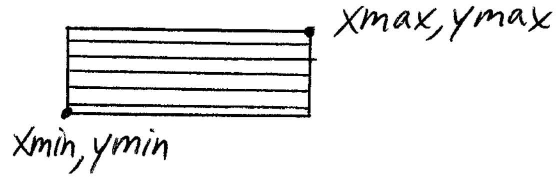
Polygon Filling

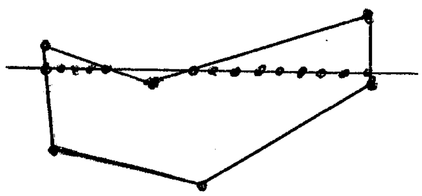
Task: assign all interior pixels to a particular color (or texture)

For rectangles:




```
for(y=ymin; y<ymax; y++)
    for(x=xmin; x<xmax; x++)
        writePixel(x, y, value);
```

For polygons:



1) Compute spans that lie between left and right edges of polygon. The span extrema are calculated by an incremental alg. that computes a scan line/edge intersection from the intersection with the previous line. This is faster than the analytic soln: $x = \frac{(x2-x1)}{(y2-y1)}(y-y1) + x1$

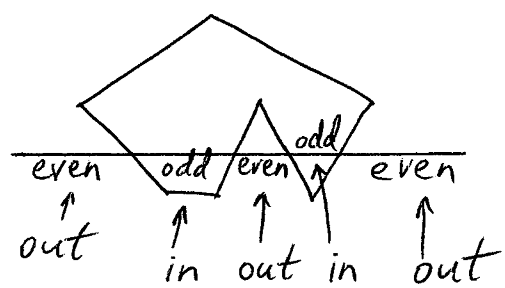
2) Determine which pixels on each scanline are within the polygon and set them to their appropriate value.

Spans can be filled in by a 3-step process:

1) Find intersections of the scanline with all edges of polygon

2) Sort intersections by increasing x-coordinates

3) Fill in all pixels in interior using odd-parity rule.

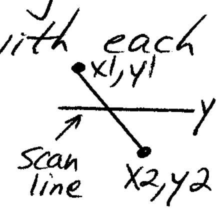


odd: draw
even: don't draw

In more detail:

<u>steps (1) + (2)</u>: Find intersections and sort

Avoid brute-force technique of testing each polygon edge for intersection with each new scanline: $x = \left(\frac{x2 - x1}{y2 - y1}\right)(y - y1) + x_1$



Only a few of the edges may intersect a scanline. Exploit edge coherence: many edges intersected by scanline i are also intersected by $i+1$.

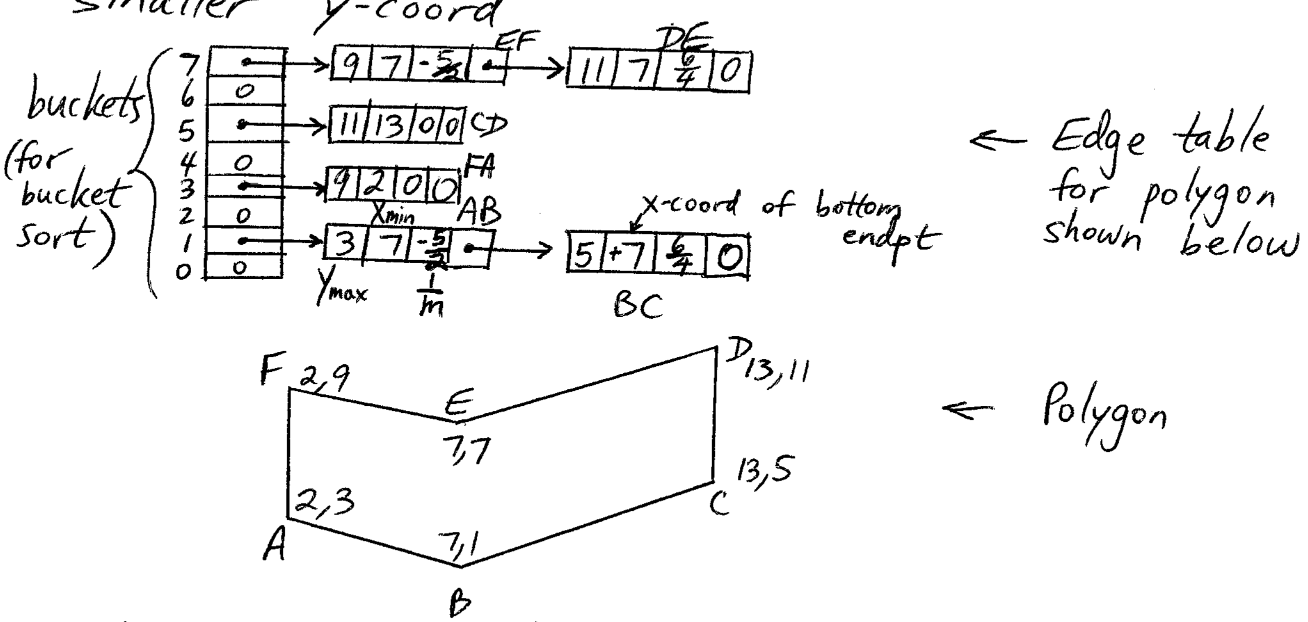
Compute intersections incrementally, like in Bresenham's algorithm.

