COSE436

Lecture 12: Clipping

Oct 23, 2024

Won-Ki Jeong
(wkjeong@korea.ac.kr)

KOREA UNIVERSITY

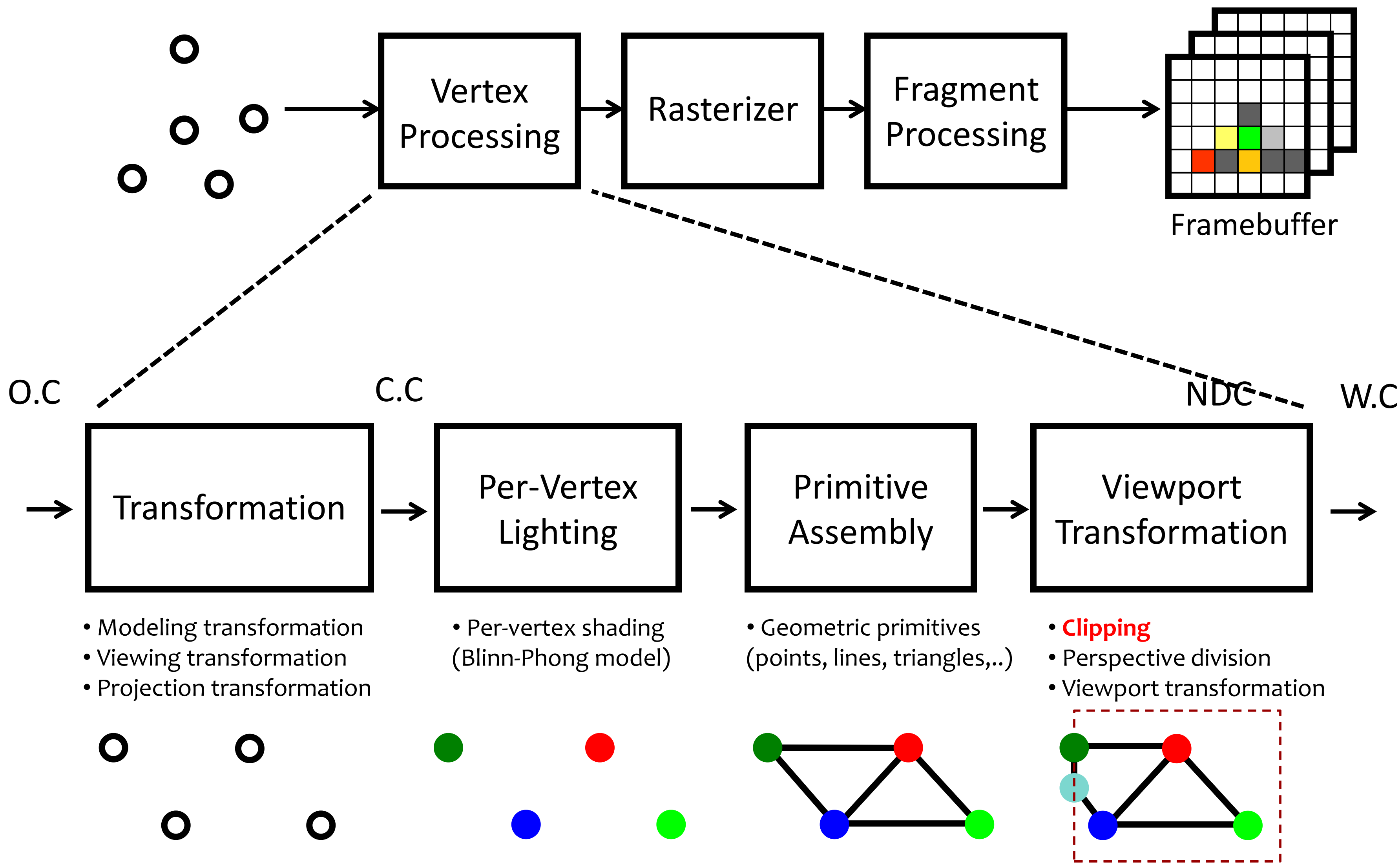
Overview

- At end of the geometry processing, vertices have been assembled into primitives
- Must clip out primitives that are outside the view frustum
  - Algorithms based on representing primitives by lists of vertices
- Must find which pixels can be affected by each primitive
  - Fragment generation
  - Rasterization or scan conversion

Required Tasks

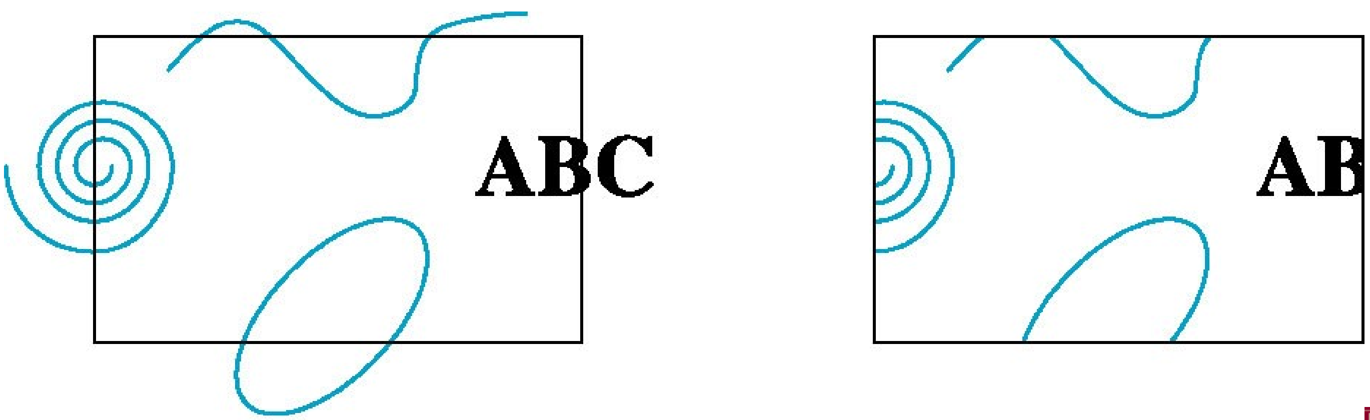
- Transformations
- Clipping
- Rasterization or scan conversion
- Some tasks deferred until fragement processing
  - Hidden surface removal
  - Antialiasing

