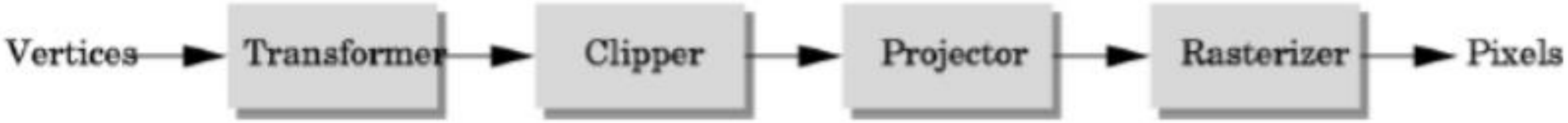
Computer Graphics
- Line & Polygon Clipping

Junjie Cao @ DLUT

Spring 2017

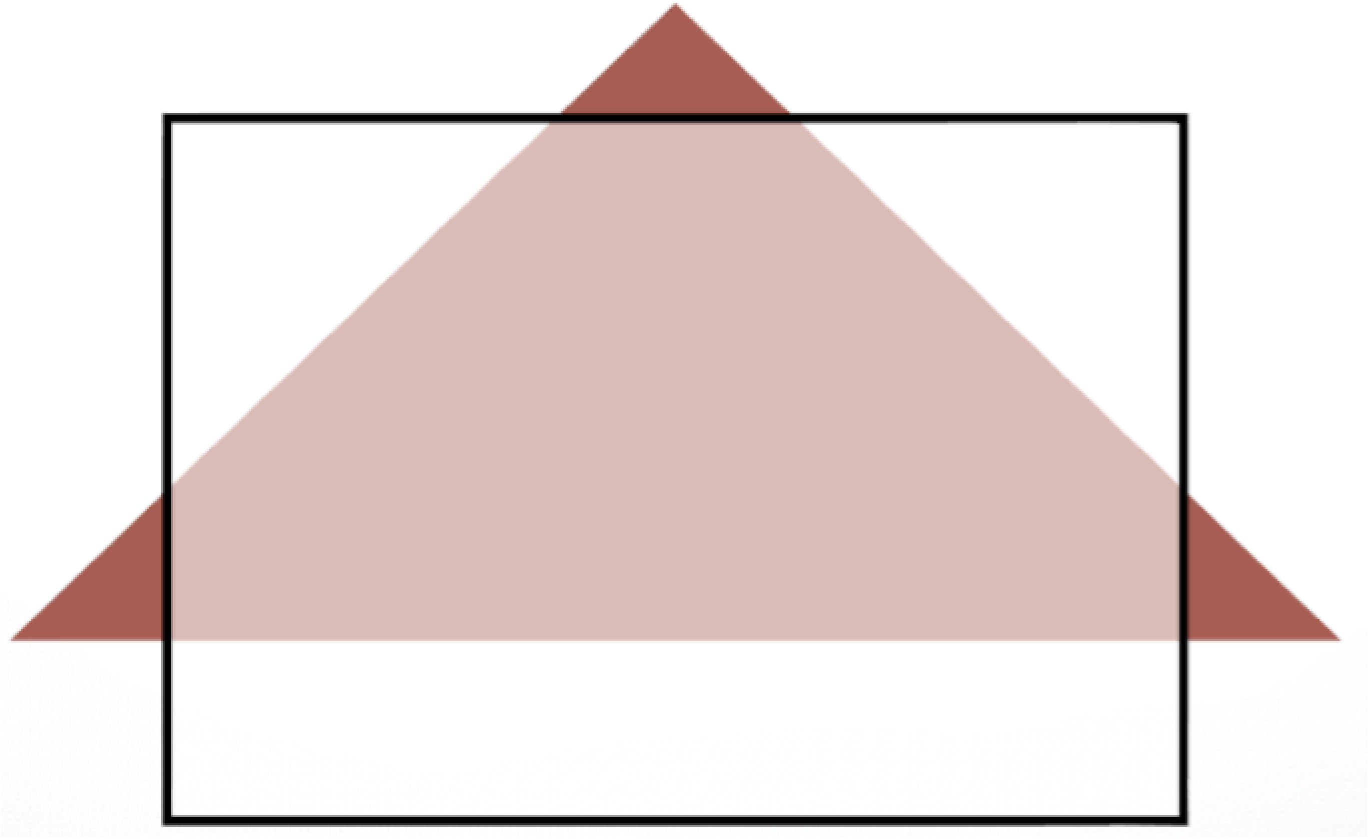
http://jjcao.github.io/ComputerGraphics/

The Graphics Pipeline, Revisited



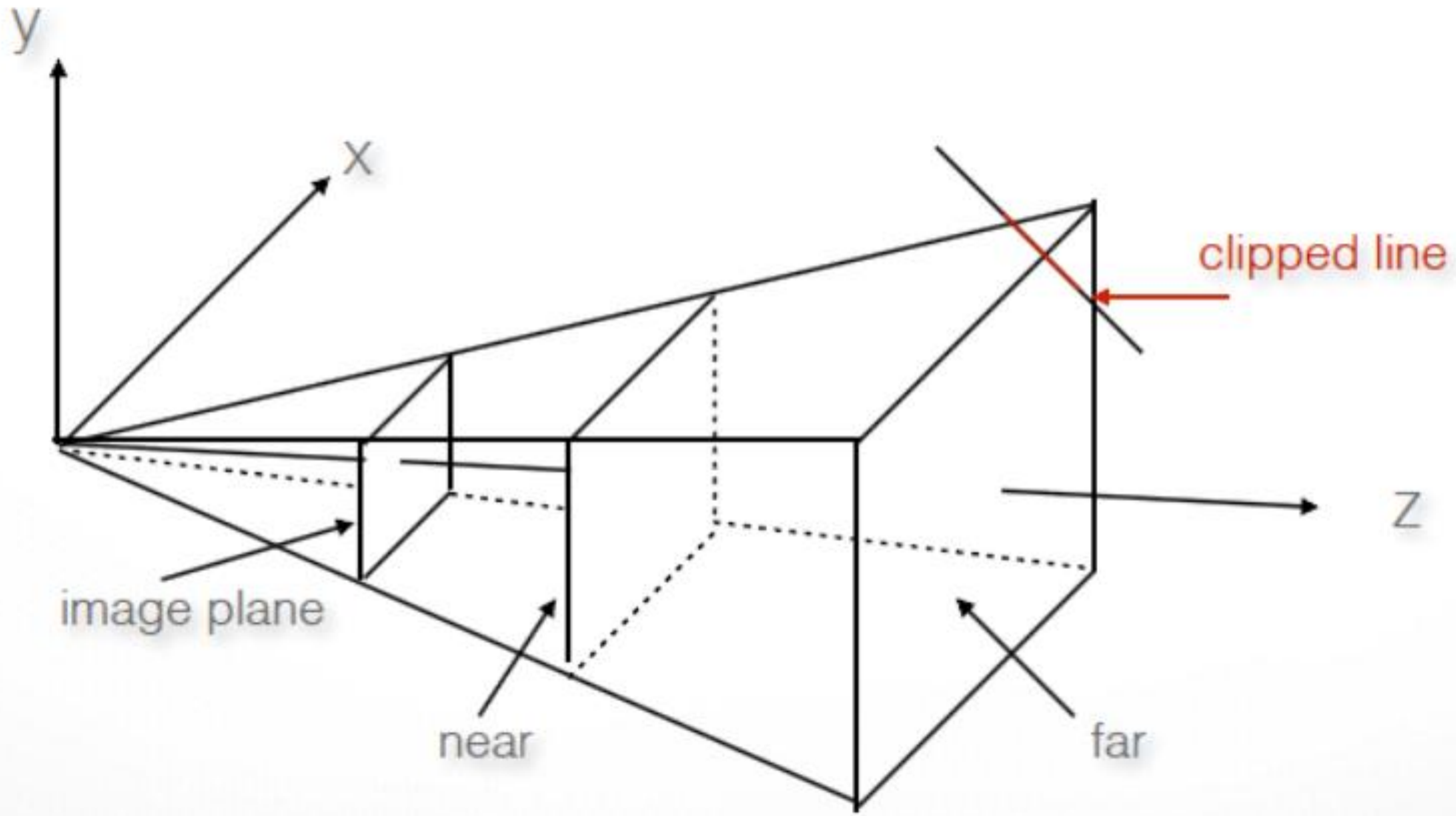
- Must eliminate objects that are outside of viewing frustum
- Clipping: object space (eye coordinates)
- Scissoring: image space (pixels in frame buffer)
  - most often less efficient than clipping
- We will first discuss 2D clipping (for simplicity)
  - OpenGL uses 3D clipping

