Introduction to Computer Graphics

4. Clipping

I-Chen Lin
National Yang Ming Chiao Tung University

Textbook: E.Angel, D. Shreiner Interactive Computer Graphics, 6th Ed., Pearson
Ref: D.D. Hearn, M. P. Baker, W. Carithers, Computer Graphics with OpenGL, 4th Ed., Pearson

Intended Learning Outcomes

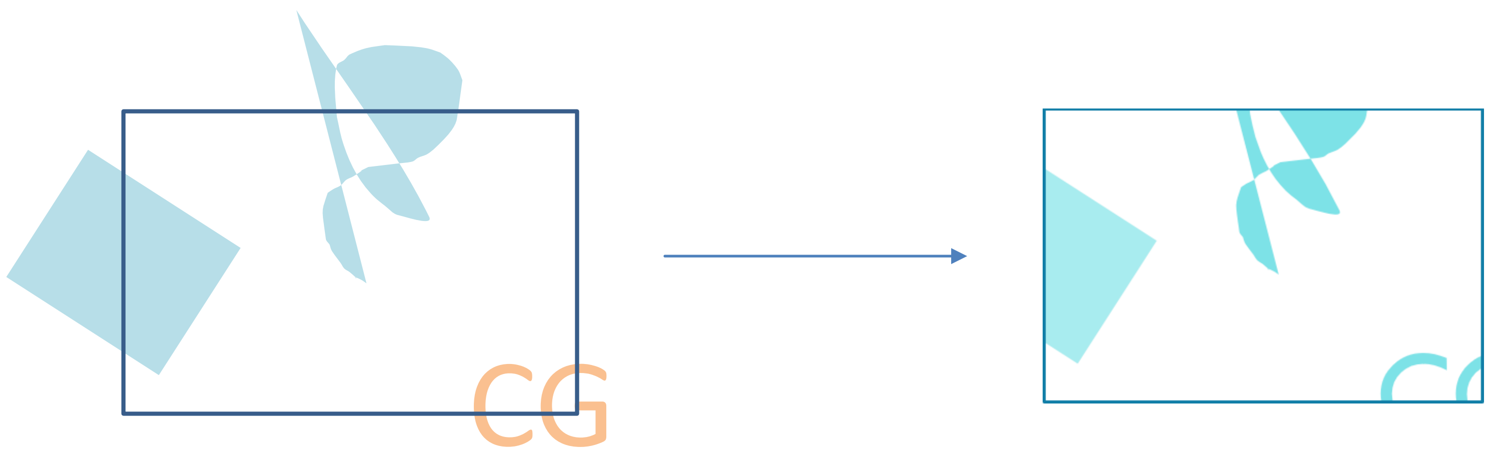
- On completion of this chapter, a student will be able to:
  - **Identify** the polygons that should be clipped and **explain** the reason behind.
  - **Describe** the typical clipping operations and algorithms.
  - **Indicate** the stage where clipping is applied within the graphics pipeline.

Objectives

- Introduce basic implementation strategies
- 2D Clipping
  - Lines
  - Polygons
- Clipping in 3D

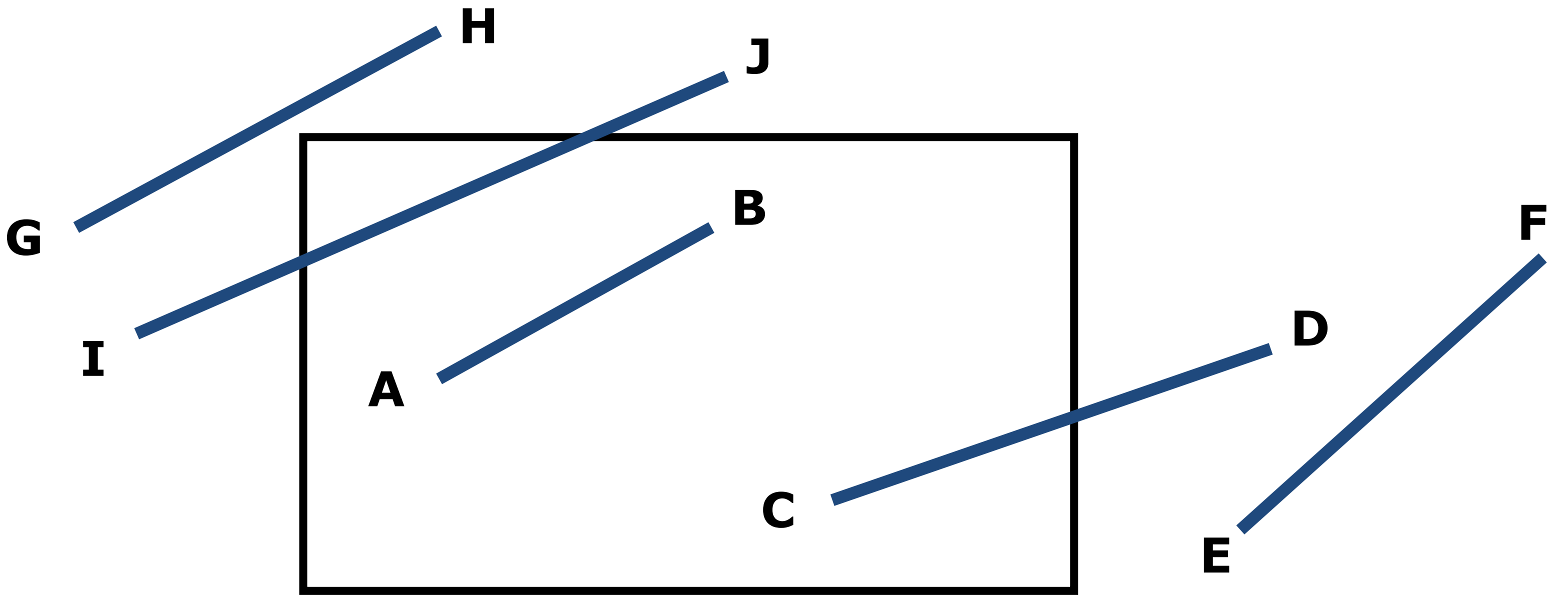
Clipping

- 2D against clipping window
- 3D against clipping volume
- Easy for line segments polygons
- Hard for curves and text
  - Convert to lines or polygons first



