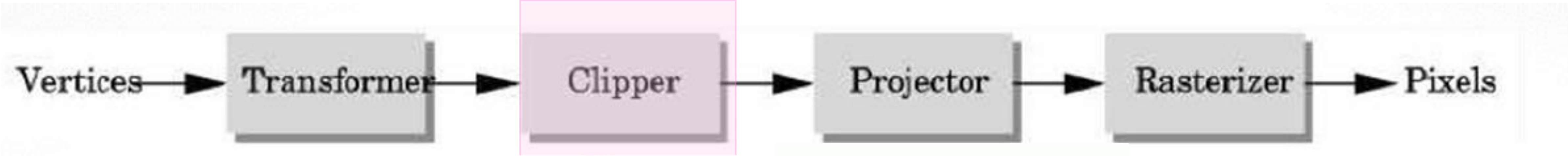
Computer Graphics
- Line & Polygon Clipping

Junjie Cao @ DLUT

Spring 2018

http://jjcao.github.io/ComputerGraphics/

The Graphics Pipeline, Revisited



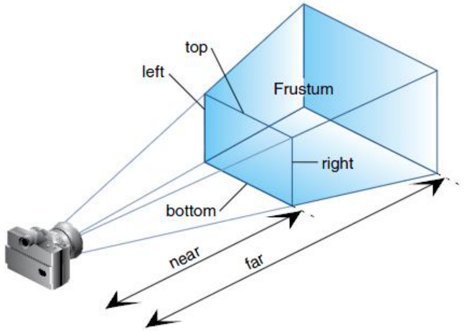
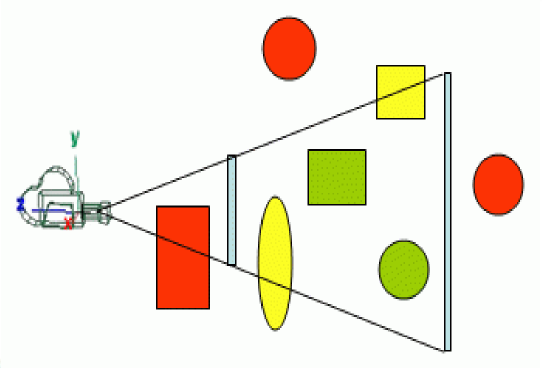


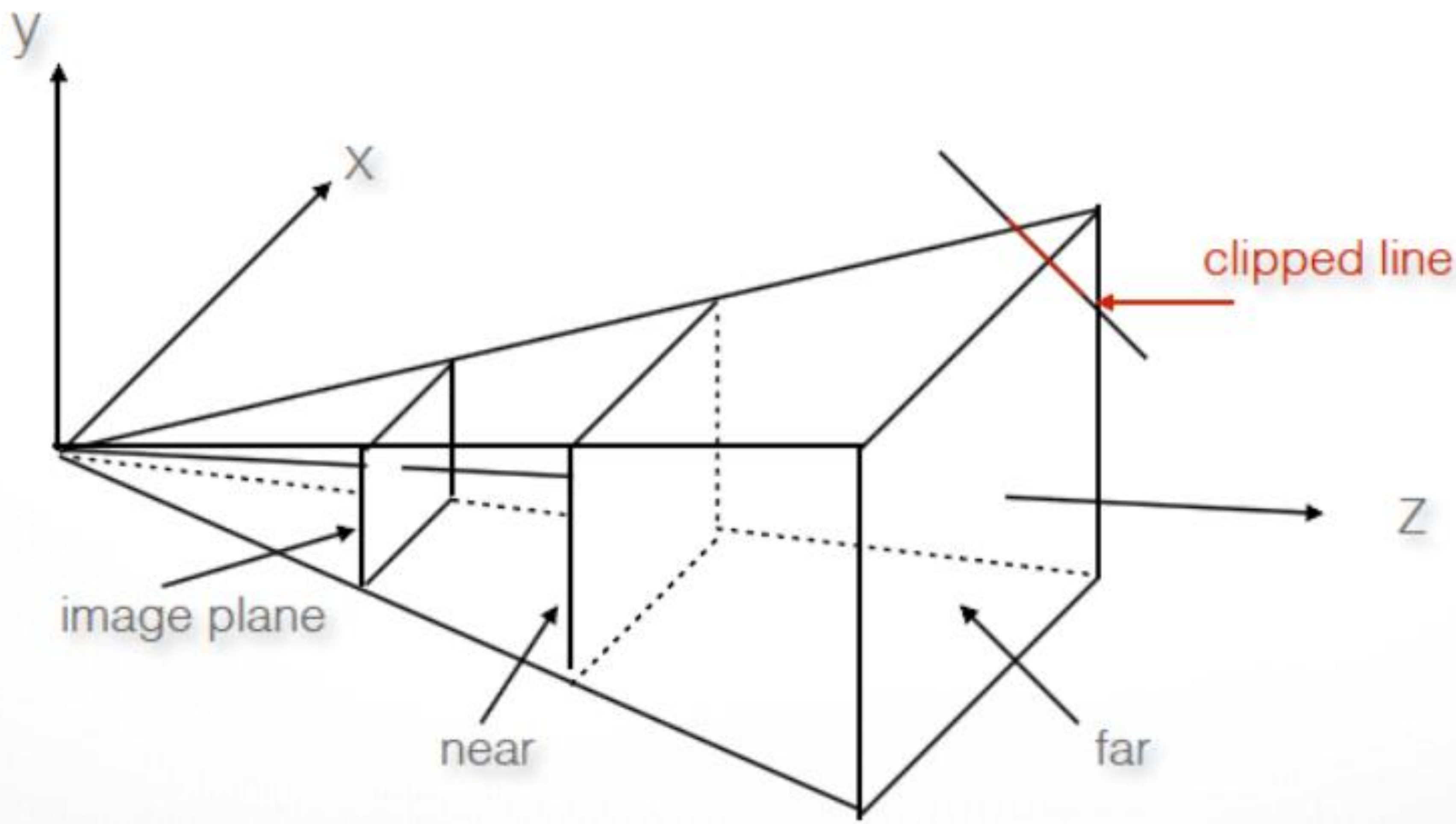
Figure 3-13 Perspective Viewing Volume Specified by glFrustum()



- Must eliminate objects that are outside of viewing frustum
- Clipping: object space (eye coordinates)
- Scissoring: image space (pixels in frame buffer)
  - most often less efficient than clipping
- We will first discuss 2D clipping (for simplicity)
  - OpenGL uses 3D clipping

Clipping Against a Frustum

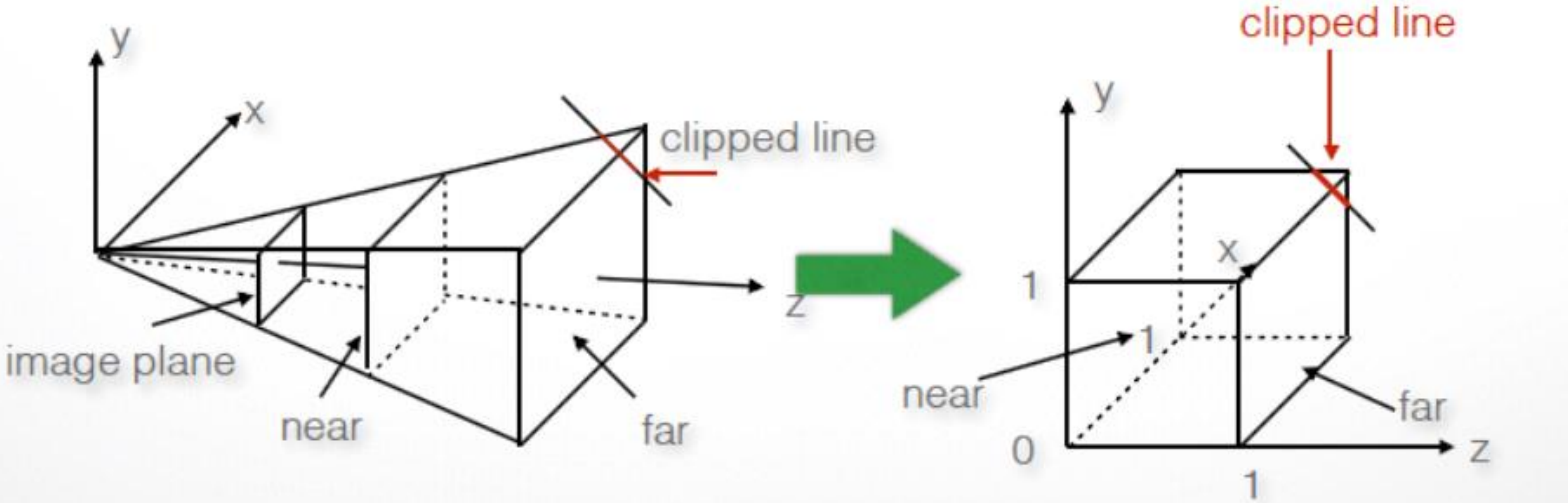
- General case of frustum (truncated pyramid)



- Clipping is tricky because of frustum shape

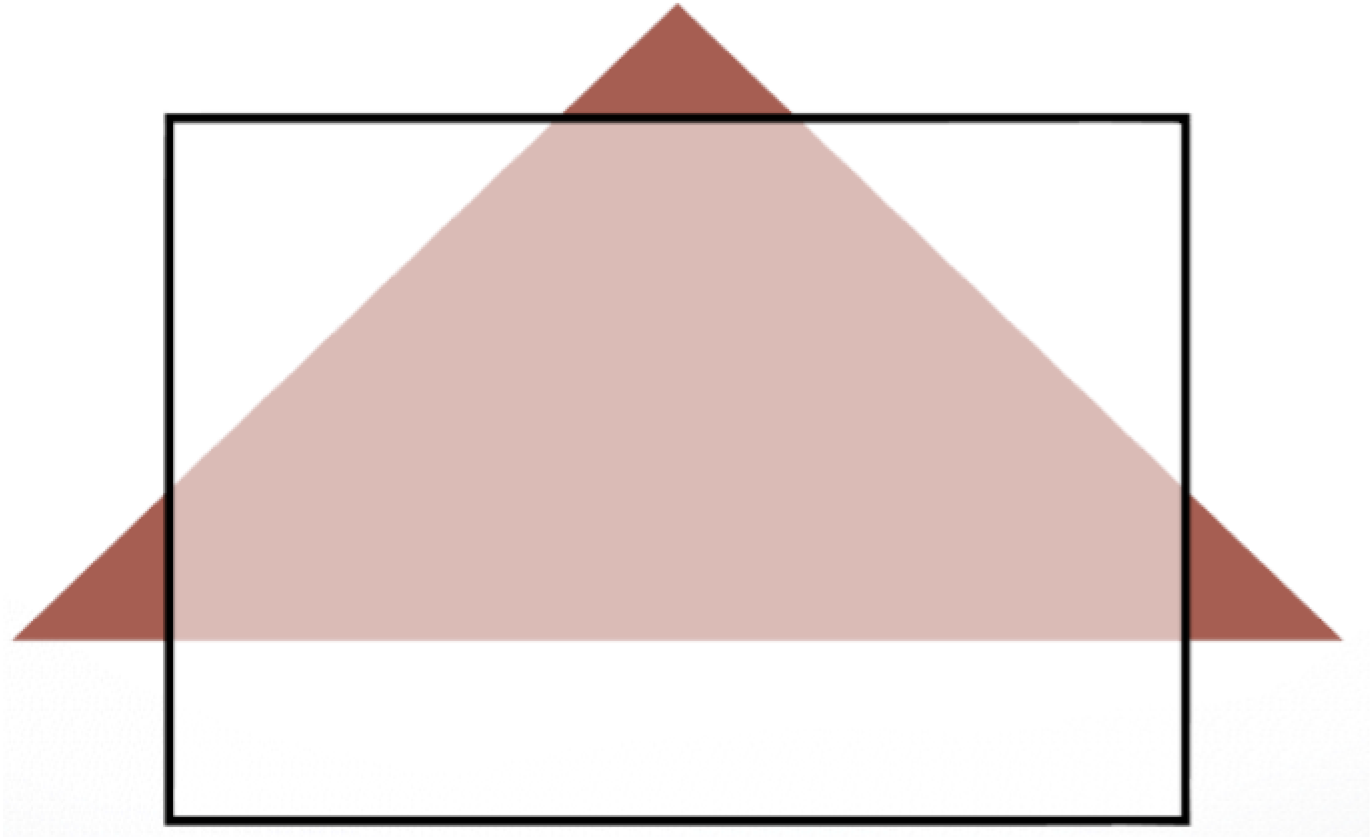
Perspective Normalization

- Solution: **Clip against resulting cube**
  - Implement perspective projection by perspective normalization and orthographic projection
  - Perspective normalization is a homogeneous transformation



