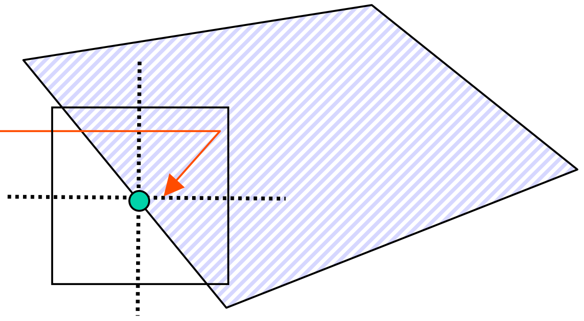
Ambiguous cases

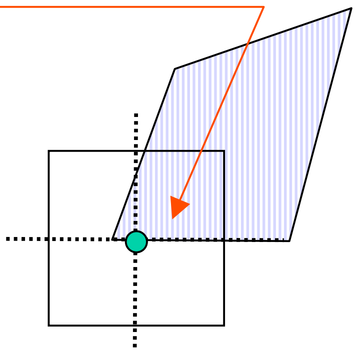
- What if a pixel is exactly on the edge?
- Polygons are usually adjacent to other polygons, so we want a **convention** which will give the pixel to *one* of the adjacent polygons or the *other* (as much as possible).
- "Draw left and bottom edges"
  - if (x+∂, y) is in, pixel is in

    (for sufficiently small, positive, ∂)



  - horizontal edge? if (x+∂, y+□) is in, pixel is in

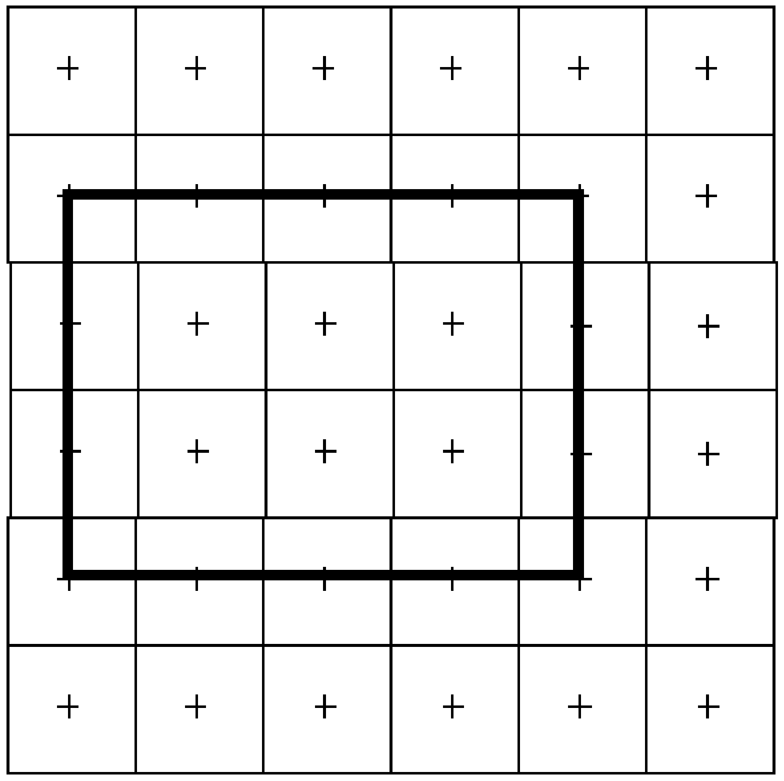
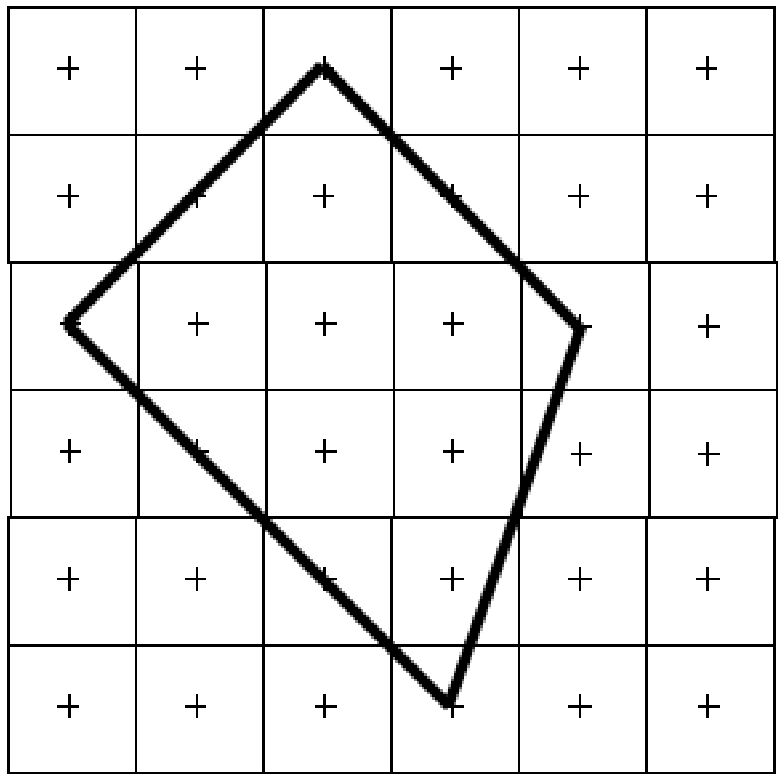
    (for sufficiently small, positive, ∂, □)



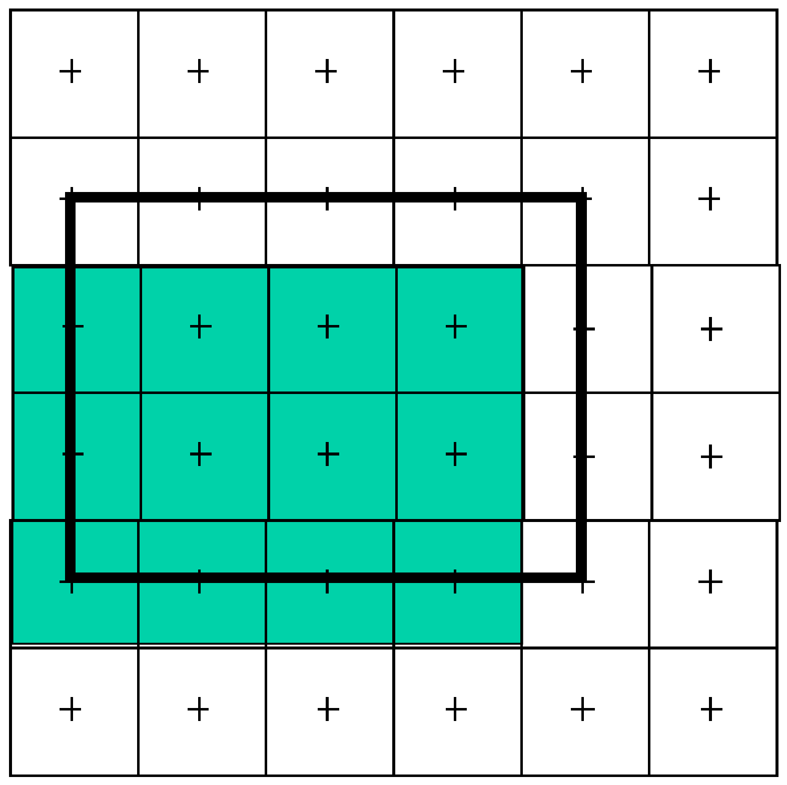
Ambiguous cases (2)