2D Clipping Problem



Clipping Against a Frustum

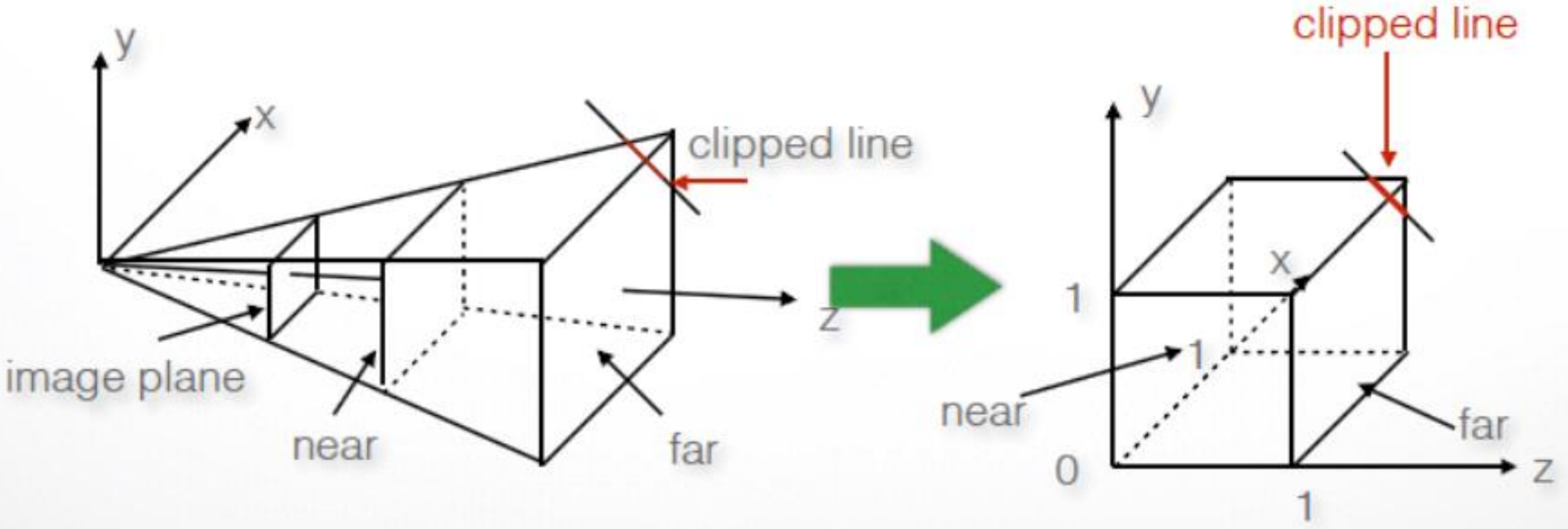
- General case of frustum (truncated pyramid)



- Clipping is tricky because of frustum shape

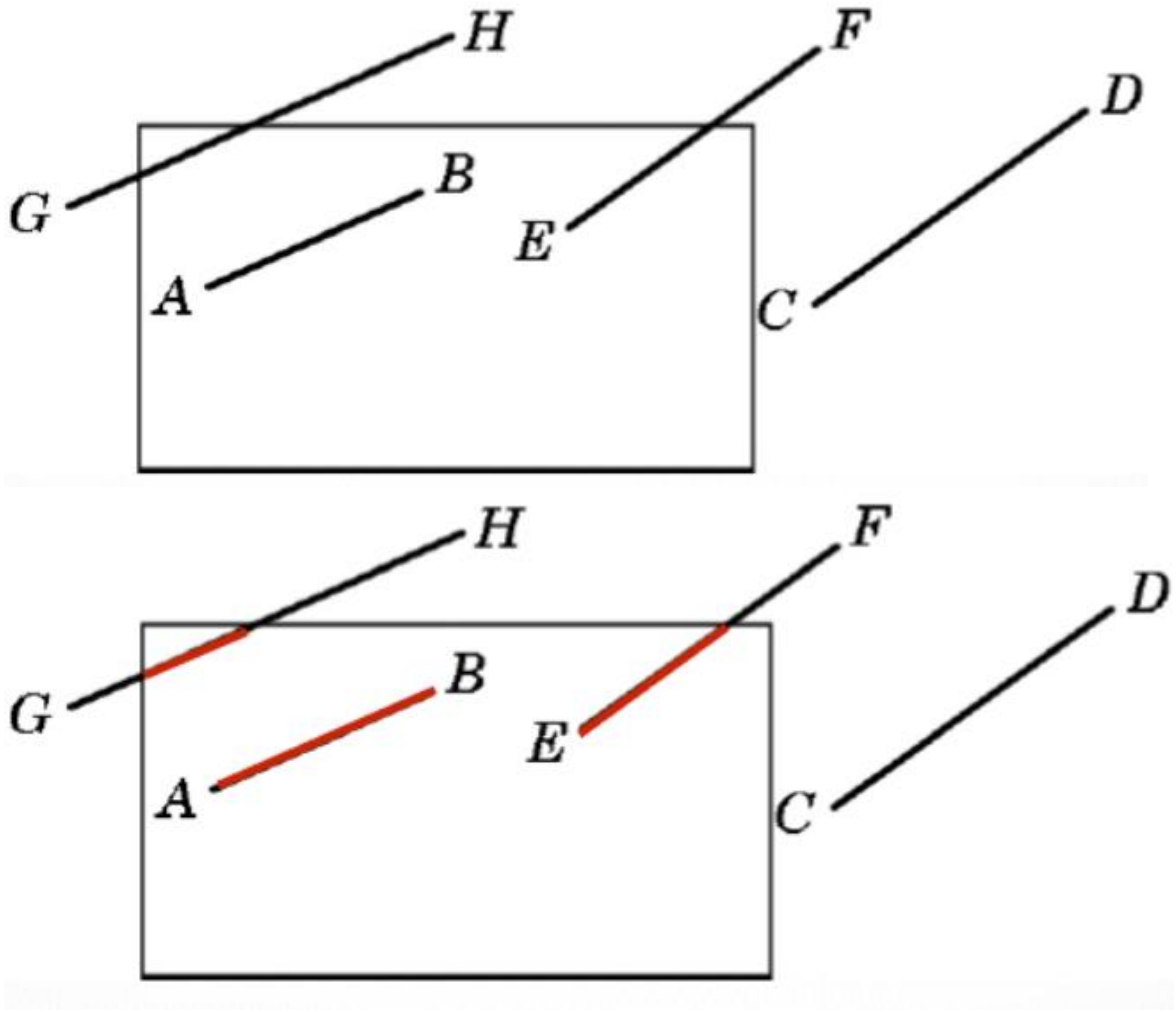
Perspective Normalization

- Solution:
  - Implement perspective projection by perspective normalization and orthographic projection
  - Perspective normalization is a homogeneous transformation



