Topics

From vertices to fragments

From Vertices to Fragments

Assign a color to every pixel

Pass every object through the system

Required tasks:

- Modeling
- Geometric processing
- Rasterization
- Fragment processing

clipping

Clipping and Visibility

Clipping has much in common with hidden-surface removal

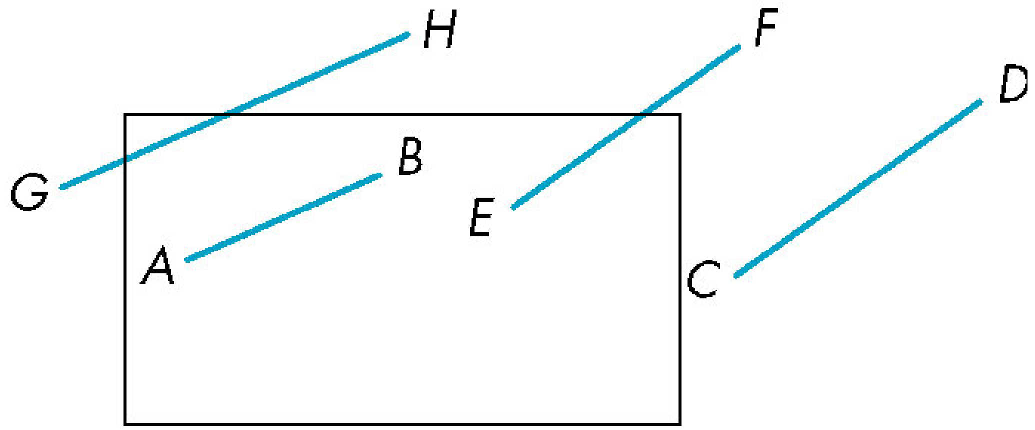
In both cases, we are trying to remove objects that are not visible to the camera

Often we can use visibility or occlusion testing early in the process to eliminate as many polygons as possible before going through the entire pipeline

Clipping 2D Line Segments

Brute force approach: compute intersections with all sides of clipping window

- Inefficient: one division per intersection

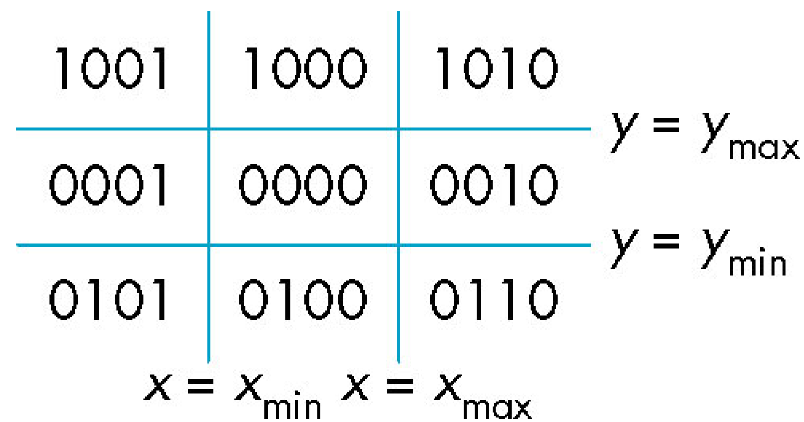


Cohen-Sutherland Algorithm

Idea: eliminate as many cases as possible without computing intersections

For each endpoint, define an outcode $b_0 b_1 b_2 b_3$

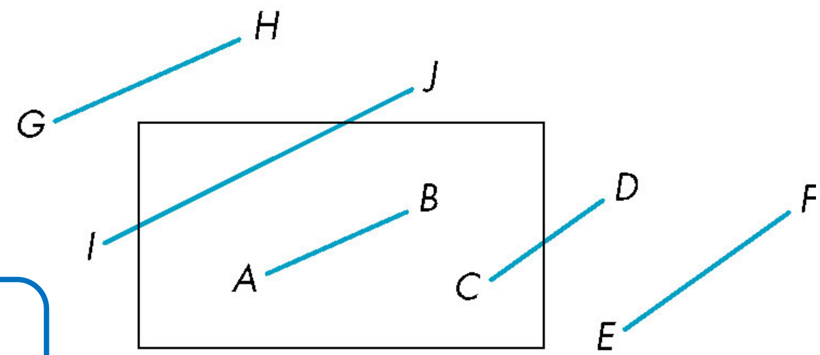
$b_0 = 1$ if $y > y_{max}$, 0 otherwise
$b_1 = 1$ if $y < y_{min}$, 0 otherwise
$b_2 = 1$ if $x > x_{max}$, 0 otherwise
$b_3 = 1$ if $x < x_{min}$, 0 otherwise



Computation of outcode requires at most 4 subtractions

Using Outcodes

Consider the 5 cases below



AB: outcode(A) = outcode(B) = 0

- Accept line segment

CD: outcode (C) = 0, outcode(D) $\neq$ 0

- Compute intersection
- Location of 1 in outcode(D) determines which edge to intersect with

Both outcodes are nonzero for other 3 cases, perform AND

- EF: outcode(E) AND outcode(F) (bitwise) $\neq$ 0
  - –reject
- GH and IJ: outcode(G) AND outcode(H) =0
  - –Shorten line segment by intersecting with one of sides of window and reexecute algorithm

Efficiency

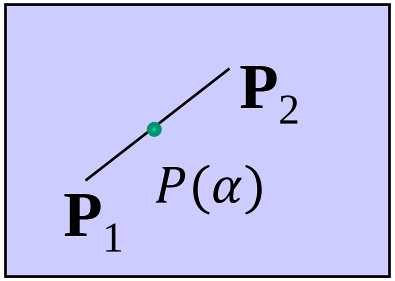
Inefficient when code has to be reexecuted for line segments that must be shortened in more than one step

➡ For the last case, use Liang-Barsky Clipping

Liang-Barsky Clipping

Consider the parametric form of a line segment

$$P(\alpha) = \begin{bmatrix} x(\alpha) \\ y(\alpha) \end{bmatrix} = (1-\alpha)P_1 + \alpha P_2 \quad 1 \geq \alpha \geq 0$$



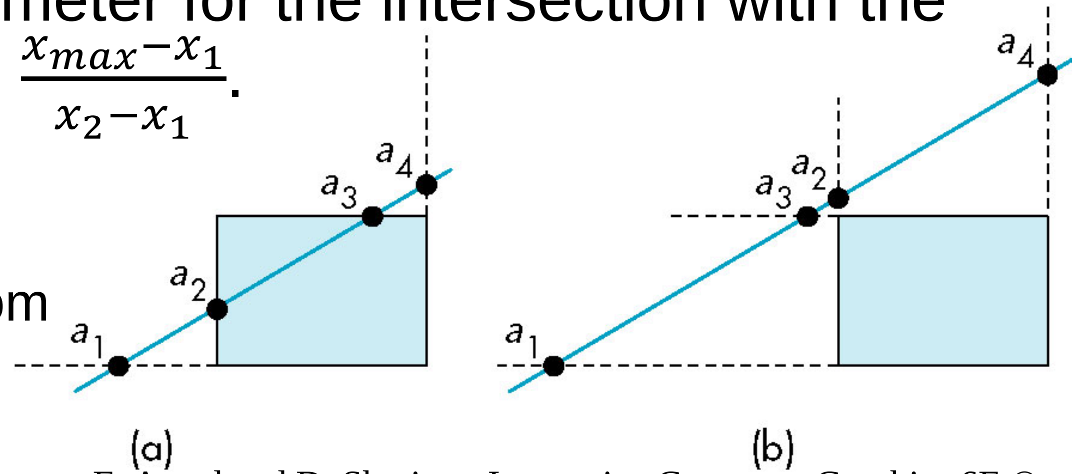
$$P(\alpha) = \begin{bmatrix} x(\alpha) \\ y(\alpha) \end{bmatrix} = \begin{bmatrix} (1-\alpha)x_1 + \alpha x_2 \\ (1-\alpha)y_1 + \alpha y_2 \end{bmatrix}$$

We can distinguish between the cases by looking at the ordering of the values of α where the line determined by the line segment crosses the lines that determine the window

Liang-Barsky Clipping

When the line is not parallel to a side of the window, compute intersections with the sides of window

For example, α_4 is the parameter for the intersection with the right side $x = x_{max} \Longrightarrow \alpha_4 = \frac{x_{max} - x_1}{x_2 - x_1}$.



