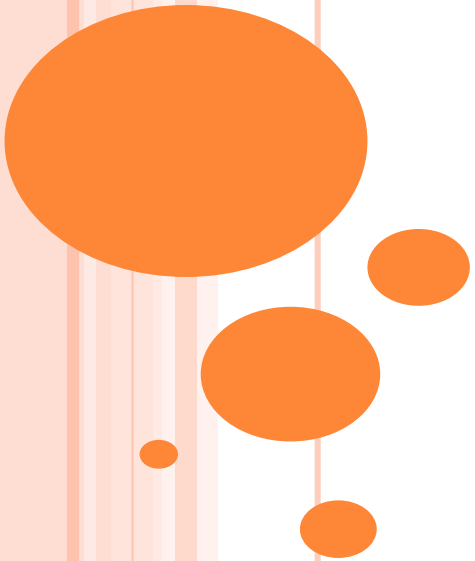
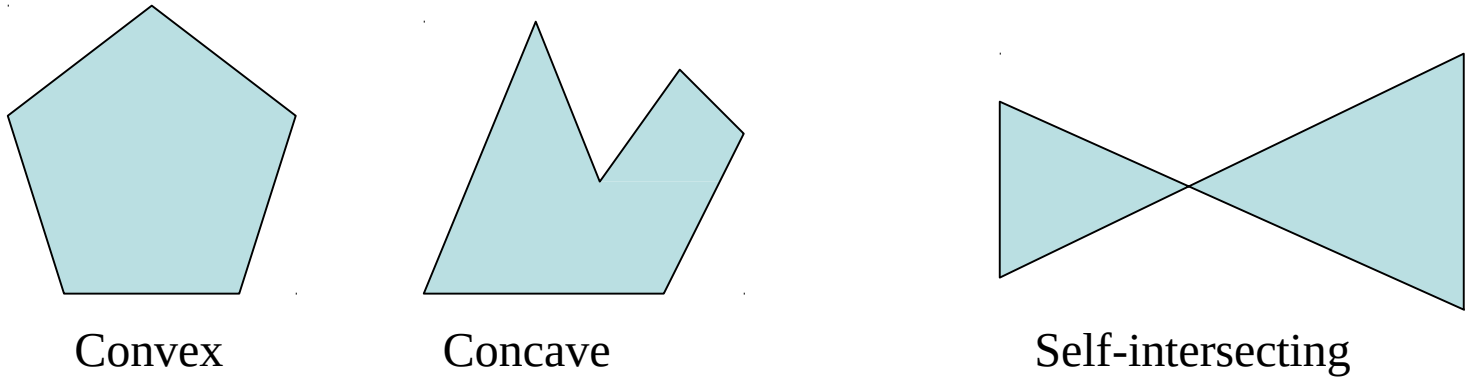


Filled Area Primitives

Polygon Types

- Different types of Polygons
  - Simple Convex
  - Simple Concave
  - Non-simple : self-intersecting
  - With holes

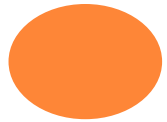


Convex Concave Self-intersecting

Scan-line Fill Algorithm

- A scan-line fill algorithm of a region is performed as follows:
  - Determining the intersection positions of the boundaries of the fill region with the screen scan lines.
  - Then the fill colors are applied to each section of a scan line that lies within the interior of the fill region.

- The simplest area to fill is a polygon, because each scan-line intersection point with a polygon boundary is obtained by solving a pair of simultaneous linear equations, where the equation for the scan line is simply y = constant.



