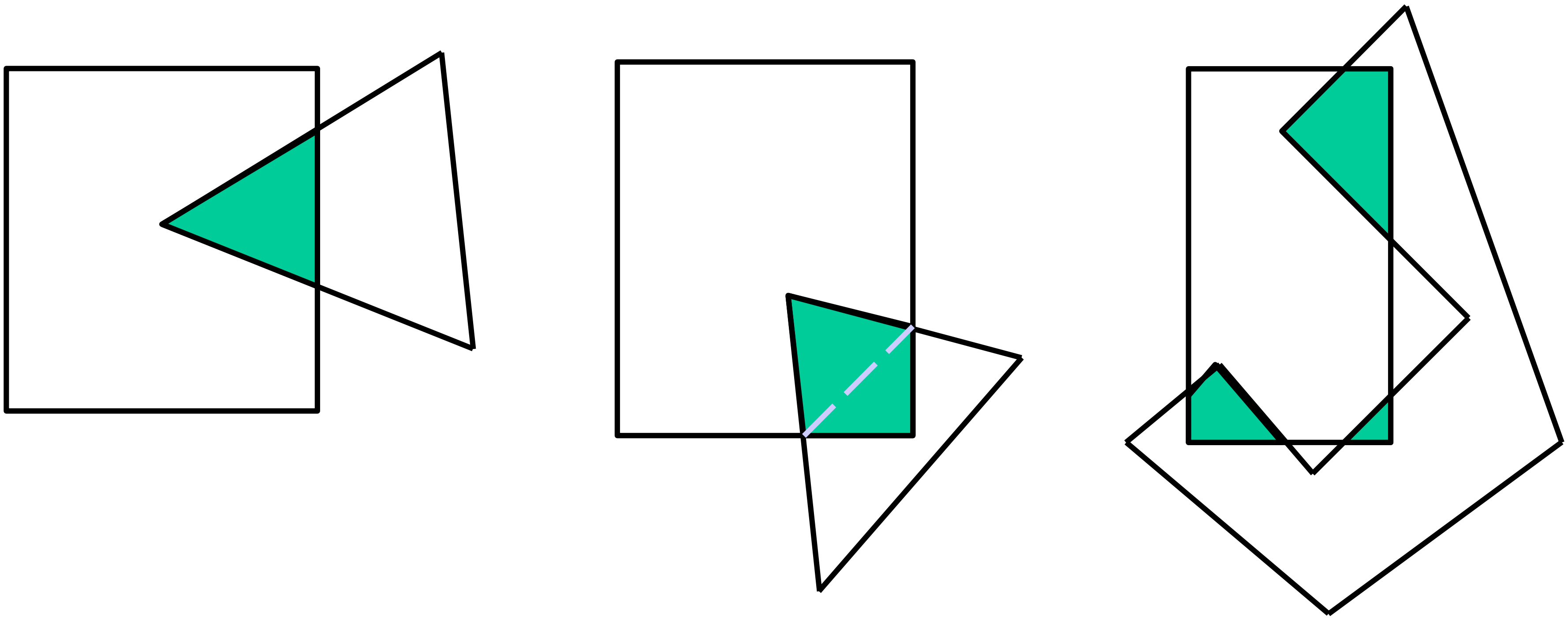
+ Efficient when majority of lines can be trivially accepted / rejected
    - Very large clip rectangles: almost all lines inside
    - Very small clip rectangles: almost all lines outside
- Repeated clipping for remaining lines
- Testing for 2D/3D point coordinates

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)