E. Angel and D. Shreiner: Interactive Computer Graphics 6E © Addison-Wesley 2012

In (a): $\alpha_4 > \alpha_3 > \alpha_2 > \alpha_1$

- Intersect right, top, left, bottom
- shorten

In (b): $\alpha_4 > \alpha_2 > \alpha_3 > \alpha_1$

- Intersect both right and left before intersecting top and bottom
- reject

Advantages

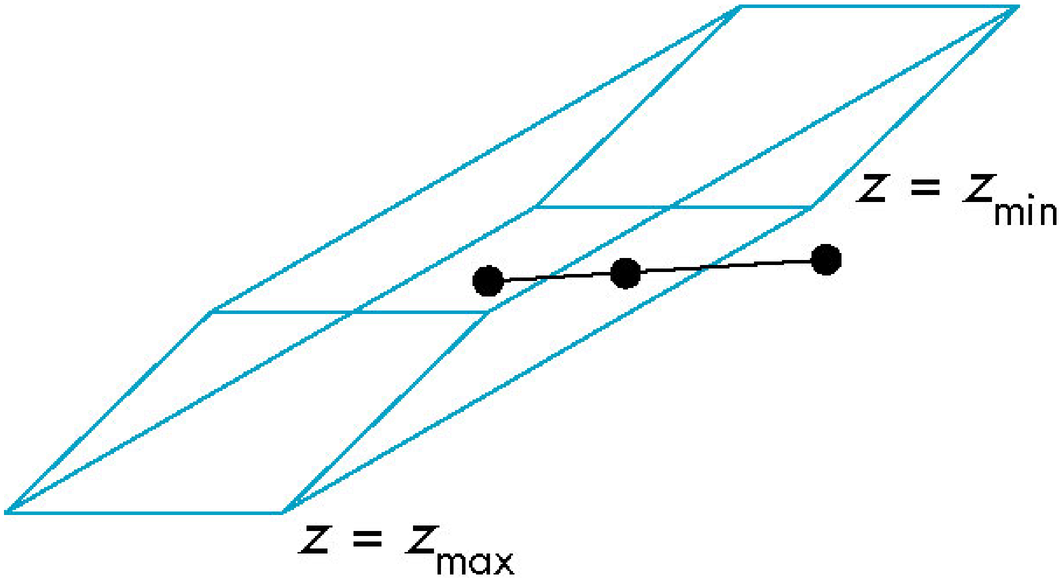
Using values of α, we do not have to use algorithm recursively as with C-S

Can be extended to 3D

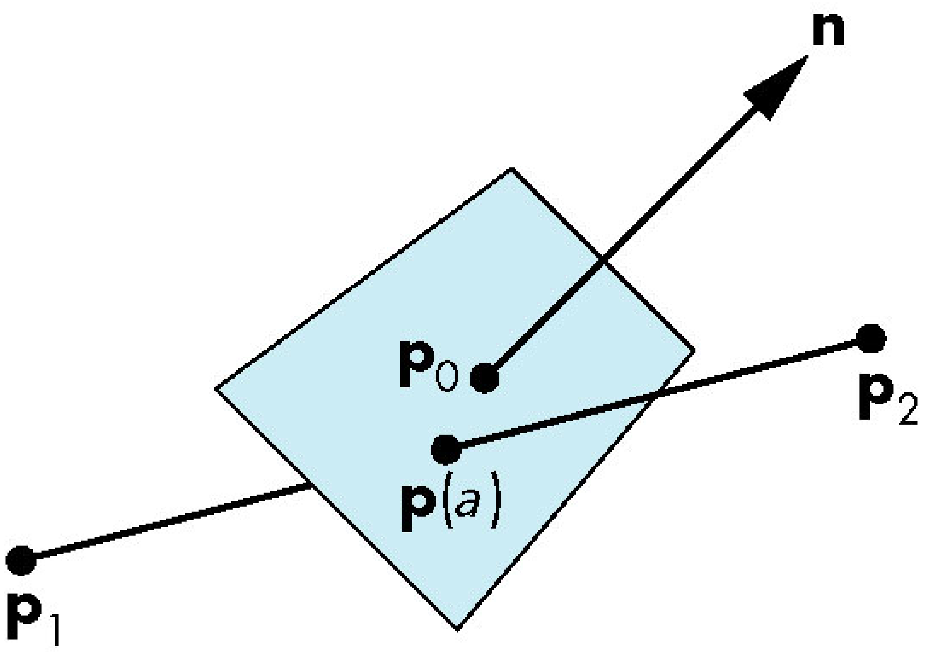
Clipping and Normalization

General clipping in 3D requires intersection of line segments against arbitrary plane

Example: oblique view



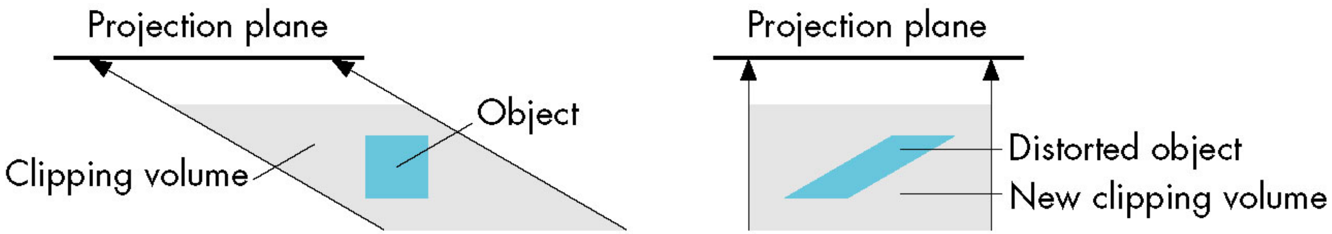
Plane-Line Intersections



$$a = \frac{n \bullet (p_o - p_1)}{n \bullet (p_2 - p_1)}$$

Normalized Form

top view



before normalization after normalization

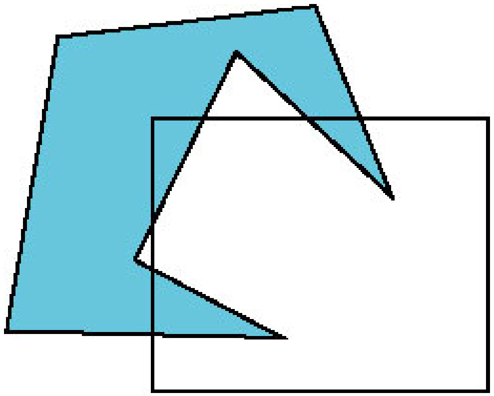
Normalization is part of viewing (pre clipping). After normalization, we clip against sides of right parallelepiped

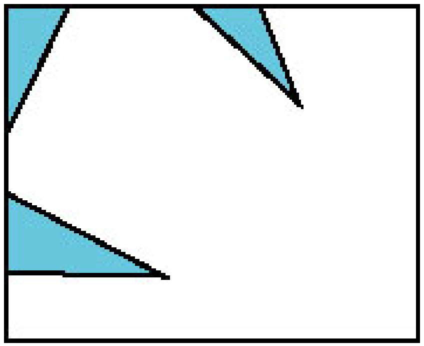
Typical intersection calculation now requires only a floating point subtraction, e.g. is $x > x_{max}$?