Scan-line Fill Algorithm: Example

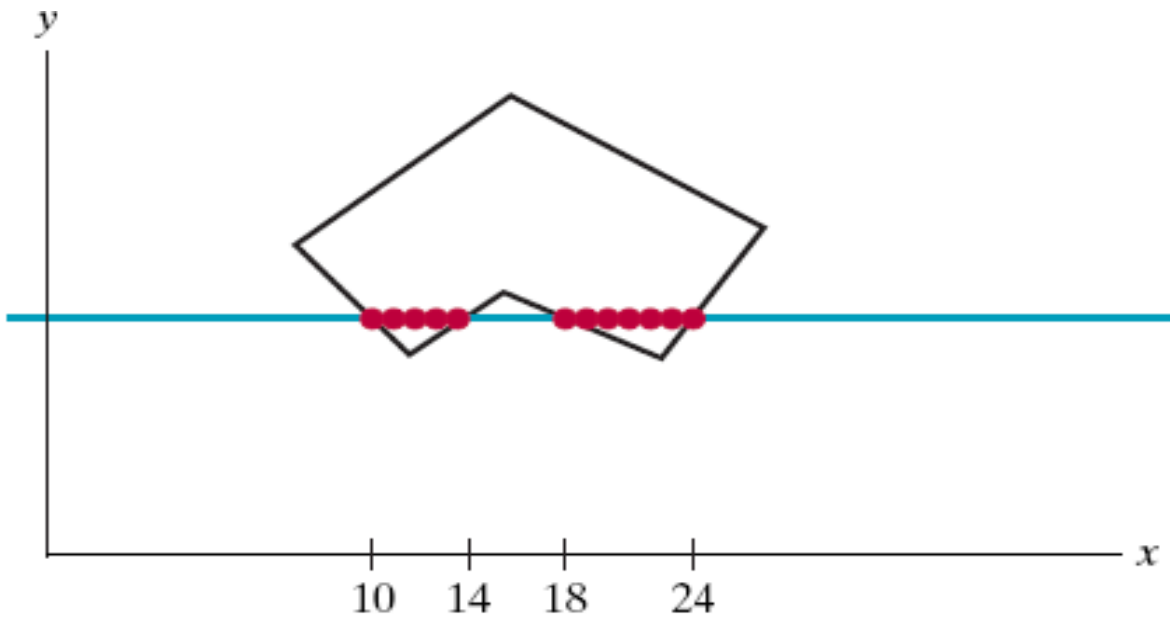
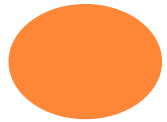


FIGURE 4–20 Interior pixels along a scan line passing through a polygon fill area.

Scan-line Fill Algorithm: Example (Contd…)

- For each scan line that crosses the polygon, the edge intersections are sorted from left to right, and then the pixel positions between, and including, each intersection pair are set to the specified fill color.

- In the previous Figure, the four pixel intersection positions with the polygon boundaries define two stretches of interior pixels.



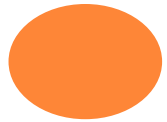
Scan-line Fill Algorithm: Example (Contd…)

- The fill color is applied to the five pixels:

  from x = 10 to x = 14 and

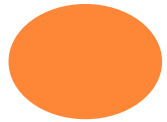
- To the seven pixels

  from x = 18 to x = 24.



Polygon Fill Algorithm

- However, the scan-line fill algorithm for a polygon is not quite as simple.
- Whenever a scan line passes through a vertex, it intersects two polygon edges at that point.
- In some cases, this can result in an odd number of boundary intersections for a scan line.



Polygon Fill Algorithm (Contd…)

- The figure shows two scan lines that cross a polygon fill area and intersect a vertex.
- Scan line y' intersects an even number of edges, and the two pairs of intersection points along this scan line correctly identify the interior pixel spans.
- But scan line y intersects five polygon edges.