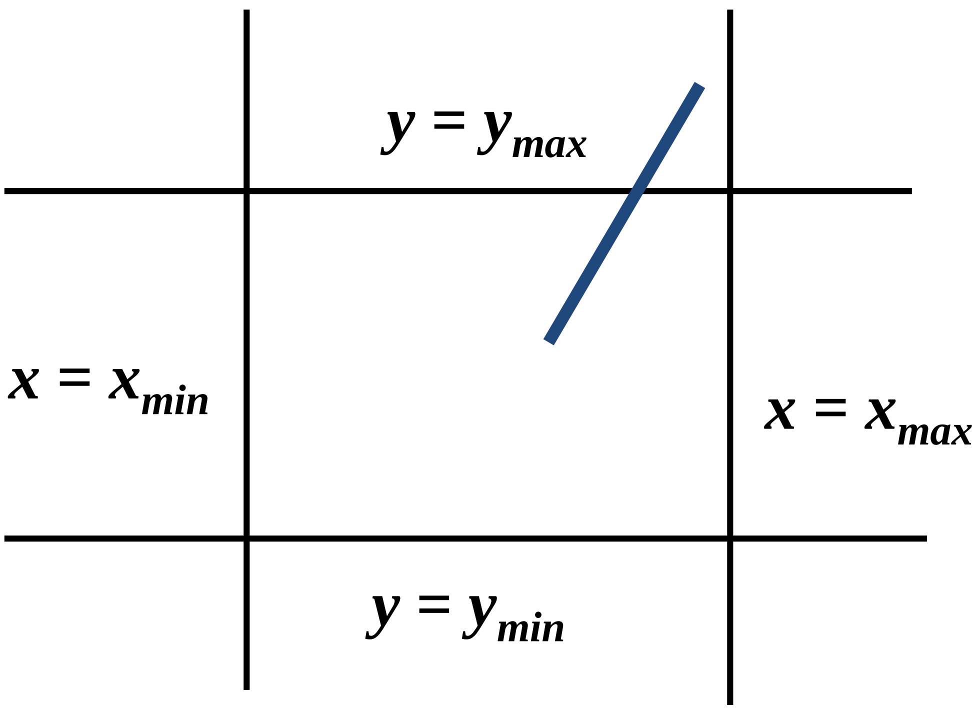
Clipping 2D Line Segments

- Brute force approach: compute intersections with all sides of clipping window
  - Inefficient: one division per intersection



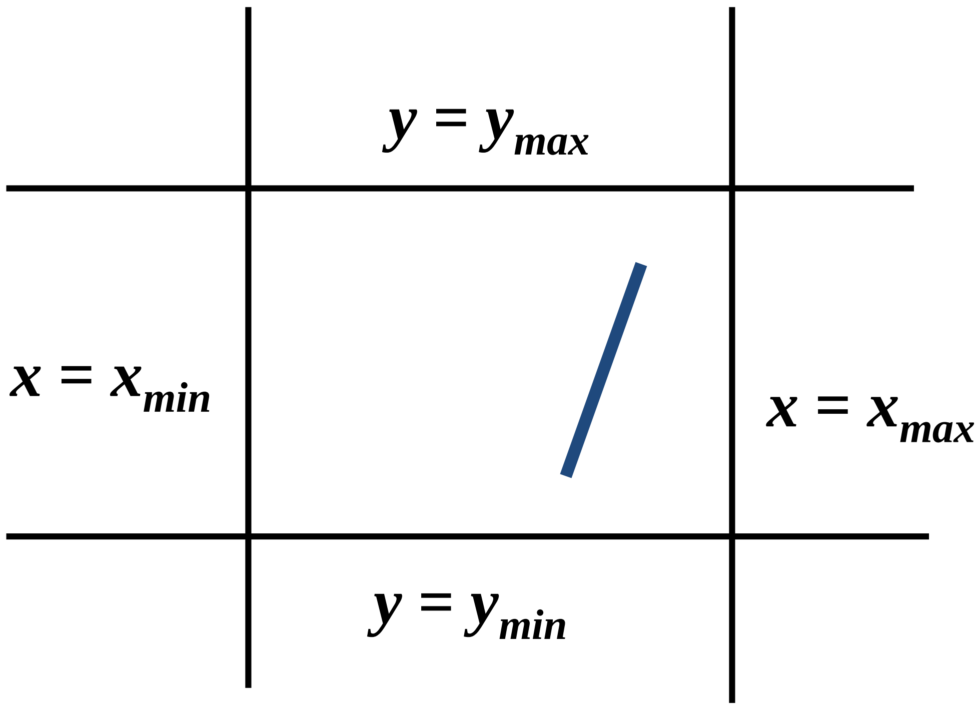
Cohen-Sutherland Algorithm

- Idea: eliminate as many cases as possible without computing intersections
- Start with four lines that determine the sides of the clipping window



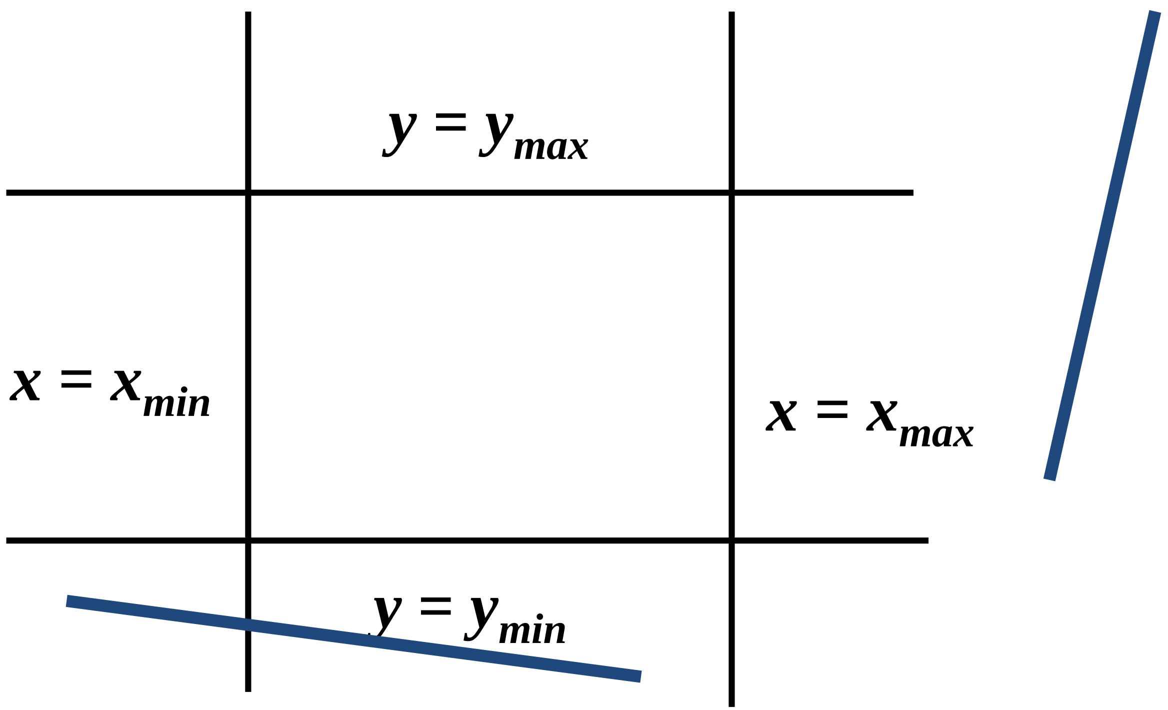
Case 1

- Case 1: both endpoints of a line segment inside all four lines
  - Draw (accept) the line segment as is