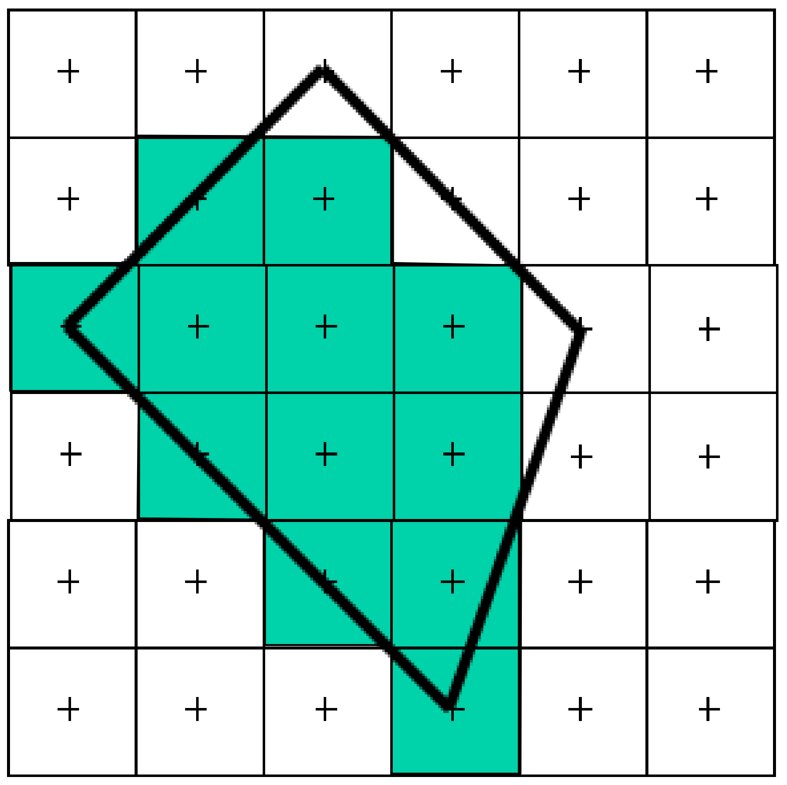
- Vertex?--essentially draw those on “left and bottom”, but detailed analysis should be done in conjunction with one’s scan conversion algorithm

Ambiguous inside cases (?)

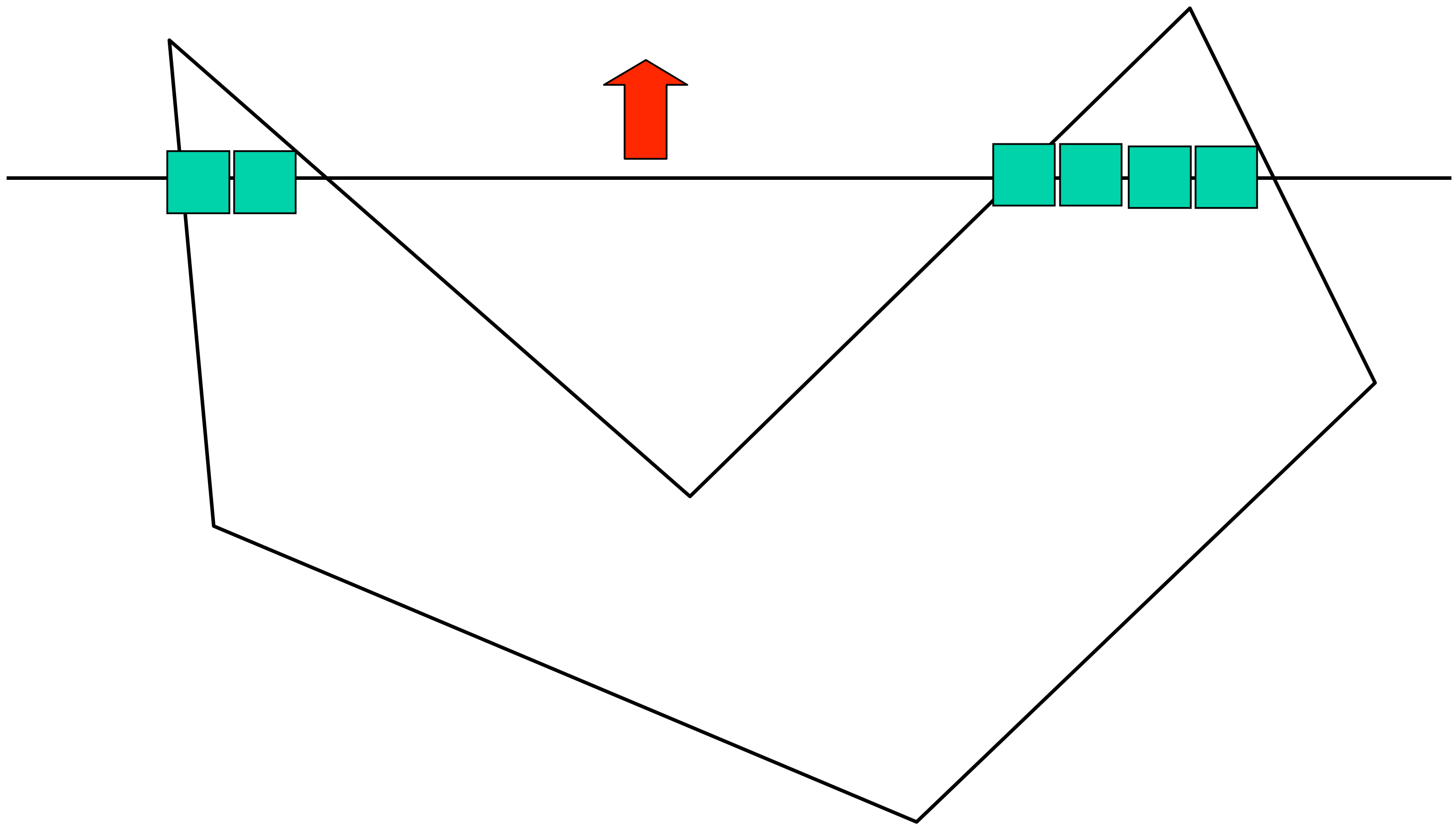



Ambiguous inside cases (answer)