Clipping


Vertex Processing
Rasterizer
Fragment Processing
Framebuffer
O.C
C.C
NDC
W.C
Transformation
Per-Vertex Lighting
Primitive Assembly
Viewport Transformation
• Modeling transformation
• Viewing transformation
• Projection transformation
• Per-vertex shading (Blinn-Phong model)
• Geometric primitives (points, lines, triangles,..)
• Clipping
• Perspective division
• Viewport transformation

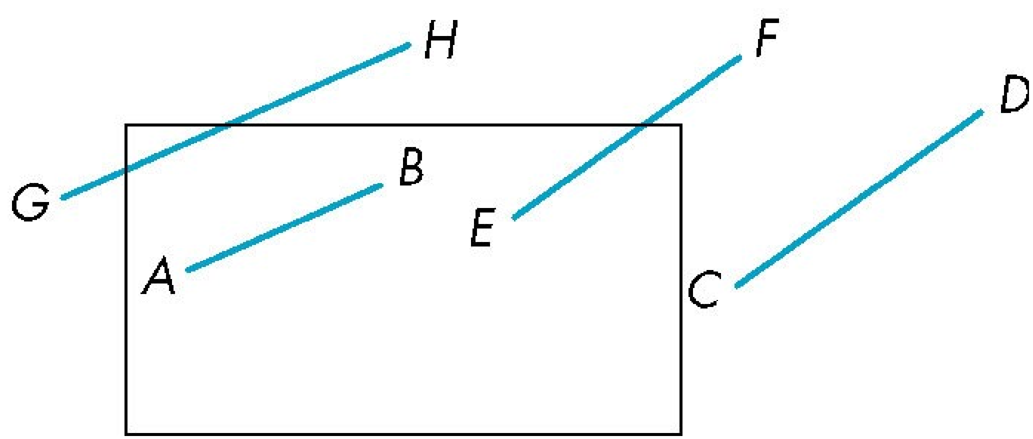
Clipping

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



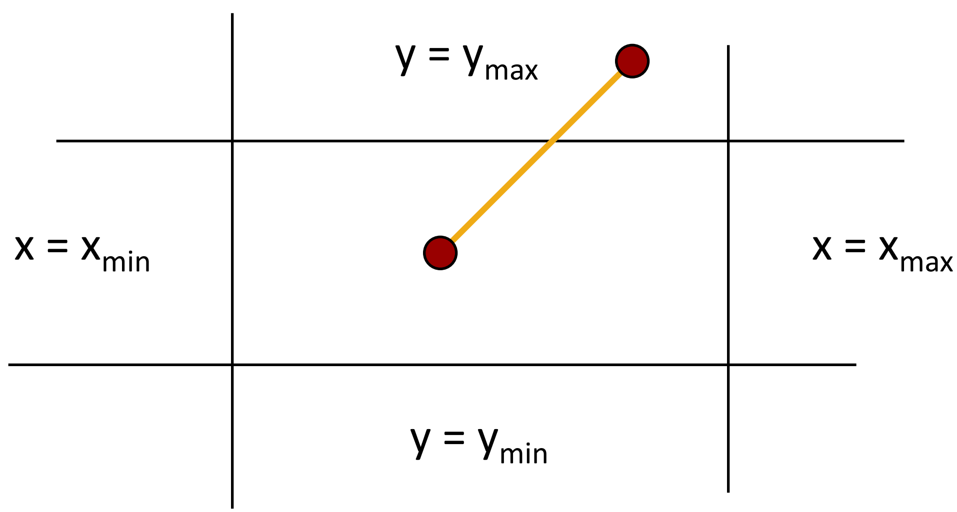
Clipping 2D Line Segments

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



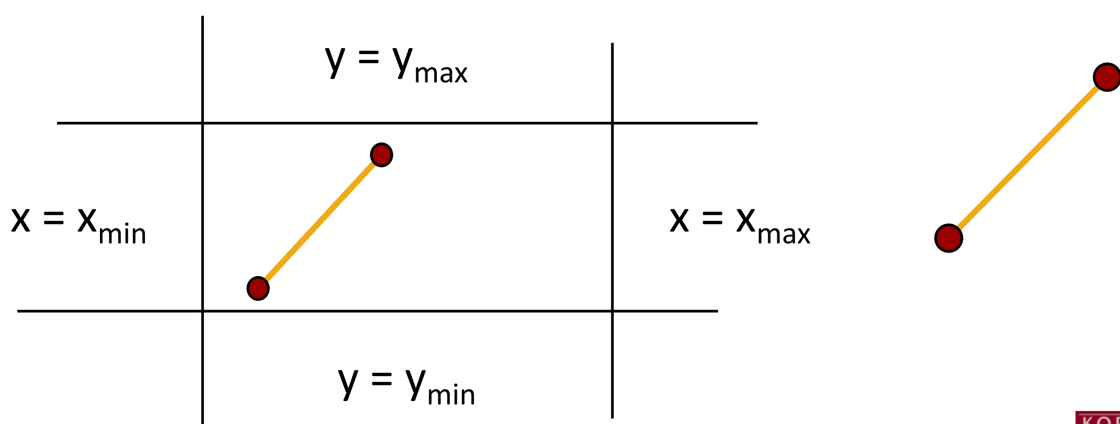
Cohen-Sutherland Algorithm

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