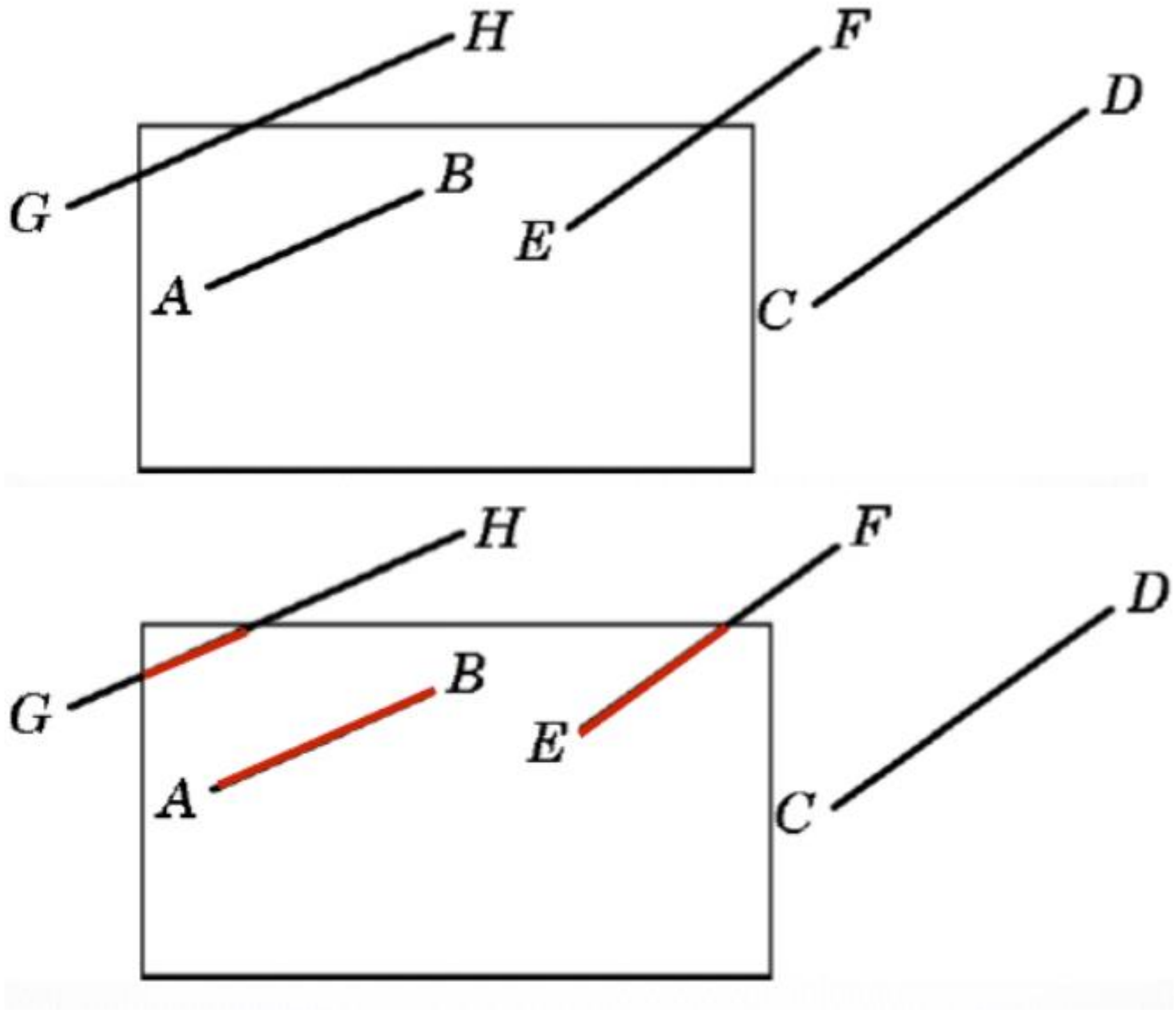
OpenGL uses -1 ≤ x,y,z ≤ 1 (others possible)

2D Clipping Problem



Clipping Against Rectangle in 2D

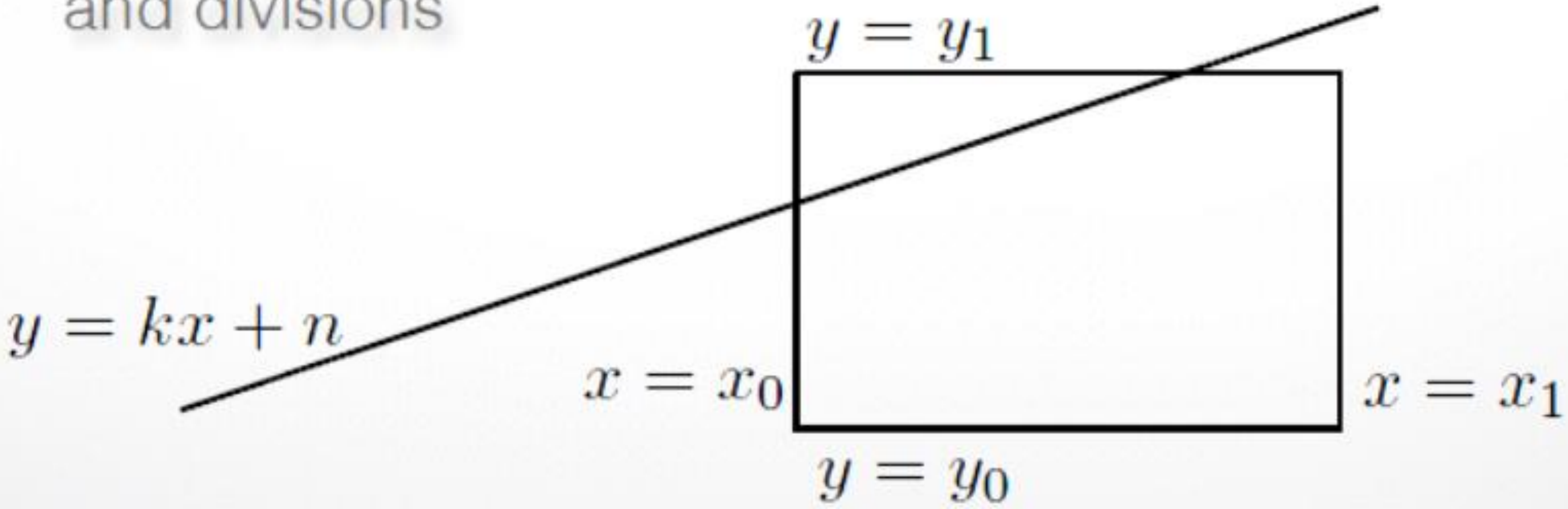
- Line-segment clipping: modify endpoints of lines to lie within clipping rectangle
- The result (in red)



Clipping Against Rectangle in 2D

- Could calculate intersections of line segments with clipping rectangle
  - expensive, due to floating point multiplications and divisions

- Want to minimize the number of multiplications and divisions



Several practical algorithms for clipping

- Main motivation:
  - Avoid expensive line-rectangle intersections (which require floating point divisions)

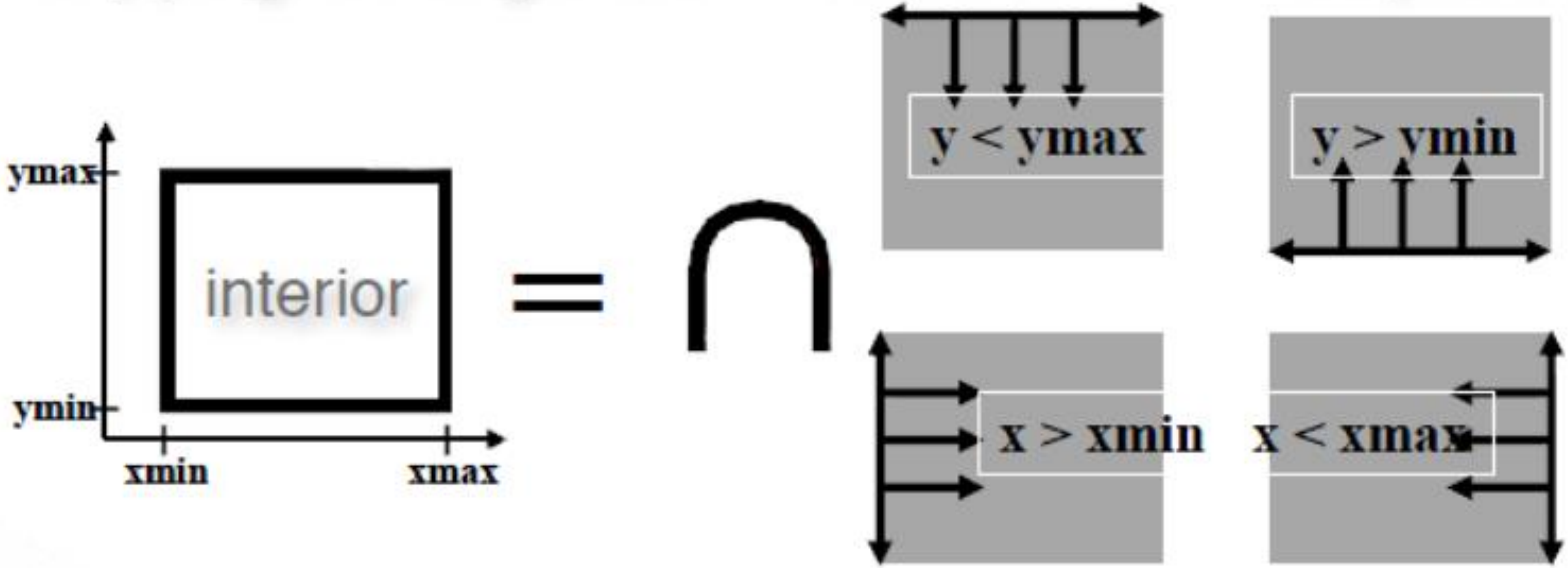
- Cohen-Sutherland Clipping (1967)

- Liang-Barsky Clipping

- There are many more (but many only work in 2D)