Sweep fill



Sweep fill

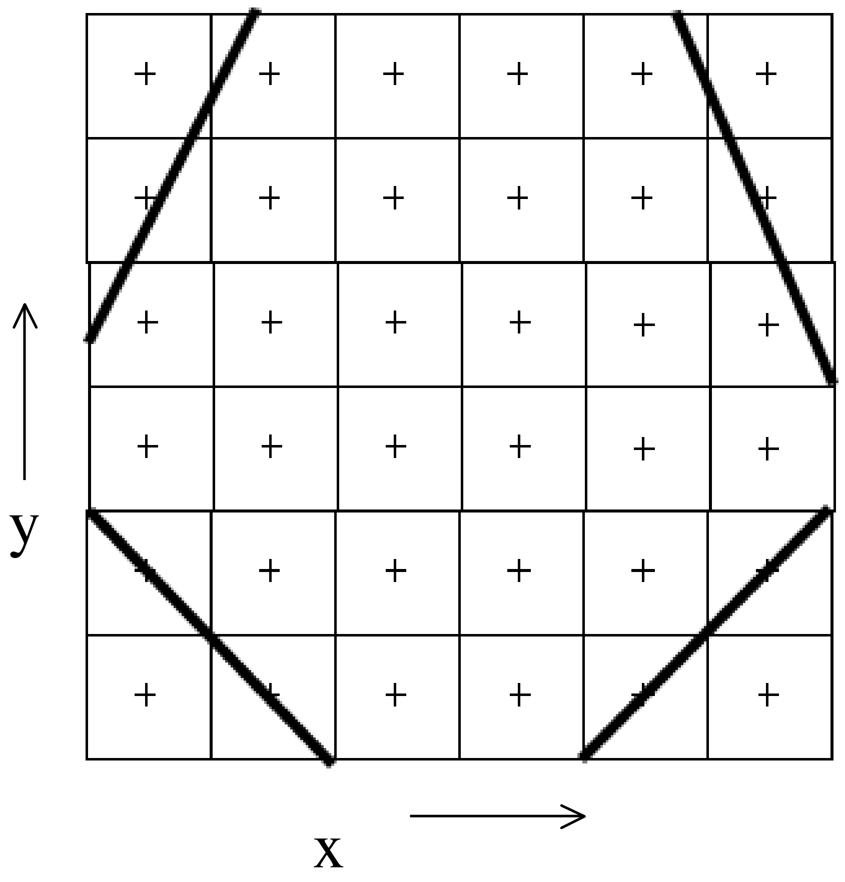
- Reduces to filling many spans
- Inside/outside parity is relatively straightforward
- Need to compute the spans, then fill
- Need to update the spans for each scan
- Need to implement “inside” rule for ambiguous cases.

Spans

- Process - fill the bottom horizontal span of pixels; move up and keep filling
- Suppose we have the span, i.e., have xmin, xmax designating the intersection of the span with the sweep line (floating point).
- Need pixels with integers greater than **or** equal to xmin, but less than xmax. (Less than **or** equal xmax would break the convention)
- In code

```
for (x=ceil(xmin); x < ceil(xmax);x++)
```

recall that ceil(x) is largest integer greater than or equal to x



- Note that two adjacent polygons do not fight each for any pixels

Algorithm

- For each row in the polygon:
  - Throw away newly irrelevant edges
  - Obtain newly relevant edges
  - Fill spans
  - Update spans

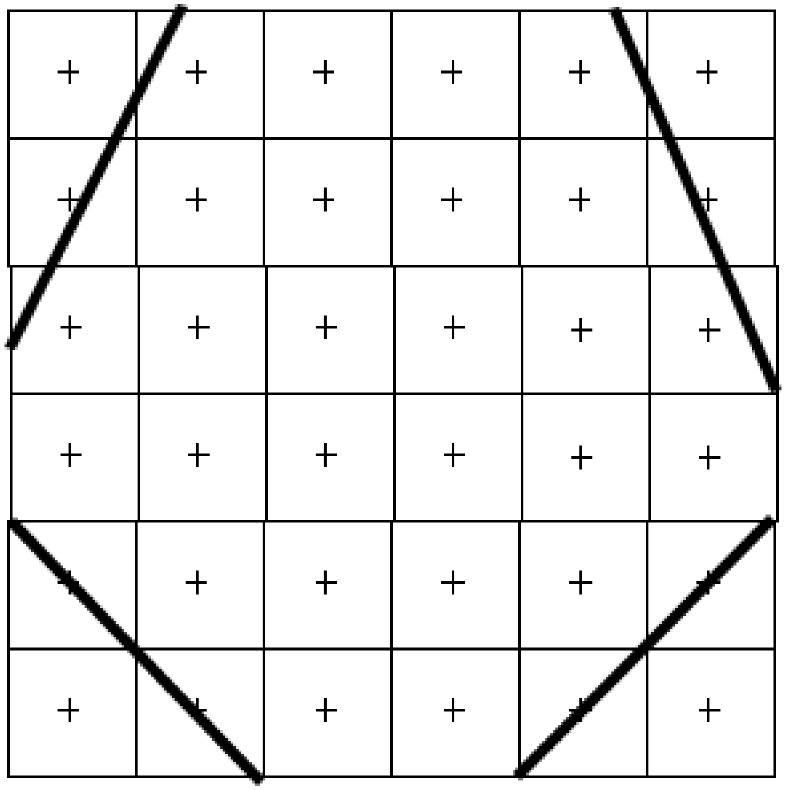
- Issues:
  - What aspects of edges need to be stored?
  - When is an edge relevant/irrelevant?