Clipping Against Rectangle in 2D

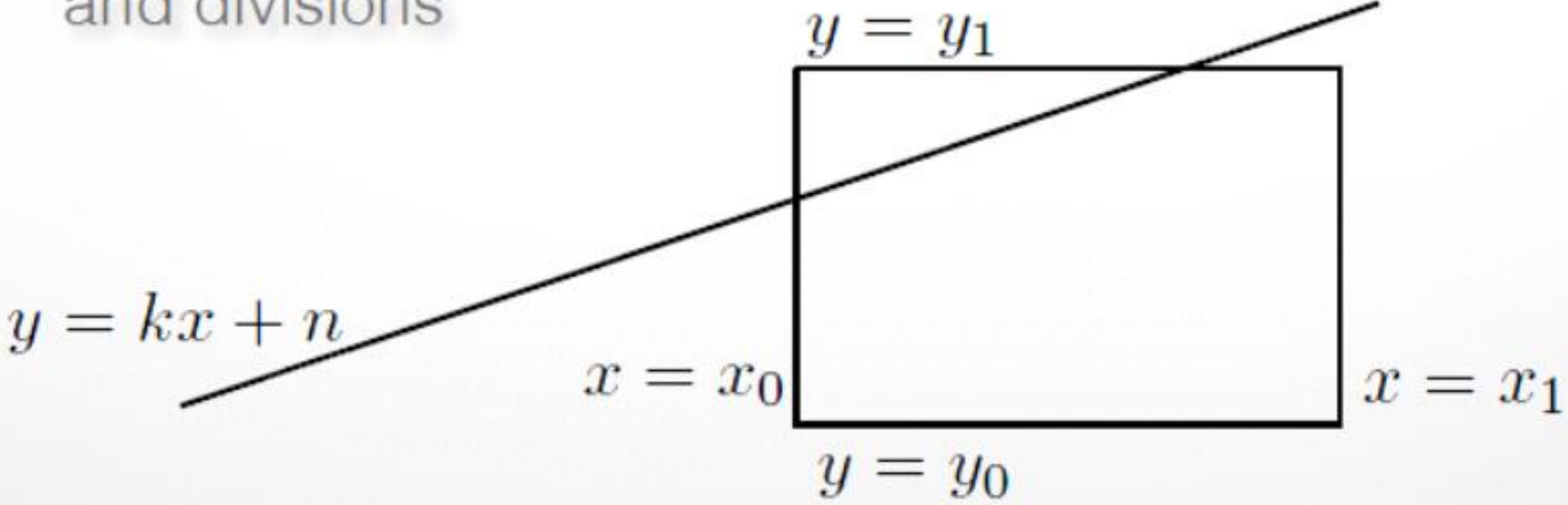
- Line-segment clipping: modify endpoints of lines to lie within clipping rectangle
- The result (in red)



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)