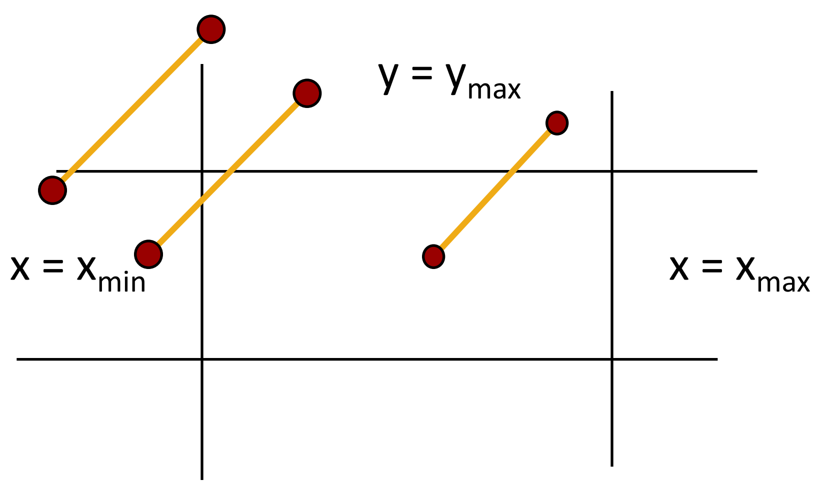
The Cases

- Case 1: both endpoints of line segment inside all four lines
  - Draw (accept) line segment as is
- Case 2: both endpoints outside all lines and on same side of a line
  - Discard (reject) the line segment



The Cases

- 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

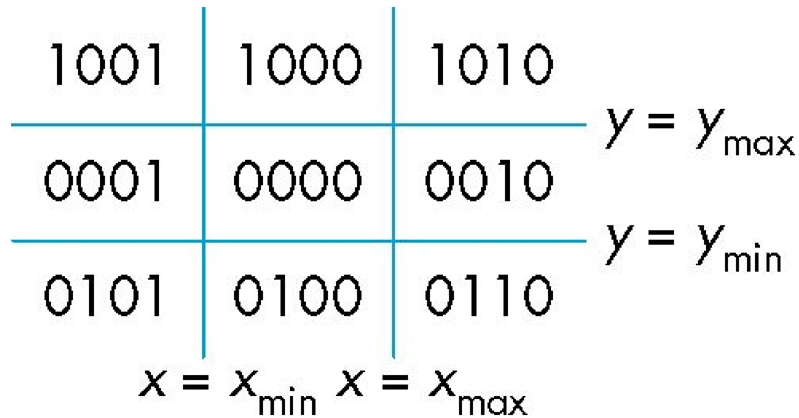


Defining Outcodes

- For each endpoint, define an outcode

$$b_0b_1b_2b_3$$

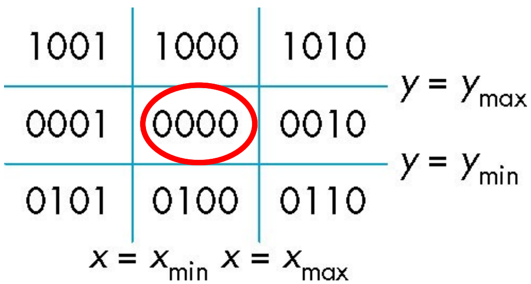
b_0 = 1 if $y > y_{max}$, 0 otherwise
b_1 = 1 if $y < y_{min}$, 0 otherwise
b_2 = 1 if $x > x_{max}$, 0 otherwise
b_3 = 1 if $x < x_{min}$, 0 otherwise

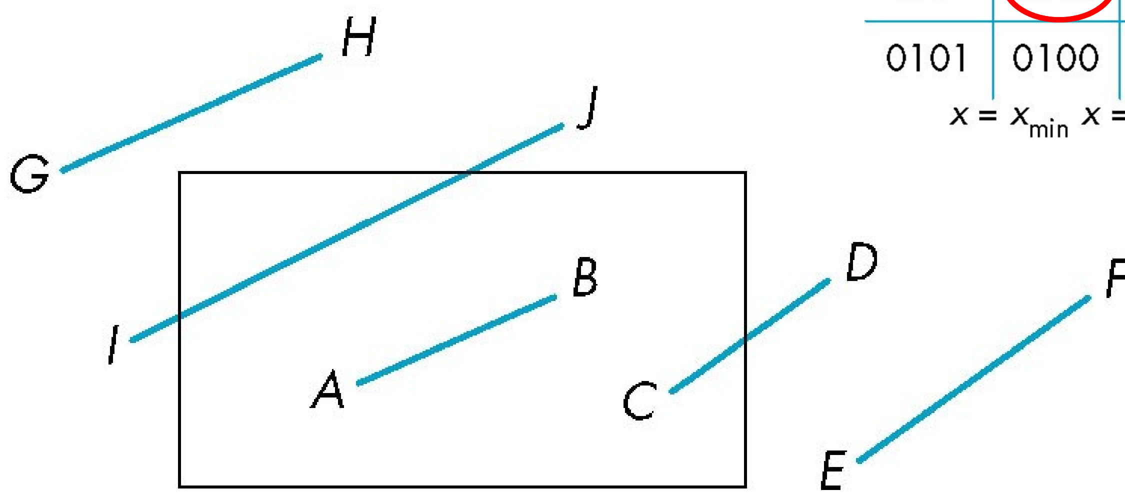


- Outcodes divide space into 9 regions
- Computation of outcode do not require fp division