The next span - 1

- For an edge, have y=mx+b
- hence, if $y_n = m\, x_n + b$, then $y_{n+1} = y_n + 1 = m\,(x_n + 1/m) + b$
- Hence, *if there is no change in the edges*, we have:

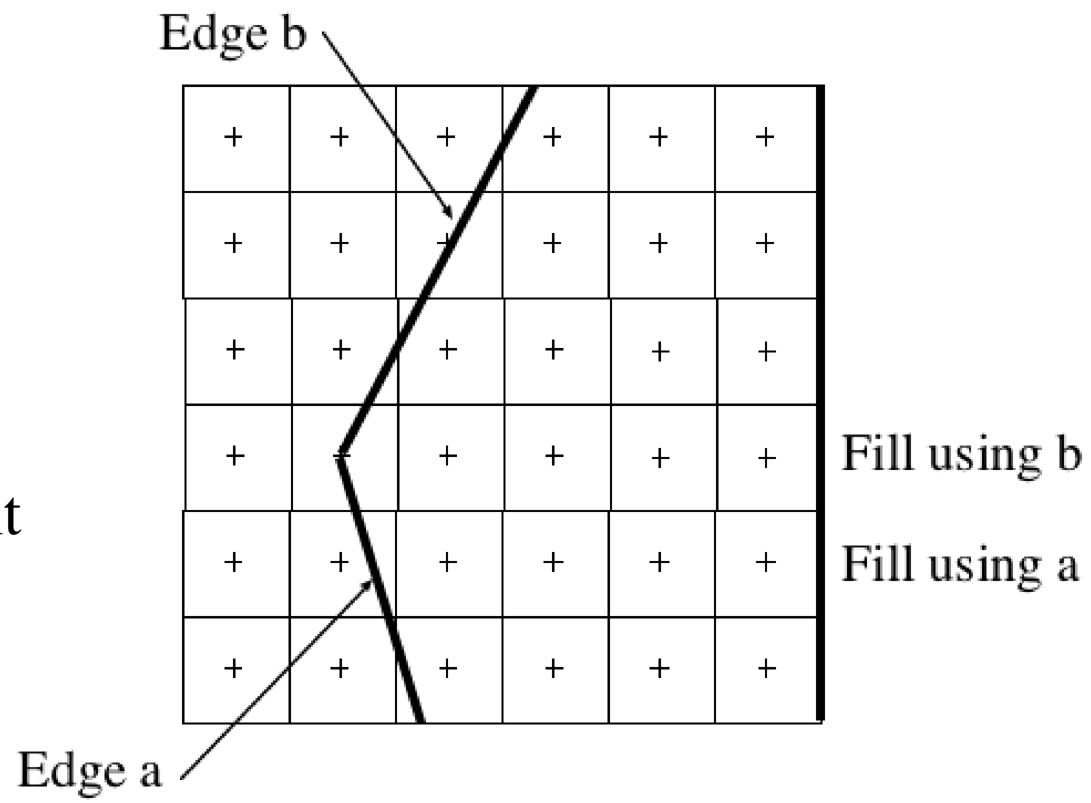
  xmax -> xmax+(1/m)(xmax)

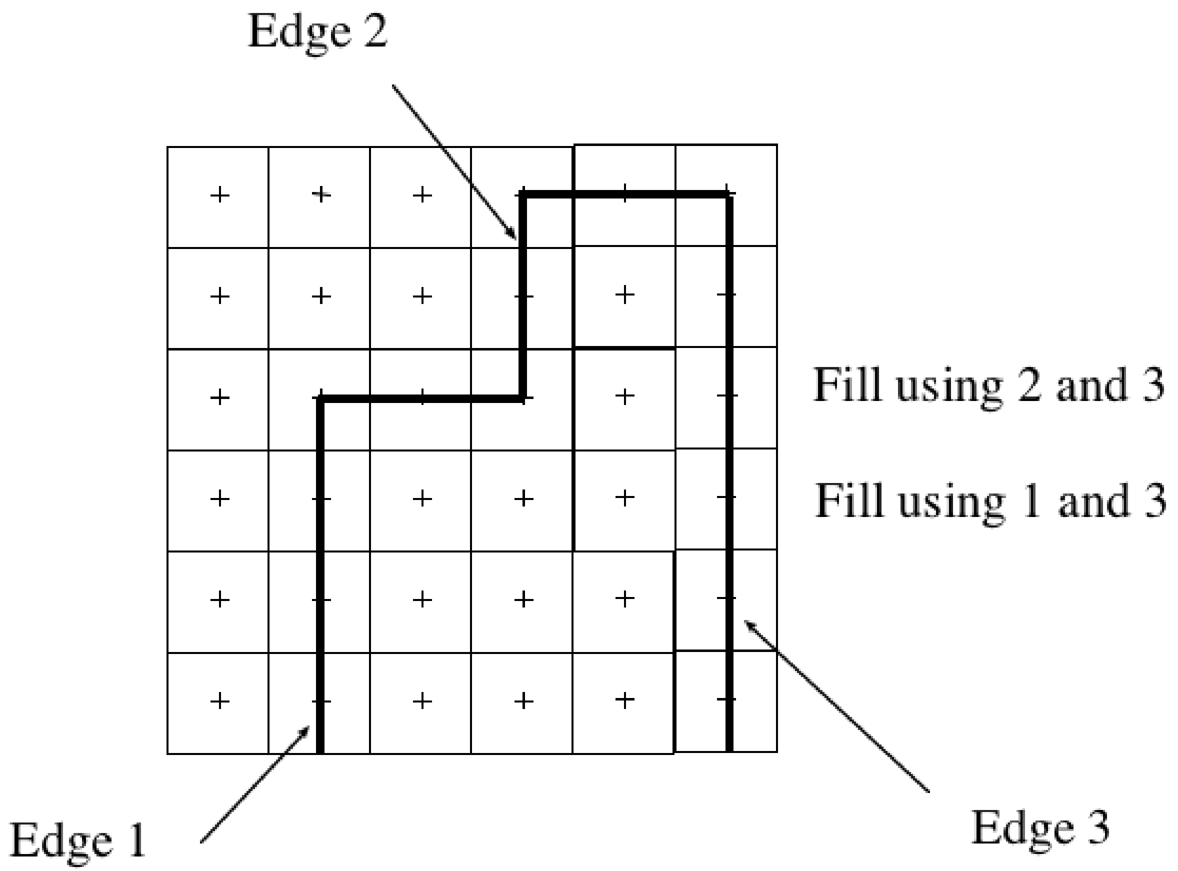
  xmin -> xmin+(1/m)(xmin)



The next span - 2

- Horizontal edges are irrelevant
- Edge is irrelevant when y>=ymax of edge (note appeal to convention)
- Similarly, edge is relevant when y>=ymin of edge