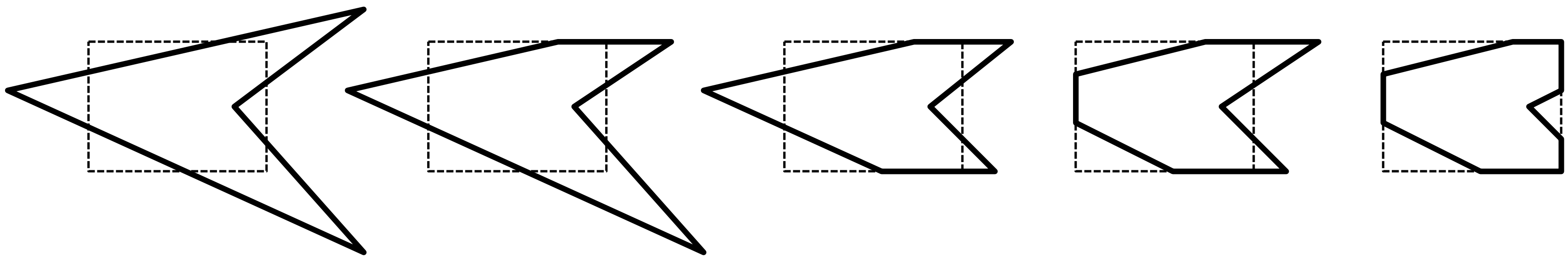
Extended version of line clipping

- Condition: polygons have to remain closed
  - Filling, hatching, shading, ...