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
  - –Increase number of polygons

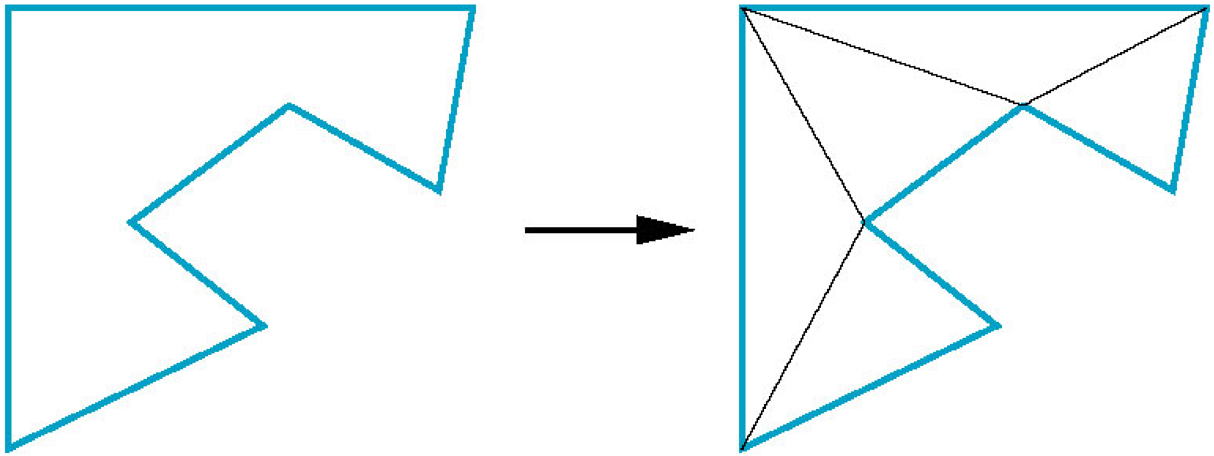




Clipping a convex polygon can yield at most one other polygon

Tessellation and Convexity

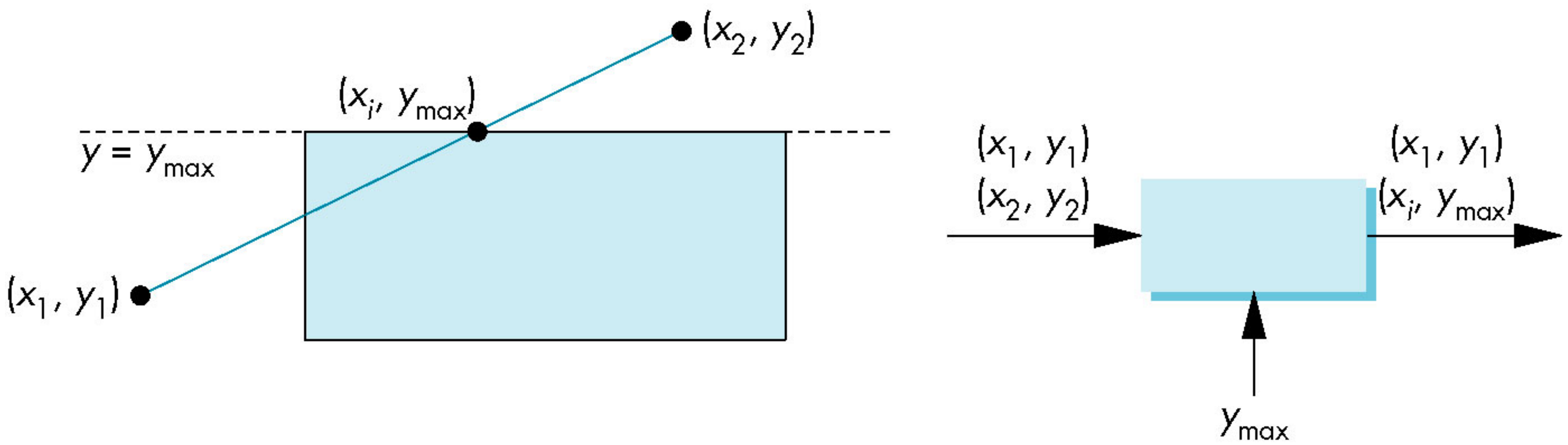
One strategy is to replace nonconvex (*concave*) polygons with a set of triangular polygons (a *tessellation*)



Apply line-segment clipping algorithms to each edge of the polygon

Clipping as a Black Box

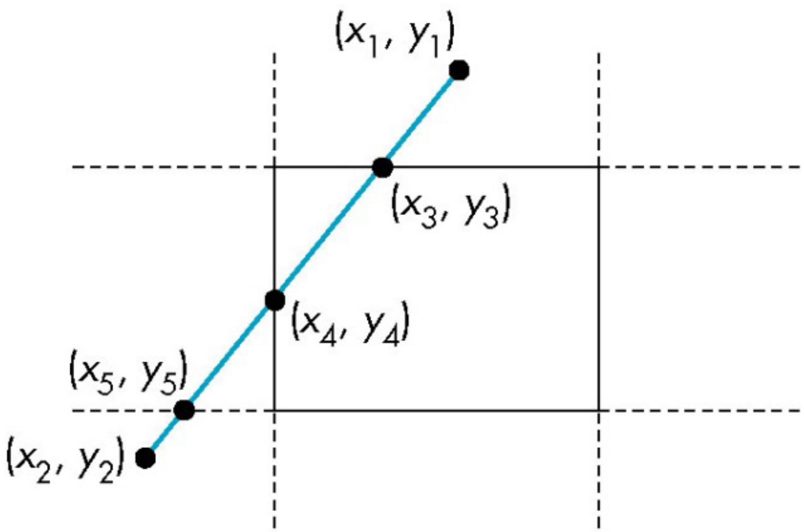
Can consider line segment clipping as a process that takes in two vertices and produces either no vertices or the vertices of a clipped line segment



Pipeline Clipping of Line Segments

Clipping against each side of window is independent of other sides

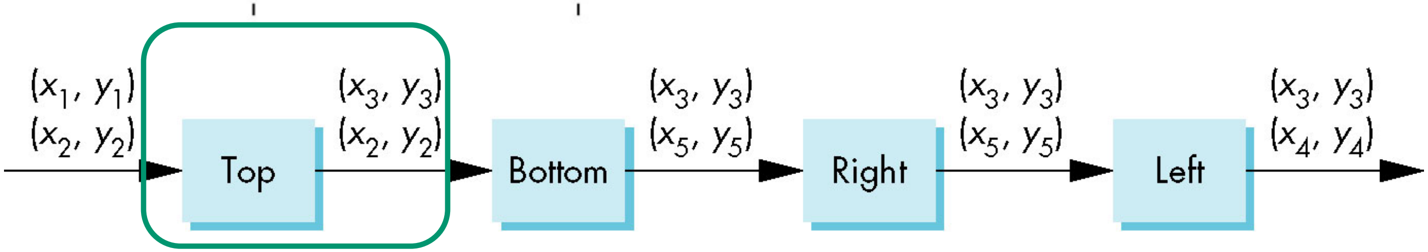
- Can use four independent clippers in a pipeline