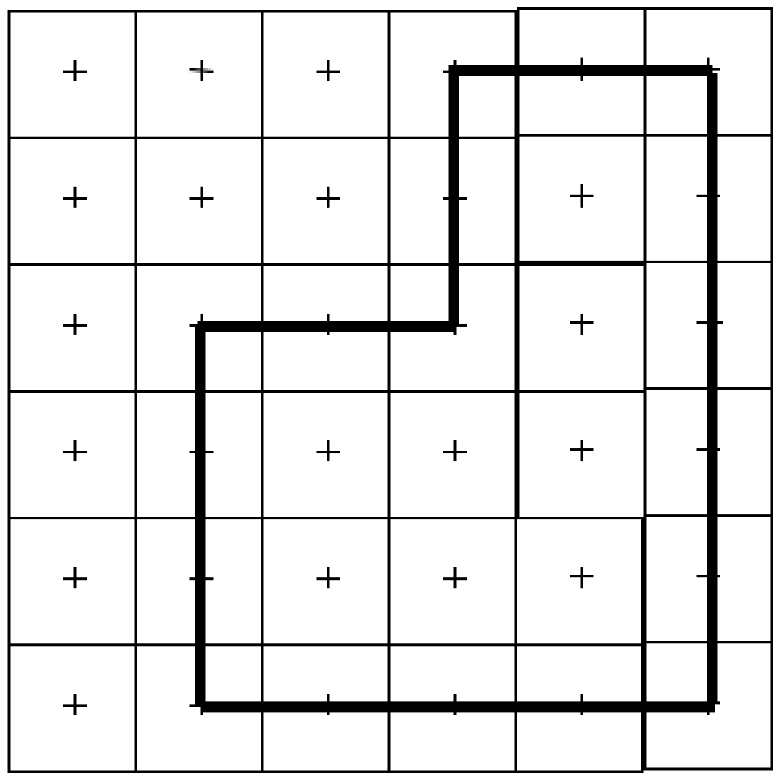



Edge 2
Fill using 2 and 3
Fill using 1 and 3
Edge 1
Edge 3

Filling in details

- Maintain a list of active edges in case there are multiple spans of pixels (Active Edge List).
- For each edge on the list, must know: x-value (current, initialize to min), max y value of edge, 1/m
- Keep edges in a table, indexed by minimum y value - Edge Table

- For row = min to row=max
  - AEL=append(AEL, ET(row));
  - remove edges whose ymax=row
  - sort AEL by x-value
  - fill spans
  - update each edge in AEL



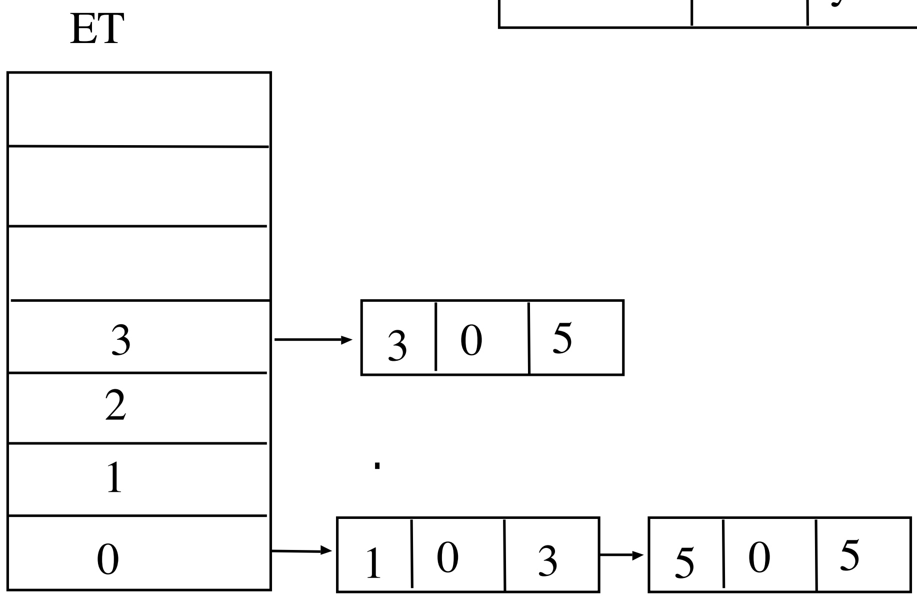
Example one

Compute the edge table (ET_ to begin. Then fill polygon and update active edge list (AEL) row by row. (Ignore horizontal egdes)

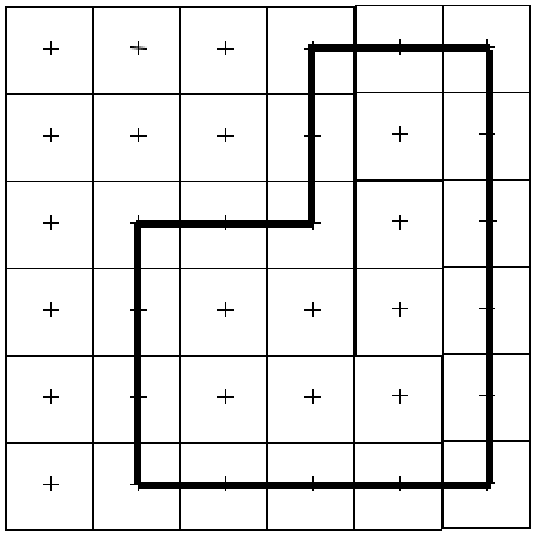
Format of AEL entries

xmin	1/m	ymax

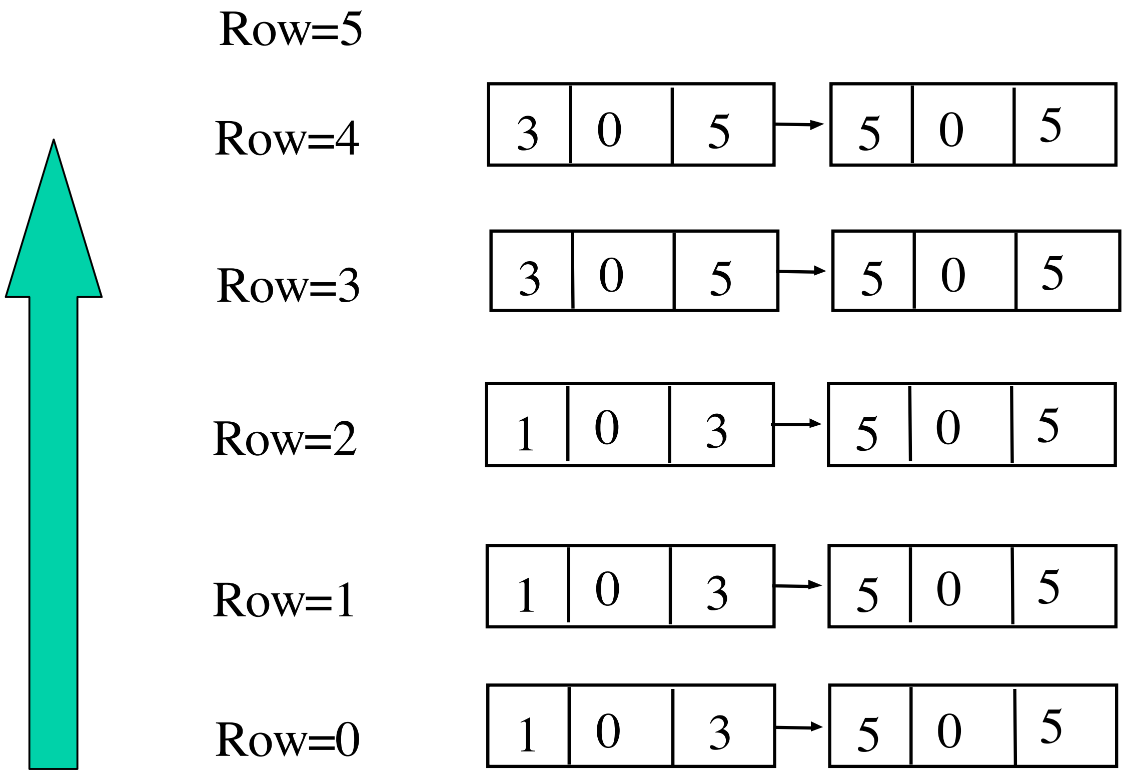
The AEL entries begin with xmin, and are initialized at row ymin