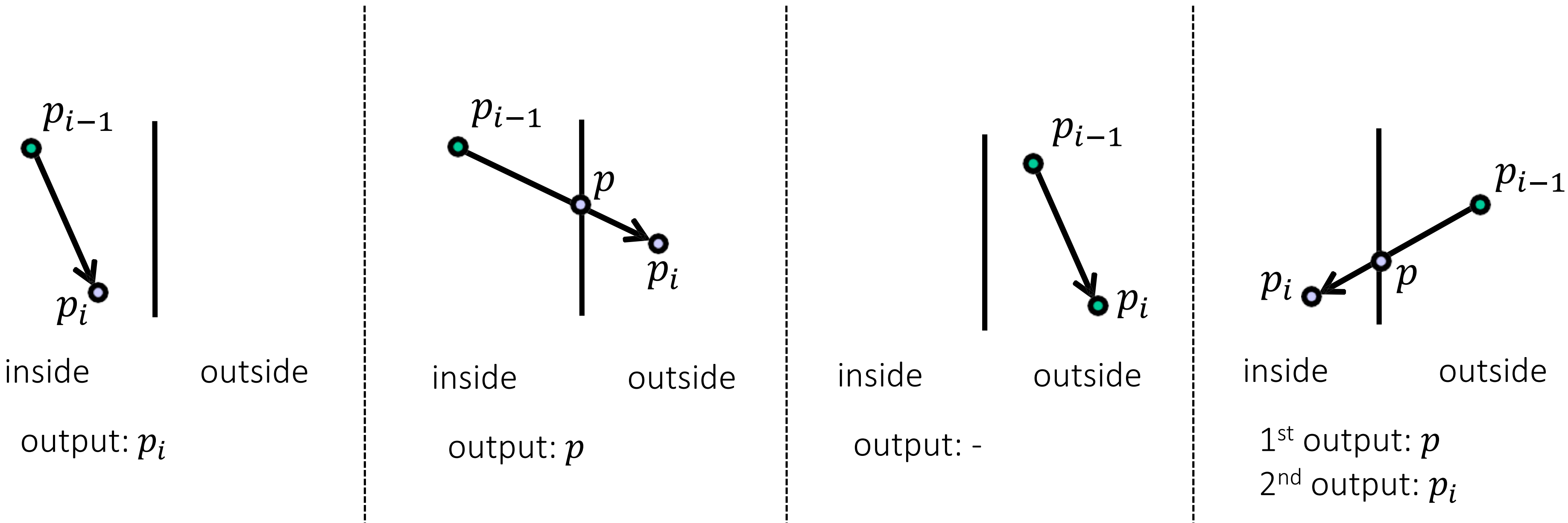


Idea

- Iterative clipping against each edge in sequence



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



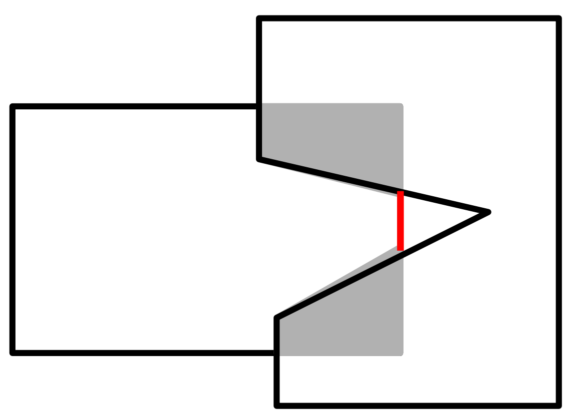
Recursive polygon clipping

- Pipelined Sutherland-Hodgeman

$p_0, p_1, \ldots$ → Top → Bottom → Left → Right → $p_0, p_1, \ldots$

Problems

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



Weiler & Atherton (´77)

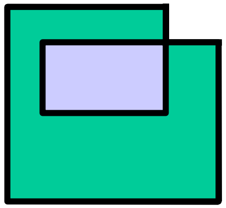
- Arbitrary concave polygons with holes against each other

Vatti (´92)

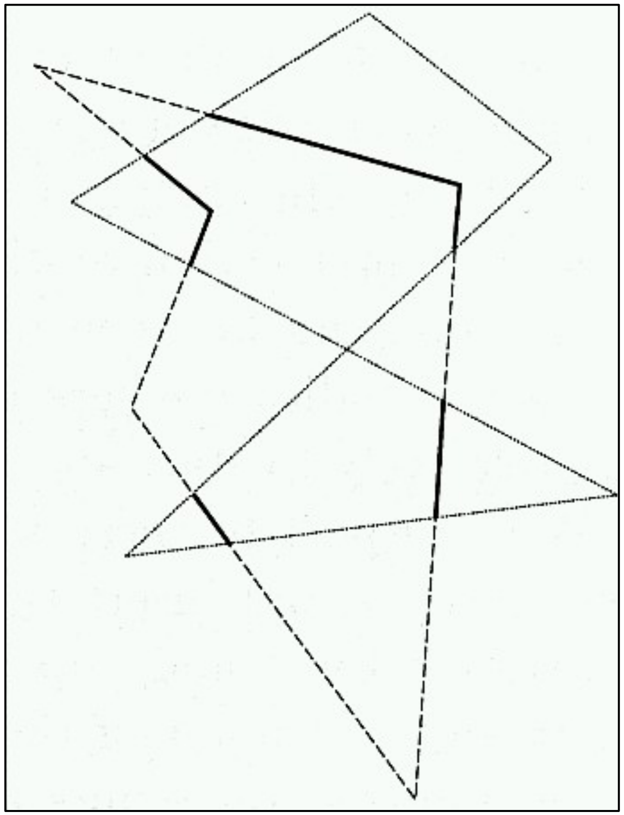
- Also with self-overlap

Greiner & Hormann (´98)