Cohen-Sutherland Clipping – 1967 flight simulation

- Clipping rectangle is an intersection of 4 half-planes

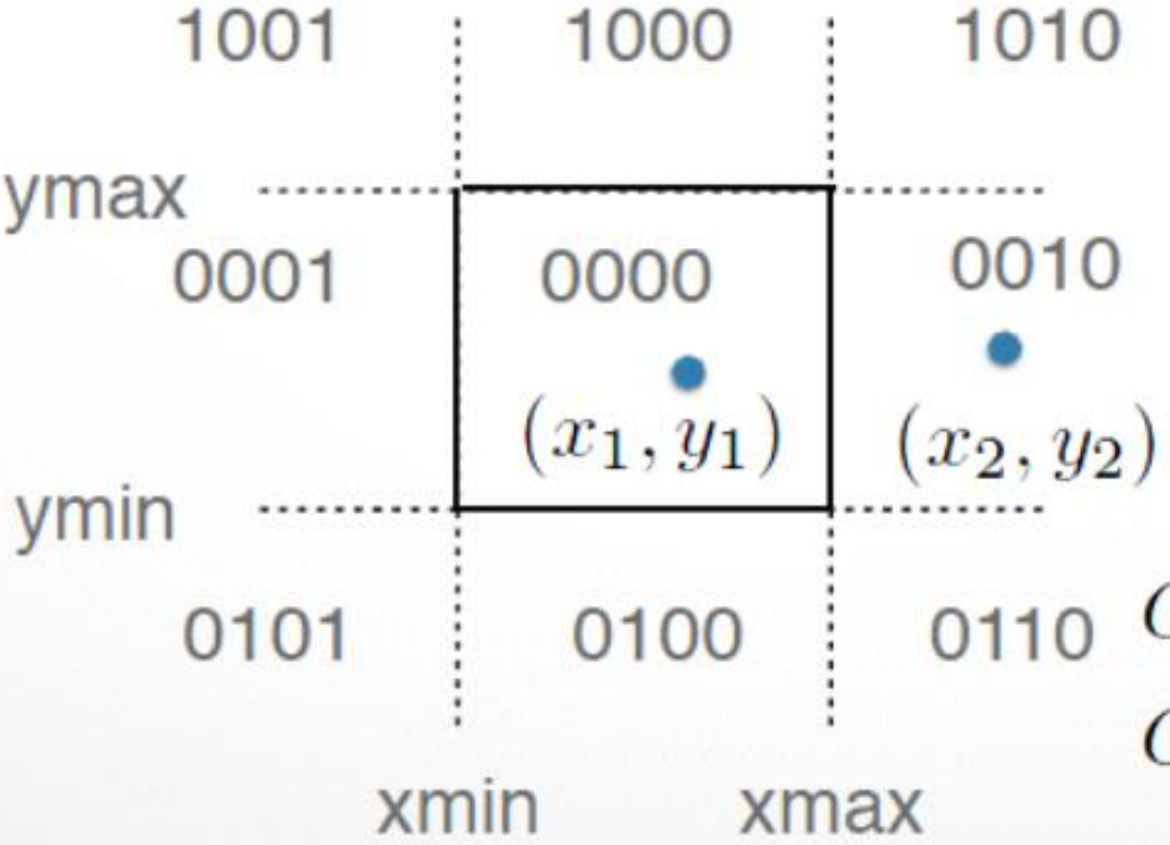


- Encode results of four half-plane tests

- Generalizes to 3 dimensions (6 half-planes)

Outcodes (Cohen-Sutherland)

- Divide space into 9 regions
- 4-bit outcode determined by comparisons (TBRL)



$$b_0 : y > y_{max}$$

$$b_1 : y < y_{min}$$

$$b_2 : x > x_{max}$$

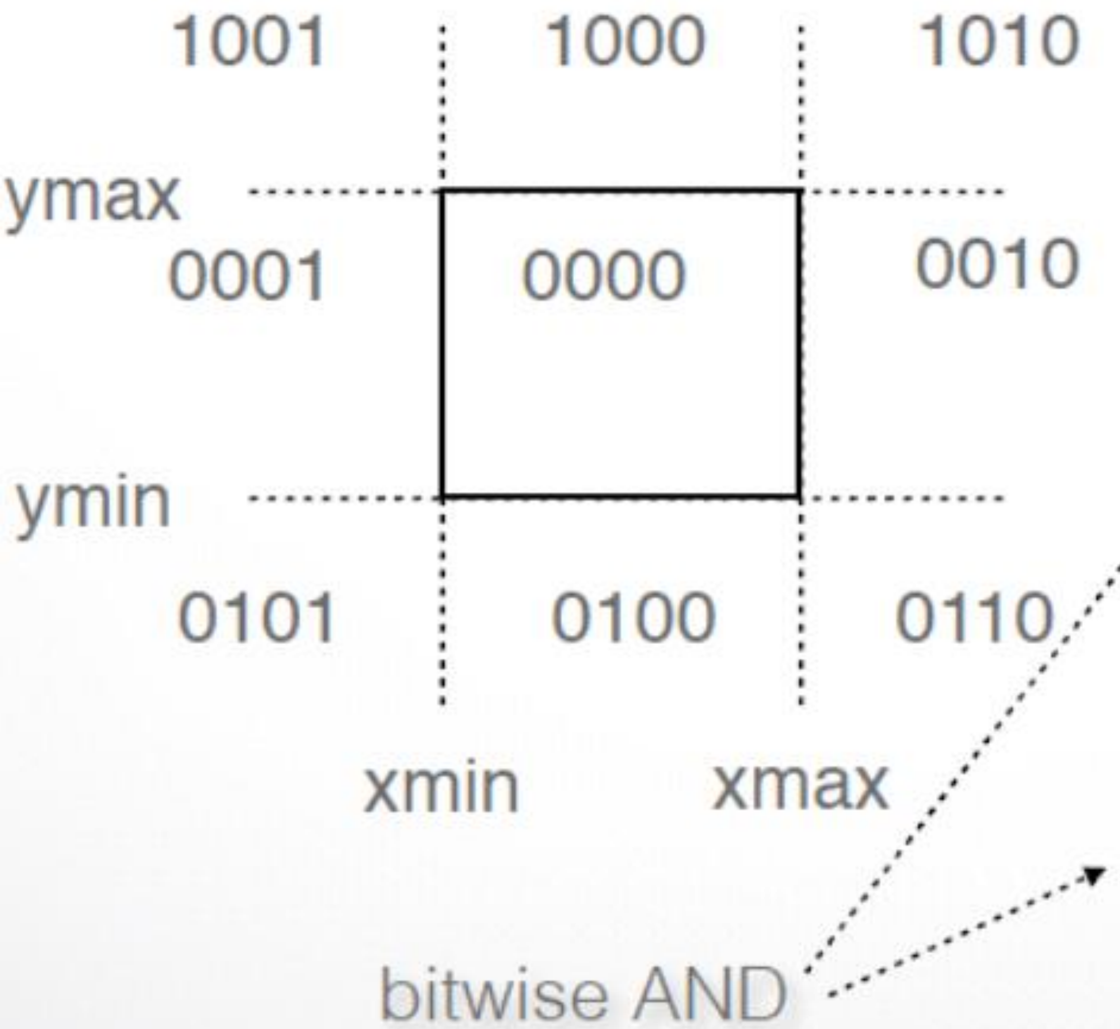
$$b_3 : x < x_{min}$$

$$O_1 = outcode(x_1, y_1)$$

$$O_1 = outcode(x_2, y_2)$$

Cases for Outcodes

- Outcomes: accept, reject, subdivide



$O_1 = O_2 = 0000$: accept entire segment

$O_1 \,\&\, O_2 \neq 0000$: reject entire segment

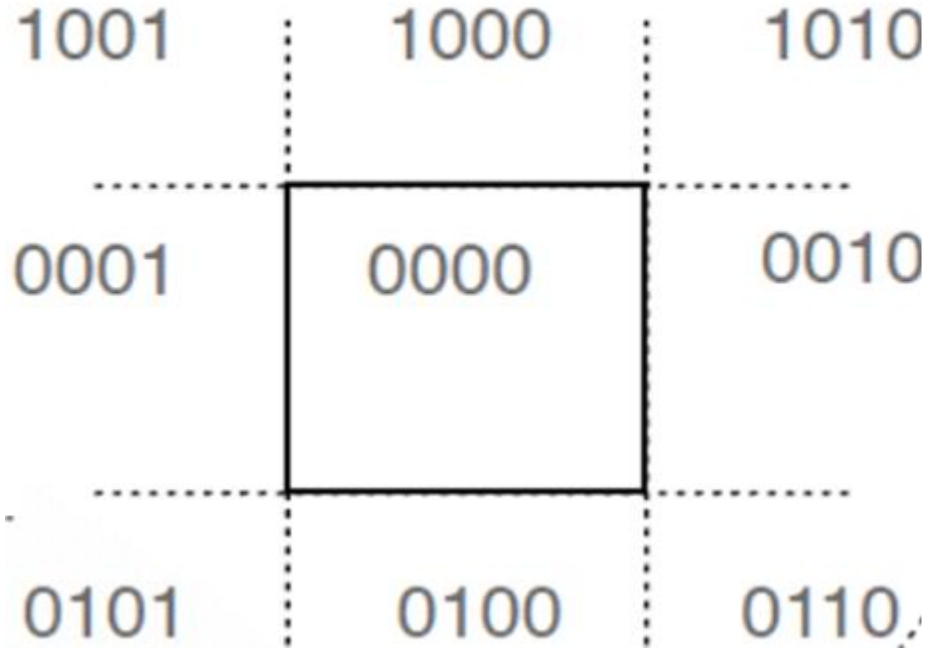
$O_1 = 0000$, $O_2 \neq 0000$: subdivide

$O_1 \neq 0000$, $O_2 = 0000$: subdivide

$O_1 \,\&\, O_2 = 0000$: subdivide

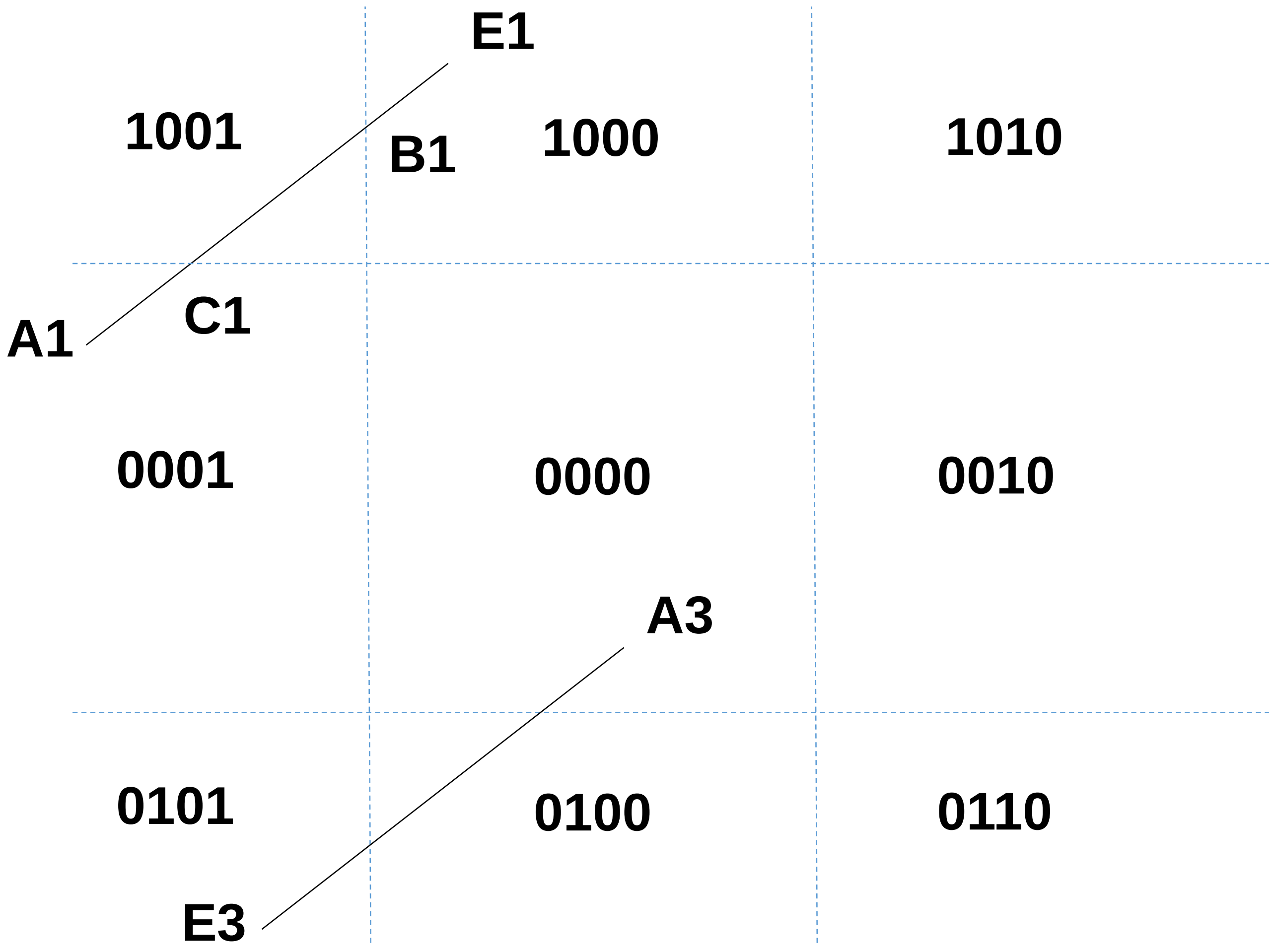
Cohen-Sutherland Subdivision

- Pick outside endpoint (o ≠ 0000)
- Pick a crossed edge (o = b0b1b2b3 and bk ≠ 0)
- Compute intersection of this line and this edge
- Replace endpoint with intersection point
- Restart with new line segment
  - Outcodes of second point are unchanged
- This algorithms converges