Example one

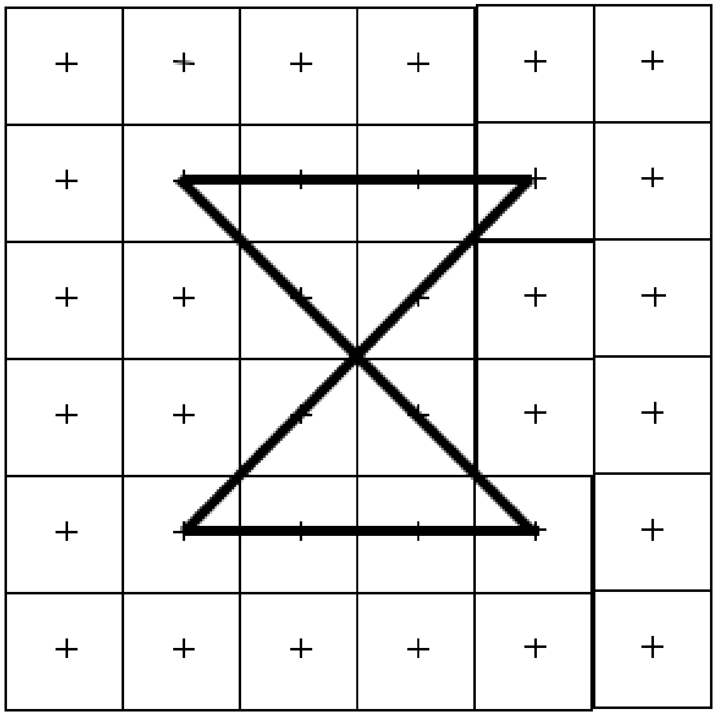


AEL just before filling listed row



Row=5		
Row=4	3 | 0 | 5 →	5 | 0 | 5
Row=3	3 | 0 | 5 →	5 | 0 | 5
Row=2	1 | 0 | 3 →	5 | 0 | 5
Row=1	1 | 0 | 3 →	5 | 0 | 5
Row=0	1 | 0 | 3 →	5 | 0 | 5