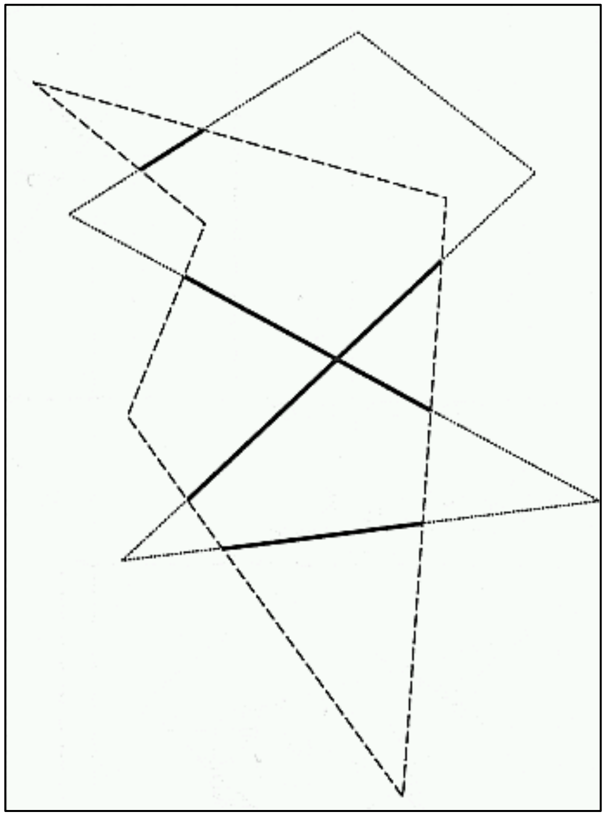
- Simpler and faster as Vatti
- Also supports Boolean operations
- Idea:
  - Odd winding number rule
    - Intersection with the polygon leads to a winding number ± 1
  - Walk along both polygons
  - Alternate winding number value
  - Mark point of entry and point of exit
  - Combine results



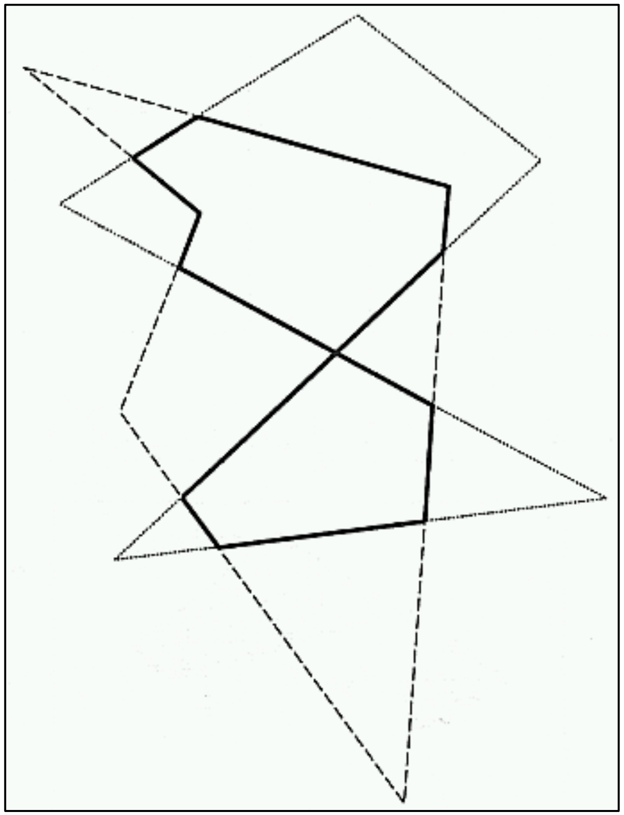
Non-zero WN: in
Even WN: out



A in B



B in A



(A in B) U (B in A)

Requirements

- Avoid unnecessary rasterization
- Avoid overflow on transformation at fixed point!

