Srinix College of Engineering, Balasore

Department of Computer Science & Engg.

Basic Concept of Region Filling Algorithm in Computer Graphics

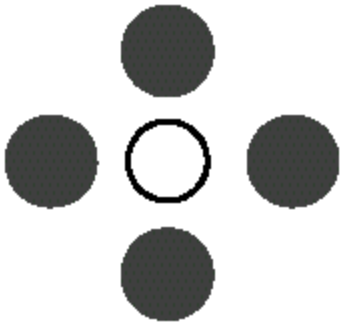
Prepared by
Krishna Bera
Asst. Professor
CSE dept
Email: krishnabera11@gmail.com

Region Filling

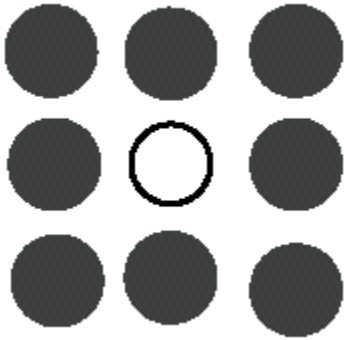
- Seed Fill Approaches
  - 2 algorithms: Boundary Fill and Flood Fill
  - works at the pixel level
  - suitable for interactive painting apllications
- Scan line Fill Approaches
  - works at the polygon level
  - better performance

Seed Fill Algorithms: Connectedness

- 4-connected region: From a given pixel, the region that you can get to by a series of 4 way moves (N, S, E and W)
- 8-connected region: From a given pixel, the region that you can get to by a series of 8 way moves (N, S, E, W, NE, NW, SE, and SW)



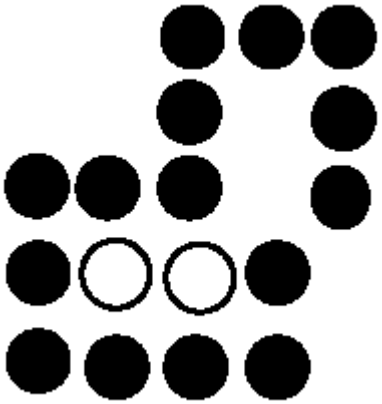
4-connected



8-connected

Boundary Fill Algorithm

- Start at a point inside a region
- Paint the interior outward to the edge
- The edge must be specified in a single color
- Fill the 4-connected or 8-connected region
- 4-connected fill is faster, but can have problems:



Boundary Fill Algorithm (cont.)

```
void BoundaryFill4(int x, int y,
                              color newcolor, color edgecolor)
{
   int current;
   current = ReadPixel(x, y);
   if(current != edgecolor && current != newcolor)
   {
      BoundaryFill4(x+1, y, newcolor, edgecolor);
      BoundaryFill4(x-1, y, newcolor, edgecolor);
      BoundaryFill4(x, y+1, newcolor, edgecolor);
      BoundaryFill4(x, y-1, newcolor, edgecolor);
   }
}
```