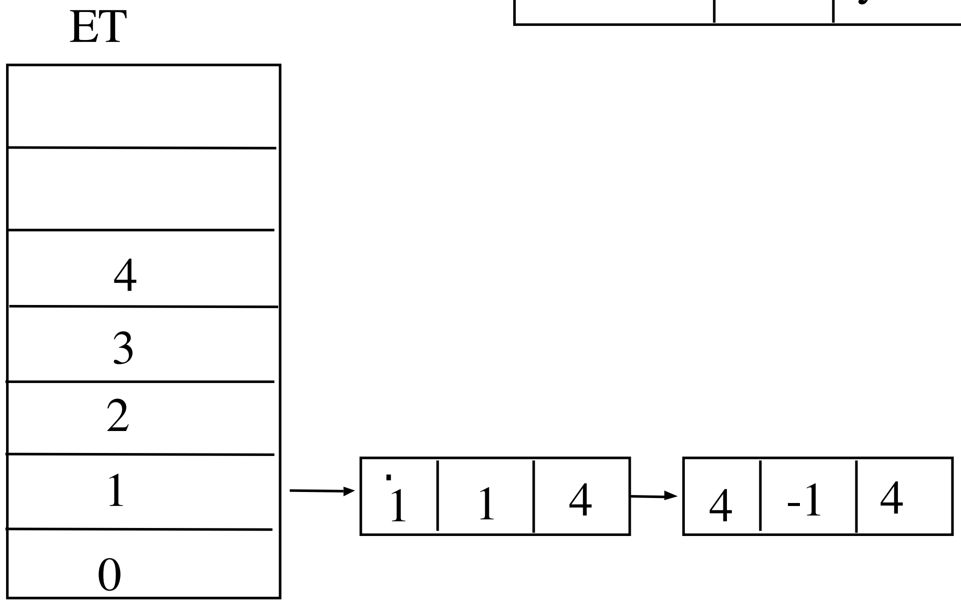
Example two



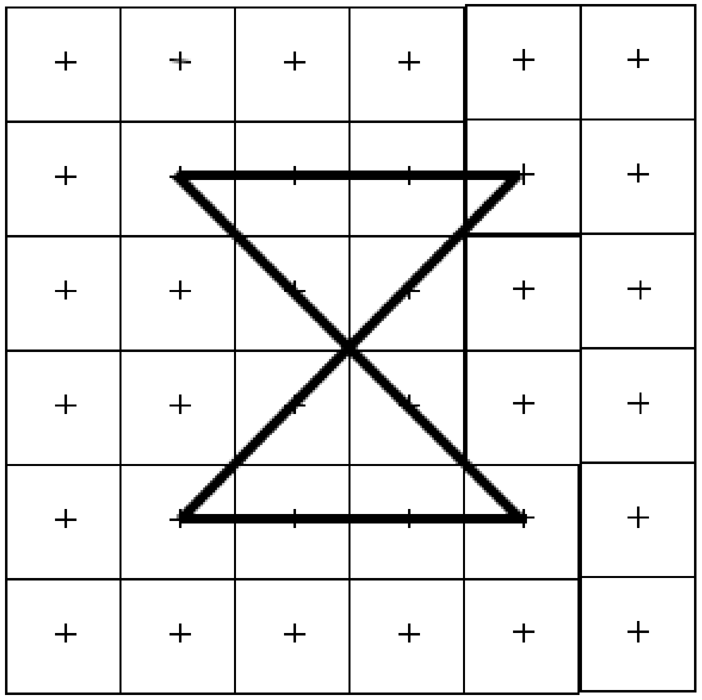
Format of AEL entries

xmin	1/m	ymax

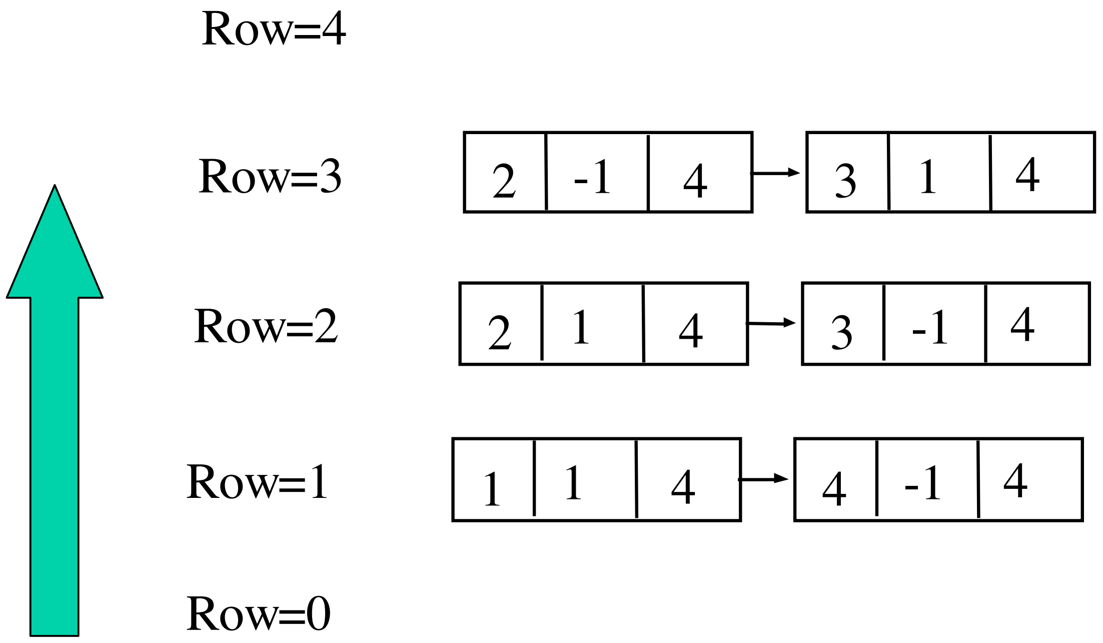
The AEL entries begin with xmin, and are initialized at row ymin



Example two



AEL just before filling listed row



Comments

- Sort is quite fast, because AEL is usually almost in order.
- Nonetheless, OpenGL limits to convex polygons, so two and only two elements in AEL at any time, and no sorting.
- With additional logic to keep track of what color to use, can fill in many polygons at a time.
- Can be done *without* floating point

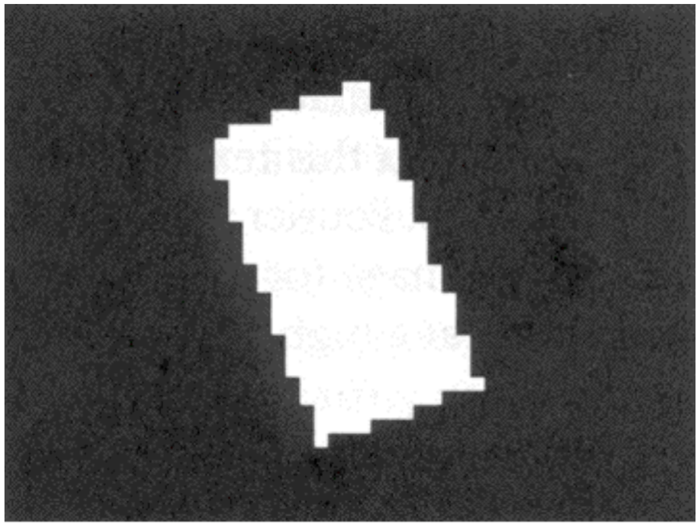
Dodging floating point

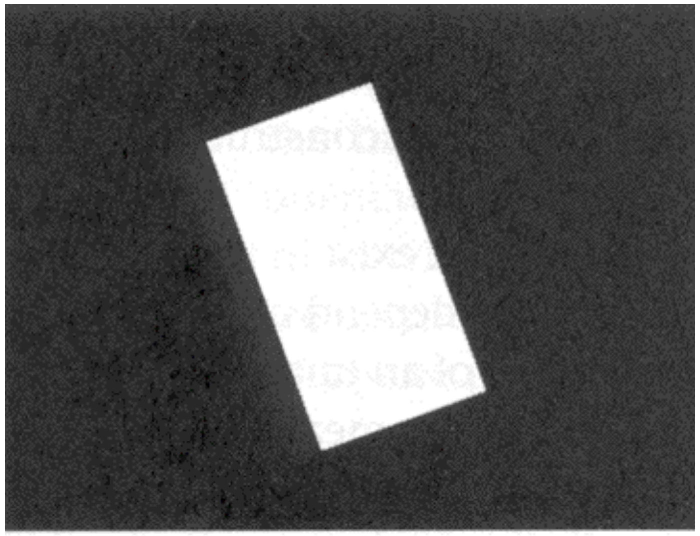
- For edge, 1/m=Dx/Dy, which is a rational number.
- Store x as x_int, x_num, x_denom=Dy
- then x->x+1/m is given by:
  - x_num=x_num+Dx
  - if x_num >= x_denom
    - x_int=x_int+1
    - x_num=x_num-x_denom

- Advantages:
  - no floating point
  - can tell if x is an integer or not (check x_num=0), and get truncate(x) easily, for the span endpoints.

Aliasing/Anti-Aliasing

- Analogous to the case of lines
- Anti-aliasing is done using graduated gray levels computed by by smoothing and sampling
- Problem with "slivers" is really an aliasing problem.