Using Outcodes

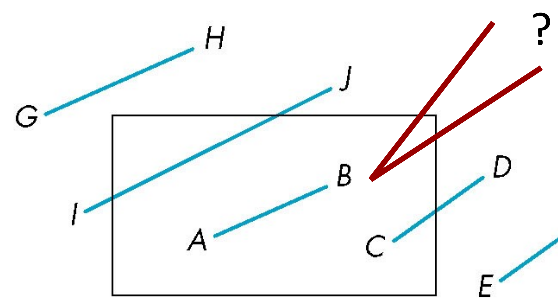
- Consider the 5 cases below
- AB: outcode(A) = outcode(B) = 0
  - Accept line segment

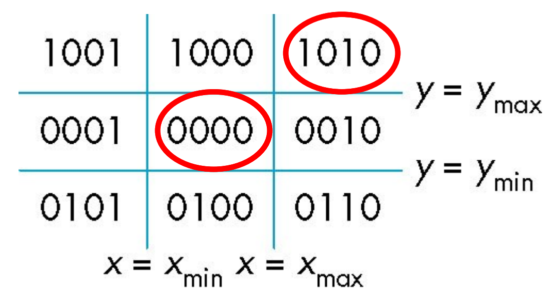




Using Outcodes

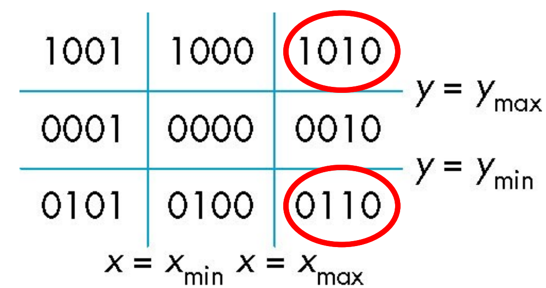
- CD: outcode (C) = 0, outcode(D) ≠ 0
  - Compute intersection
  - Location of 1 in outcode(D) determines which edge to intersect with
  - Note if there were a segment from C to a point in a region with 2 ones in outcode, we might have to do two intersections

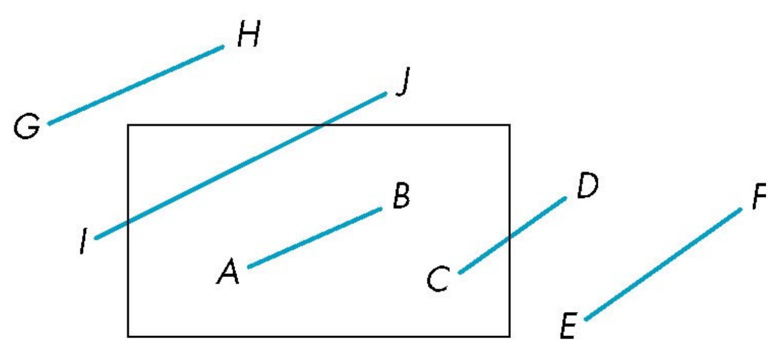




Using Outcodes

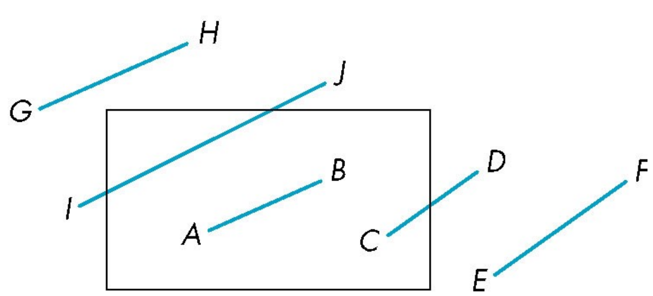
- EF: both non-zero, AND is non-zero
  - Both outcodes have a 1 bit in the same place
  - Line segment is outside of corresponding side of clipping window
  - reject