For example,

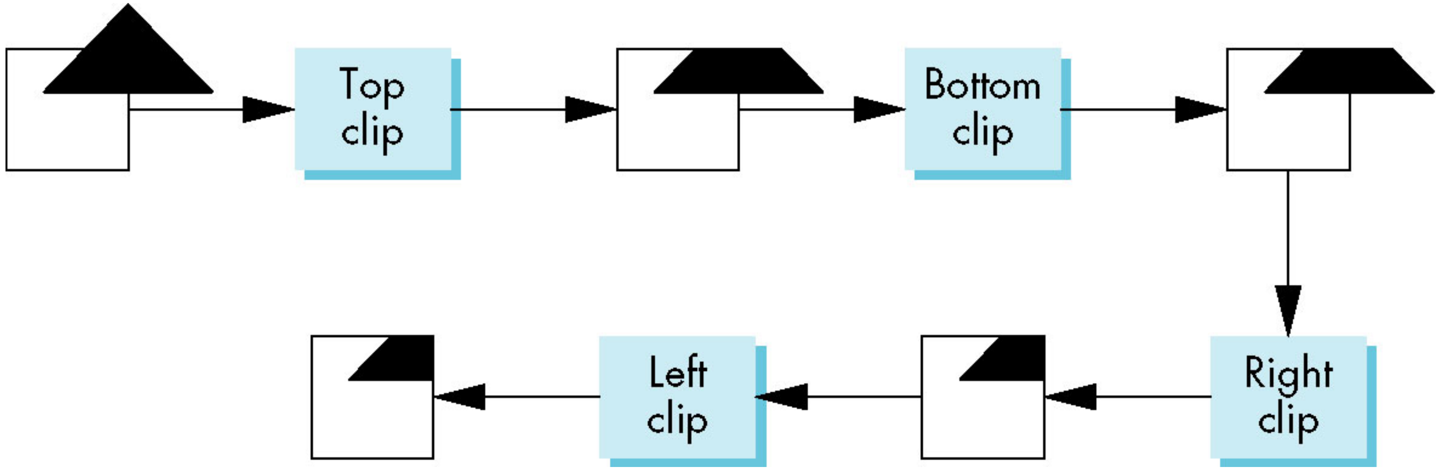
$$x_3 = x_1 + (y_{max} - y_1)\frac{x_2 - x_1}{y_2 - y_1}$$

$$y_3 = y_{max}$$



Pipeline Clipping of Polygons

For all edges of polygon, run the pipeline



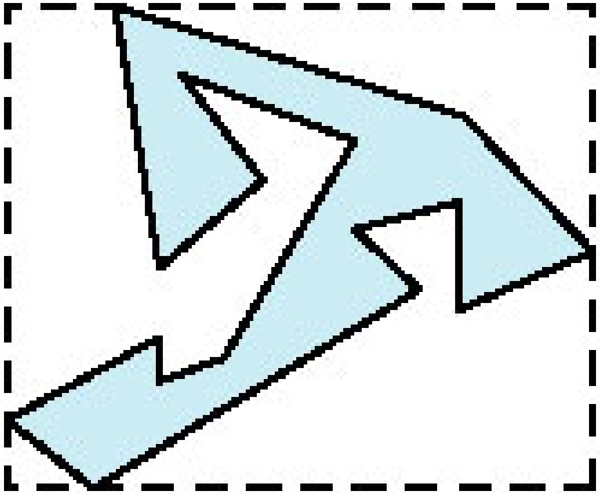
Three dimensions: add front and back clippers

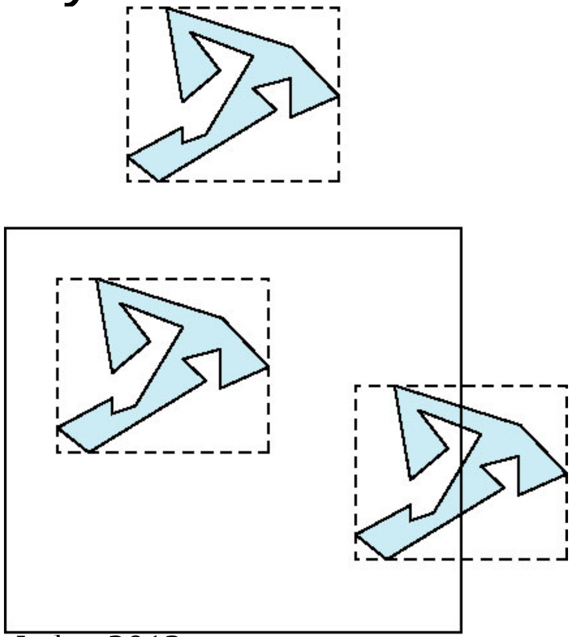
Not efficient for many-sided polygon

Bounding Boxes

Rather than doing clipping on a complex polygon, we can use an ***axis-aligned bounding box*** **or** ***extent***

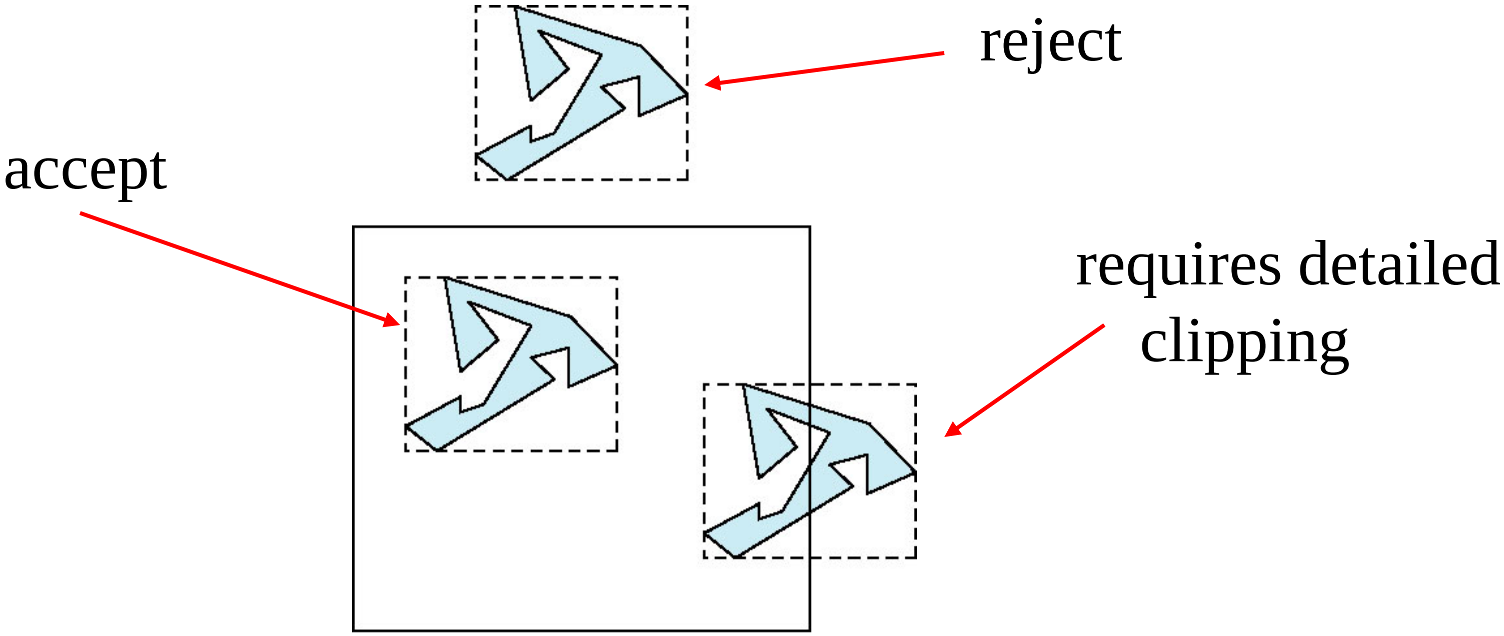
- Smallest rectangle aligned with axes that encloses the polygon
- Simple to compute: max and min of x and y
- Avoid detailed clipping for all cases