Clipping against viewing frustum

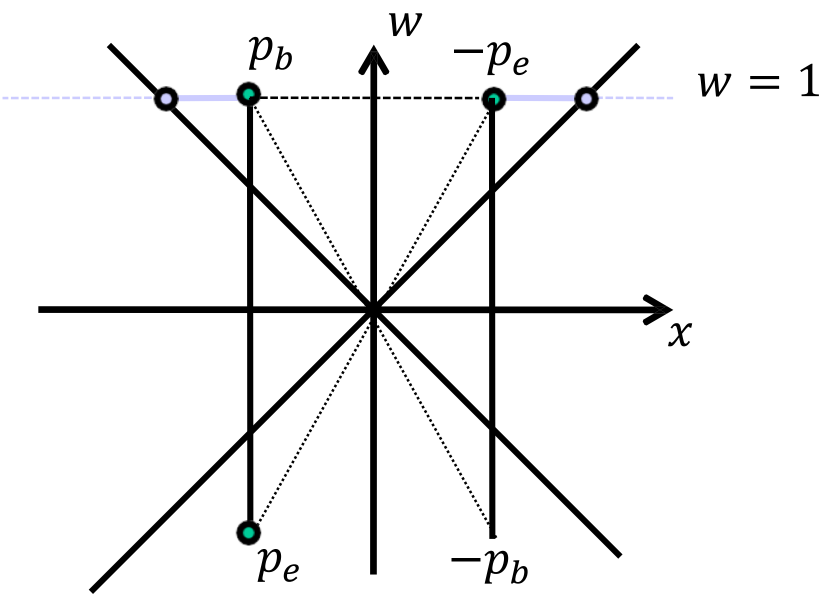
- Enhanced Cohen-Sutherland with 6-bit outcode
- After perspective division
  - $-1 < y < 1$
  - $-1 < x < 1$
  - $-1 < z < 0$
- Clip against side planes of the canonical viewing frustum
- Works analogously with Liang - Barsky or Sutherland - Hodgeman

Clipping in homogeneous coordinates

- Use canonical view frustum, but avoid costly division by W
- Inside test with a linear distance function (WEC)
  - Left: $^X/_W > -1 \longrightarrow W + X = WEC_L(p) > 0$
  - Top: $^Y/_W < 1 \longrightarrow W - Y = WEC_T(p) > 0$
  - Back: $^Z/_W > -1 \longrightarrow W + Z = WEC_B(p) > 0$
  - …