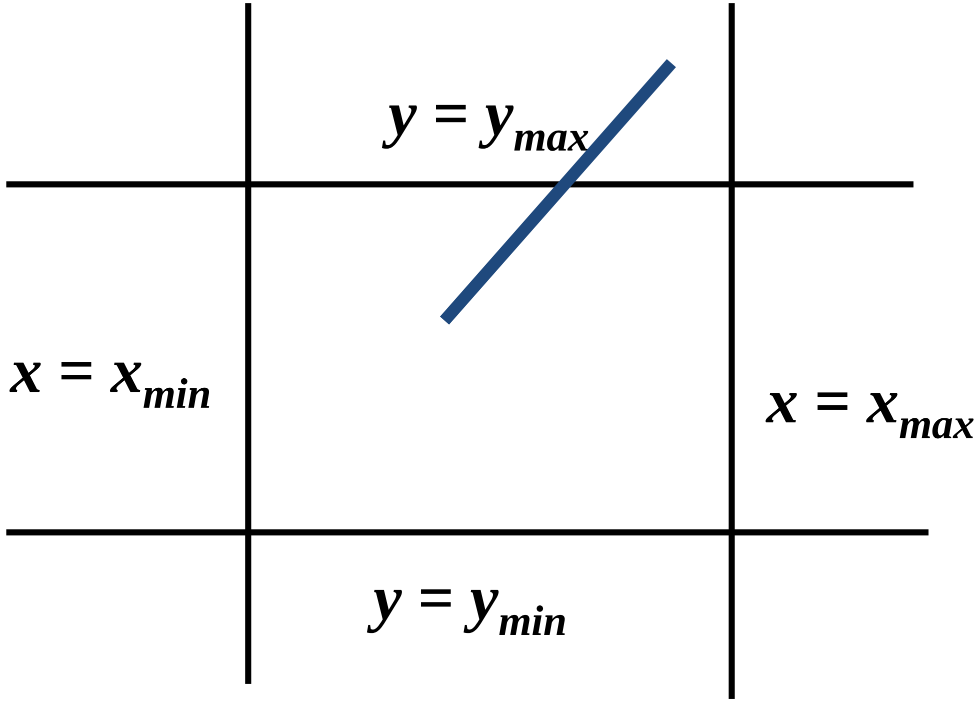
Case 2

- Case 2: both endpoints outside all lines and on same side of a line
  - Discard (reject) the line segment



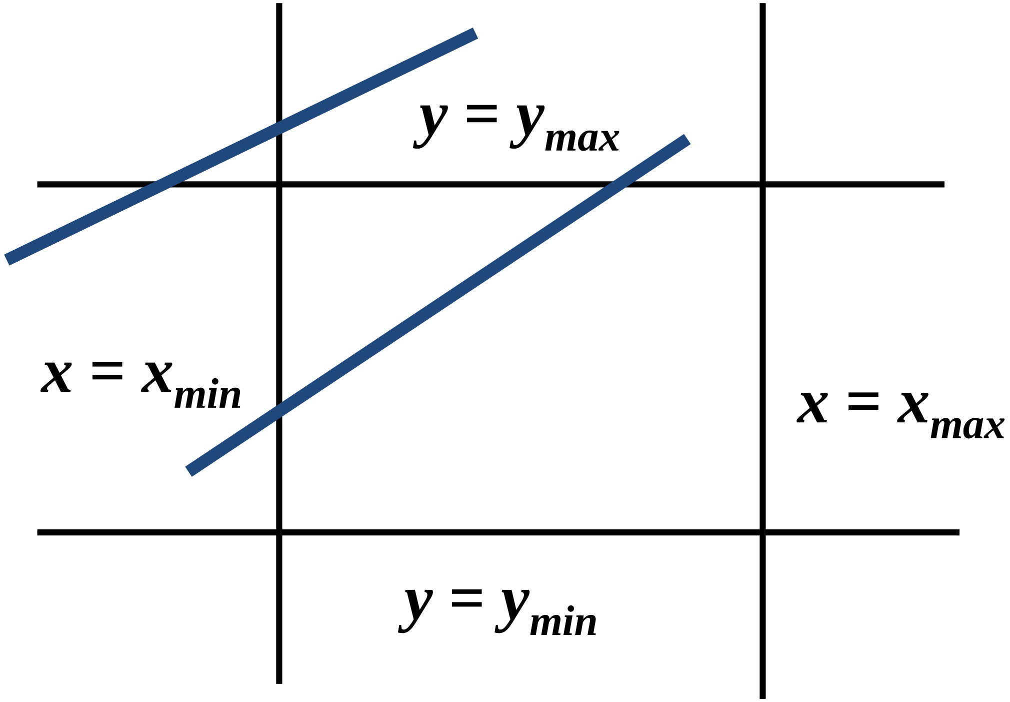
Case 3

- One endpoint inside, one outside
  - Must do at least one intersection



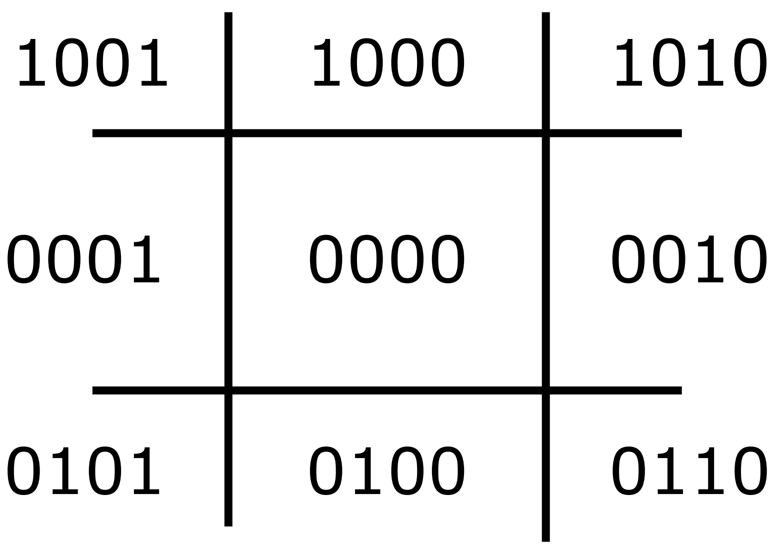
Case 4

- Both outside
  - May have part inside
  - Must do at least one intersection



Defining Outcodes

- For each endpoint, define an outcode
  - [b0 b1 b2 b3]
- Outcodes divide space into 9 regions
- Computation of outcode requires at most 4 subtractions