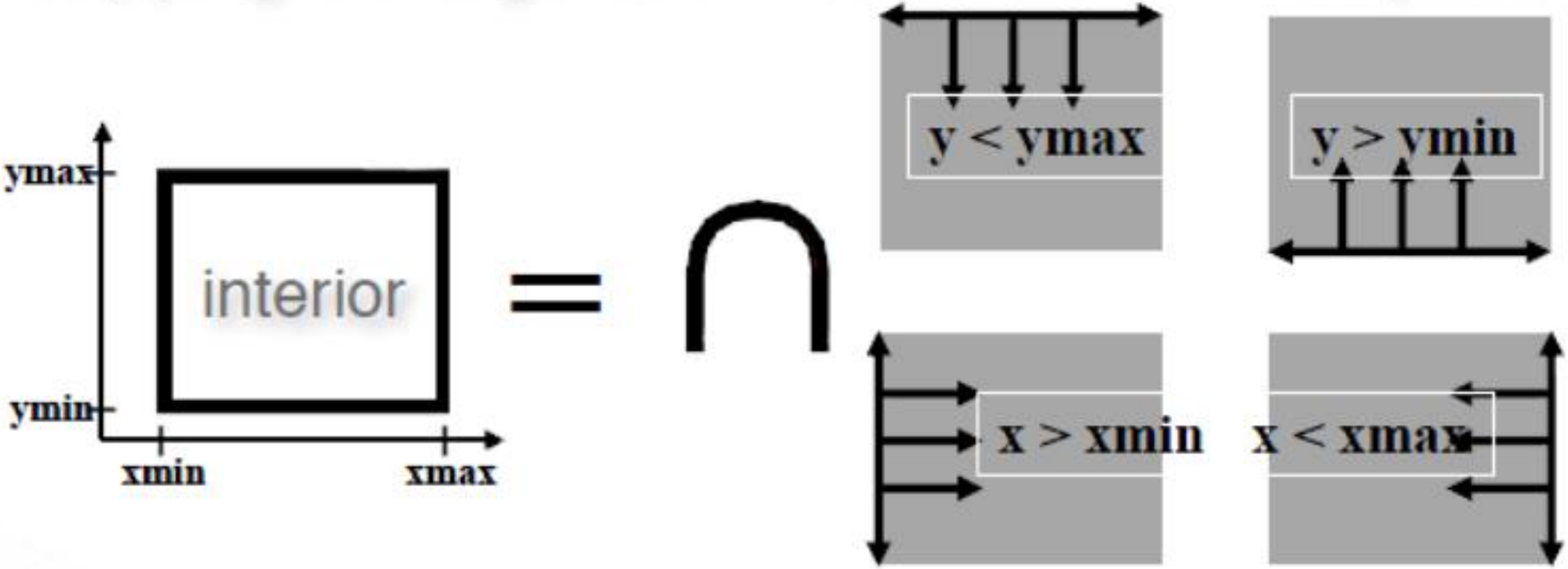
- Cohen-Sutherland Clipping

- Liang-Barsky Clipping

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

Cohen-Sutherland Clipping

- Clipping rectangle is an intersection of 4 half-planes

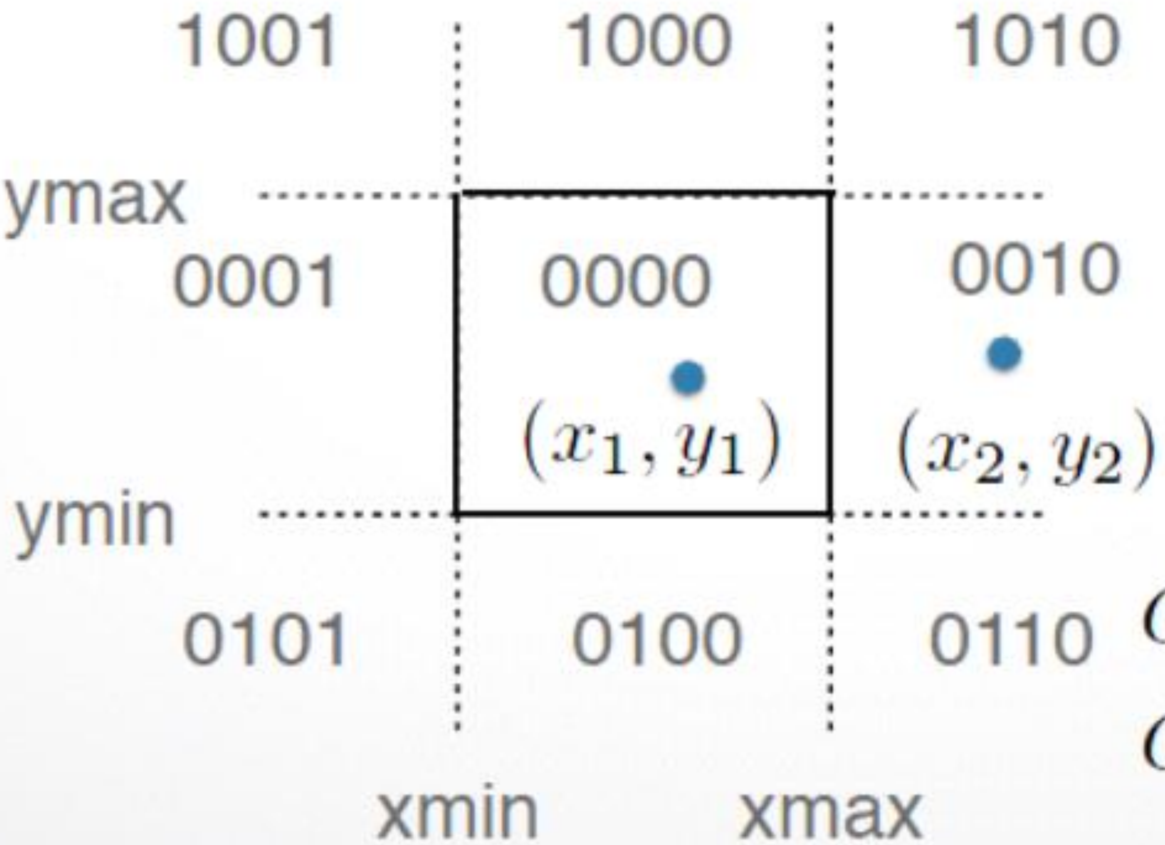


- Encode results of four half-plane tests

- Generalizes to 3 dimensions (6 half-planes)

Outcodes (Cohen-Sutherland)

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



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

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

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

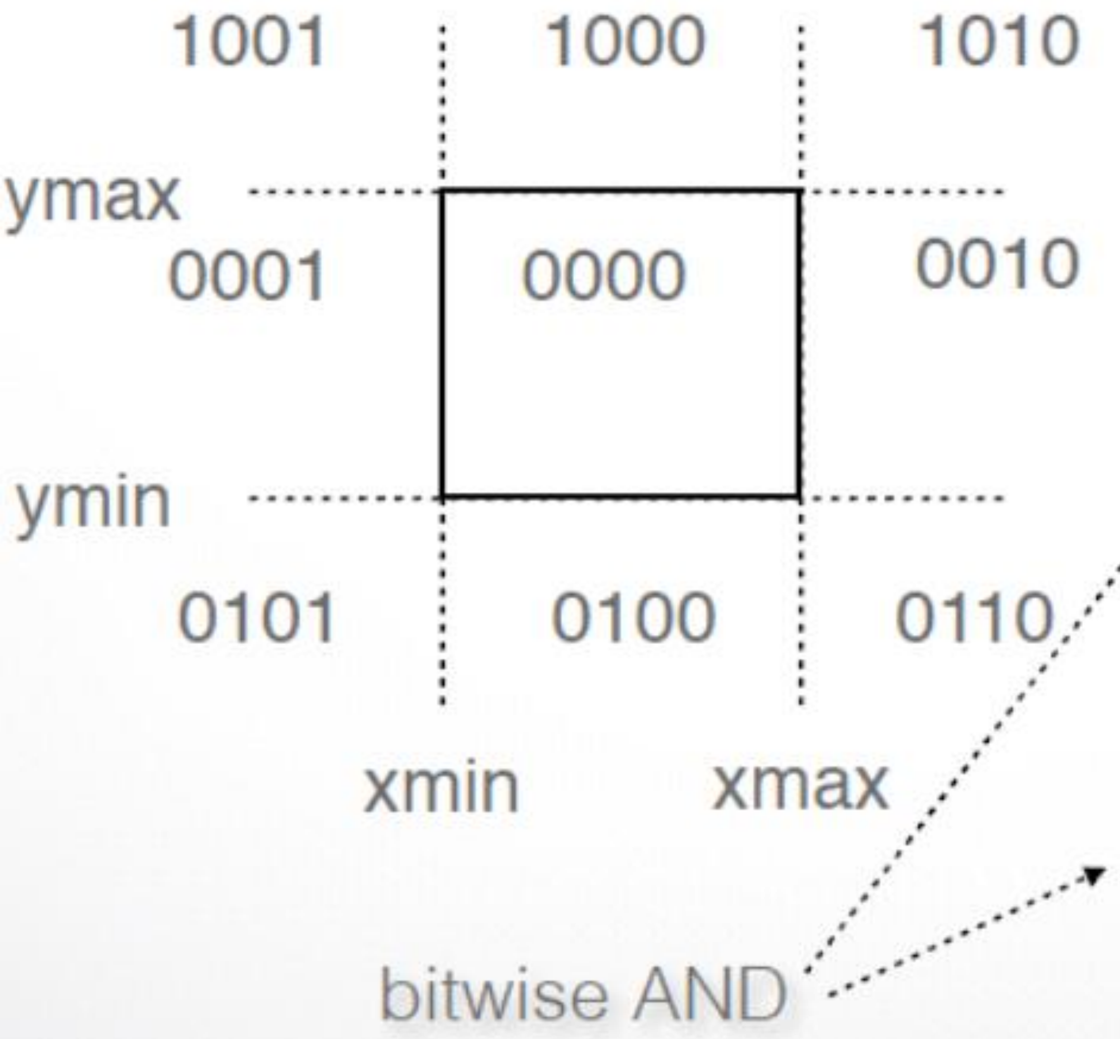
$$b_3 : x < x_{min}$$

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

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

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 ($o \neq 0000$)
- Pick a crossed edge ($o = b_0b_1b_2b_3$ and $b_k \neq 0$)
- 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