Bounding boxes

Can usually determine accept/reject based only on bounding box



Rasterization

After clipping, the remaining primitives are inside the view volume

The color buffer is an $n \times m$ array, (0,0) for the lower-left corner

- Pixels are discrete
- Square centered at halfway between integers in OpenGL

Rasterization (scan conversion)

- Determine which pixels are inside primitive specified by a set of vertices
- Produces a set of fragments
- Fragments have a location (pixel location) in the buffer and other attributes such color and texture coordinates that are determined by interpolating values at vertices

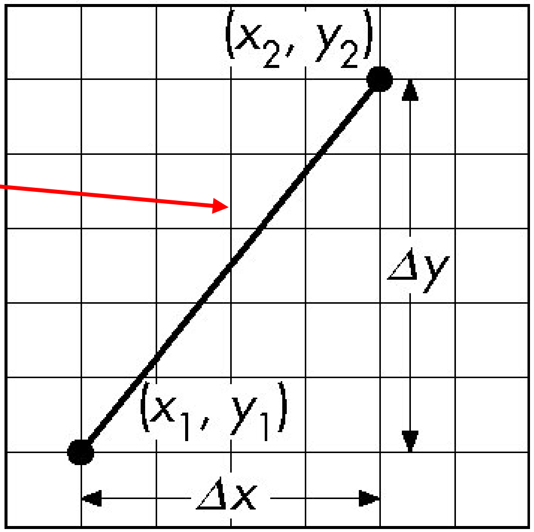
Scan Conversion of Line Segments

Start with line segment in window coordinates with integer values for endpoints

Assume implementation has a `write_pixel` function

$$m = \frac{\Delta y}{\Delta x}$$

y = mx + h



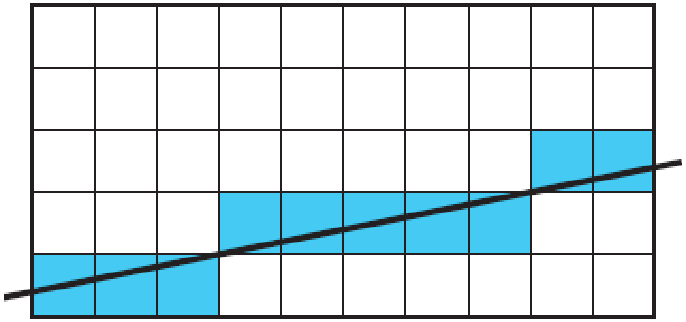
DDA Algorithm

<u>D</u>igital <u>D</u>ifferential <u>A</u>nalyzer

- DDA was a mechanical device for numerical solution of differential equations
- Line $y = mx + h$ satisfies differential equation

$$\frac{dy}{dx} = m = \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1} \longrightarrow \text{two endpoints}$$

Along scan line Δx = 1

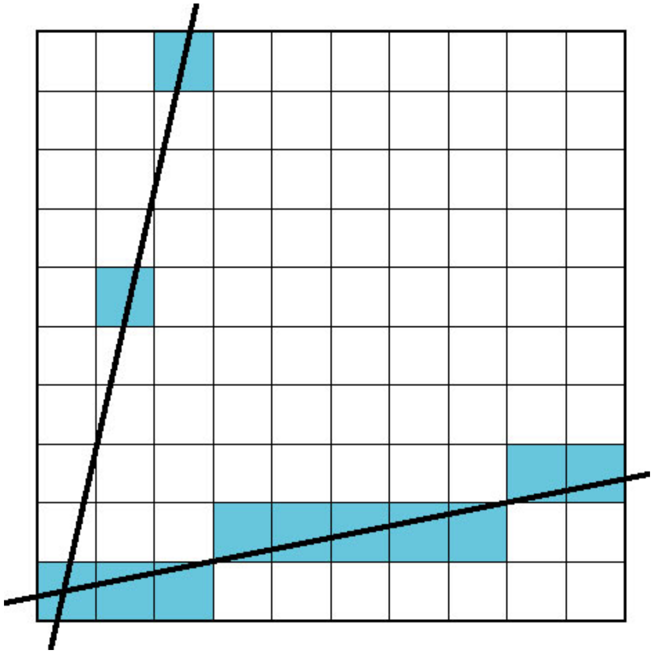


```
For(x=x1; x<=x2,ix++) {
   y+=m;
  write_pixel(x, round(y), line_color)
}
```

Problem

DDA = for each x plot pixel at closest y

- Problems for steep lines

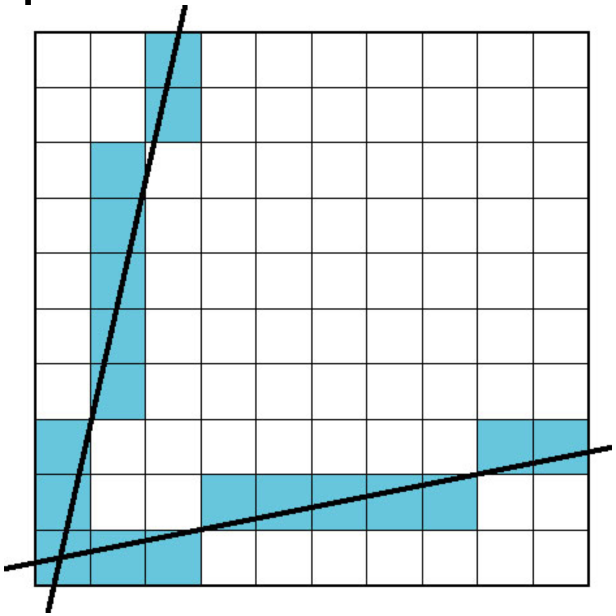


Using Symmetry

Use for $0 \leq m \leq 1$, for each x plot pixel at closest y

For m > 1, swap role of x and y

- For each y, plot closest x



Bresenham's Algorithm

m is a floating point

DDA requires one floating point addition per step

Bresenham's algorithm eliminates all fp calculations

- Standard algorithm for rasterizers

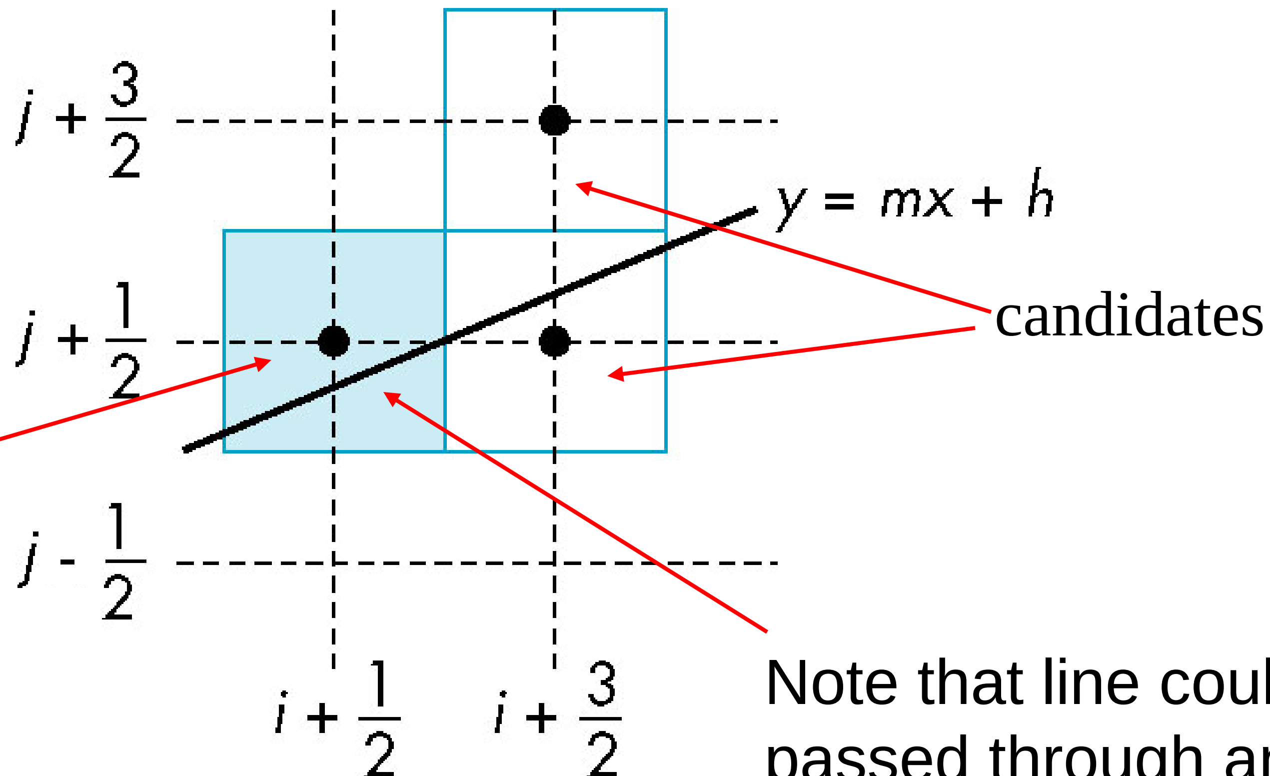
Consider only $0 \leq m \leq 1$, other cases by symmetry