b0 = 1 if y > ymax, 0 otherwise

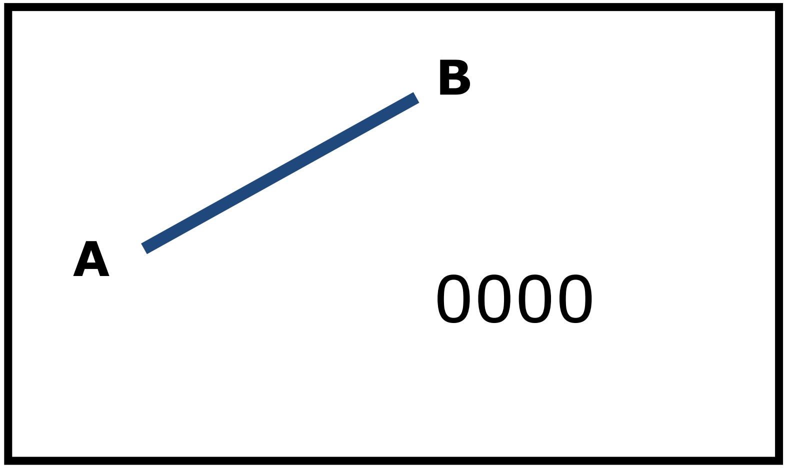
b1 = 1 if y < ymin, 0 otherwise

b2 = 1 if x > xmax, 0 otherwise

b3 = 1 if x < xmin, 0 otherwise

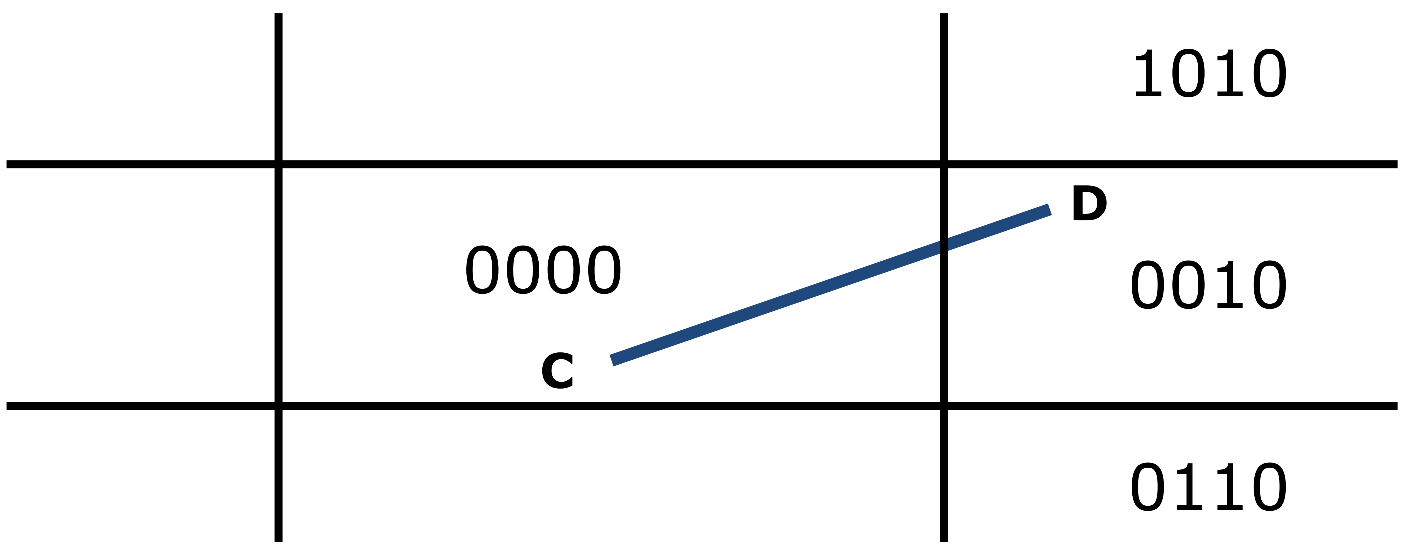
Using Outcodes

- AB: outcode(A) = outcode(B) = 0
  - Accept the line segment



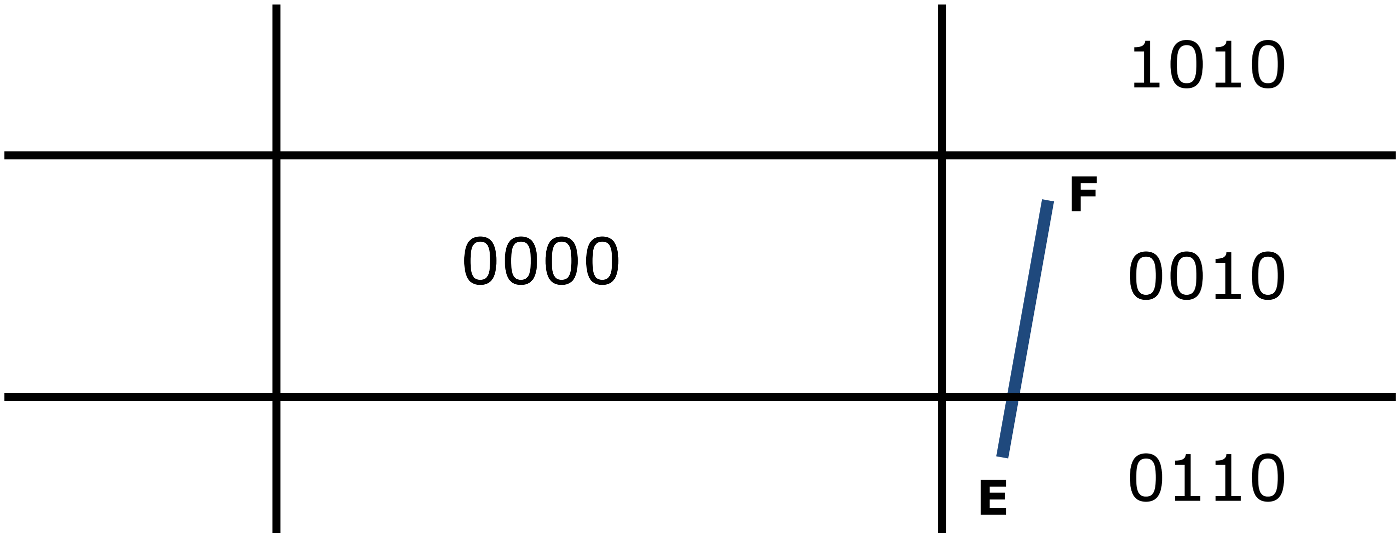
Using Outcodes

- CD: outcode (C) = 0, outcode(D) ≠ 0
  - Compute intersection
  - Location of 1 in outcode(D) determines which edge to intersect with
  - Note if there were a segment from C to a point in a region with 2 ones in outcode, we might have to do two intersections