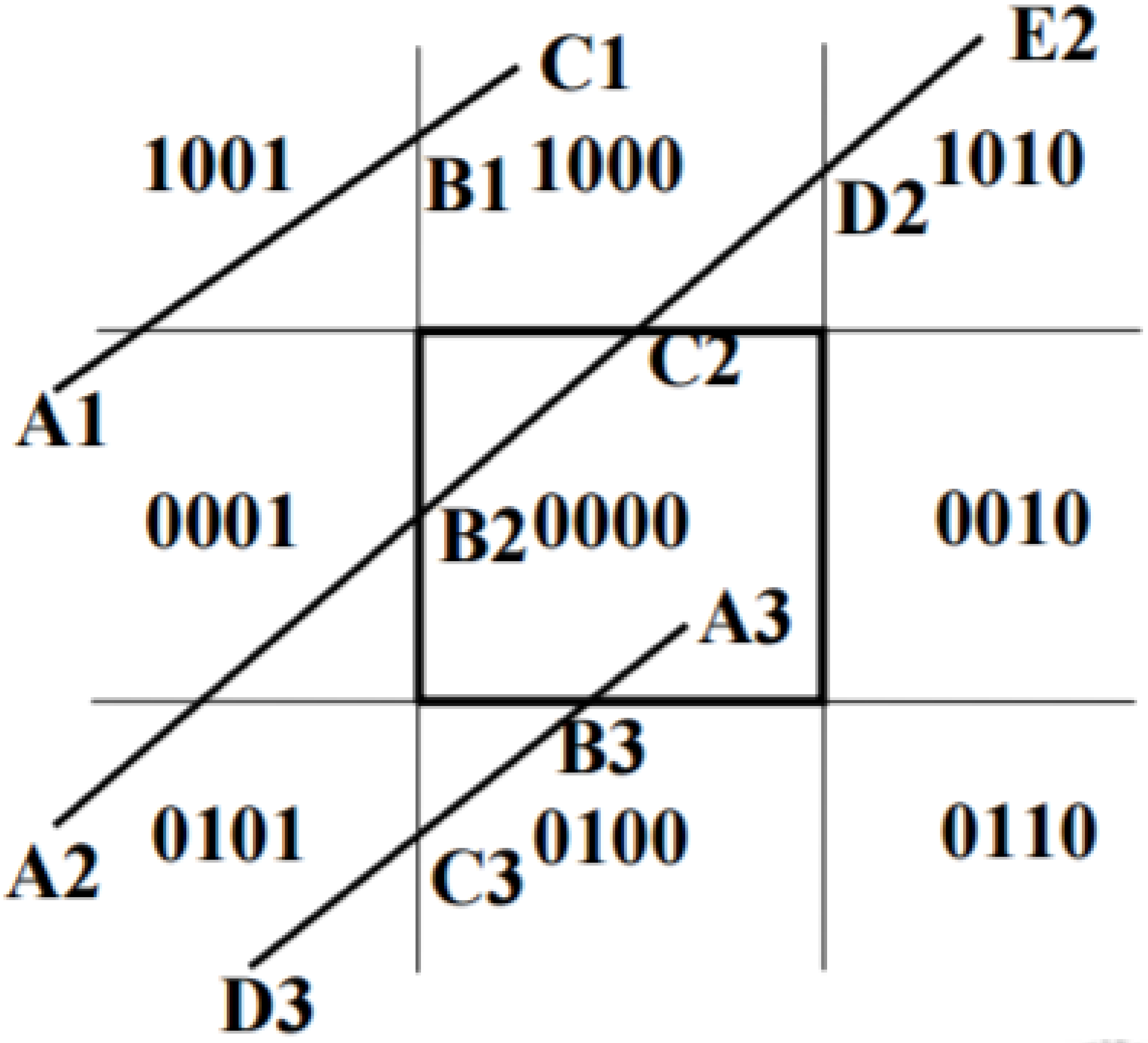
Cohen-Sutherland Line-Clipping

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 Line-Clipping

- Will do unnecessary clipping.
- Not the most efficient.

- Clipping and testing are done in fixed order.
- Efficient when most of the lines to be clipped are either rejected or accepted (not so many subdivisions).

- Easy to program.
- Parametric clipping are more efficient.

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

- More efficient than Cohen-Sutherland

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

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

- Can be described as

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

Liang-Barsky Clipping

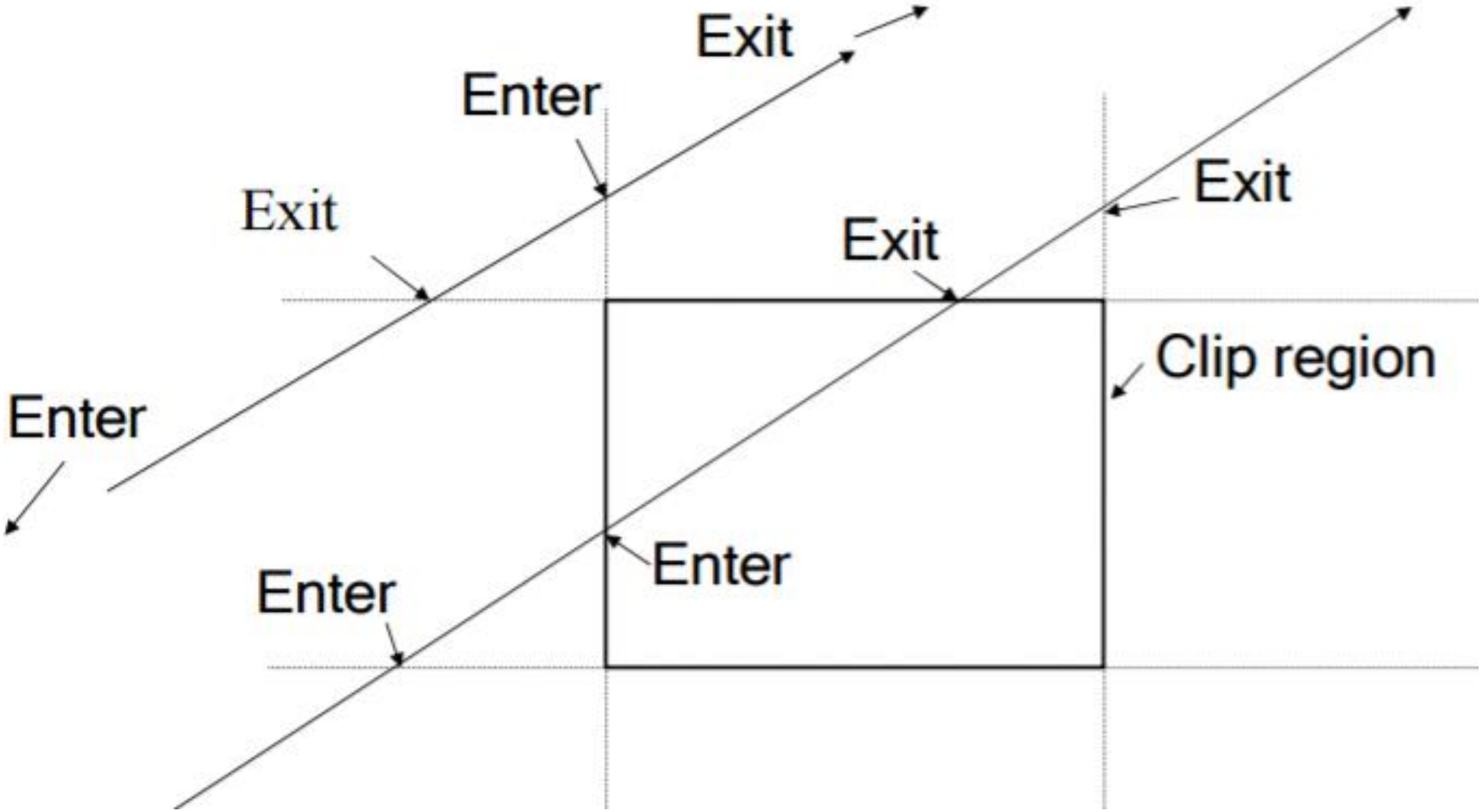
The infinitely line intersects the clip region edges when:

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

Liang-Barsky Clipping

- When $p_k < 0$, as u increases
  - line goes from outside to inside - entering
- When $p_k > 0$,
  - line goes from inside to outside - exiting
- When $p_k = 0$,
  - 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