Assume pixel centers are at half integers

If we start at a pixel that has been written, there are only two candidates for the next pixel to be written into the frame buffer

Candidate Pixels

$0 \leq m \leq 1$



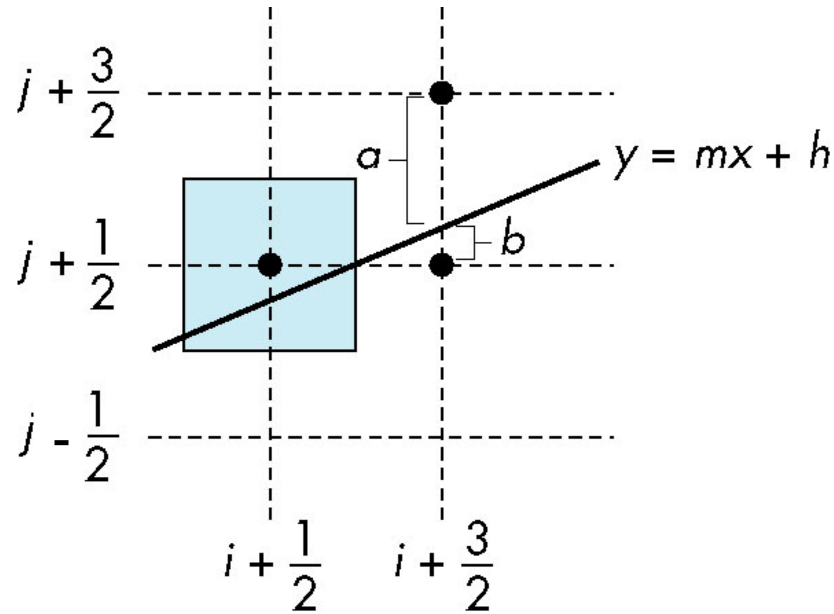
last pixel

Note that line could have passed through any part of this pixel

Decision Variable

a and b are the distances between the line and the upper/lower candidate

$$
\begin{aligned}
d &= (x_2 - x_1)(a - b) \\
&= \Delta x(a - b)
\end{aligned}
$$



d is an integer, why?
d < 0 use upper pixel
d > 0 use lower pixel

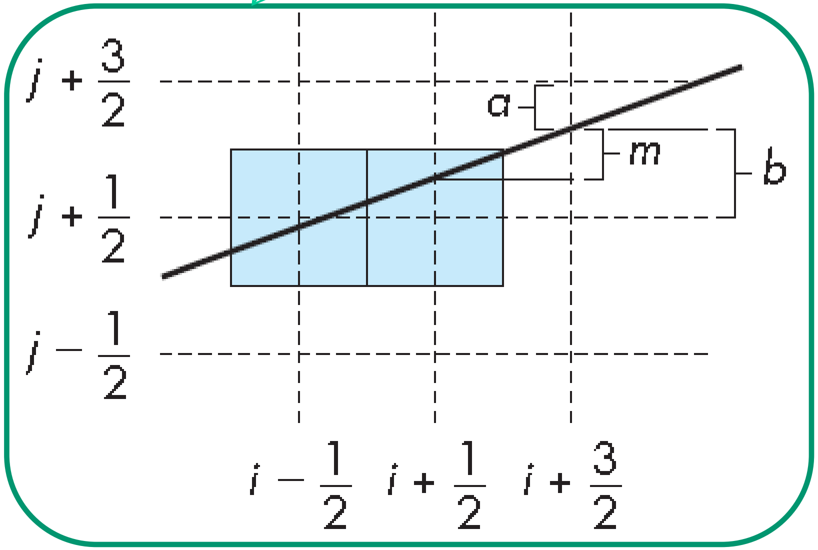
Replacing floating-point with fixed-point operations

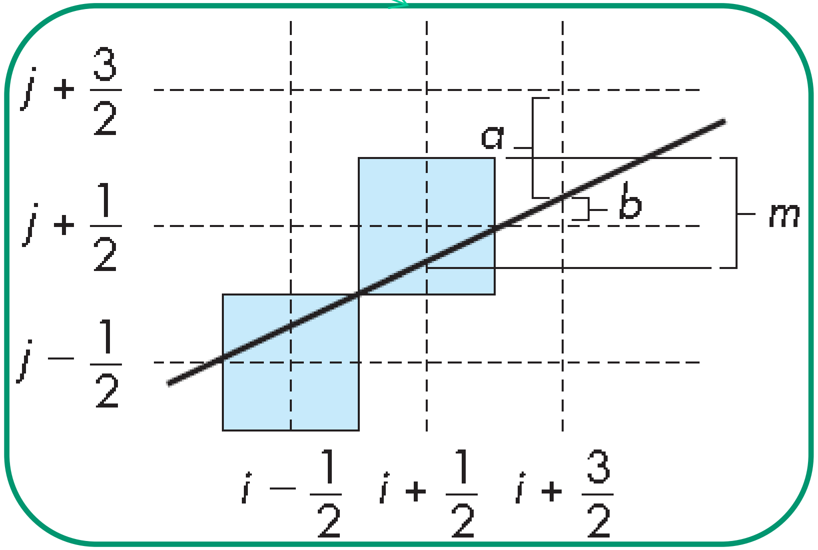
Incremental Form

Look at d_k, the value of the decision variable at $x = k + 0.5$

More efficient if we compute d_{k+1} incrementally from d_k

If $d_k > 0$, $\quad d_{k+1} = d_k - 2\Delta y$;
otherwise, $d_{k+1} = d_k - 2(\Delta y - \Delta x)$