Flood Fill Algorithm

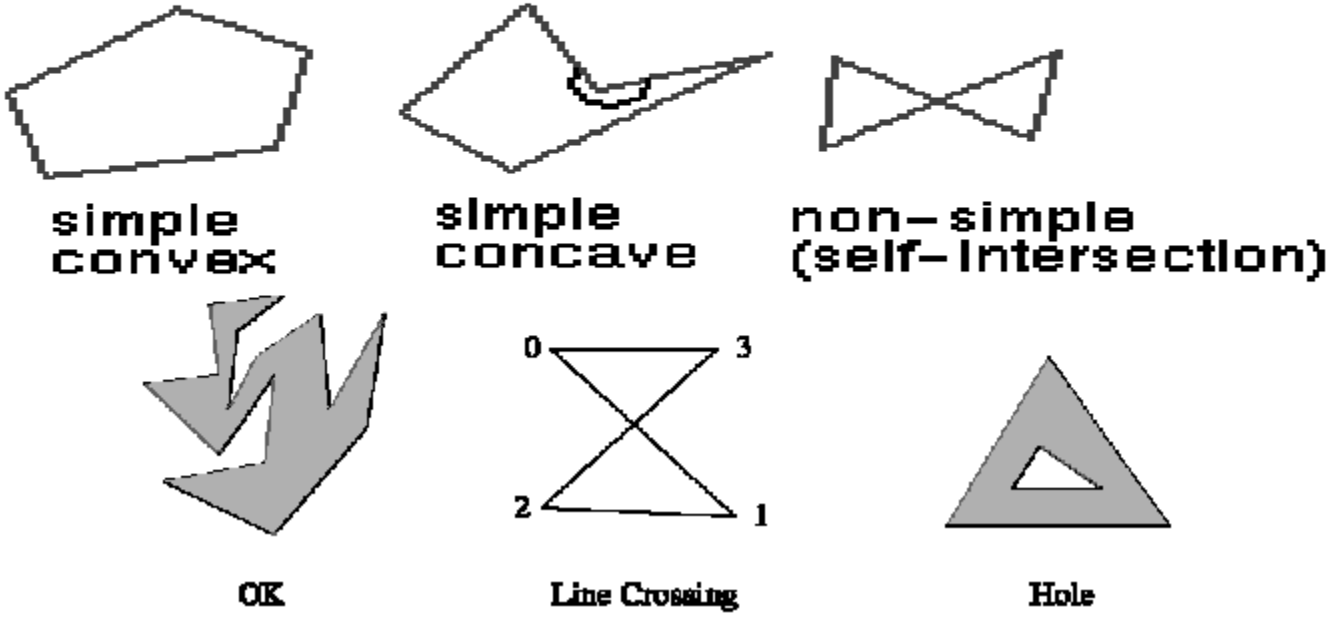
- Used when an area defined with multiple color boundaries
- Start at a point inside a region
- Replace a specified interior color (old color) with fill color
- Fill the 4-connected or 8-connected region until all interior points being replaced

Flood Fill Algorithm (cont.)

```
void FloodFill4(int x, int y, color newcolor, color oldColor)
{
   if(ReadPixel(x, y) == oldColor)
   {
      FloodFill4(x+1, y, newcolor, oldColor);
      FloodFill4(x-1, y, newcolor, oldColor);
      FloodFill4(x, y+1, newcolor, oldColor);
      FloodFill4(x, y-1, newcolor, oldColor);
   }
}
```

Polygon Types

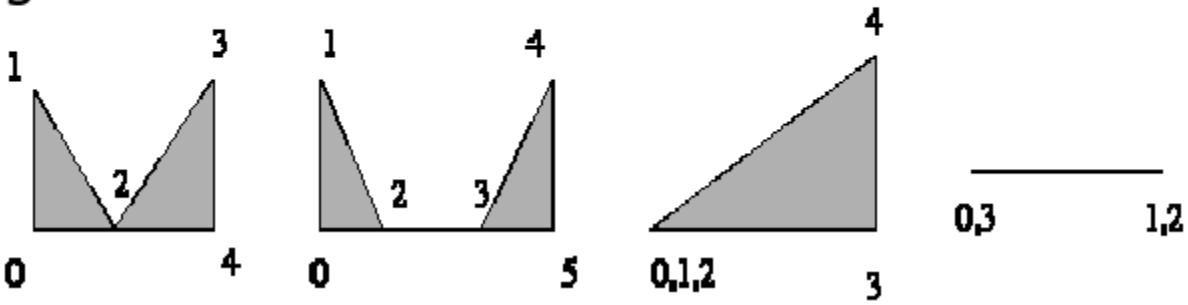
- simple convex, simple concave, non-simple (self-intersecting)
- want no holes, no intersections (line crossings)
- rectangles and triangles always simple convex



Convex, Concave, Degenerate

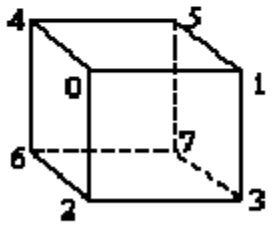
- Convex polygons are preferable to concave
  - Polygon is convex if for any two points inside polygon, the line segment joining these two points is also inside.