Liang-Barsky Clipping

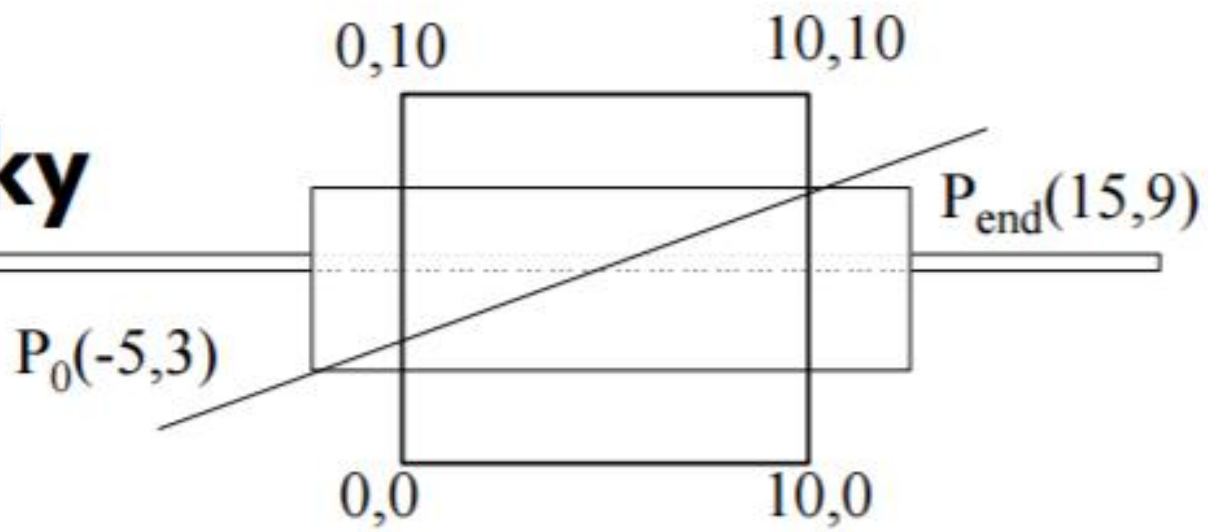


Liang-Barsky Clipping

1. Set $u_{min} = 0$ and $u_{max} = 1$.
2. Calculate the u values:
3. If $u < u_{min}$ or $u > u_{max}$ ignore it. Otherwise classify the u values as entering or exiting.
4. If $u_{min} < u_{max}$ then draw a line from:

$$(x_0 + \Delta x \cdot u_{min}, y_0 + \Delta y \cdot u_{min})$$ to
$$(x_0 + \Delta x \cdot u_{max}, y_0 + \Delta y \cdot u_{max})$$

Example Liang-Barsky



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

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 $u_{max} = 3/4$

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

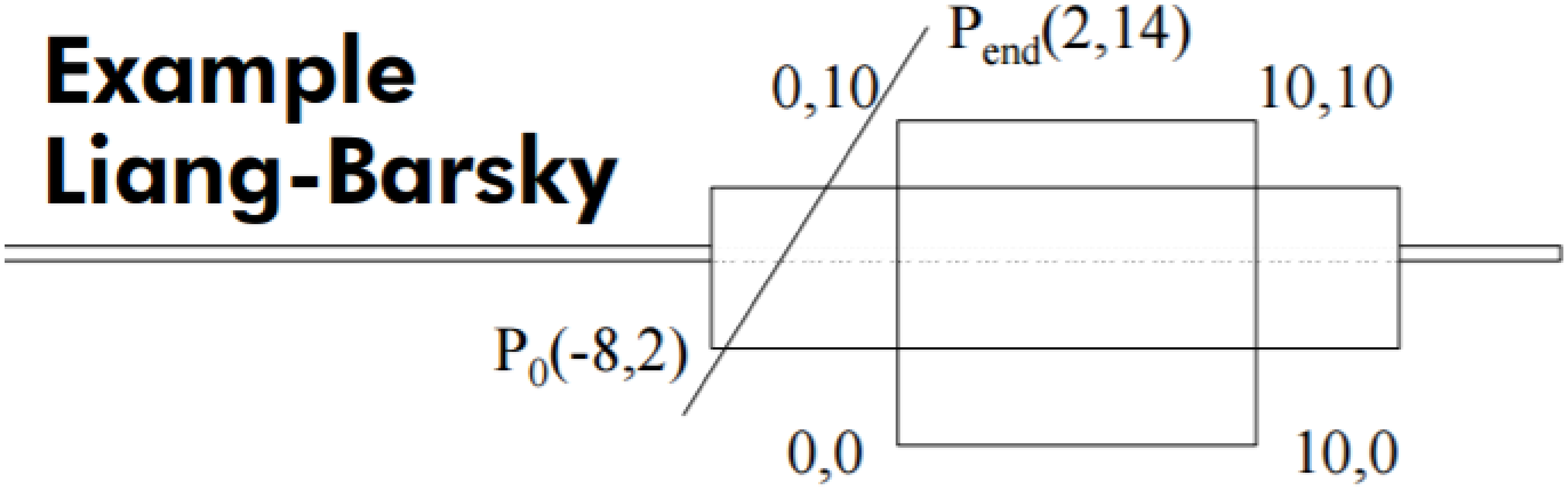
- If $u_{min} < u_{max}$, there is a line segment
  - compute endpoints by substituting u values
- Draw a line from

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

  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

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

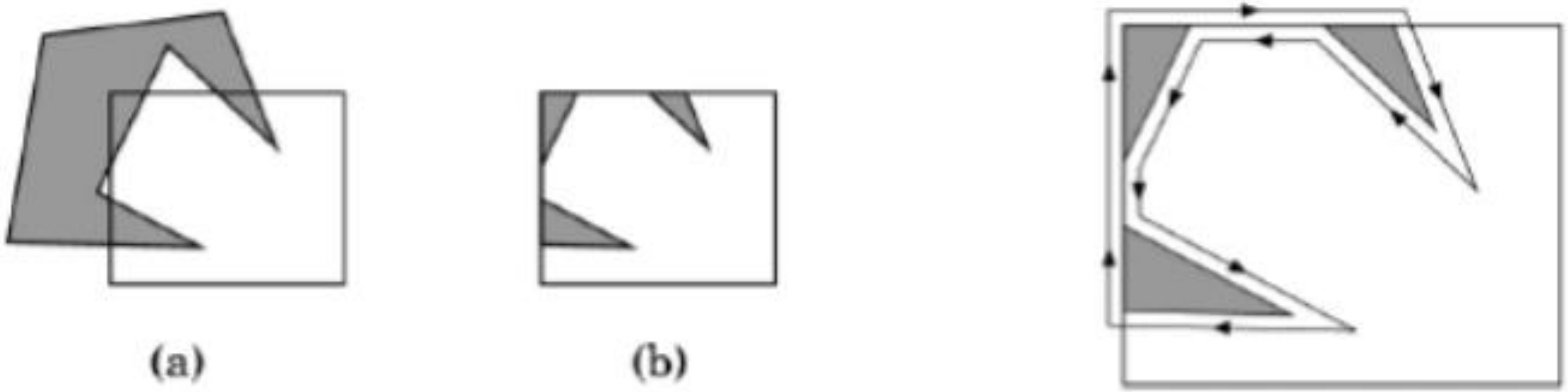
- Clipping in Three Dimensions

Polygon Clipping

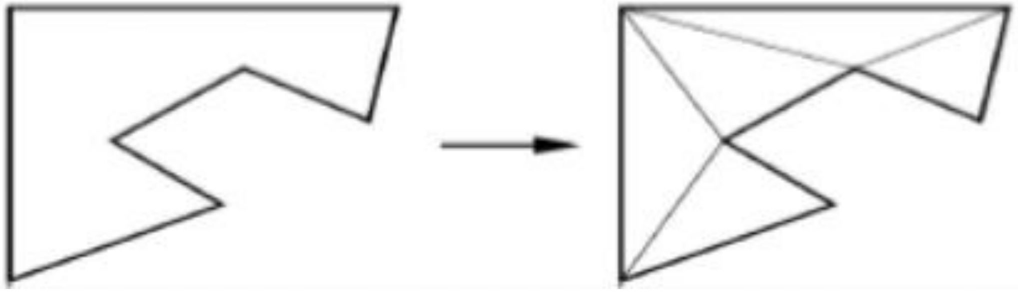
- Convert a polygon into one or more polygons
- Their union is intersection with clip window
- Alternatively, we can first tesselate concave polygons (OpenGL supported)