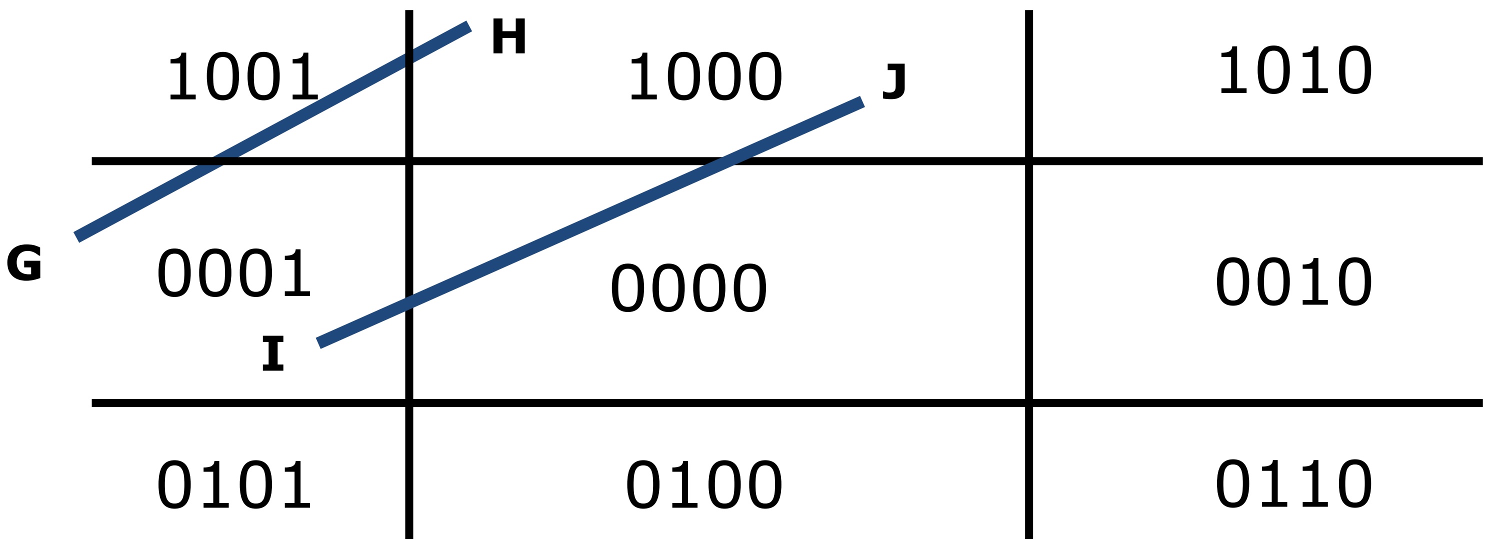
Using Outcodes

- EF: outcode(E) logically ANDed with outcode(F) (bitwise) ≠ 0
  - Both outcodes have a 1 bit in the same place
  - The line segment is outside of corresponding side of clipping window
  - reject



Using Outcodes

- GH and IJ: same outcodes, neither zero but logical AND yields zero
  - Shorten line segment by intersecting with one of sides of window
  - Compute outcode of intersection (newendpoint of shortened line segment)
  - Re-execute algorithm

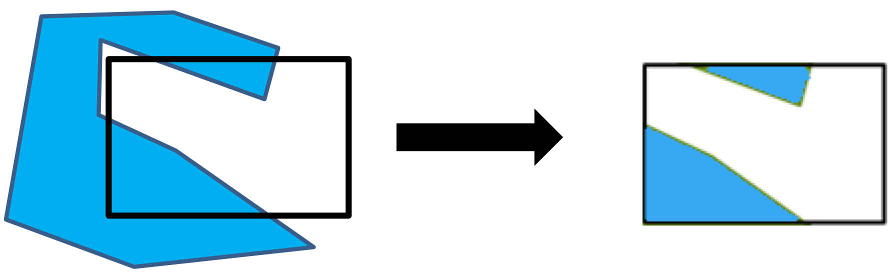


Efficiency

- In many applications, the clipping window is small relative to the size of the entire data base
  - Most line segments can be eliminated based on their outcodes.
- Inefficiency when code has to be re-executed for line segments that must be shortened in more than one step

Polygon Clipping

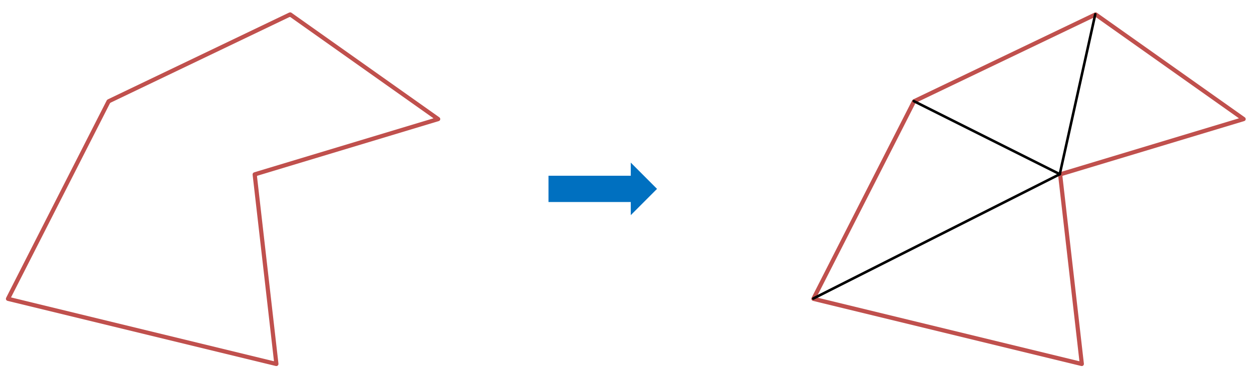
- Not as simple as line segment clipping
  - Clipping a line segment yields at most one line segment
  - Clipping a polygon can yield multiple polygons



