Polygon Filling

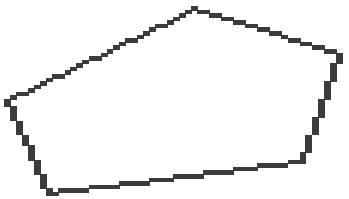
- 2 parts to task
  - which pixels to fill
  - what to fill them with
- First consider filling unclipped primitives with solid color
- Which pixels to fill
  - consider scan lines that intersect primitive
  - fill in *spans* of pixels lying inside primitive
  - exploit *spatial coherence, span coherence, scan-line coherence, edge coherence*
- Can write frame buffer one word at time rather than one bit

Line Segments Sharing Pixels

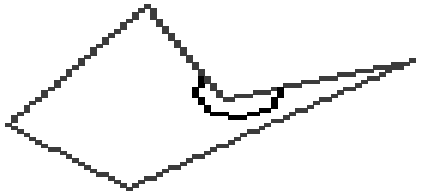
- Important not to write pixels twice
  - could be set to background color if **xor** used
  - could be double intensity (film writer)
- Is boundary part of an area defining primitive
  - mathematically yes
  - graphically more complicated
- Rule: left and bottom edges are part of primitive
  - shared vertical edges belong to rightmost primitive
- Still problems
  - bottom left point still drawn twice
  - rectangles missing right and top edges !!

Polygon Types

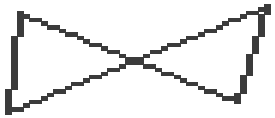
- simple convex, simple concave, non-simple (self-intersecting)
- want no holes, no intersections (line crossings)
- rectangles and triangles always simple convex



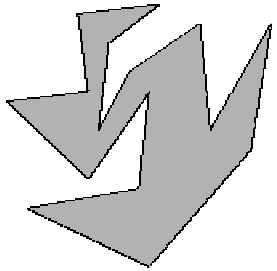
simple convex



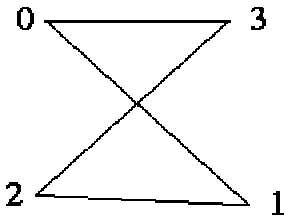
simple concave



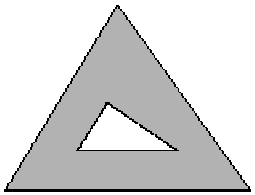
non-simple (self-intersection)



OK



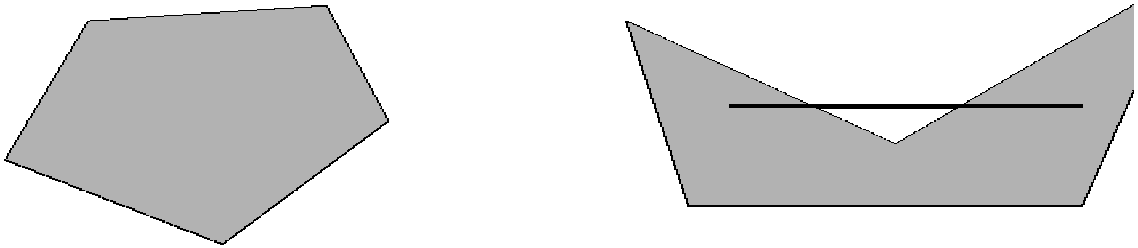
Line Crossing



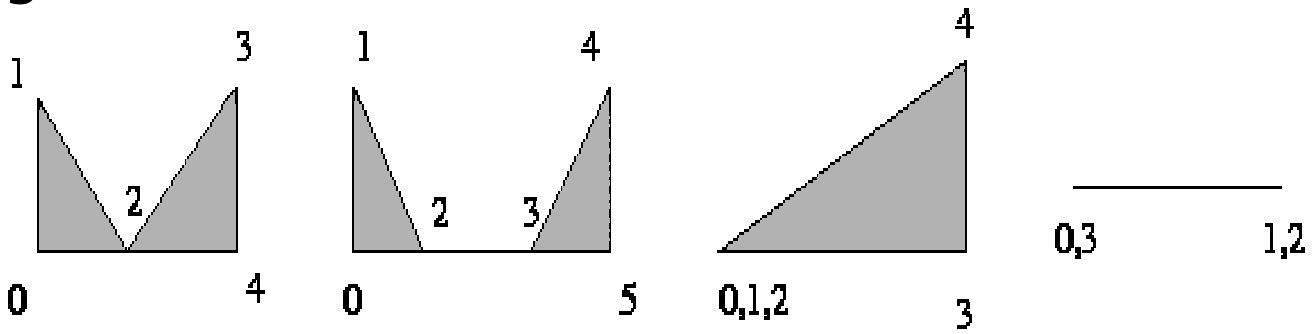
Hole

Convex, Concave, Degenerate

- Convex polygons are preferable to concave
  - Polygon is convex if for any two points inside polygon, the line segment joining these two points is also inside.