TBRL

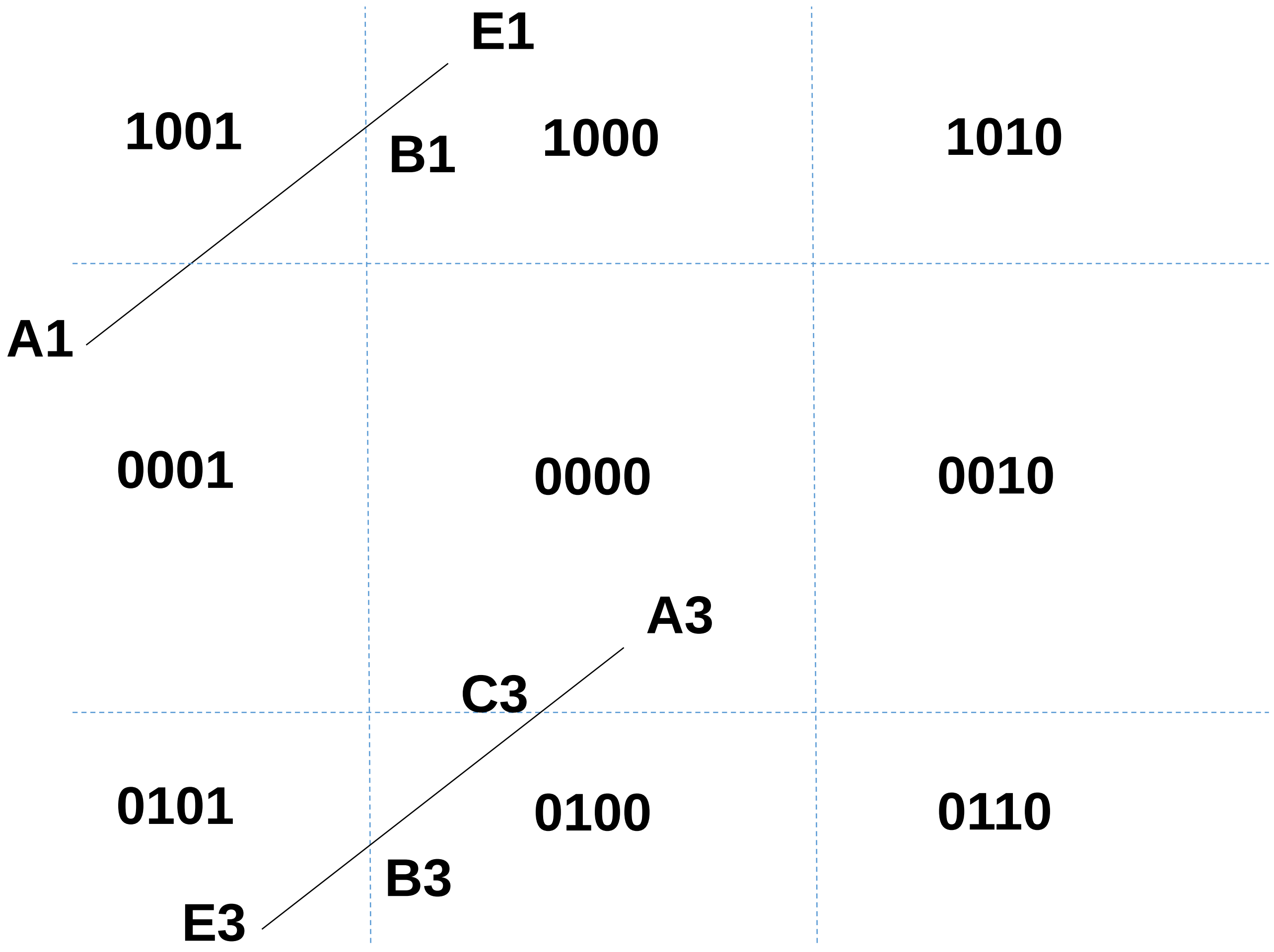
Cohen-Sutherland Line-Clipping (example)



a) A1E1
b) B1E1
c) Reject

d) A1E1
e) E1C1
f) Reject

Cohen-Sutherland Line-Clipping (example)



a) A1E1
b) B1E1
c) Reject

d) A3E3
e) A3B3 (0,1,0,0)
f) A3C3
g) accept

a) A3E3
b) A3C3
c) accept

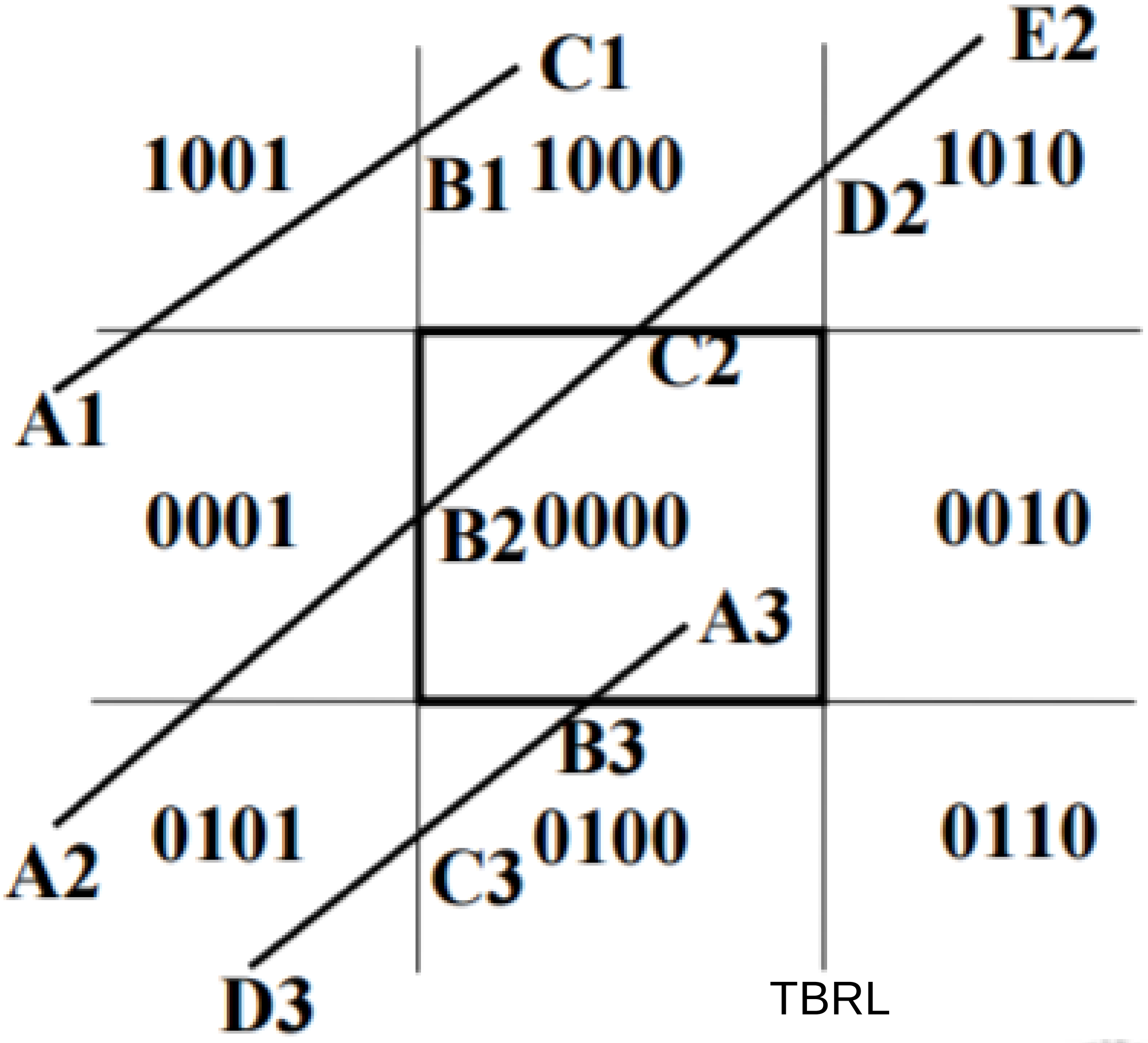
Cohen-Sutherland Line-Clipping

Clip order: Left, Right, Bottom, Top

1) A1C1
2) B1C1
3) reject

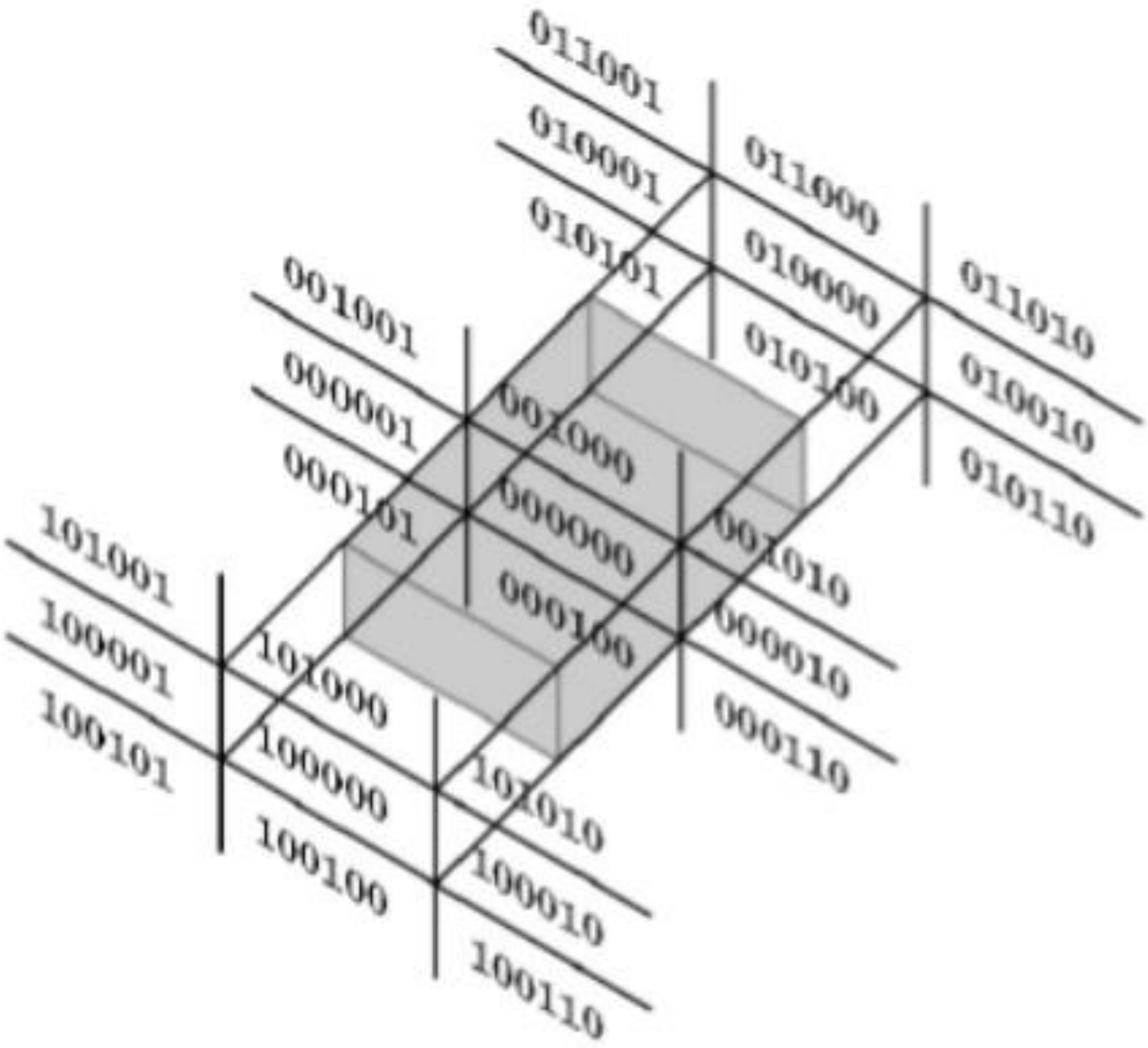
1) A2E2
2) B2E2
3) B2D2
4) B2C2
5) accept

1) A3D3
2) A3C3
3) A3B3
4) accept



Cohen-Sutherland in 3D

- Use 6 bits in outcode
  - b_4: $z > z_{max}$
  - b_5: $z < z_{min}$
- Other calculations as before



Cohen-Sutherland Line-Clipping

- Advantages:
  - Simple to program
  - Extends to 3-D cubic volumes
  - Efficient when most of the lines to be clipped are either rejected or accepted (not so many subdivisions).