- However, clipping a convex polygon can yield at most one other polygon

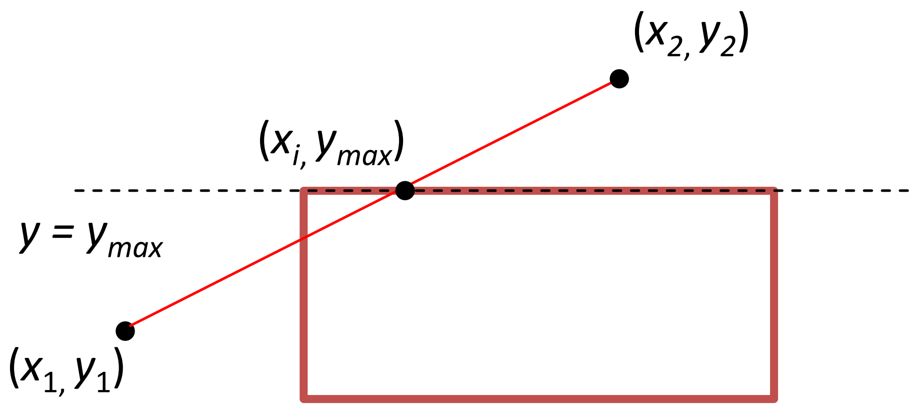
Tessellation and Convexity

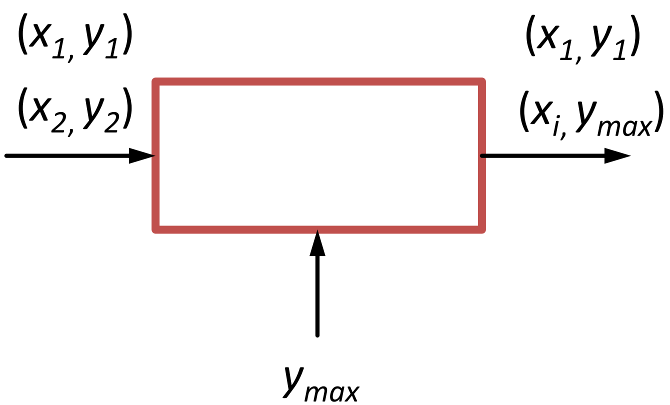
- One strategy is to replace nonconvex (concave) polygons with a set of triangular polygons (a tessellation)
- Also makes fill easier



Clipping as a Black Box

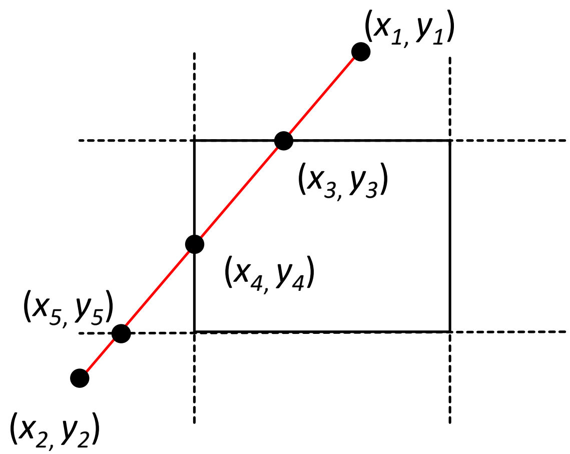
- Consider line segment clipping as a process that takes in two vertices and produces either no vertices or the vertices of a clipped line segment.

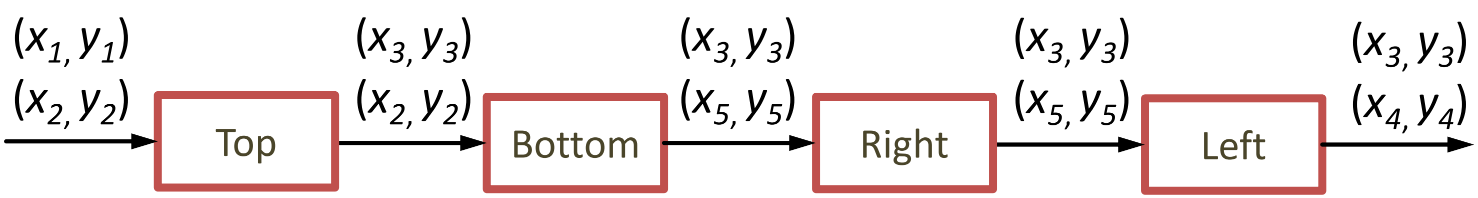




Pipeline Clipping of Line Segments

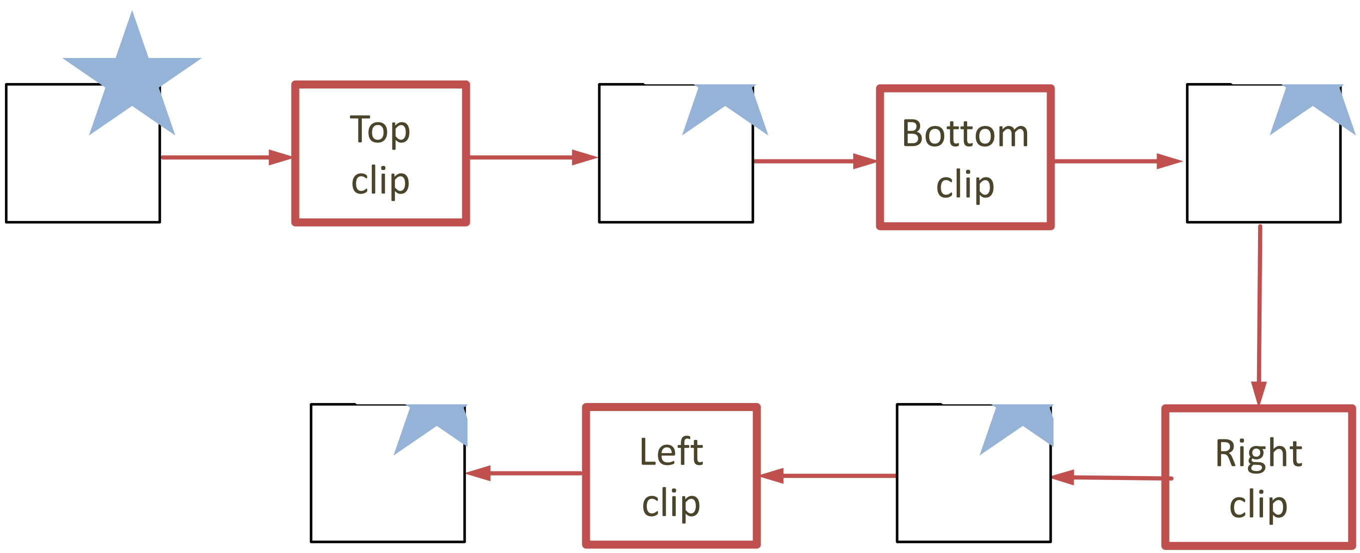
- Clipping against each side of window is independent of other sides
  - Can use four independent clippers in a pipeline