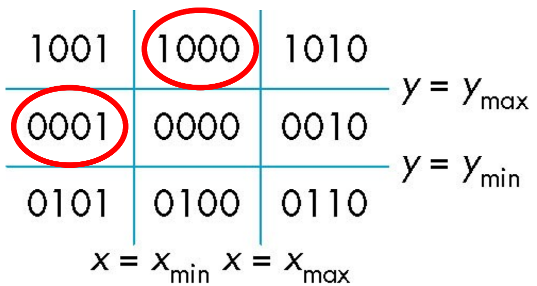




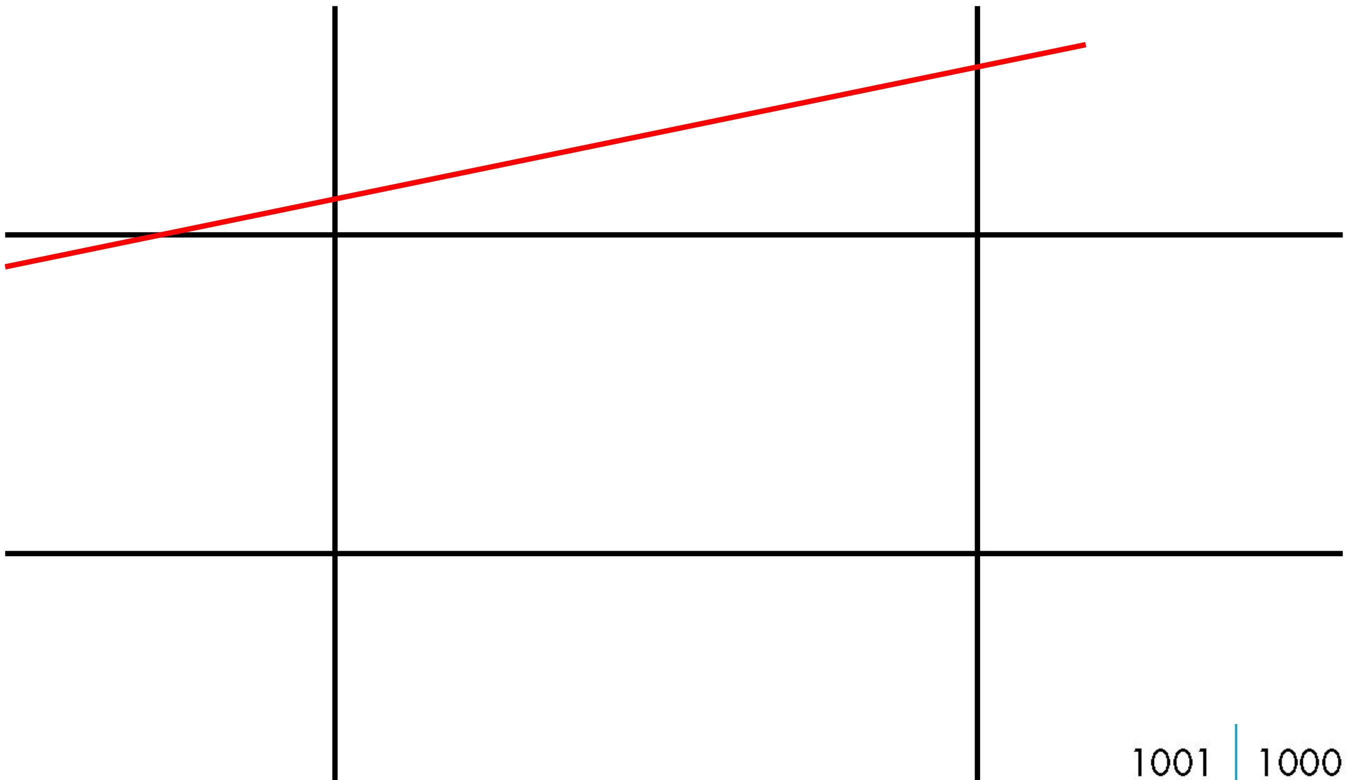
Using Outcodes

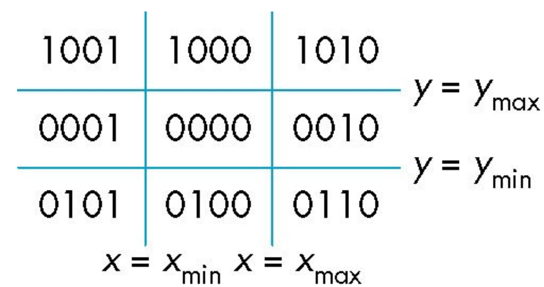
- GH and IJ: both non-zero, AND is zero
- Shorten line segment by intersecting with one of sides of window
- Compute outcode of intersection (new endpoint of shortened line segment)
- Re-execute algorithm