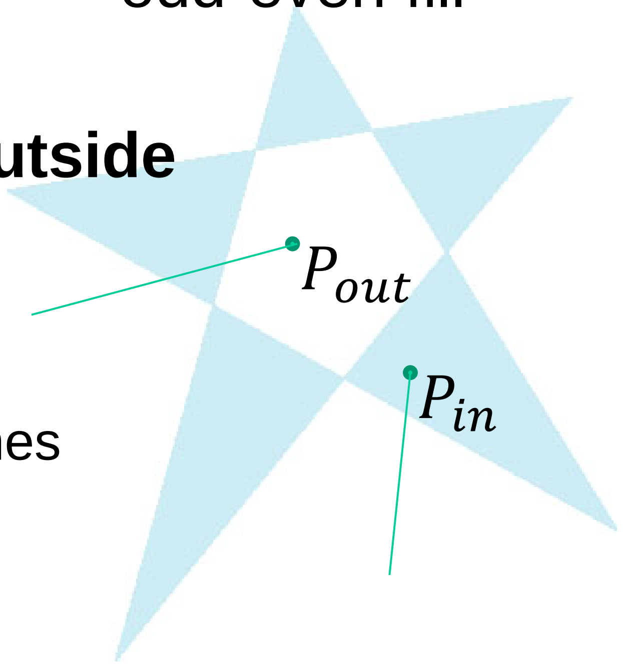


Polygon Rasterization

Polygon properties:

- Simple: edges cannot cross, i.e., only meet at the end points
- Convex: All points on line segment between two points in a polygon are also in the polygon
- Flat: all vertices are in the same plane

odd-even fill



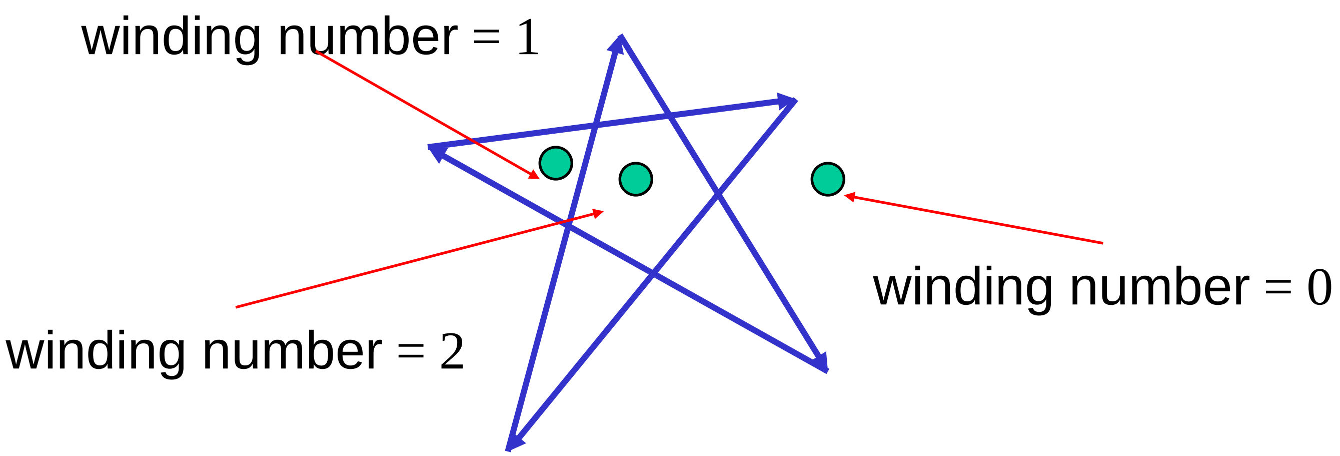
How to tell inside from outside – inside-outside testing

- Convex easy
- Nonsimple difficult
- Odd even test: count edge crossings with scanlines
  - Inside: odd crossings
  - Outside: even crossings

Winding Test: Winding Number

Traverse the edges of the polygon from any starting vertex and going around the edge in a particular direction until reaching the starting point

Winding number: number of times of a point encircled by the edges



Alternate definition of inside: inside if winding number ≠ 0

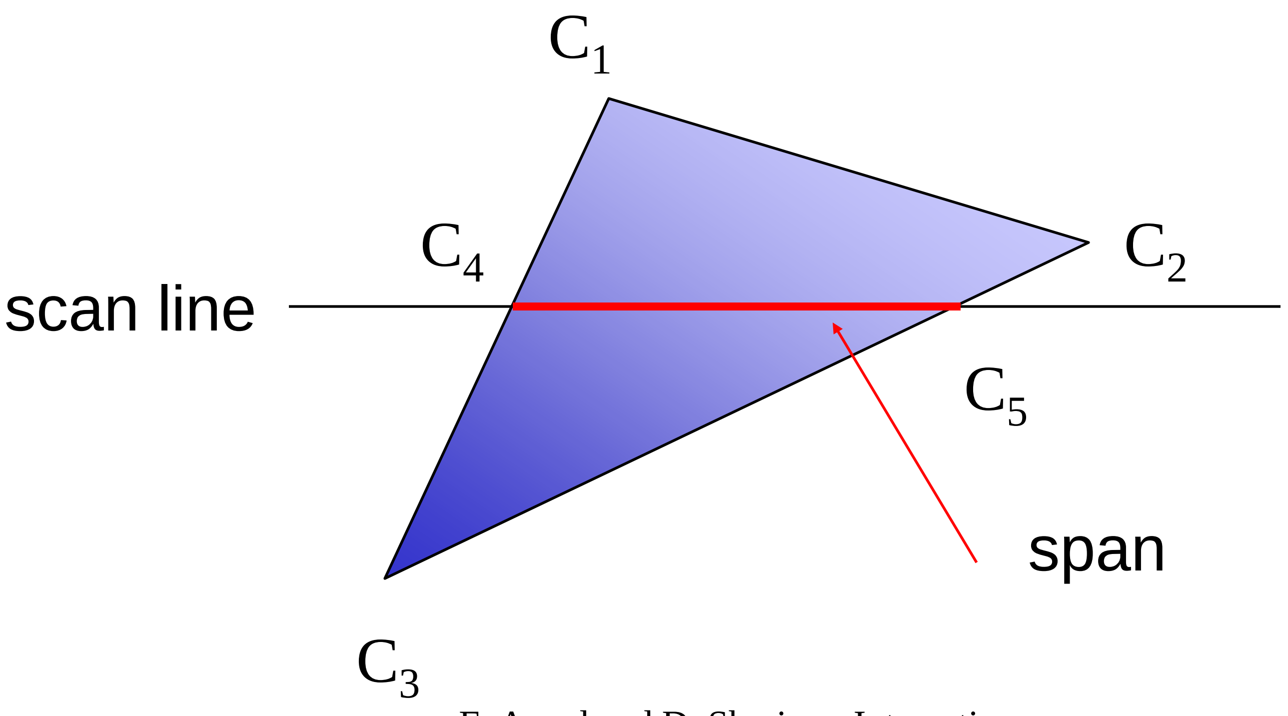
Filling in the Frame Buffer

Fill at end of pipeline: coloring a point with the inside color if it is inside the polygon

- Convex Polygons only
- Nonconvex polygons assumed to have been tessellated
- Shades (colors) have been computed for vertices (Gouraud shading)
- Scanline fill
- Flood fill

Scanline Fill: Using Interpolation

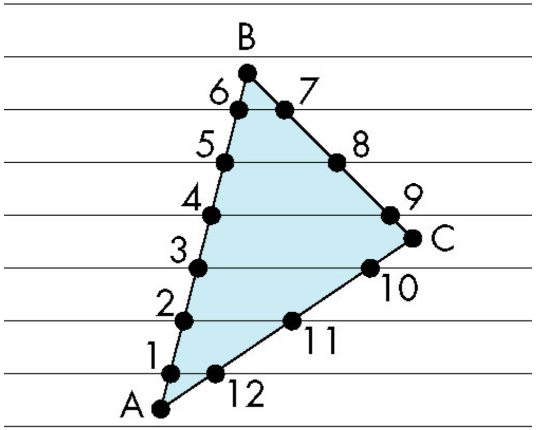
C_1 C_2 C_3 specified by `glColor` or by vertex shading
C_4 determined by interpolating between C_1 and C_2
C_5 determined by interpolating between C_2 and C_3
Interpolate points between C_4 and C_5 along span