  - Degeneracies



Polygon Representation

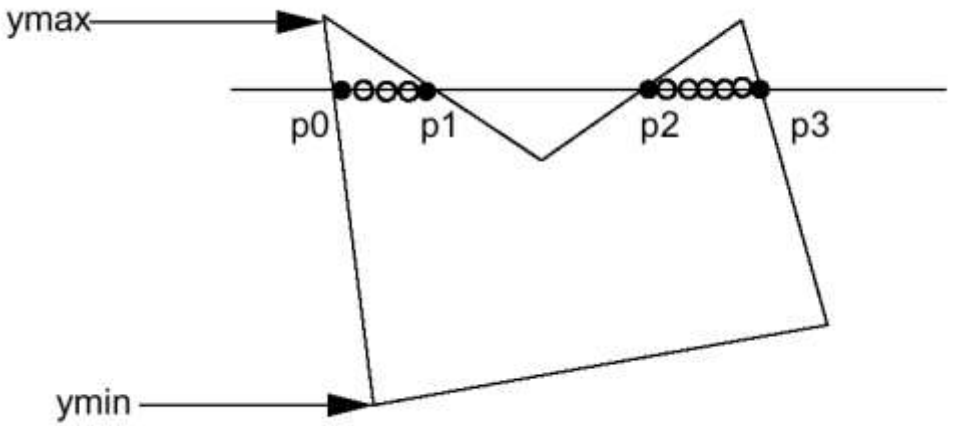
- Polygon Representation
  - ordered list of vertices
  - avoids redundant storage and computations
  - associate other information with vertices
    - colors, normals, textures



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

Scanline Fill Algorithm

- Intersect scanline with polygon edges
- Fill between pairs of intersections
- Basic algorithm:
  For y = ymin to ymax
  1) intersect scanline y with each edge
  2) sort intersections by increasing x [p0,p1,p2,p3]
  3) fill pairwise (p0->p1, p2->p3, …)



Spacial Handling

- Make sure we only fill the interior pixels

  Define interior:

  For a given pair of intersection points (Xi, Y), (Xj, Y)

  -> Fill ceilling(Xi) to floor(Xj)

  important when we have polygons adjacent to each other.

Spacial Handling (cont.)

- Intersection has an integer X coordinate

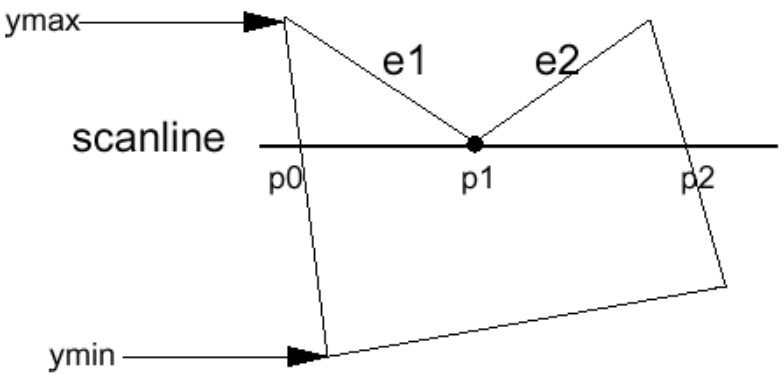
  ->if Xi is integer, we define it to be interior

  ->if Xj is integer, we define it to be exterior
  (so don't fill)

Spacial Handling (cont.)

- Intersection is an edge end point

Case 1



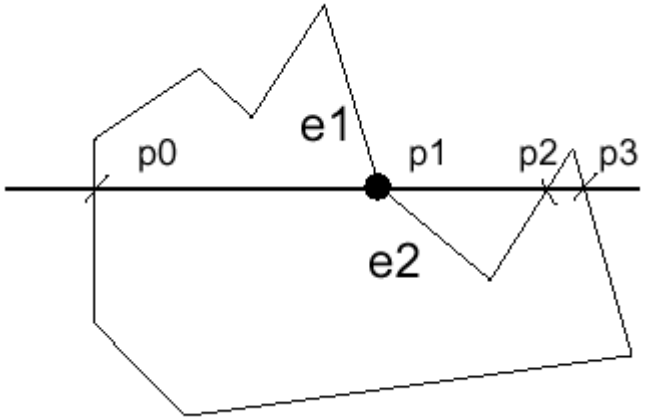
Intersection points: (p0, p1, p2) ???