Example



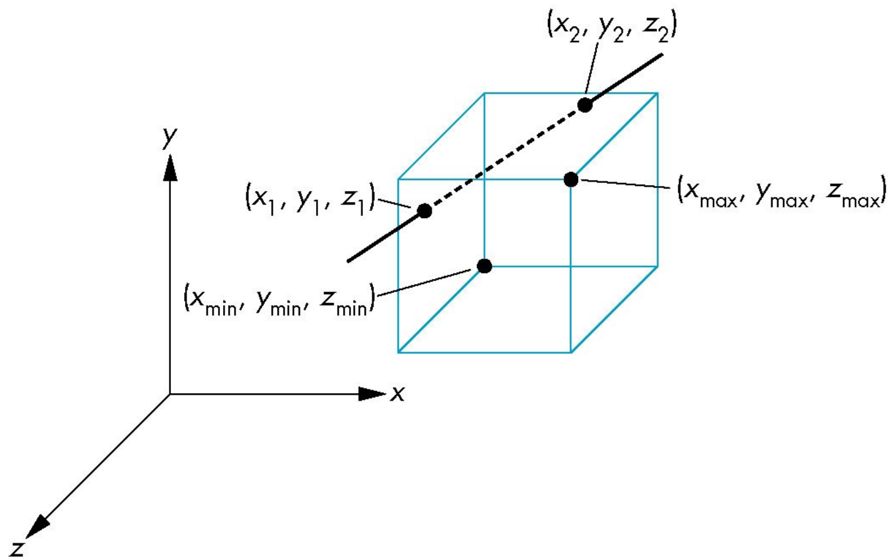

1001
1000
1010
0001
0000
0010
0101
0100
0110
y = ymax
y = ymin
x = xmin
x = xmax

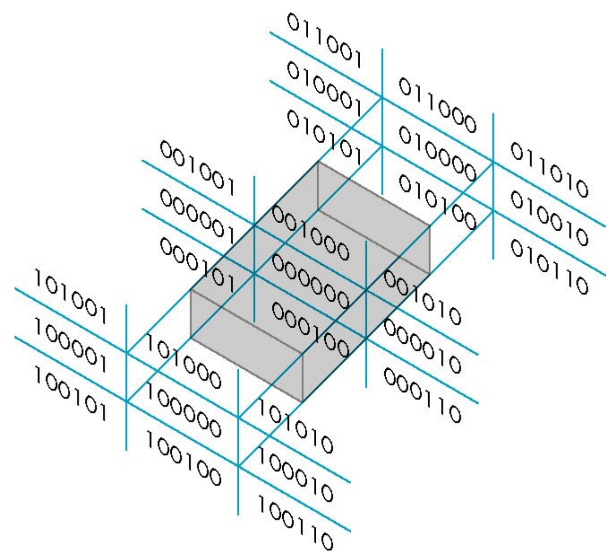
Efficiency

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

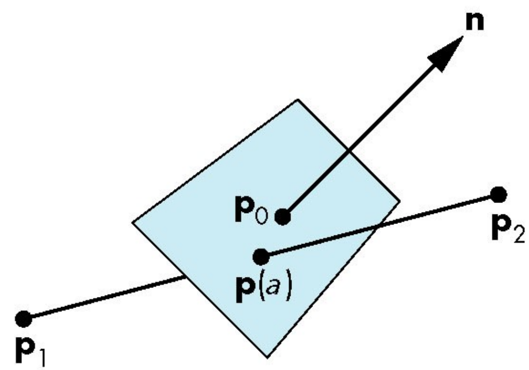
Cohen Sutherland in 3D

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





Plane-Line Intersections



$$p(\alpha) = (1-\alpha)p_1 + \alpha p_2$$

$$n \cdot (p(\alpha) - p_o) = 0$$

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

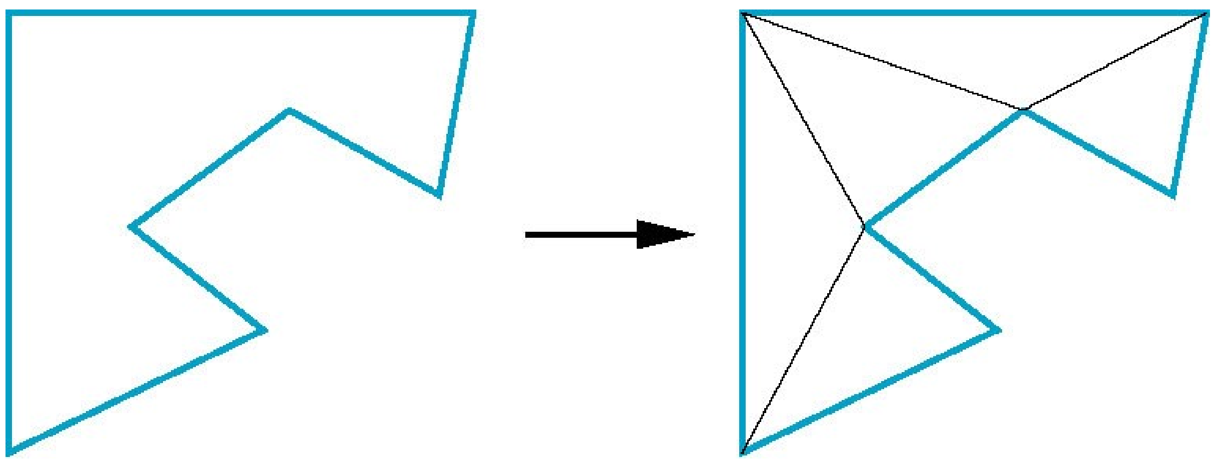
Polygon Clipping

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