- Disadvantages:
  - Fixed-order decision can do needless work
  - Can improve using more regions (Nicholl-Lee-Nicholl Alg)

- **Parametric clipping are more efficient.**

Liang-Barsky Clipping

- **You-Dong Liang** (梁友栋) is a mathematician and educator, best known for his contributions in geometric modeling and the Liang-Barsky algorithm.
- Born on July 19, 1935
- Design Liang-Barsky clipping when he is 49

- Liang, Y.D., and Barsky, B., "A New Concept and Method for Line Clipping", **ACM Transactions on Graphics**, 3(1):1-22, January **1984**.
- Liang, Y.D., B.A., Barsky, and M. Slater, Some Improvements to a Parametric Line Clipping Algorithm, CSD-92-688, Computer Science Division, University of California, Berkeley, 1992.
- Fu-Chung Huang, Gordon Wetzstein, Brian A. Barsky, and Ramesh Raskar. "**Eyeglasses-free Display: Towards Correcting Visual Aberrations with Computational Light Field Displays**", *Proceedings of ACM SIGGRAPH 2014*, Vancouver, 10-14 August 2014, *ACM Transactions on Graphics (TOG)*, Volume 33, Issue 4, July 2014, Article No. 59.

Parametric form - **Liang-Barsky Clipping**

- A line segment with endpoints

  (x_0, y_0) and (x_{end}, y_{end})

  we can describe in the parametric form

$$x = x_0 + u\Delta x$$
$$y = x_0 + u\Delta y \qquad\qquad 0 \le u \le 1$$

  where

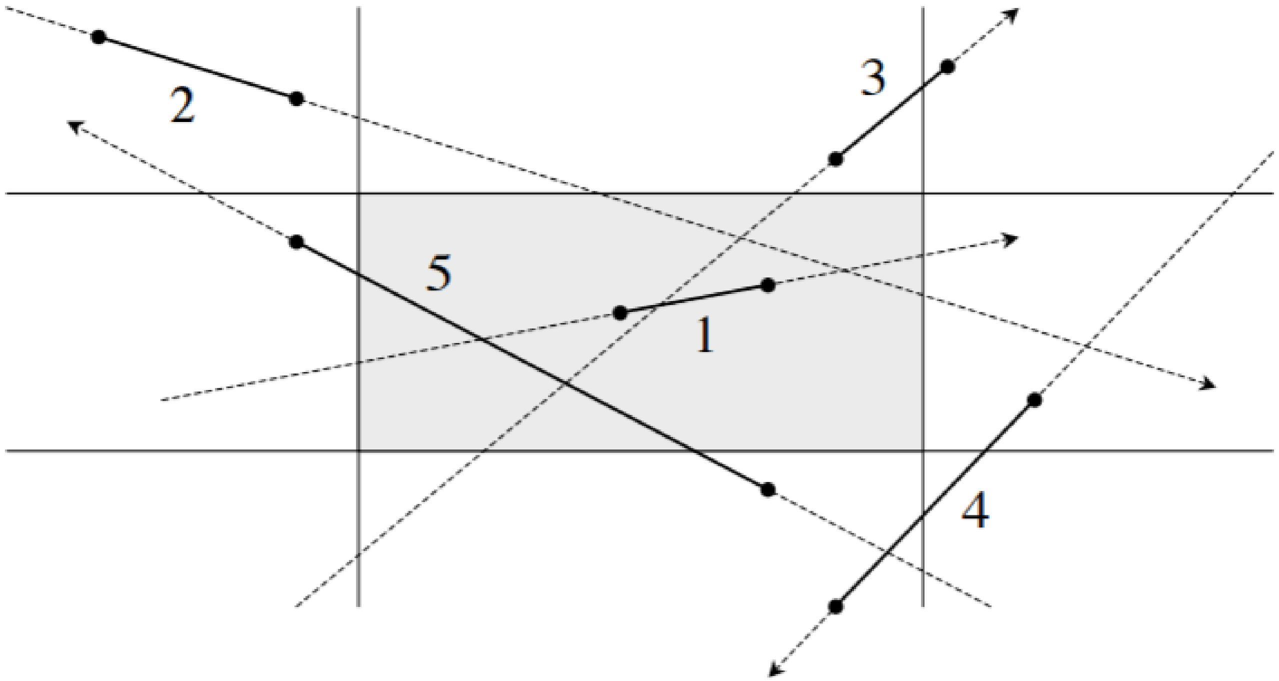
$$\Delta x = x_{end} - x_0$$
$$\Delta y = y_{end} - y_0$$

Liang-Barsky Clipping

- Liang-Barsky asks: for what values of u does a line segmen enter or exit the **bounds**?

- There can be, at most, two of each; we care about the **maximum entry** value and the **minimum exit** value

$$x = x_0 + u\Delta x$$
$$y = x_0 + u\Delta y$$



Line 1: max entry < 0, min exit > 1 — *accept*
Line 2: max entry > 1, min exit > 1 — *reject*
Line 3: max entry < 0, min exit < 0 — *reject*
Line 4: max entry $>$ min exit — *reject*
Line 5: max entry > 0, min exit < 1, max entry $<$ min exit — *clip*

Liang-Barsky Clipping

- A line is inside clipping region for values of u such that:

$$xw_{\min} \le x_0 + u\Delta x \le xw_{\max} \qquad \Delta x = x_{\text{end}} - x_0$$
$$yw_{\min} \le y_0 + u\Delta y \le yw_{\max} \qquad \Delta y = y_{\text{end}} - y_0$$

- Can be described as

$$u\,p_{\mathrm{k}} \le q_{\mathrm{k}}, \qquad \mathrm{k} = 1, 2, 3, 4$$

- The infinitely line intersect the clip region edges when

$$u_k = \frac{q_k}{p_k} \text{ where } \begin{array}{lll} p_1 = -\Delta x & q_1 = x_0 - xw_{\min} & \text{Left boundary} \\ p_2 = \Delta x & q_2 = xw_{\max} - x_0 & \text{Right boundary} \\ p_3 = -\Delta y & q_3 = y_0 - yw_{\min} & \text{Bottom boundary} \\ p_4 = \Delta y & q_4 = yw_{\max} - y_0 & \text{Top boundary} \end{array}$$

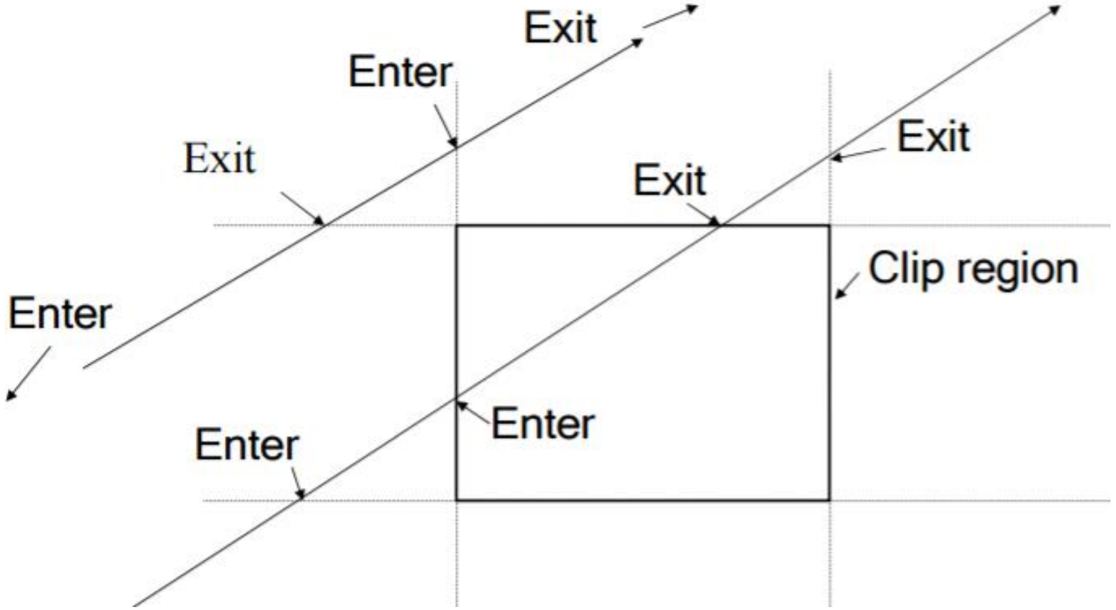
Liang-Barsky Clipping

- Liang-Barsky asks: for what values of u does a line segment enter or exit the bounds?
- **How to measure enter or exit via pk, qk instead of u?**

$\Delta \mathrm{x} = x_{end} - x_0$

$p_1 = -\Delta x$	$q_1 = x_0 - xw_{\min}$	Left boundary
$p_2 = \Delta x$	$q_2 = xw_{\max} - x_0$	Right boundary
$p_3 = -\Delta y$	$q_3 = y_0 - yw_{\min}$	Bottom boundary
$p_4 = \Delta y$	$q_4 = yw_{\max} - y_0$	Top boundary

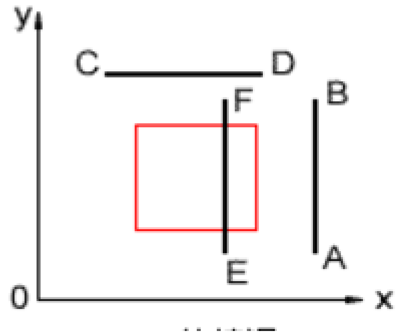
- When $\mathrm{p_k} < 0$, as u increases
  - line goes from outside to inside - entering
- When $\mathrm{p_k} > 0$,
  - line goes from inside to outside - exiting
- When $\mathrm{p_k} = 0$,
  - line is parallel to an edge



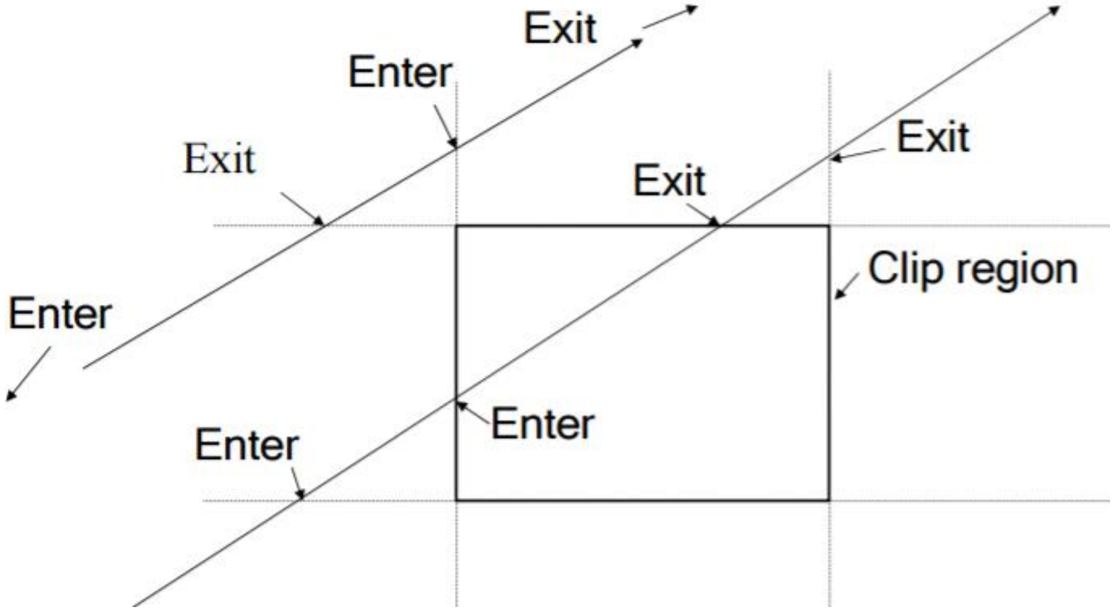
If there is a segment of the line inside the clip region, a sequence of infinite line intersections must go:
Entering, entering, exiting, exiting.