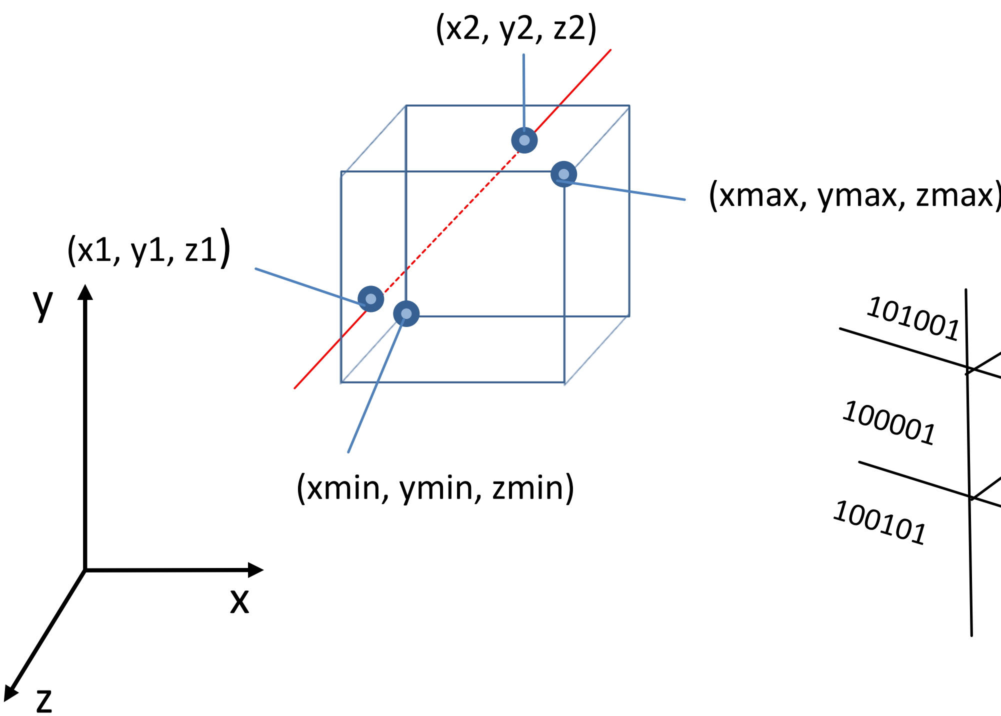
Pipeline Clipping of Polygons

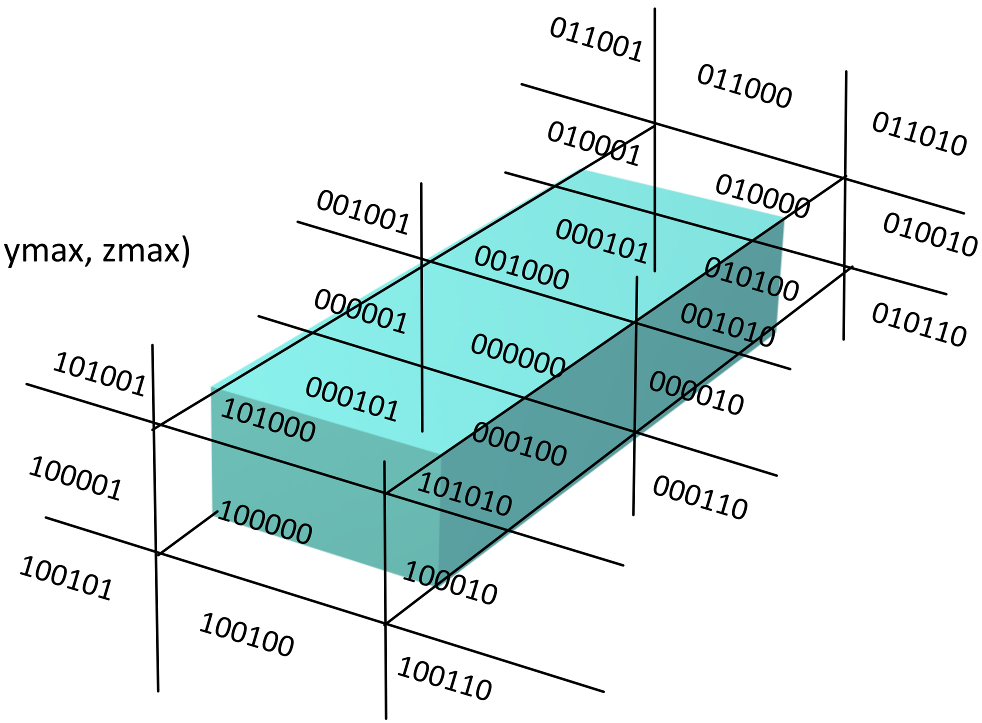
- Sutherland-Hodgman algorithm
- Strategy used in SGI Geometry Engine
- Small increase in latency



Cohen-Sutherland Method in 3D

- Use 6-bit outcodes
  - When needed, clip line segments against planes





Cohen-Sutherland Method in 3D

Check for outcodes:

$$-1 \le x_p \le 1,\ -1 \le y_p \le 1,\ -1 \le z_p \le 1$$

Since

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \xRightarrow{\text{SRT...}} \cdots\cdots \Rightarrow \begin{bmatrix} x_h \\ y_h \\ z_h \\ h \end{bmatrix} \xRightarrow{\text{Divide h}} \cdots\cdots \Rightarrow \begin{bmatrix} x_h/h \\ y_h/h \\ z_h/h \\ 1 \end{bmatrix} = \begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix}$$

To avoid unnecessary float division, We can check

$$-h \le x_h \le h,\ -h \le y_h \le h,\ -h \le z_h \le h$$

Cohen-Sutherland Method in 3D

- If outcode(A)==outcode(B)==0
  - Accept the whole line segment.
- If(outcode(A) and outcode(B))!=0
  - Reject the line segment.