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

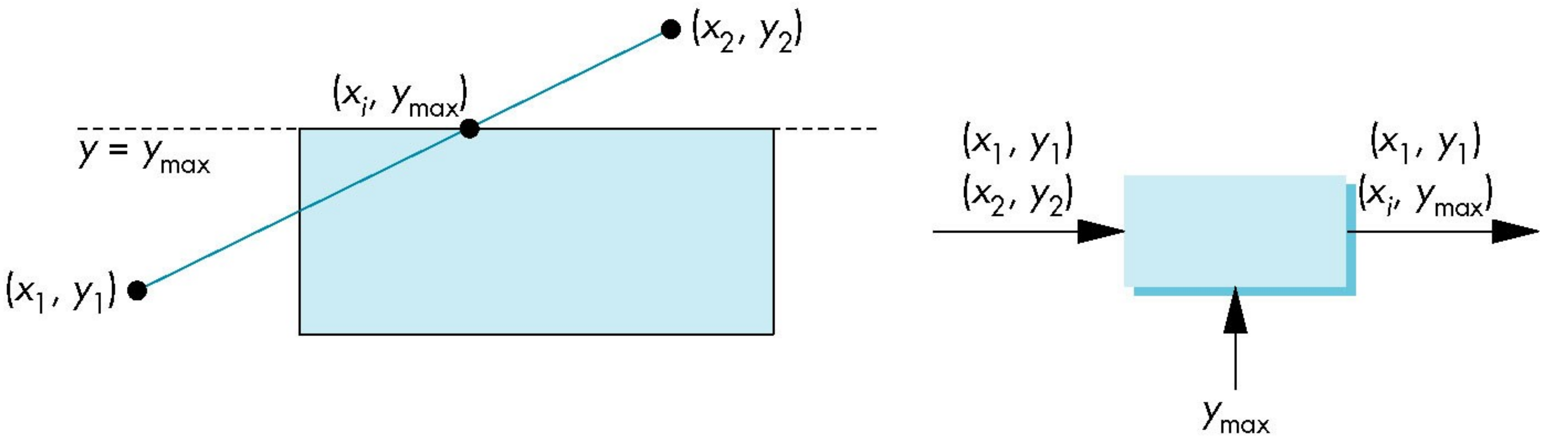
Tessellation and Convexity

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



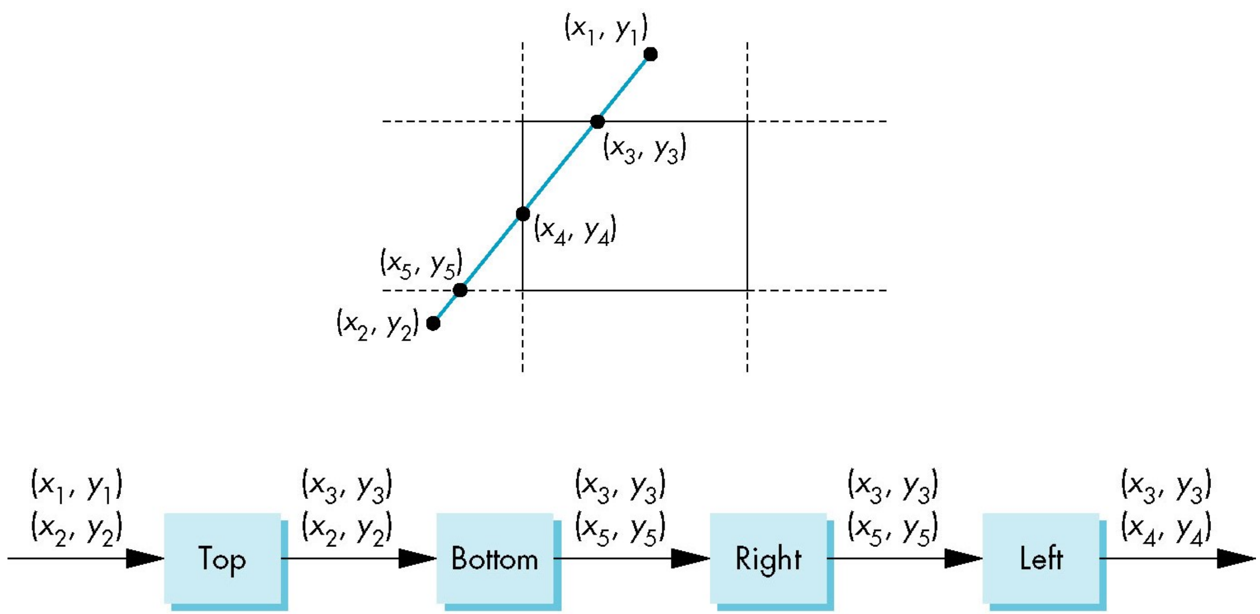
Clipping as a Black Box

- Can 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



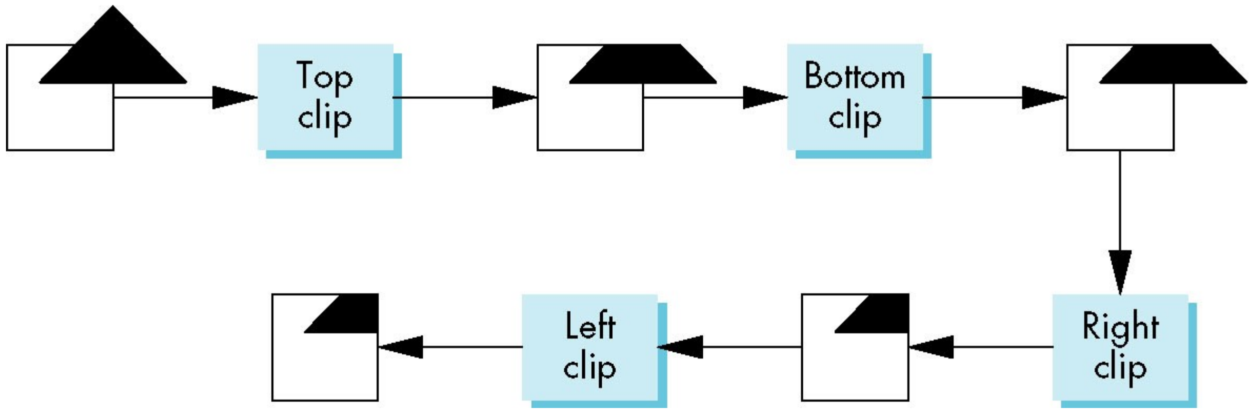
Pipeline Clipping of Line Segments

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



Sutherland & Hodgeman Algorithm

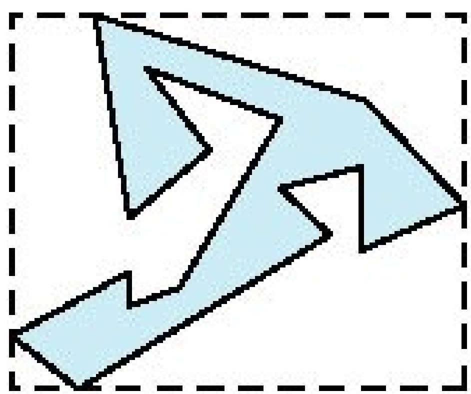
Pipeline Clipping of Polygons



- Three dimensions: add front and back clippers
- Strategy used in SGI Geometry Engine
- Small increase in latency