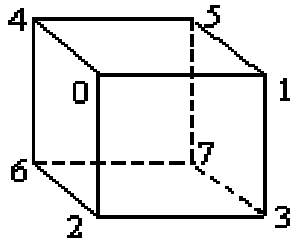


  - Degeneracies



Polygon Representation

- Polygon Representation
  - ordered list of vertices
  - avoids redundant storage and computations
  - associate other information with vertices
    - colors, normals, textures

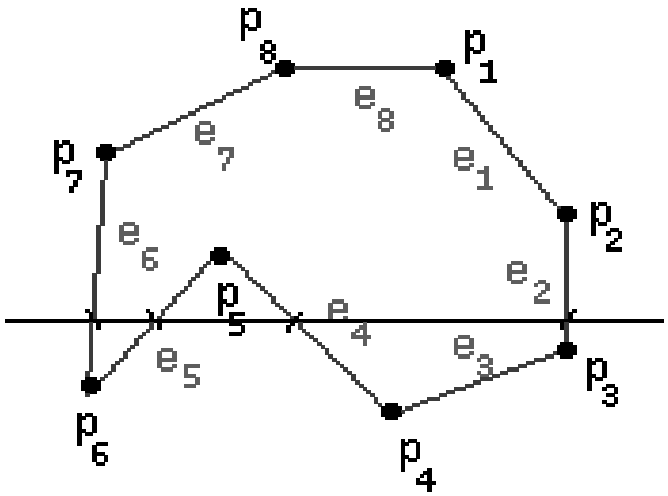



```
faces                 vertex list
#  vertex list        #  x,y,z

0  0,2,3,1            0  0,1,1
1  1,3,7,5            1  1,1,1
2  5,7,6,4            2  0,0,1
3  4,6,2,0            3  1,0,1
4  4,0,1,5            4  0,1,0
5  2,6,7,3            5  1,1,0
                      6  0,0,0
                      7  1,0,0
```

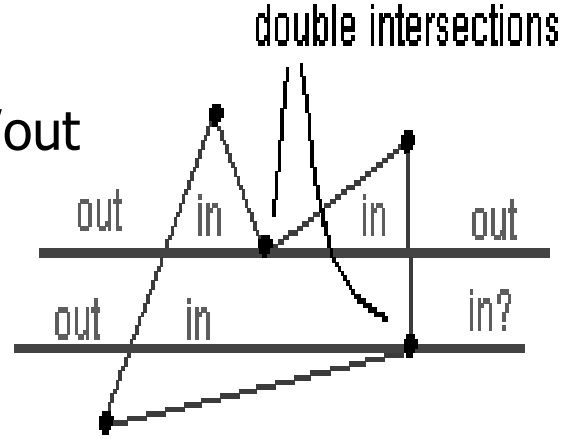
Scan Conversion

- Scan conversion: shade pixels lying within a closed polygon efficiently
  - fill color will in general depend on the lighting, texture, and visibility of the polygon
- Assume polygon is closed and has ordered edges



Scan Conversion Algorithm

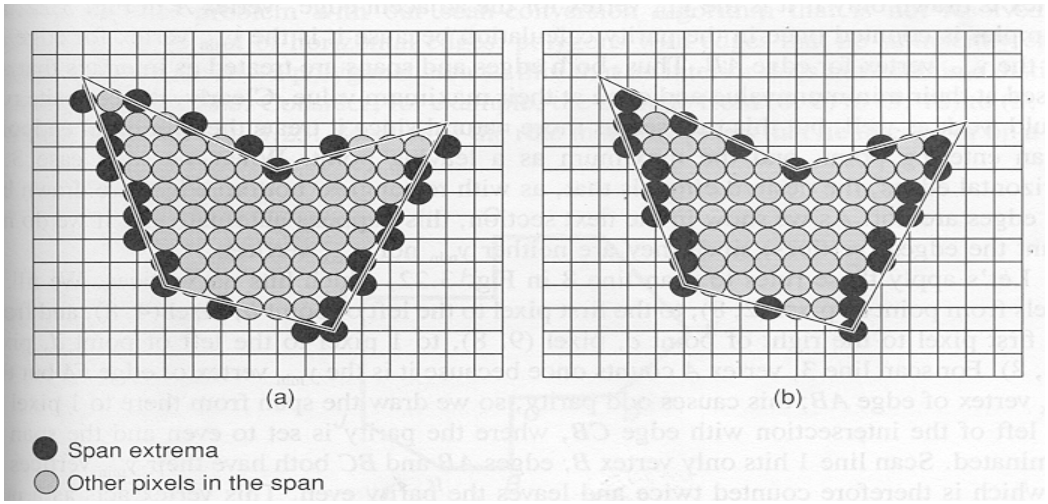
- intersect each scan-line with all edges
- sort intersections by increasing x coordinate
- calculate parity of intersections to determine in/out
  - parity starts even - each intersection inverts
- fill the 'in' pixels - those with odd parity