Conditions

- If (pk =0) – parallel line to one of the clipping edge
  - (qk <0) – outside, **reject**
  - (qk ≥0) – inside



- If (pk <0) – from outside to inside
  - Intersection at: t_1 = qk/pk (OIP)
- If (pk >0) – from inside to outside
  - Intersection at: t_2 = qk/pk (IOP)

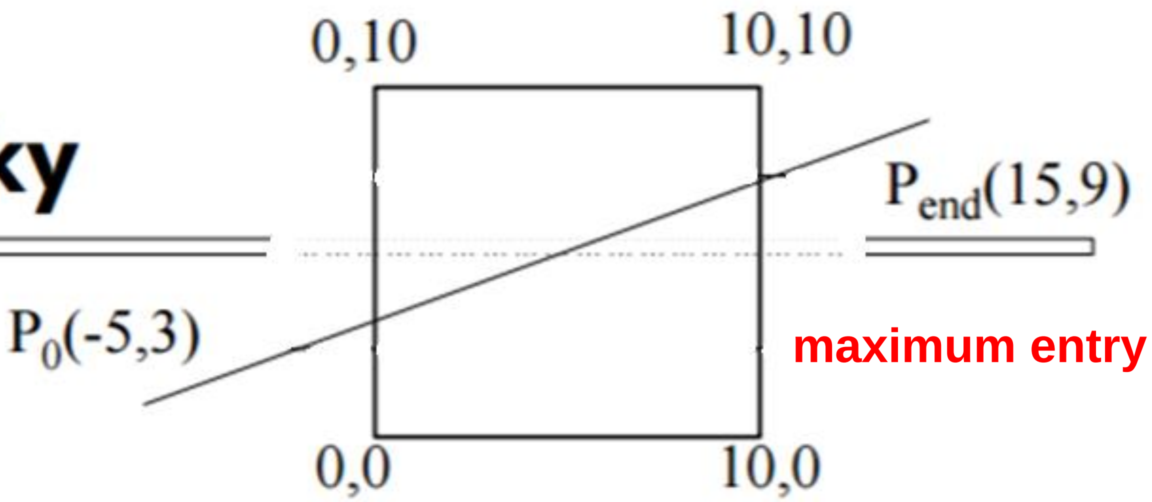


- Key Insight: we need to find two intersection points, one OIP, one IOP.

Intersection Point Computation

- For ($p_k < 0$) (Outside to inside point)
  - $r_k = q_k/p_k$
  - OIP $t_1 = \max(r_k, 0)$
- For ($p_k > 0$) (Inside to outside point)
  - $r_k = q_k/p_k$
  - IOP $t_2 = \min(r_k, 1)$
- If ($t_1 > t_2$), complete outside, REJECT
- Else clipped line is between (t_1, t_2)

Example Liang-Barsky



maximum entry value and the **minimum exit** value

$$u_{left} = \frac{q_1}{p_1} = \frac{x_0 - xw_{\min}}{-\Delta x} = \frac{-5-0}{-(15-(-5))} = \frac{1}{4}$$

Entering $\Rightarrow$ $u_{\min} = 1/4$

$$u_{right} = \frac{q_2}{p_2} = \frac{xw_{\max} - x_0}{\Delta x} = \frac{10-(-5)}{15-(-5)} = \frac{3}{4}$$

Exiting $\Rightarrow$ $u_{\max} = 3/4$

$$u_{bottom} = \frac{q_3}{p_3} = \frac{y_0 - yw_{\min}}{-\Delta y} = \frac{3-0}{-(9-3)} = -\frac{1}{2}$$

$u < 0$ then ignore

$$u_{top} = \frac{q_4}{p_4} = \frac{yw_{\max} - y_0}{\Delta y} = \frac{10-3}{9-3} = \frac{7}{6}$$

$u > 1$ then ignore

Liang-Barsky Clipping

- We have $u_{min} = 1/4$ and $u_{max} = 3/4$

$$P_{end} - P_0 = (\underbrace{15+5}_{\Delta x},\underbrace{9-3}_{\Delta y}) = (20,6)$$

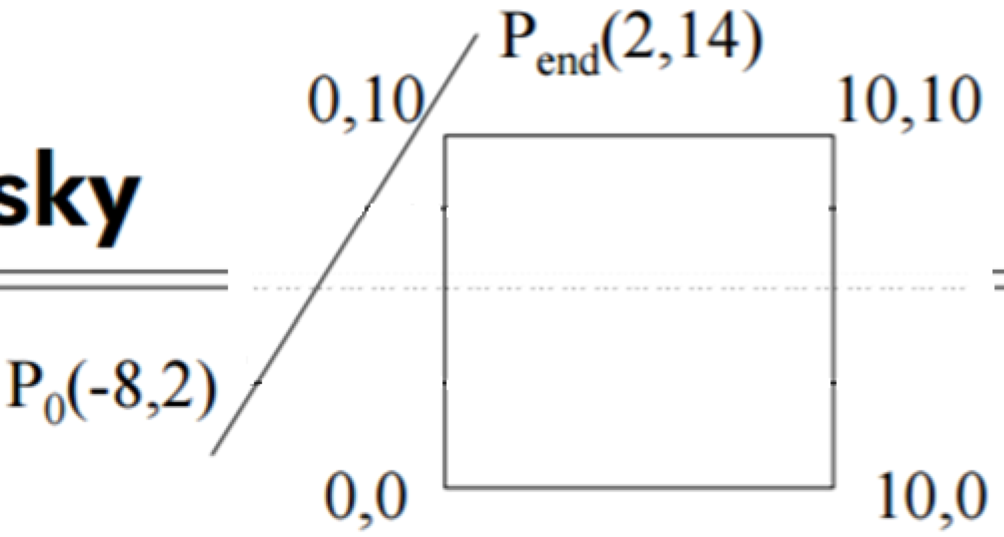
- If $u_{min} < u_{max}$, there is a line segment
  - compute endpoints by substituting u values
- Draw a line from

  $(-5+(20)\cdot(1/4),\ 3+(6)\cdot(1/4))$

  to

  $(-5+(20)\cdot(3/4),\ 3+(6)\cdot(3/4))$

Example Liang-Barsky



$$u_{left} = \frac{q_1}{p_1} = \frac{x_0 - xw_{\min}}{-\Delta x} = \frac{-8-0}{-(2-(-8))} = \frac{4}{5}$$

Entering $\Rightarrow$ $u_{\min} = 4/5$

$$u_{right} = \frac{q_2}{p_2} = \frac{xw_{\max} - x_0}{\Delta x} = \frac{10-(-8)}{2-(-8)} = \frac{9}{5}$$

$u > 1$ then ignore

$$u_{bottom} = \frac{q_3}{p_3} = \frac{y_0 - yw_{\min}}{-\Delta y} = \frac{2-0}{-(14-2)} = -\frac{1}{6}$$

$u < 0$ then ignore

$$u_{top} = \frac{q_4}{p_4} = \frac{yw_{\max} - y_0}{\Delta y} = \frac{10-2}{14-2} = \frac{2}{3}$$

Exiting $\Rightarrow$ $u_{\max} = 2/3$

Liang-Barsky Clipping

- We have $u_{min} = 4/5$ and $u_{max} = 2/3$

  $$P_{end} - P_0 = (2+8,\ 14-2) = (10,\ 12)$$

- $u_{min} > u_{max}$,
  there is no line segment do draw

Summary: Liang-Barsky

- (x,y) coordinates are only computed for the two final intersection points
- At most 4 parameter values (division) are computed
- This is a non-iterative algorithm
- Can be extended to 3-D

Line-Segment Clipping Assessment

- Cohen-Sutherland
  - Works well if many lines can be rejected early
  - Recursive structure (multiple subdivisions) is a drawback

- Liang-Barsky
  - Avoids recursive calls
  - Many cases to consider (tedious, but not expensive)
  - In general **much faster** than Cohen-Sutherland