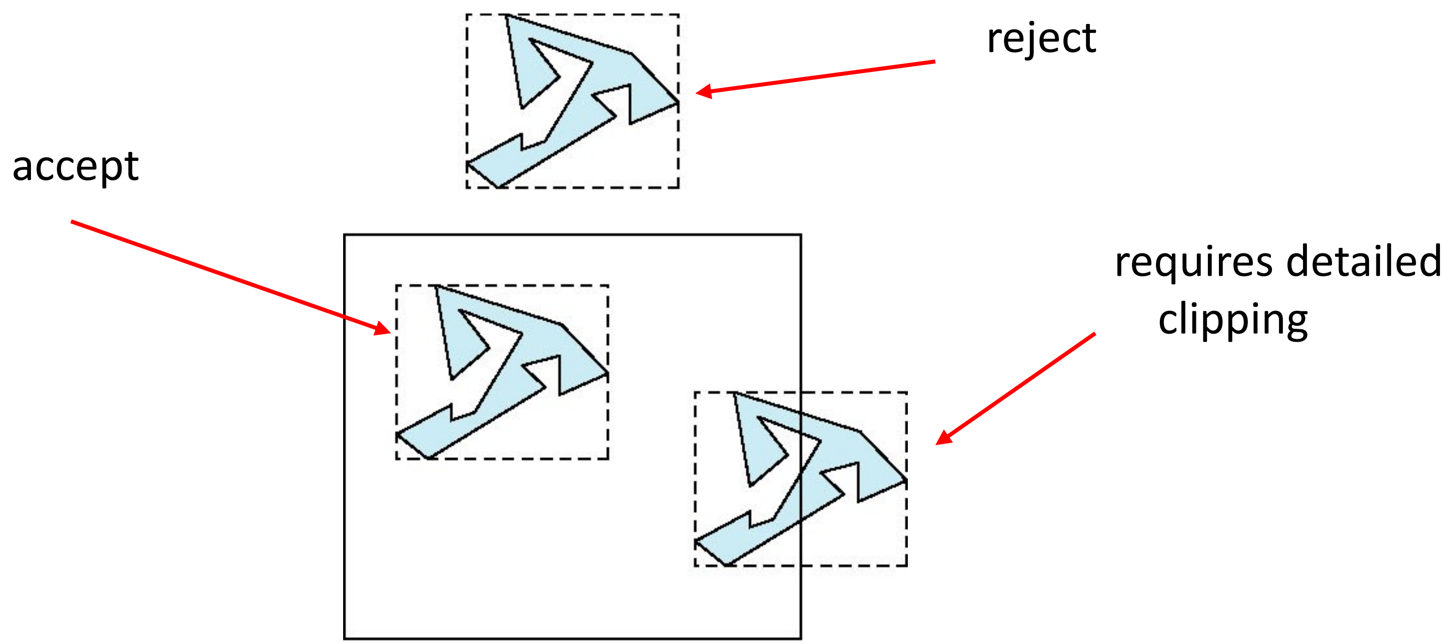
Bounding Boxes

- 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



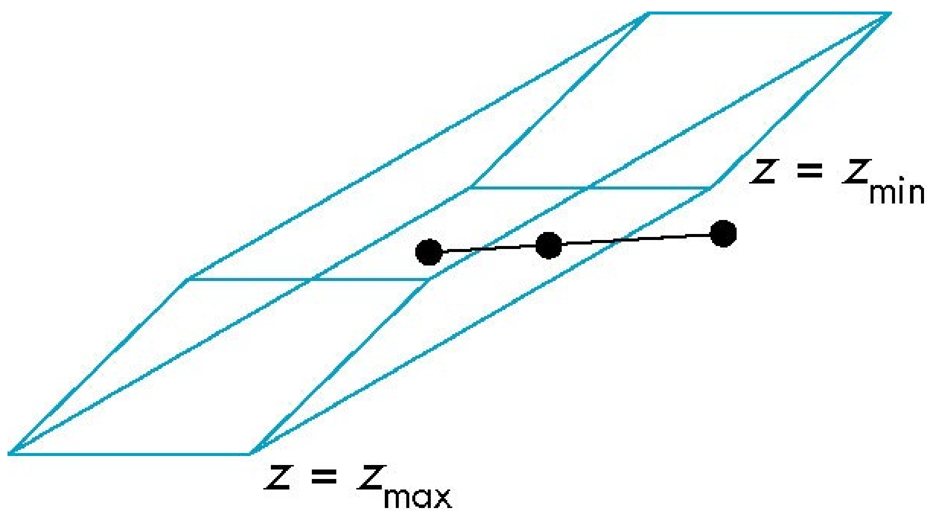
Bounding Boxes

- Can usually determine accept/reject based only on bounding box



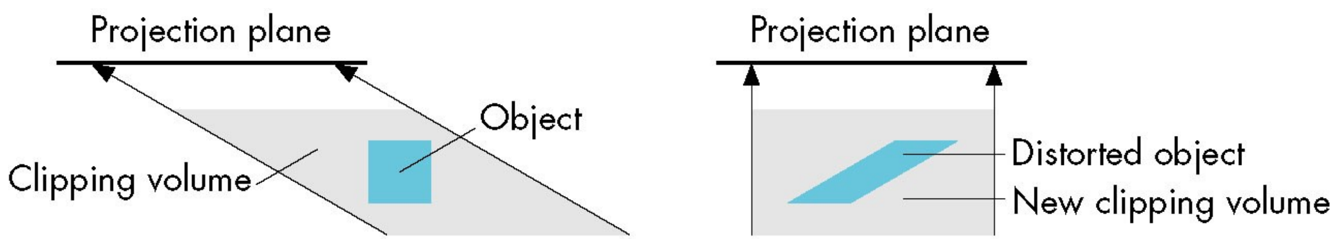
Clipping and Normalization

- General clipping in 3D requires intersection of line segments against arbitrary plane
- Example: oblique view



Normalized Form

top view



before normalization after normalization

Normalization is part of viewing (pre clipping) but after normalization, we clip against sides of right parallelepiped

Typical intersection calculation now requires only a floating point subtraction, e.g. is $x > x_{max}$?

Projection Matrix

- Orthogonal Projection
  - Near, far, left, right, top, bottom are w.r.t eye coordinate

$$\mathrm{M_P} = \mathrm{ST} = \begin{bmatrix} \frac{2}{right - left} & 0 & 0 & -\frac{right + left}{right - left} \\ 0 & \frac{2}{top - bottom} & 0 & -\frac{top + bottom}{top - bottom} \\ 0 & 0 & -\frac{2}{far - near} & -\frac{far + near}{far - near} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Projection Matrix

- Perspective Projection
  - Near, far, left, right, top, bottom are w.r.t eye coordinate

$$\begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

Clipping in Homogeneous Coordinate

- Clipping happens in *clip coordinate*
- Why not clipping in normalized device coordinate?
  - (x,y,z,w)->(x/w,y/w,z/w)->(±1,±1,±1)
  - What if w = 0?

Clipping in Homogeneous Coordinate

- Clipping plane: $x=\pm w$, $y=\pm w$, $z=\pm w$
- Example: clip to $x=-w$ (left)
  - Homogeneous coordinate: $x/w=-1$
  - Homogeneous plane: $w + x = 0$
  - Intersection between line P_1P_2 and plane $x+w=0$
    - $P=(1-u)P_1 + uP_2$ on $x+w=0$
    - $[(1-u)w_1+uw_2]+[(1-u)x_1+ux_2]=0$
    - $u=(w_1+x_1)/\{(w_1+x_1)-(w_2+x_2)\}$

Questions?