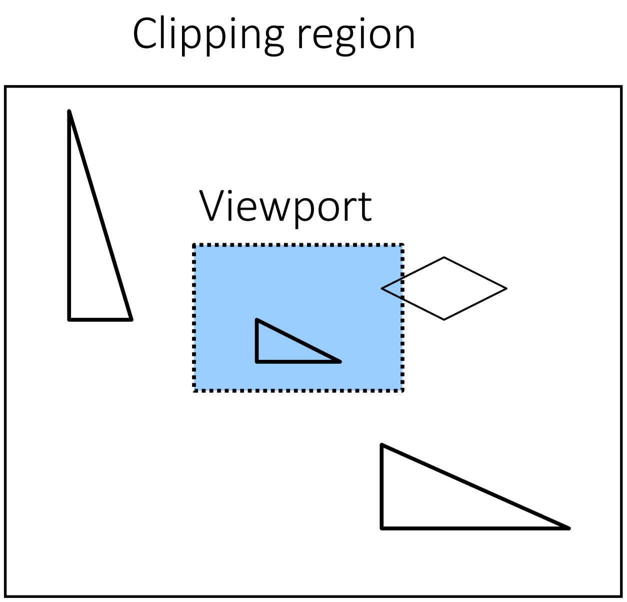
Negative W

- 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



Combining clipping and scissoring

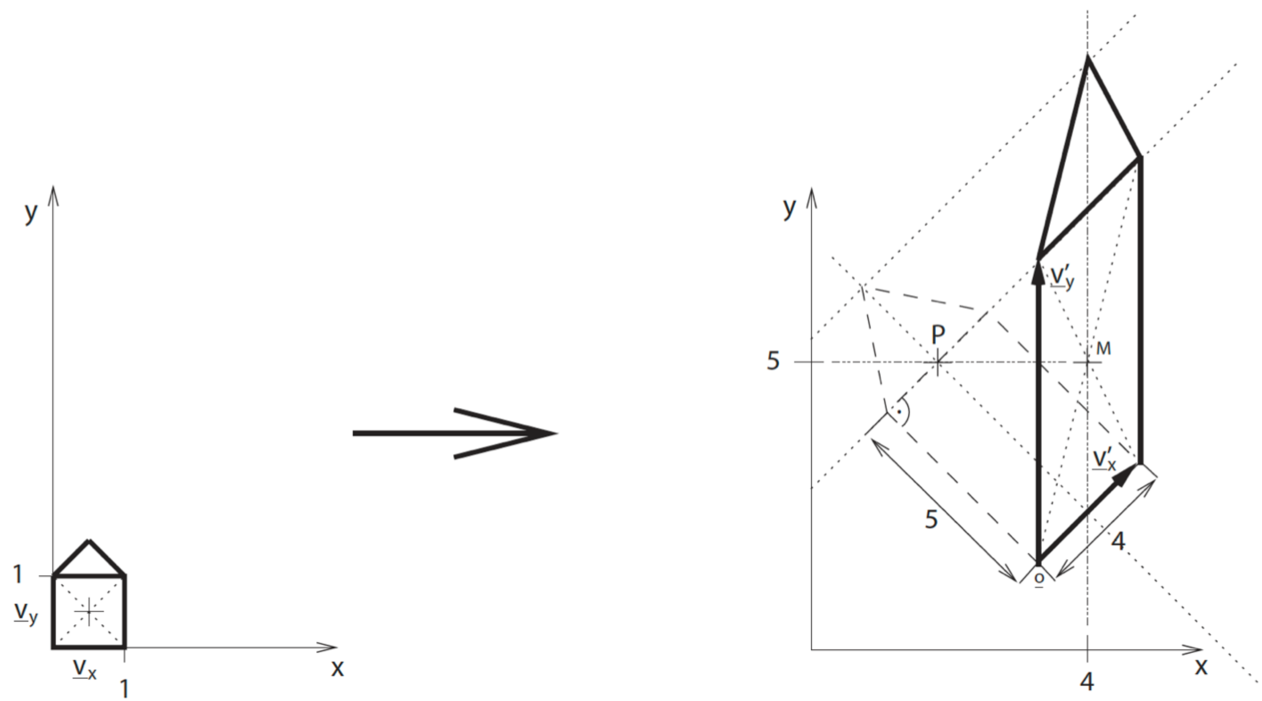
- Clipping is expensive and should be avoided
  - Intersection calculation
  - Variable number of new points, new triangles
- Enlargement of clipping region
  - (Much) larger than viewport, but
  - Still avoiding overflow due to fixed-point representation
- Result
  - Less clipping
  - Applications should avoid drawing objects that are outside of the viewport / viewing frustum
  - Objects that are partially outside will be implicitly clipped during rasterization
  - Slight penalty because they will still be processed (triangle setup)



Submission deadline: Friday, 29. November 2019 9:45 (before the lecture)
Written solutions have to be submitted in the lecture room before the lecture. Every assignment sheets counts 100 points (theory and practice)

6.1 Transformations (50 Points)

In the picture below the left house should be transformed into the house on the right. The point M is at (4, 5) and lines that look to be parallel are parallel. Please specify the complete transformation matrix as a sequence of primitive transformations (there's no need to calculate the final matrix). Do not guess any numbers.



6.2 Affine Spaces (20 Points)

Prove that the set of points $A = \{(x, y, z, w) \in \mathbb{R}^4 \mid w = 1\}$ is an affine space. What is the associated vector space? You do *not* have to show that the associated vector space is a vector space. What is the difference between a point and a vector in that affine space?

Definition of an affine space: An affine space consists of a set of points P, an associated vector space V and an operation $+ \in P \times V \to P$ that fulfills the following axioms:

1) For $p \in P$ and $v, w \in V$: $(p + v) + w = p + (v + w)$
2) for $p, q \in P$ there exists a unique $v \in V$ such that: $p + v = q$

6.3 Rotations (30 Points)

Show that an arbitrary rotation around the origin in 2D can be represented by a combination of a shearing in y, a scaling in x and y and a shearing in x in this order. You have to derive the shearing and scaling matrices to an arbitrary rotation T.

$$T(x) = \begin{pmatrix} \cos\phi & \sin\phi \\ -\sin\phi & \cos\phi \end{pmatrix}$$