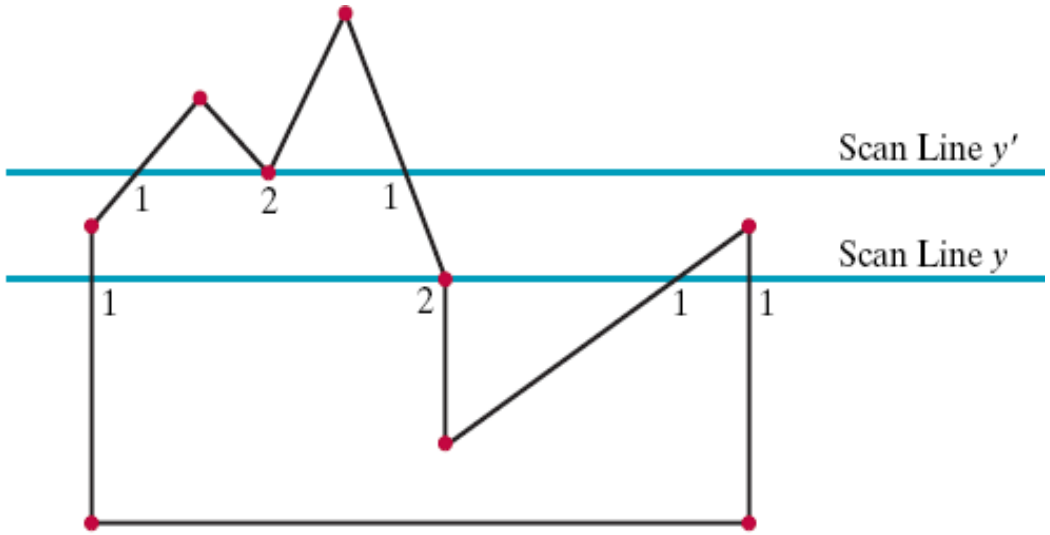


FIGURE 4-21 Intersection points along scan lines that intersect polygon vertices. Scan line y generates an odd number of intersections, but scan line y' generates an even number of intersections that can be paired to identify correctly the interior pixel spans.

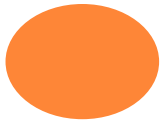
Polygon Fill Algorithm (Contd…)

- To identify the interior pixels for scan line y, we must count the vertex intersection as only one point.
- Thus, as we process scan lines, we need to distinguish between these two cases.
- We can detect the difference between the two cases by noting the position of the intersecting edges relative to the scan line.
- For scan line y, the two edges sharing an intersection vertex are on opposite sides of the scan line.
- But for scan line y', the two intersecting edges are both above the scan line.

Polygon Fill Algorithm (Contd…)

- A vertex that has adjoining edges on opposite sides of an intersecting scan line should be counted as just one boundary intersection point.

- We can identify these vertices by tracing around the polygon boundary in either clockwise or counterclockwise order and observing the relative changes in vertex y coordinates as we move from one edge to the next.



Polygon Fill Algorithm (Contd…)

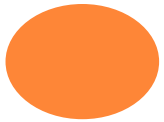
- If the endpoint y values of two consecutive edges increase or decrease, we need to count the shared (middle) vertex as a single intersection point for the scan line passing through that vertex.

- Otherwise, the shared vertex represents a local extremum (minimum or maximum) on the polygon boundary, and the two edge intersections with the scan line passing through that vertex can be added to the intersection list.

Inside-Outside Tests

- In CG applications often interior regions of objects have to be identified.

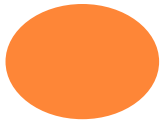
- Approaches:
  - Odd-even rule:
    - Draw a line from a point to outside of coordinate extents
    - Count line segments of the object crossing this line
    - If the number is odd then the point is interior, else exterior