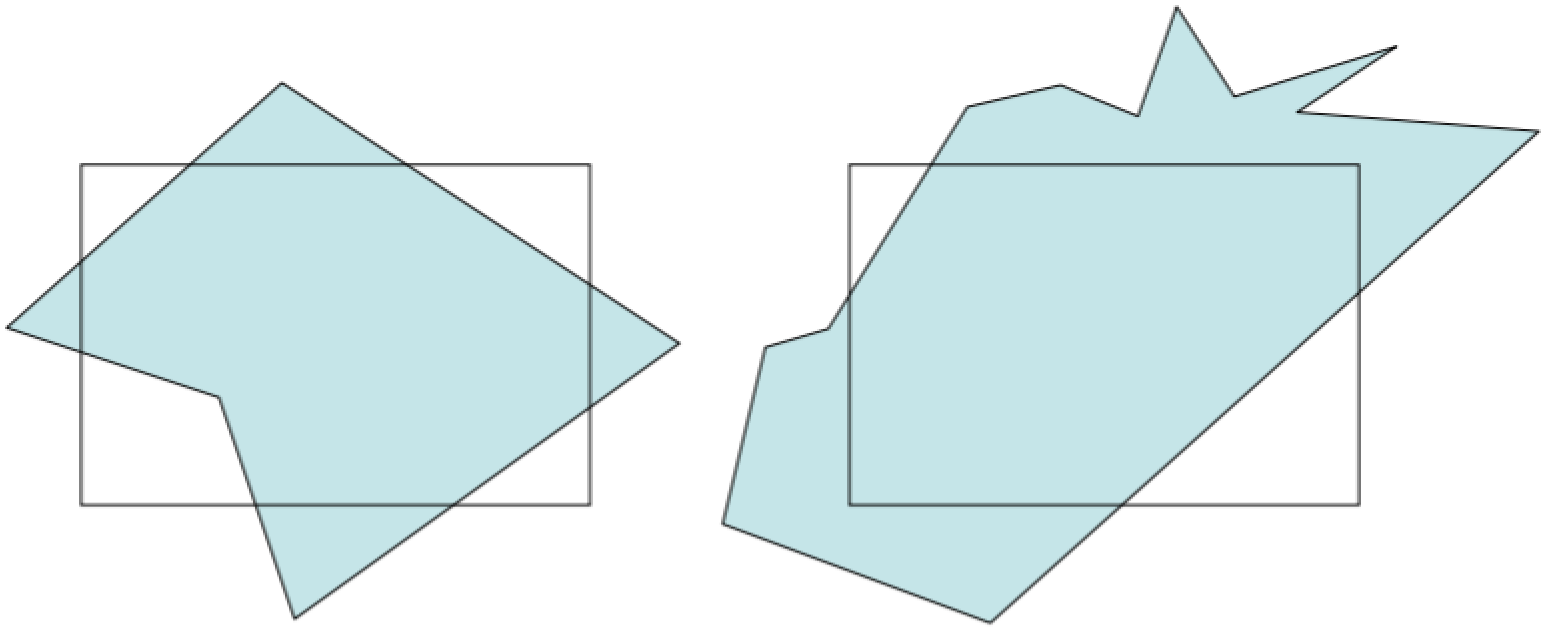
Outline

- Line-Segment Clipping
  - Cohen-Sutherland
  - Liang-Barsky

- Polygon Clipping
  - Sutherland-Hodgeman
  - Weiler-Atherton

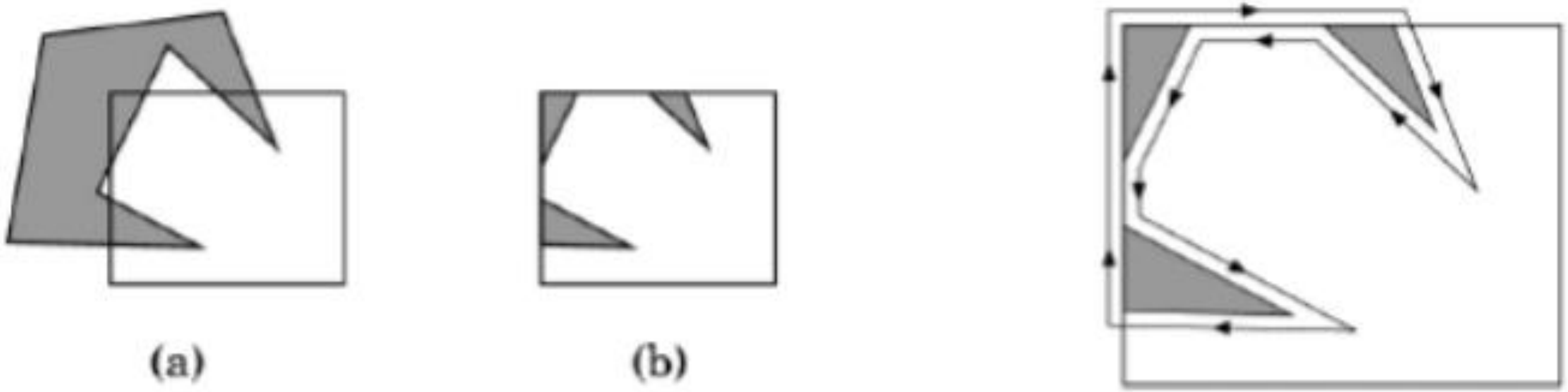
Polygon Clipping

- Clipping polygons complicates the problem because we have to take account the enclosed area
- Also, where line segment clipping always maps two points to two (possibly other) points, polygon clipping may actually change the number of points after clipping

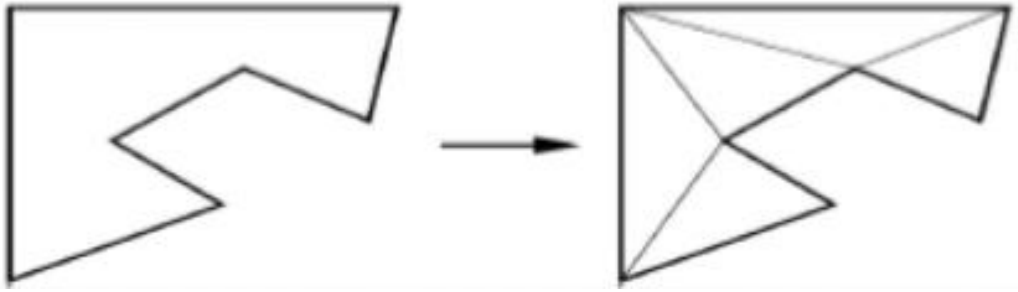


Concave Polygons

- Approach 1: clip, and then join pieces to a single polygon
  - often difficult to manage

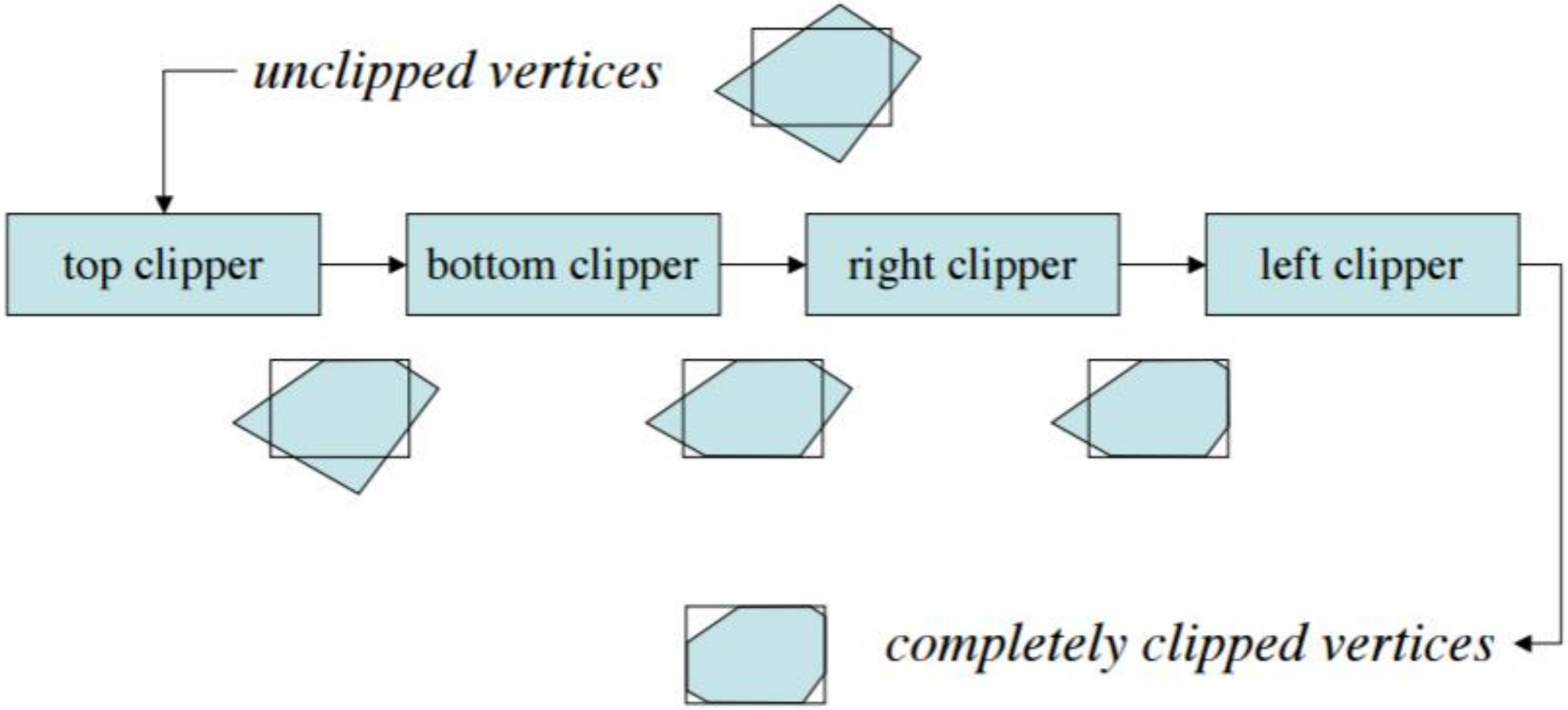


- Approach 2: tesselate and clip triangles
  - this is the common solution



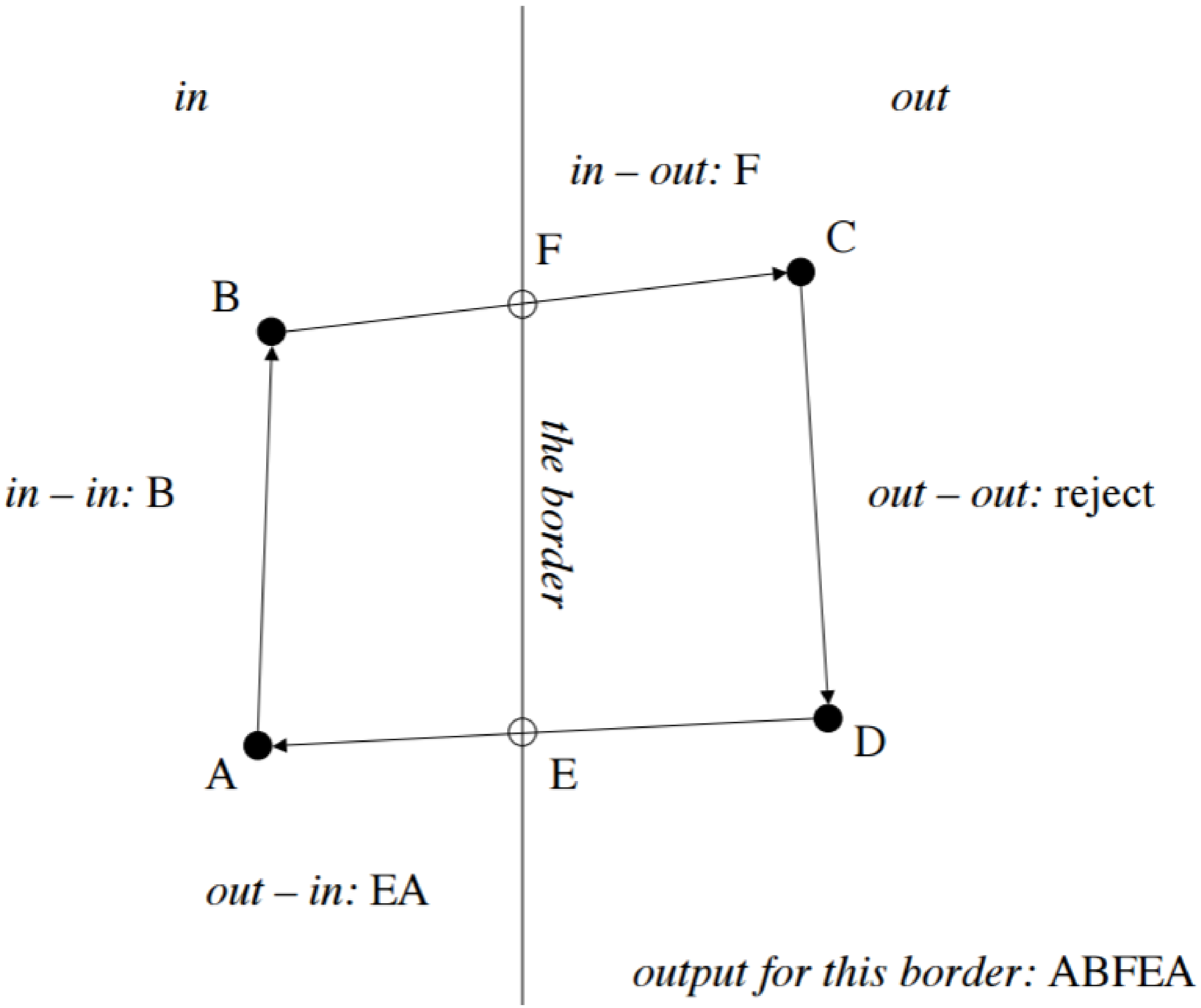
Sutherland-Hodgeman (part 1)

- Subproblem:
  - Input: polygon (vertex list) and single clip plane
  - Output: new (clipped) polygon (vertex list)
- Apply once for each clip plane
  - 4 in two dimensions
  - 6 in three dimensions
  - Can arrange in pipeline



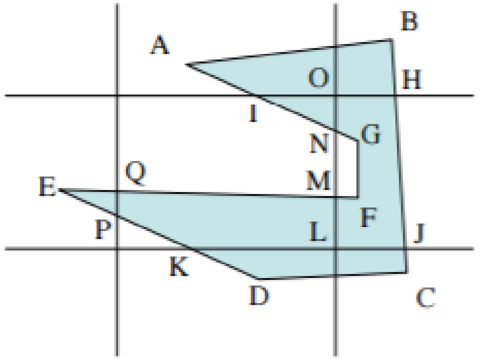
Sutherland-Hodgeman (part 2)