- General issues - how to handle intersection at integer and fractional x values
- Special cases:
  - shared vertices lying on scan-lines - double intersections
    - count y_{min} vertices but not y_{max} vertices in parity count
  - do NOT count vertices of horizontal edges

Fractional and Integer Intersections

- Fractional intersections
  - if approaching intersection to the right to determine inside pixel
    - take floor if inside, ceil if outside
- Integer intersections
  - if leftmost pixel make interior, rightmost exterior

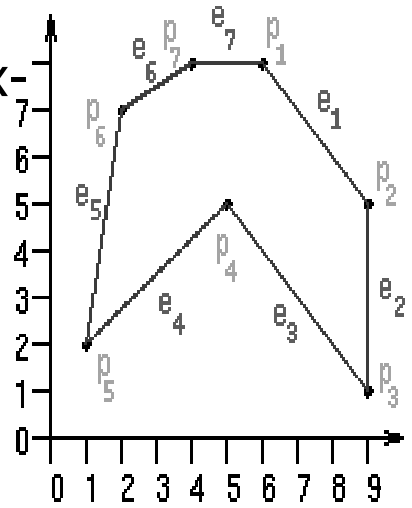


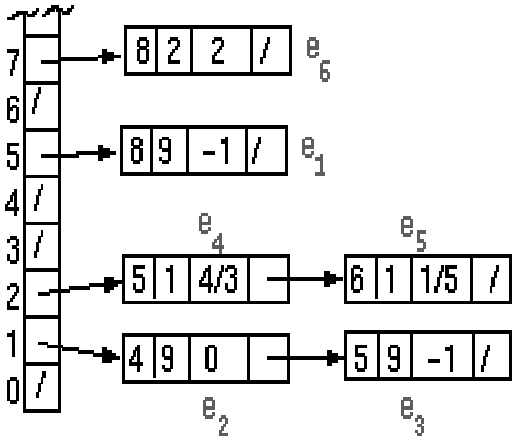
Using Spatial Coherence

- Efficiency can be improved by using *spatial coherence*
- Edges that intersect scan-line i are likely to intersect i+1
- x_i changes predictably from scan-line i to i+1
- use an incremental algorithm that calculates the scan-line extrema from extrema of previous scan line by using
  - $x_{i+1} = x_i + 1/m$ where m is slope

Data Structures for Scan Conversion

- Edge Table:
- eliminate horizontal edges
- sorted by smaller y coordinate
  - use bucket sort with #buckets = #scan-lines
- in bucket order by increasing x-coordinate of lower endpoint
- add edge to linked-list for the scan-line corresponding to the lower vertex, storing:
  - y_upper: last scanline to consider
  - x_lower: starting x coordinate for edge
  - 1/m: for incrementing x; compute before shortening





Data Structures for Scan Conversion

- Active Edge List (AEL) or Active Edge Table (AET)
  - Linked list of active edges on the current scan-line, y sorted by their x intersection values.
  - Fill spans defined by (suitably rounded pairs)
  - Each active edge has the following information:
    - y_upper: last scanline to consider
    - x: edge's intersection with current y
    - 1/m: for incrementing x
- When move to next scan-line y+1 update AEL
  - remove those with y_upper = y
  - add new edges with y_min = y+1
  - compute new x intersections

Scan Conversion Algorithm

- Start with smallest y coordinate - lowest bucket
- initialize AEL to empty
- repeat until AEL and ET are empty
  - add edges in edge table with y_min=y to AEL
  - if AEL <> NULL
    - sort AEL by x - insertion or bubble sort
    - fill pixels between edge pairs
    - delete finished edges - y_upper=y
    - increment y by 1
    - update each non-vertical edge's x using 1/m

Generalizations

- Slivers - thin areas whose interior does not contain distinct spans for each scan line
  - missing pixels - a variant of the aliasing problem
- Multiple overlapping polygons
- Filling Ellipse Arcs
- Pattern Filling
- see Foley & van Dam 3.6 ; Hearn & Baker 3.11