Concave Polygons

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



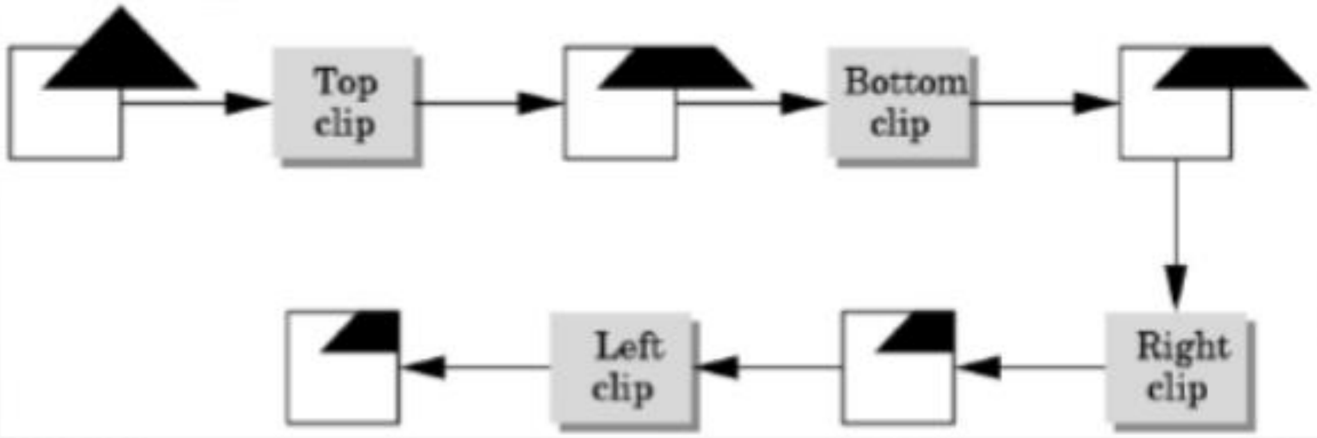
- Approach 2: tesselate and clip triangles
  - this is the common solution



Sutherland-Hodgeman (part 1)

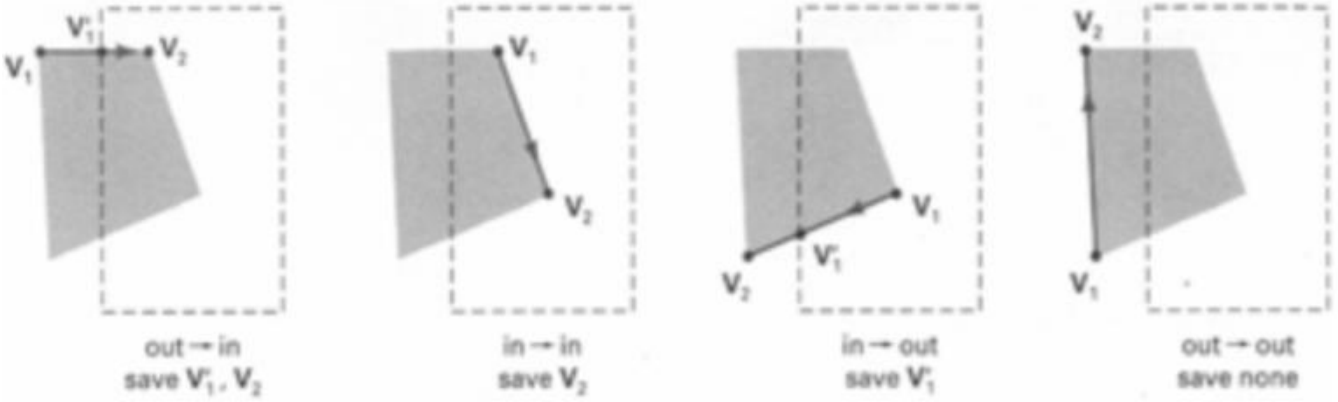
- Subproblem:
  - Input: polygon (vertex list) and single clip plane
  - Output: new (clipped) polygon (vertex list)

- 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 and vertex
        - out-to-out: no output
    - Will output clipped polygon as vertex list



- Concave polygons may be displayed with extra lines => need some cleanup
- Can combine with Liang-Barsky idea

Weiler-Atherton Polygon Clipping

- Clips concave polygons correctly.

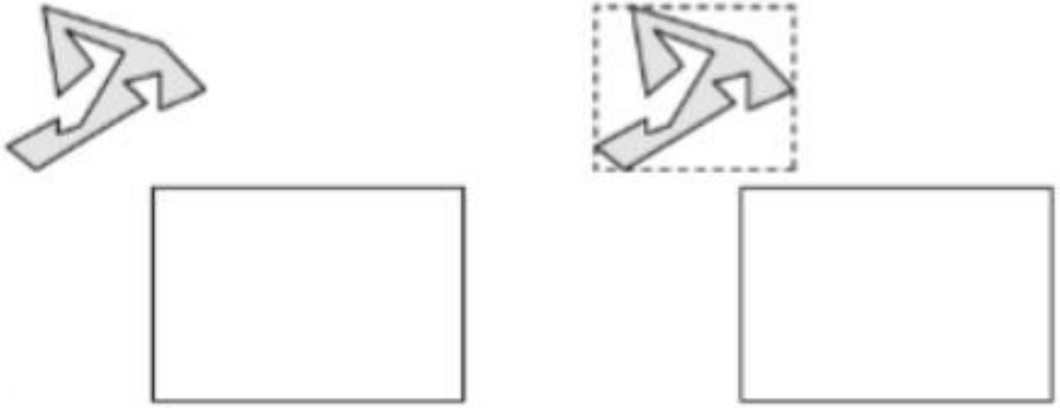
- Instead of always going around the polygon edges, we also, want to follow window boundaries.

- 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.