- Other cases
  - Calculate an intersection (according to outcode bits)
  - Then check outcode again
- Note: use parametric forms

$$x_h = x_{ha} + (x_{hb} - x_{ha})u$$

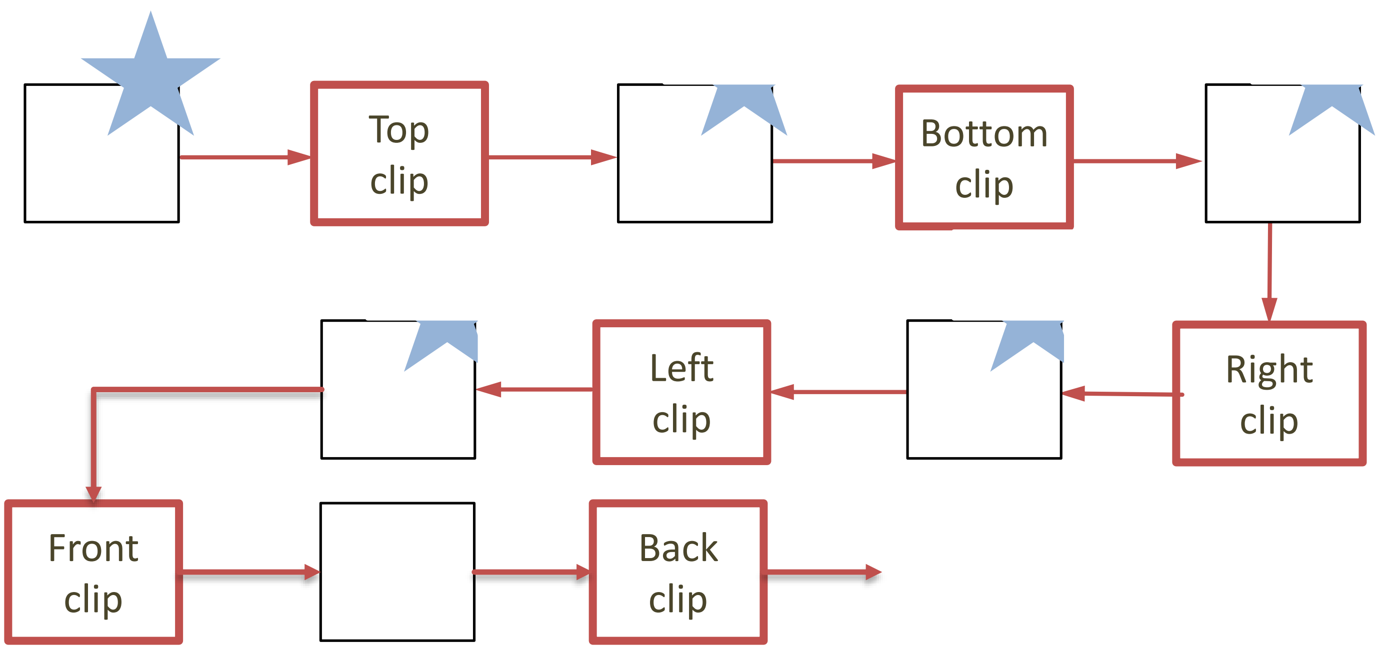
$$y_h = y_{ha} + (y_{hb} - y_{ha})u$$

$$z_h = z_{ha} + (z_{hb} - z_{ha})u$$

$$h = h_a + (h_b - h_a)u$$

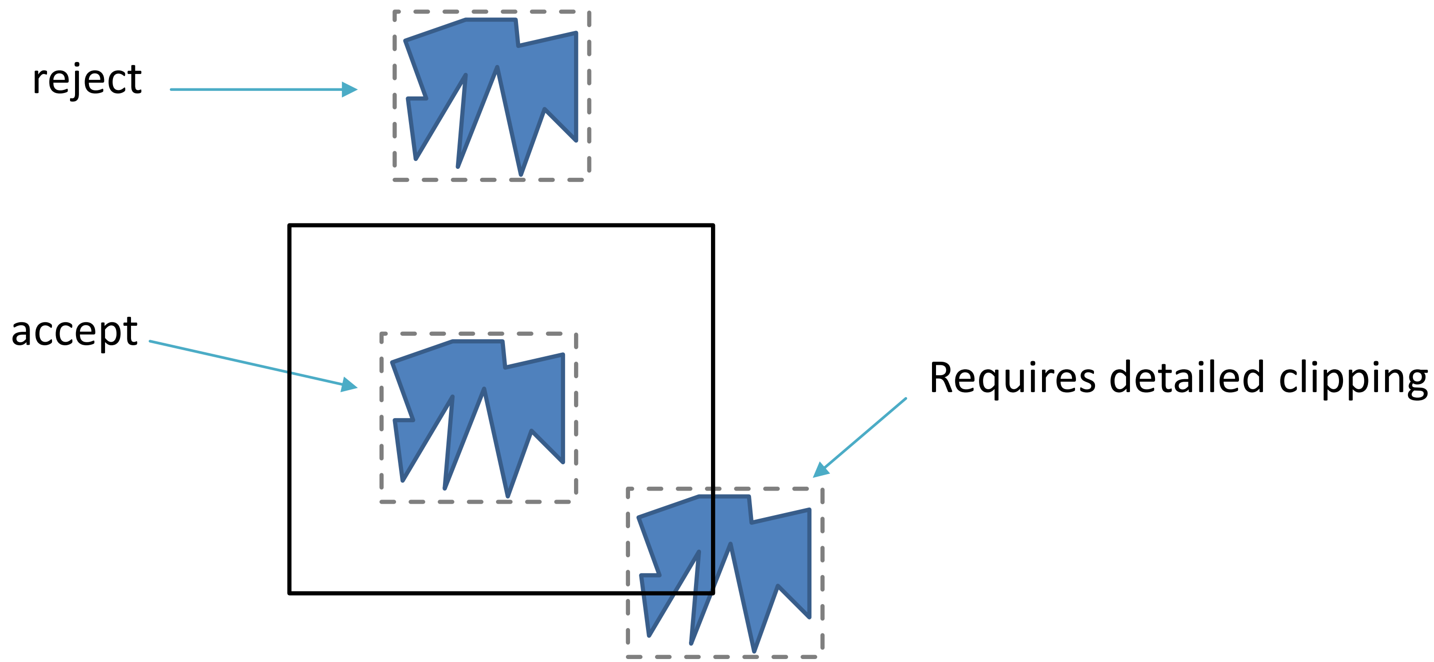
Polygon Clipping in 3D

- Similar to 2D clipping
  - Bounding box
  - Clipping with each clipping plane
  - Etc.....



Bounding Boxes

- Rather than doing clipping on a complex polygon, we can use an axis-aligned bounding box or extent
  - Smallest rectangle aligned with axes that encloses the polygon
  - Simple to compute: max and min of x and y



Usually handled by your own data structure/program, ***not within the pipeline*** *!!*