Outline
- Filling problem
  - Flood fill algorithm
- Polygon filling
  - Point-in-polygon test
  - Per-pixel filling approach
- Per-scanline filling approach
- Scan conversion algorithm

Friday, January 15, 2016

Filling Problem
- How to display a filled 2D shape on a monitor?
  - Select all the pixels that are inside the 2D shape
  - Do it efficiently
- The shape can be:
  - Rectangle
  - Circle
  - Polygon

Flood Fill Algorithm
- Fill arbitrary inter-connected shapes
- Assumptions:
  - The boundary pixels of the shape are already drawn or labeled
    - By line or circle drawing algorithms
  - At least one inside pixel is known
    - Called seed pixel
    - May require point-in-shape test

Pseudocode for Flood Fill
- Push the seed pixel into the stack;
- while the stack is not empty {
  - Pop a pixel (u,v) from the stack;
  - Fill pixel (u,v) with required color;
  - for each pixel P in 
    
    \{(u-1,v),(u+1,v),(u,v-1),(u,v+1)\} 
  
    if P is a boundary pixel
    - continue;
    - if P has already been filled
      - continue;
    - Push pixel P into the stack;

- }

Polygon Filling
- Polygon is defined by a set of vertices:
  - The order of vertices is important
- Different types of polygons
  - Convex
  - Concave
  - Self-intersecting
  - With interior holes
**Point-in-Polygon Test**
- Shot a ray from the point P
- Can be arbitrary direction, but normally horizontal to the right
- Count the number of intersections
  - Odd -> inside
  - Even -> outside
- Complexity: \( O(k) \)
  - \( k \): number of edges

**Handle Special Cases**
- Ignore all horizontal edges
  - Edge ab
- Ignore vertex v, if v is on the ray and its y-coordinate is local maximum or minimum
  - Vertices c & e
- Count the rest intersections:
  - d & f

**Per-Pixel Filling Approach**
- Go through each pixel on screen
  - If the pixel is inside the polygon, draw it
  - The bounding rectangle of the polygon can be used to speed up
- Redundancy:
  - If a pixel is inside, its left & right neighbors on the same scanline are likely inside as well

**Per-Scanline Filling Approach**
- Uses inter-scanline coherence
  - For each scanline:
    - Find its intersections with all edges
      - Should be an even number
    - Sort the intersections by X coordinate
    - Pair up all the intersections
    - Fill all pixels between each pair

**Find Intersections**
- Requires calculating the coordinates in fractional value
  - Round up or down based on which side is interior
  - Cannot use the closest pixels found by line generation algorithm
- Redundancy:
  - The intersections for adjacent scanlines are related

**Parity Calculation**
- Requires sorting all intersections
- Need to consider special cases:
  - Ignore horizontal edges
  - Ignore vertex that is local minimum or maximum
- Redundancy:
  - Order of intersections doesn’t change much for adjacent scanlines
Scan Conversion Algorithm

- Utilize both inter-scanline & intra-scanline coherences:
  - When move to the next scanline, no need to recalculate & resort all the intersections
  - Intersection positions can be updated based on slope of the edge
    \[ m = \frac{y_a - y_b}{x_a - x_b}; \]
    \[ x_{i+1} = x_i + \frac{1}{m}; \]

Represent Edges with Nodes

- Each node keeps 3 variables:
  - \( y_{\text{last}} \): y coordinate of the lower vertex
  - \( x_{\text{cur}} \): x coordinate of the edge's intersection with current scanline
  - Initially set to the x-coordinate of the higher vertex
  - \( 1/m \): x increment between adjacent scanlines

Edge Table (ET)

- An array of linked-lists that stores all edges initially
- Each linked-list corresponds to one scanline
- Stores edges that start at this scanline
- When a given scanline is processed:
  - The associated edges are moved to the active edge list

Active Edge List (AEL)

- A linked-list of edges that intersect the current scanline
- Edges are sorted by the \( x_{\text{cur}} \) value
- When moving from one scanline to the next:
  - Add starting edges
  - Remove ending edges
  - Update \( x_{\text{coordinates}} \) of remaining edges
  - Reorder edges if necessary

Psuedocode for Scan Conversion

- for each edge \( k \) in the polygon
  - if \( k \) is not a horizontal edge
    - Add \( k \) to the ET at location of higher vertex;
  - Set \( y = \text{largest} \ y \text{ coordinate with an entry in ET}; \)
  - Initialize AEL to be empty;
  - while AEL or ET is not empty {
    - Move edges at location \( y \) of the ET to the AEL;
    - Remove edges whose \( y_{\text{last}} = y \) from the AEL;
    - Fill pixels on \( y \) using edge pairs in the AEL;
    - \( y \) --;
    - for each edge remaining in the AEL
      - \( x_{\text{cur}} += \frac{1}{m}; \)
  }