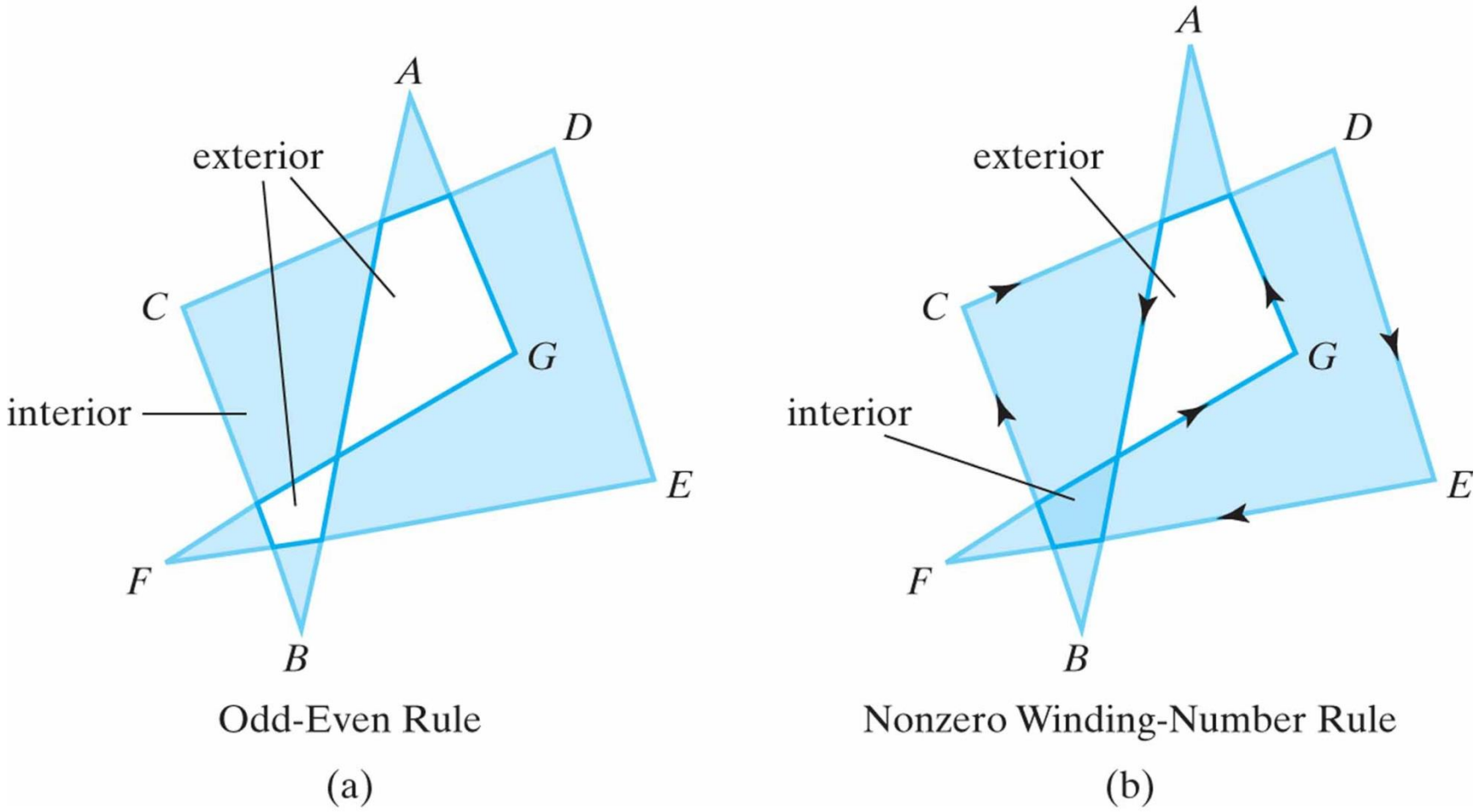
Inside-Outside Tests (Contd…)

- Nonzero winding-number rule:
  - Init winding-number to 0
  - Draw a line from a point
  - Move along the line
  - Count line segments of object crossing this line
  - If crossing line is from bottom-to-top; winding-number + 1, otherwise top-to bottom – 1
  - If winding-number ≠ 0 then point interior, else exterior

- But: How to determine directional boundary crossings?
- (Hint: Using vectors)



Inside-Outside Tests (Contd…)



Thank You

☺