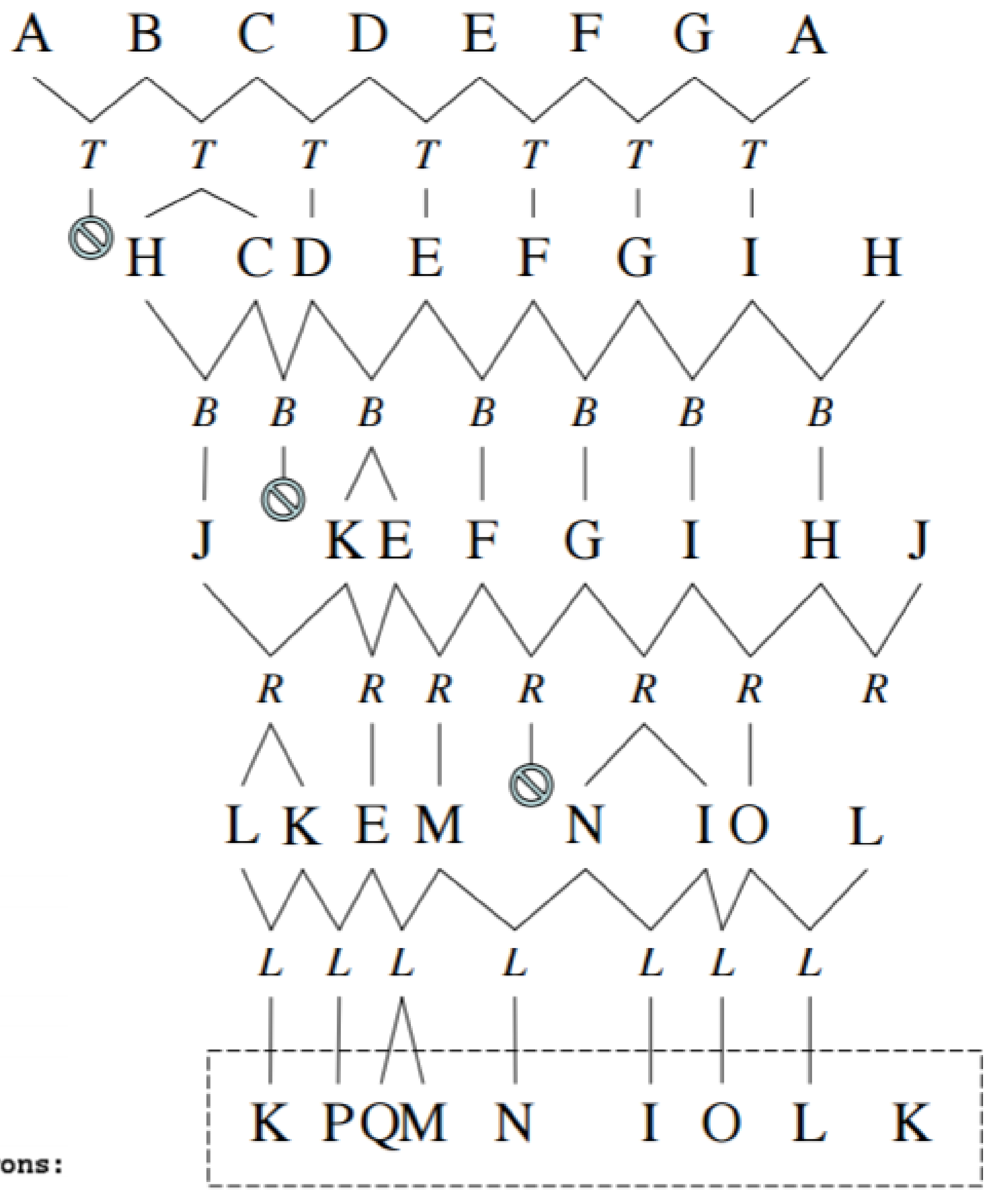
- To clip vertex list (polygon) against a **half-plane**:
  - Test first vertex. Output if inside, otherwise skip.
  - Then loop through list, testing transitions
    - In-to-in: output vertex
    - In-to-out: output intersection
    - out-to-in: output intersection & vertex
    - out-to-out: no output
  - Will output clipped polygon as vertex list



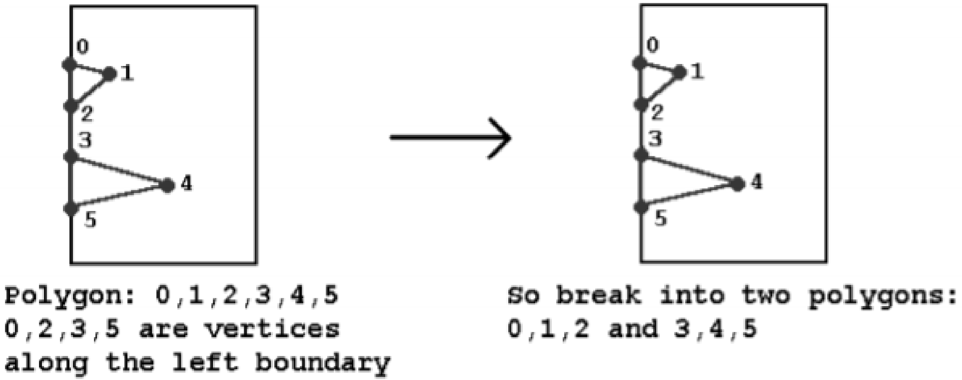
- Can combine with Liang-Barsky idea

Example & problem





- Works fine with convex polygons
- Concave polygons problematic
  - **Coincident edges** along a clip boundary may be generated as part of the output polygon
    - Such as MN, OL
  - Could cause problems with **polygon filling**, shadows, etc



Weiler-Atherton Polygon Clipping

- Clips concave polygons correctly
  - produces **separate polygons** for each visible fragment

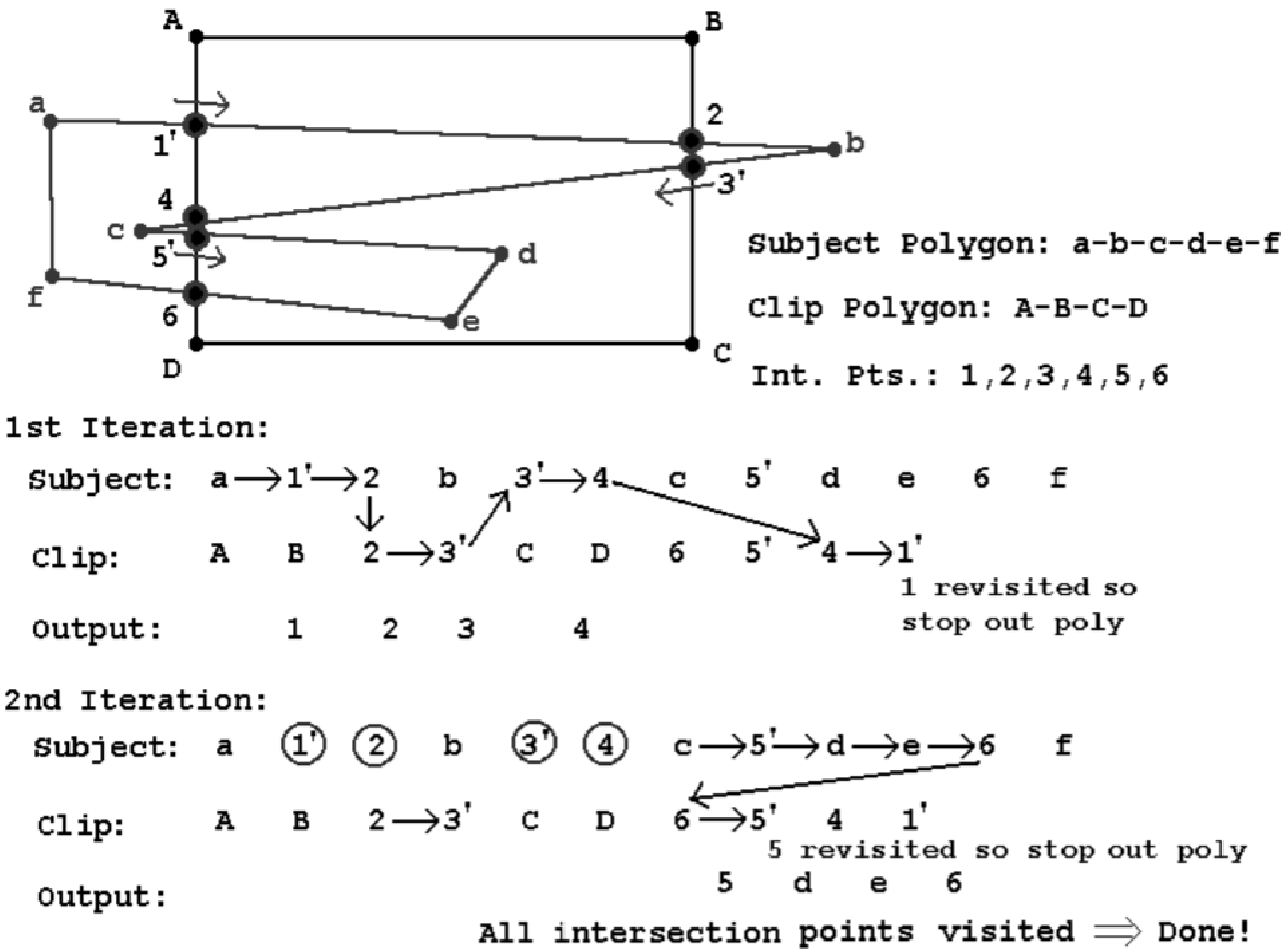
- Instead of always going around the polygon edges, we **also**, want to **follow window boundaries**.

  - For an outside-to-inside pair of vertices, follow the polygon boundary.

  - For an inside-to-outside pair of vertices, follow the window boundary in a clockwise direction.

Example

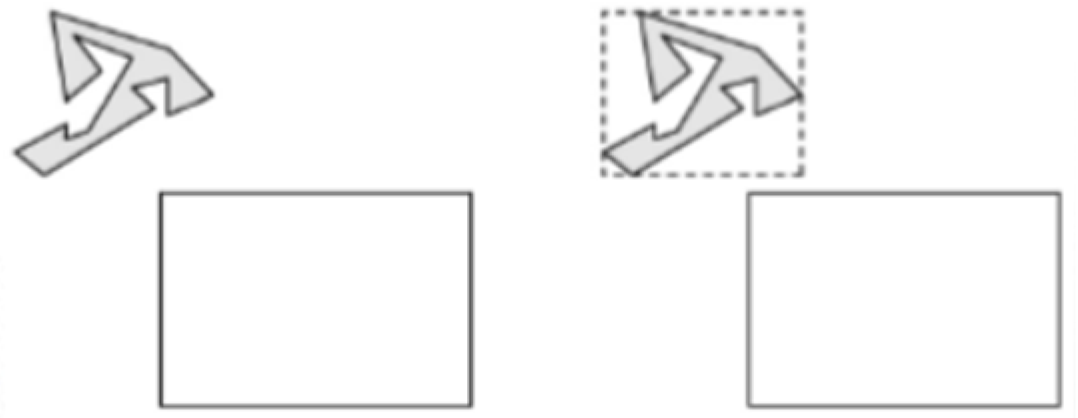
- outside-to-inside, follow the polygon boundary.
- inside-to-outside, follow the window boundary in a clockwise direction.



Other Cases and Optimizations

- Curves and surfaces
  - Do it analytically if possible
  - Otherwise, approximate curves / surfaces by lines and polygons

- Bounding boxes
  - Easy to calculate and maintain
  - Sometimes big savings



Summary: Clipping

- Clipping line segments to rectangle or cube
  - **Lessen** expensive multiplications and divisions
  - Cohen-Sutherland or Liang-Barsky

- Polygon clipping
  - Sutherland-Hodgeman pipeline
  - Weiler-Atherton Polygon Clipping

- Clipping in 3D
  - essentially extensions of 2D algorithms