->(p0,p1,p1,p2) so we can still fill pairwise

->In fact, if we compute the intersection of the scanline with edge e1 and e2 separately, we will get the intersection point p1 twice. Keep both of the p1.

Spacial Handling (cont.)

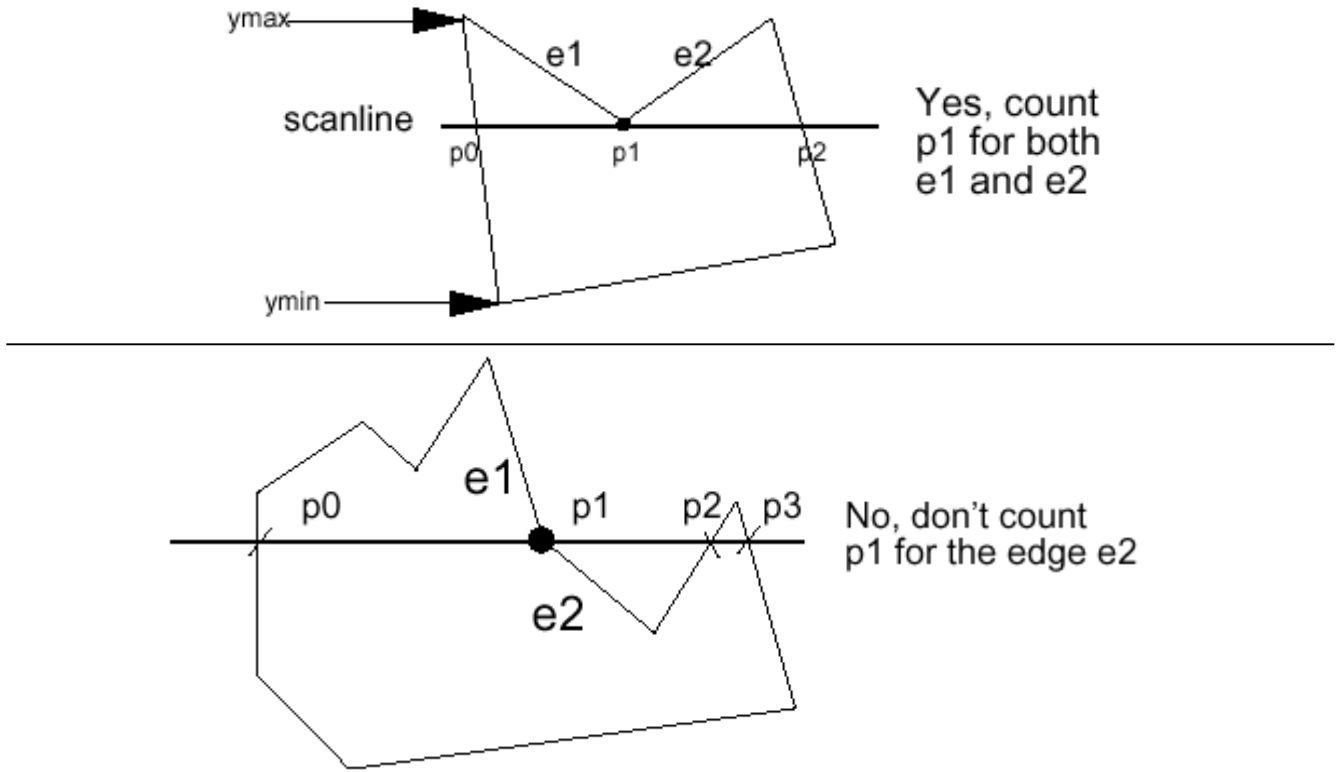
Case 2



However, in this case we don't want to count p1 twice (p0,p1,p1,p2,p3), otherwise we will fill pixels between p1 and p2, which is wrong.

Spacial Handling (cont.)

Summary: If the intersection is the ymin of the edge's endpoint, count it. Otherwise, don't.



References:

- Computer Graphics: Principles and Practice in C, by J. D. Foley, A. Van Dam, S. K. Feiner, J. F. Hughes. Addison-Wesley, 2nd ed.
- Essential Mathematics for Computer Graphics, fast, by John Vince. Springer.