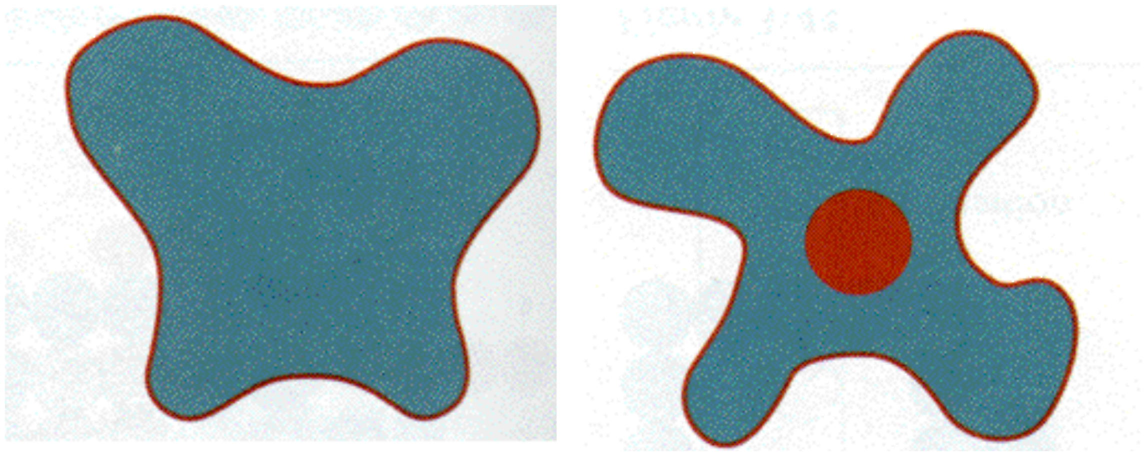

Aliasing


Ideal

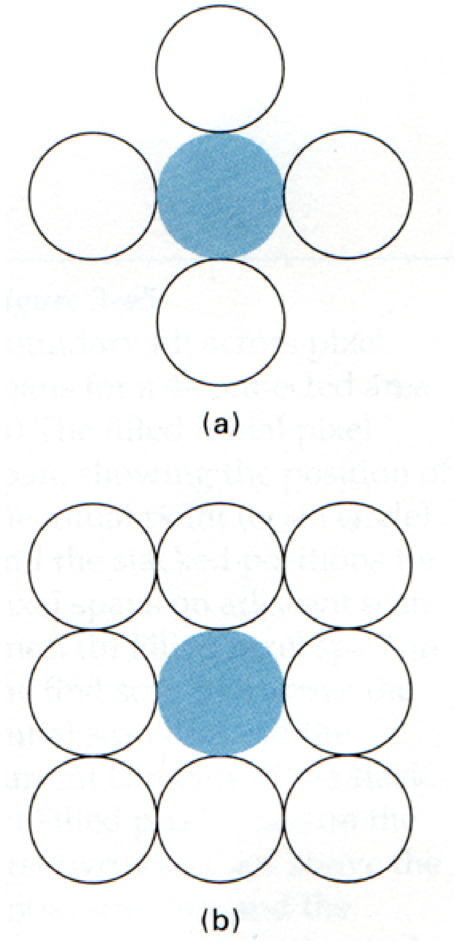
Boundary fill

- Basic idea: fill in pixels inside a boundary

- Recursive formulation:
  - to fill starting from an inside point
    - if point has not been filled,
      - fill
      - call on all neighbours that are not boundary pixels.



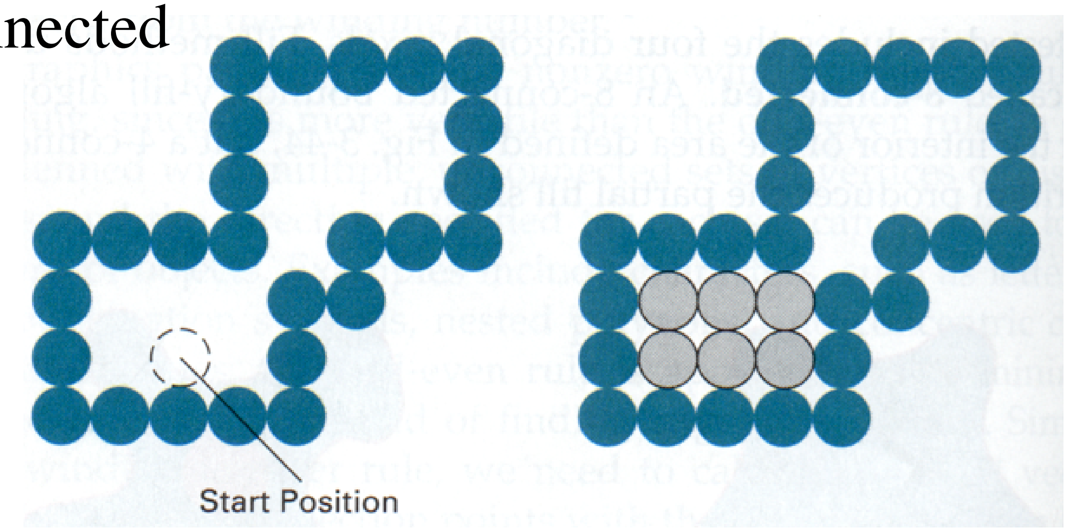
Choice of neighbours is important



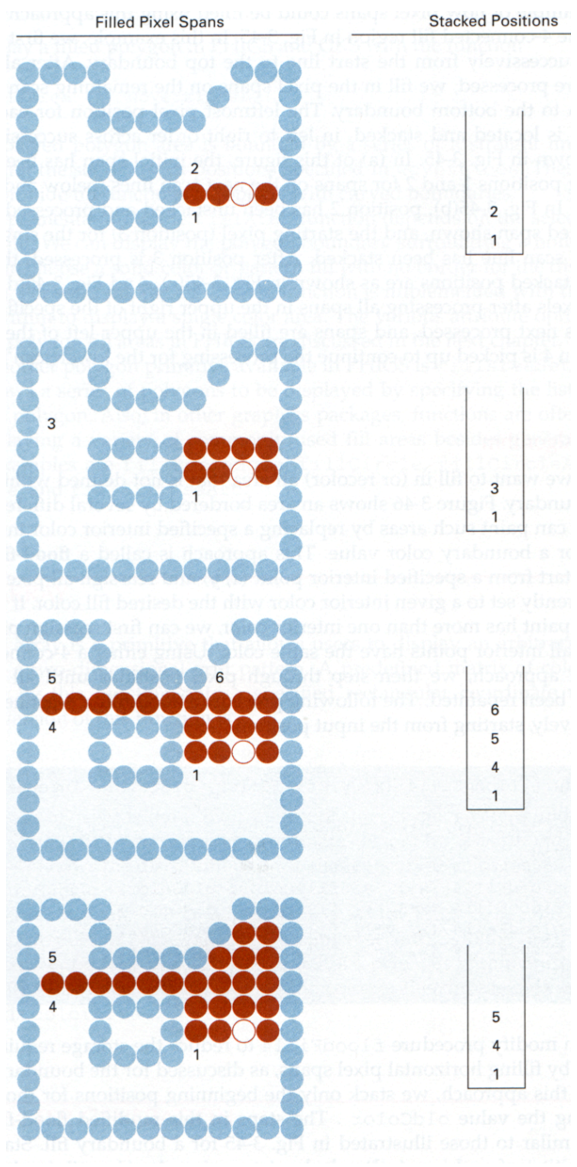
4-connected

8 connected

4 connected fill of a four connected boundary doesn't work



- Using spans for boundary fill means a less messy stack (due to less recursion)



Pattern fill

- Use index into screen as index into pattern