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



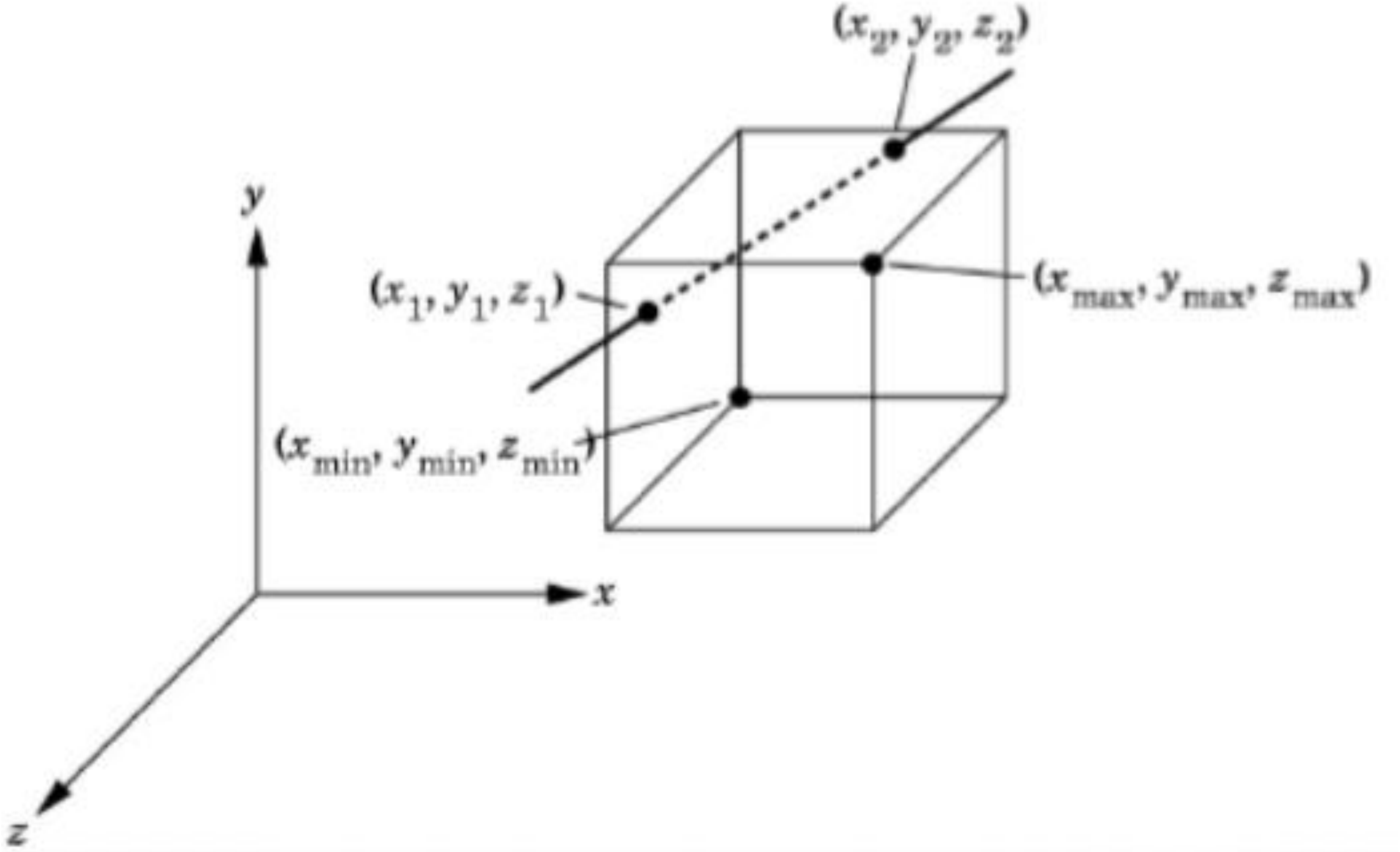
(a) (b)

Outline

- Line-Segment Clipping
  - Cohen-Sutherland
  - Liang-Barsky
- Polygon Clipping
  - Sutherland-Hodgeman
  - Weiler-Atherton
- Clipping in Three Dimensions

Clipping Against Cube

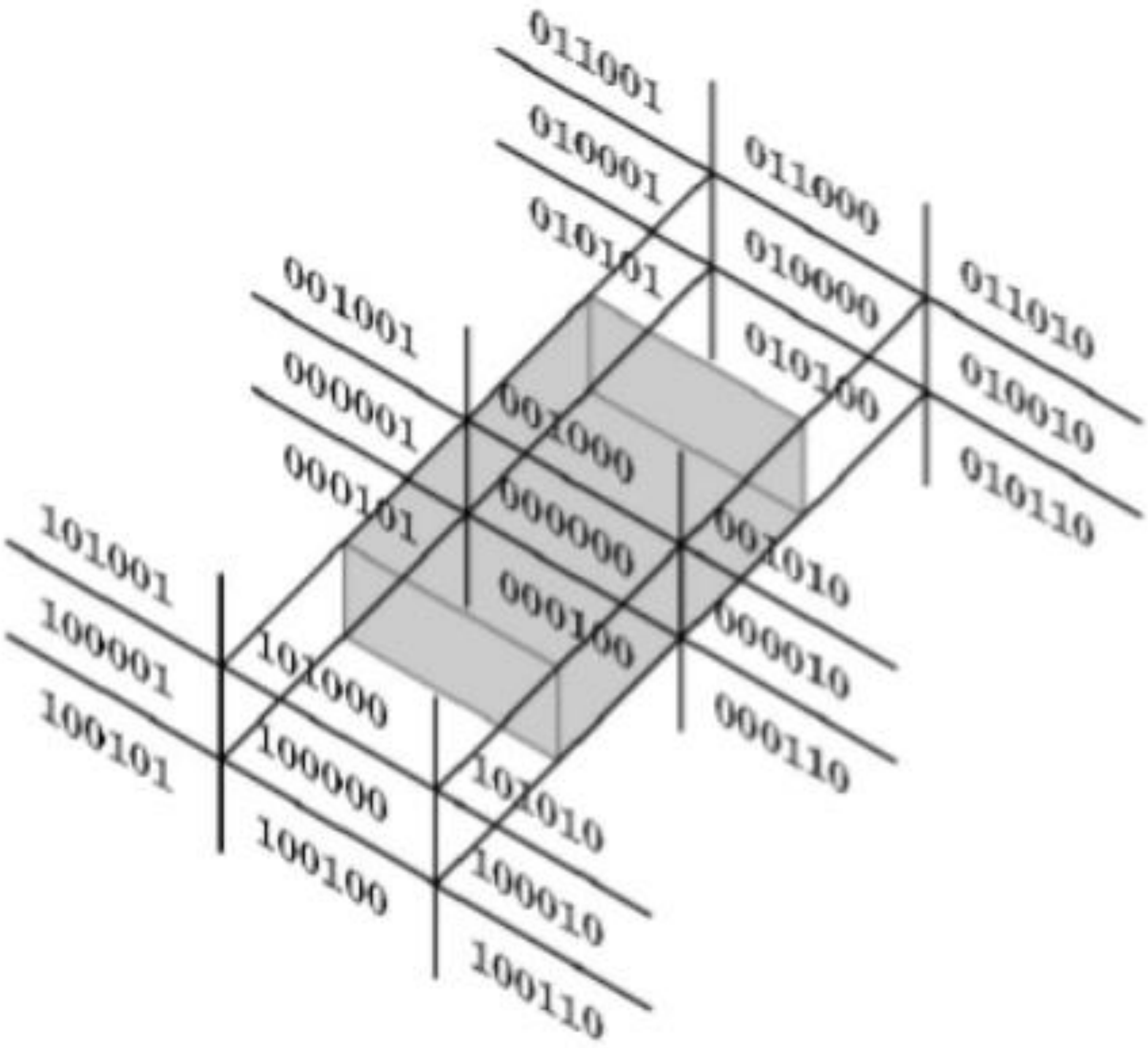
- Derived from earlier algorithms
- Can allow right parallelepiped



Cohen-Sutherland in 3D

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

- Other calculations as before



Liang-Barsky in 3D

- Add equation $z(\alpha) = (1-\alpha)z_1 + \alpha z_2$
- Solve, for $\mathbf{p}_0$ in plane and normal $\mathbf{n}$:

$$p(\alpha) = (1-\alpha)p_1 + \alpha p_2$$
$$n \cdot (p(\alpha) - p_0) = 0$$

- Yields

$$\alpha = \frac{n \cdot (p_0 - p_1)}{n \cdot (p_2 - p_1)}$$

- Optimizations as for Liang-Barsky in 2D

Summary: Clipping

- Clipping line segments to rectangle or cube
  - Avoid expensive multiplications and divisions
  - Cohen-Sutherland or Liang-Barsky
- Polygon clipping
  - Sutherland-Hodgeman pipeline
- Clipping in 3D
  - essentially extensions of 2D algorithms