Data structures:

Edge table (ET)

Active-edge table (AET)

ET contains all edges sorted by their smaller y-coord



Using ET, create AET:

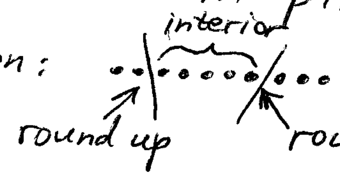
1) set y to smallest y-coord that has an entry in ET (y for first nonempty bucket)
2) Init AET to be empty (0)
3) Repeat until the AET and ET are empty
   - 3.1) Move from ET bucket y to the AET those edges whose $y_{min} = y$ (entering edges) and then sort the AET on x
   - 3.2) Fill in pixels on scanline y by using odd-parity rule
   - 3.3) Remove from AET those entries for which $y = y_{max}$ (edges no longer involved in next scanline)
   - 3.4) Increment y by 1 for next scanline
   - 3.5) For each nonvertical edge remaining in the AET, update x for the new y.

Example

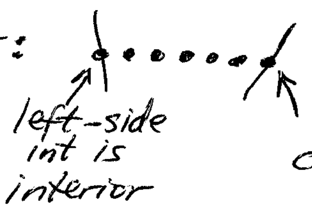
	AET ptr			
$y=1$	AB: 3 | 7 | $\frac{-5}{2}$ | •	BC: 5 | 7 | $\frac{6}{4}$ | 0		
$y=2$	AB: 3 | 4.5 | $\frac{-5}{2}$ | •	BC: 5 | 8.5 | $\frac{6}{4}$ | 0		
$y=3$	AB: 3 | 2 | $\frac{-5}{2}$ | •	AF: 9 | 2 | 0 | •	BC: 5 | 10 | $\frac{6}{4}$ | 0	
$y=4$	AF: 9 | 2 | 0 | •	BC: 5 | 11.5 | $\frac{6}{4}$ | 0		
$y=5$	AF: 9 | 2 | 0 | •	CD: 11 | 13 | 0 | •	BC: 5 | 13 | $\frac{6}{4}$ | 0	
$y=6$	AF: 9 | 2 | 0 | •	CD: 11 | 13 | 0 | 0		
$y=7$	AF: 9 | 2 | 0 | •	EF: 9 | 7 | $\frac{-5}{2}$ | •	DE: 11 | 7 | $\frac{6}{4}$ | •	CD: 11 | 13 | 0 | 0
$y=8$	AF: 9 | 2 | 0 | •	EF: 9 | 4.5 | $\frac{-5}{2}$ | •	DE: 11 | 8.5 | $\frac{6}{4}$ | •	CD: 11 | 13 | 0 | 0
$y=9$	EF: 9 | 2 | $\frac{-5}{2}$ | •	AF: 9 | 2 | 0 | •	DE: 11 | 10 | $\frac{6}{4}$ | •	CD: 11 | 13 | 0 | 0
$y=10$	DE: 11 | 11.5 | $\frac{6}{4}$ | •	CD: 11 | 13 | 0 | 0		
$y=11$	CD: 11 | 13 | 0 | •	DE: 11 | 13 | $\frac{6}{4}$ | 0		

(sort on x,
For tie: sort on $\frac{1}{m}$)

Step (3): Fill in pixel spans

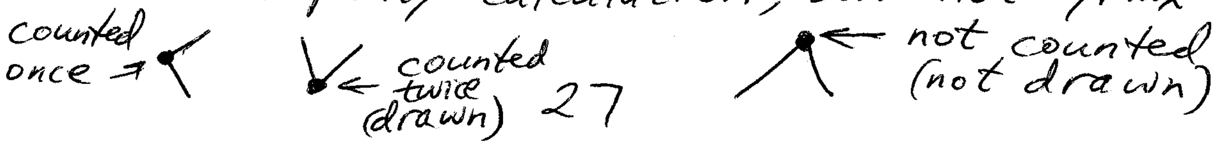
Fraction:


$\Longrightarrow$ only fill <u>interior</u> pixels

Integer:


$\Longrightarrow$ avoid conflicts at shared edges

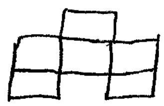
Shared vertex: count y_{min} vertex of an edge in parity calculation, but not y_{max}



Fill Algorithms

Instead of polygon fill, where we are given a list of edges and require ET and AET data structures, we may use a fill algorithm $\Rightarrow$ no need to define boundary with edges, just different color.

Defs: A region is a collection of pixels. A region is 4-connected if every 2 pixels can be joined by a sequence of pixels using only up, down, left, or right moves (N, S, W, E)



4-connected region (also 8-connected)

NW	N	NE
W	X	E
SW	S	SE

A region is 8-connected if every 2 pixels can be joined by a sequence of pixels using N, S, W, E, NE, NW, SE, or SW moves.

Note: every 4-connected region is also 8-connected.

 ← 8-connected region (but not 4-connected)

Consider a starting pixel P.

The <u>interior-defined</u> region is the largest connected region of points whose value is that of P.

The <u>boundary-defined</u> region is the largest connected region of pixels whose value is not some given boundary value.

Algorithms that fill interior-defined regions are called <u>flood-fill</u> algs. Those that fill boundary-defined regions are called <u>boundary-fill</u> algs.

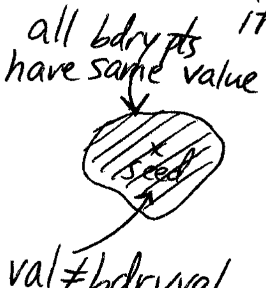
Since both start from a pixel within the region, they are sometimes both called <u>seed-fill</u> algs.

for 4-connected regions

```
floodfill4(x, y, oldval, newval)
int x, y, oldval, newval;
{
    if(readPixel(x,y) == oldval) {
        writePixel (x, y, newval);
        floodfill4(x, y-1, oldval, newval);
        floodfill4(x, y+1, oldval, newval);
        floodfill4(x-1, y, oldval, newval);
        floodfill4(x+1, y, oldval, newval);
    }
}
```

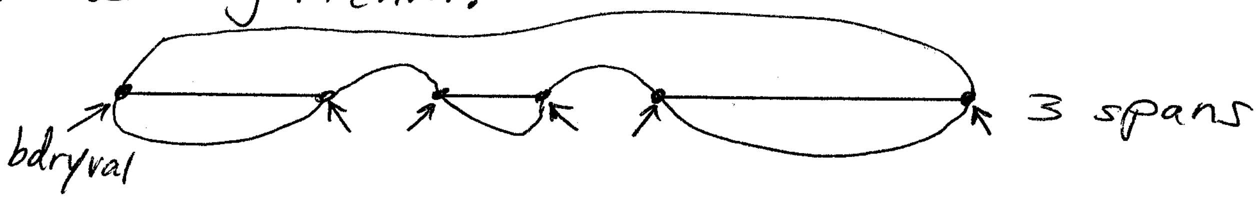
all interior pixels have same value (oldval)

```
boundaryfill4(x, y, bdryval, newval)
int x, y, bdryval, newval;
{
      int c;
      c = readPixel(x, y);
      if(c != bdryval && c != newval){
            writePixel(x, y, newval);
            boundaryfill4(x, y-1, bdryval, newval);
            boundaryfill4(x, y+1, bdryval, newval);
            boundaryfill4(x-1, y, bdryval, newval);
            boundaryfill4(x+1, y, bdryval, newval);
      }
}
```



Both floodfill4() and boundaryfill4() are simple, but very highly recursive (rel. slow) functions. Recursion may cause stack overflow.

Better algorithm:



Spans are filled in iteratively:

```
for(x=x1; x<x2; x++) writePixel (x, y, newval);
```

A span is identified by its rightmost pixel. After a span is filled, the row above is examined from right to left to find the rightmost pixel of each span.
These pixel addresses are stacked. Same for below.
After a span is processed, the pixel address at the top of the stack is used as the new starting point.
The algorithm ends when the stack is empty.
p.440 (Hill text)