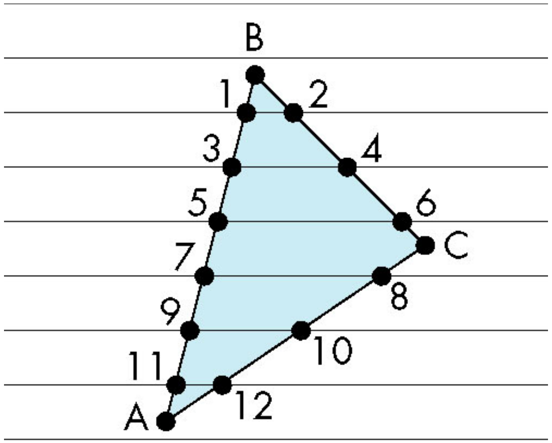
Scan Line Fill

Can also fill by maintaining a data structure of all intersections of polygons with scan lines

- Sort by scan line
- Fill each span



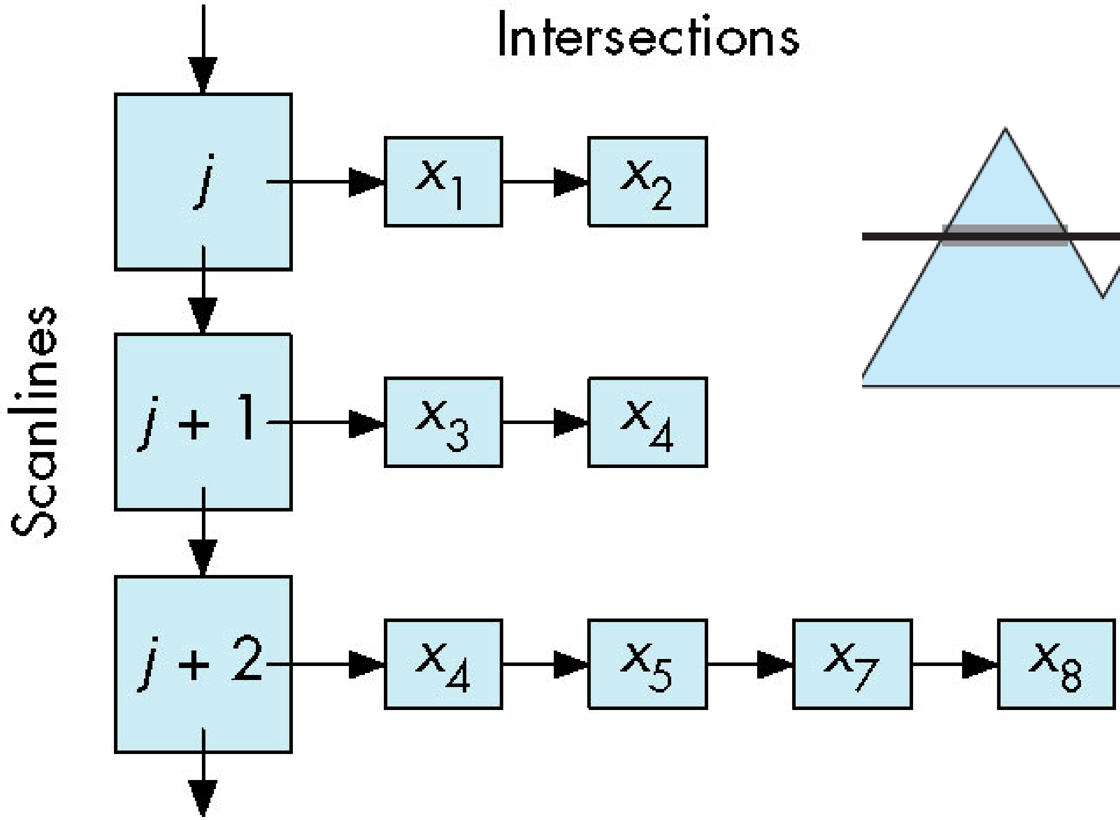
vertex order generated by vertex list

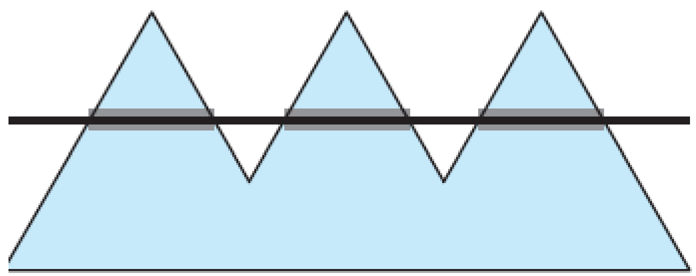


desired order

Data Structure

Insertion sort is applied on the x-coordinates for each scanline



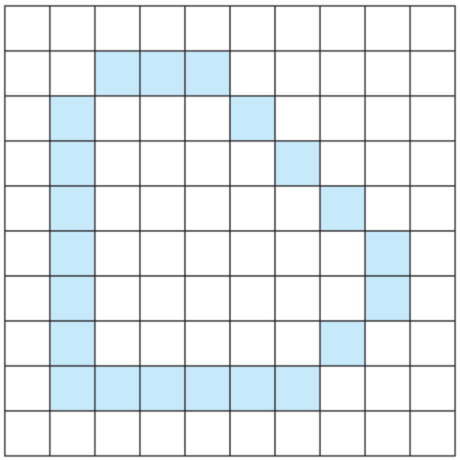


Flood Fill

Starting with an unfilled polygon, whose edges are rasterized into the buffer, fill the polygon with inside color (BLACK)

Fill can be done recursively if we know a seed point located inside. Color the neighbors to (BLACK) if they are not edges.

```
flood_fill(int x, int y) {
    if(read_pixel(x,y)= = WHITE) {
       write_pixel(x,y,BLACK);
       flood_fill(x-1, y);
       flood_fill(x+1, y);
       flood_fill(x, y+1);
       flood_fill(x, y-1);
}   }
```



Back-Face Removal (Culling)

Only render front-facing polygons

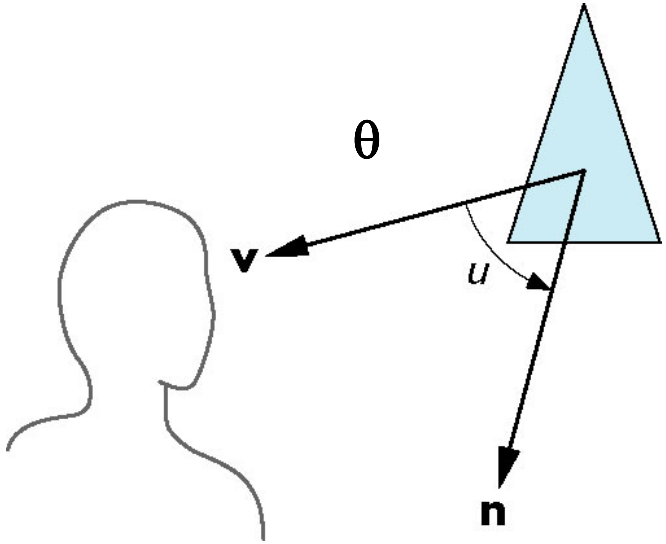
Reduce the work by hidden surface removal

Face is visible iff $-\frac{\pi}{2} \leq \theta \leq \frac{\pi}{2}$

equivalently

$\cos\theta \geq 0$ **or** $\boxed{\mathbf{v} \bullet \mathbf{n} \geq 0}$

Easy to compute



E. Angel and D. Shreiner: Interactive Computer Graphics 6E © Addison-Wesley 2012