2D Graphics Primitives II

Additional issues in scan converting lines

1) Endpoint order

Want algorithms to draw the same pixels for each line

How handle?

a) draw only in one order - switch endpoints

b) algorithm to draw in both directions
   only problem is for error = 1/2 pixel
   e.g.: if choose E then when going in one direction, choose SW when going in opposite direction

Solution (a) can be a problem when drawing a patterned line - want pattern to start at specified start end point

   e.g. 001100110011 versus 110011001100
2) Starting at the edge of a clip rectangle

To clip line:
- can analytically clip with the sides of the clip rectangle
- on right - get integer coordinates
- on left - get non integer y
  - can't draw lines with non integer coordinates
  - could use closest pixel
    - Now scan-converted clipped line has different slope!
    - Correct by initializing decision variable to midpoint of next two points

If find intersection of line and \( y = y_{\text{min}} \) and then round \( x \),
  - don't get all the pixels

Thus intersect \( y = y_{\text{min}} - 0.5 \) with line, and round up the \( x \)
3) Varying intensity of line as a function of slope

Diagonal lines have less intensity than horizontals and verticals

If bi-level display - no solution

If multilevel display - can vary intensity of each point as function of average distance between points (thus of slope)

4) Outline primitives composed of lines

shared vertices should be drawn only once

Why:
  a) if write in XOR mode (sometimes done) then
      if background color initially set
      write pixel once and sets to foreground color
      write pixel twice and sets to background color

      Source 0 0 1 1
      Dest 0 1 0 1

      or 0 1 1 1
      xor 0 1 1 0 (draw & undraw)
      copy 0 0 1 1 (default)
      invert 1 0 1 0
      clear 0 0 0 0

  b) reduce number of memory accesses (writes)
Scan Conversion of Polygons

Rectangles

Polygons in general

How draw polygon?

Do we need a special point plotting routine for polygons?
What is OpenGL’s point plotting function for a line?
(GL_Lines)
Would you use this to draw a polygon?
Why GL_LINE_STRIP rather than GL_LINES?

How to draw a filled polygon

1) Draw polygon boundary (scan-conversion of lines)
   Then fill the boundary
2) Draw a filled polygon (scan-conversion of polygons)

Issues:

1) Edge adjacent polygons
   Example: Rectangles
   How specify?
      two opposite corners: (0,0) (20,10)
      upper left corner, width, height: 0, 0, 20, 10
         (easier to move)
      If have two rectangles: 0, 0, 20, 10
      20, 0, 20, 10
   Discretization Problem
      Shared edge
      Continuous
      Discrete
Define which pixels belong to a primitive

Interior pixels obviously belong
What about boundary pixels?
   Problem above, as middle column could belong to both
   Don't want to scan convert twice
   Need to decide what color to display

Solution:
   Left and bottom edges drawn and belong to rectangle
   Right and top edges not drawn and not belong

Applies to rectangles and to any other polygon
   How apply to any polygon?
   Go round polygon in counter clockwise direction
   and assign directions to edges

   \[ 0 \leq \text{angle} < \pi \]
   draw and include

   \[ \pi \leq \text{angle} < 2\pi \]
   don't draw or include
Drawing Rule

Still draws some points twice
Which points?

Can apply rule to filled and unfilled polygons
When would you apply it to unfilled?

Each span misses it's rightmost pixel
Each rectangle misses it's topmost span

Scan converting rectangles

Write rectangle a scan-line at a time
a span at a time
\[ y = y_c; \text{ } x_{\text{min}} < x \leq x_{\text{max}} \]

Exploits Spatial Coherence
Nearby pixels generally have the same value
When not true?

Exploits Span Coherence
All pixels on a span have the same value
Neighboring spans generally have the same value
Is this true for all polygons?

Edge Coherence
All pixels on an edge have the same value

Can bundle pixels together into words to reduce memory access
Draw and then fill Algorithms

Pixel defined filling versus polygon defined filling

Define pixel connectivity

Neighbors

4-neighbors

<table>
<thead>
<tr>
<th>i, j-1</th>
<th>i, j</th>
<th>i+1, j</th>
</tr>
</thead>
<tbody>
<tr>
<td>i-1, j</td>
<td>i, j</td>
<td>i+1, j</td>
</tr>
<tr>
<td>i, j+1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

8-neighbors

| i-1, j-1 | i, j-1 | i+1, j-1 |
| i-1, j | i, j | i+1, j |
| i, j+1 |

6-neighbors

| i-1, j-1 | i, j-1 | i+1, j |
| i-1, j | i, j | i+1, j |
| i, j+1 |

Path

A path from the pixel at \([i_0, j_0]\) to the pixel at \([i_n, j_n]\) is a sequence of pixel indices \([i_0, j_0], [i_1, j_1], \ldots, [i_n, j_n]\) s.t. the pixel at \([i_k, j_k]\) is a neighbor of the pixel at \([i_{k+1}, j_{k+1}]\) for all \(k\) with \(0 \leq k \leq n-1\).
A region (polygon) is n-connected if there exists an n-path between every pair of points in the region.

Define different connectivities for boundary and interior pixels

- If boundary is 8 connected and interior is 8 connected, then interior of polygon is connected to background.
- If boundary is 4 connected, then two boundary regions.
- If boundary is 8 connected and interior is 4 connected, then okay.

Here okay to have 4 connected boundary and 8 connected interior.

Must use opposite connectivity for boundary and interior in general.
Recursive Flood Fill:

Label (x, y, interior, new: Integer) ;
Begin
   If (pixel(x,y) == interior) Then
      pixel(x,y) = new;
      Label(x+1, y, interior, new);
      Label(x-1, y, interior, new);
      Label(x, y+1, interior, new);
      Label(x, y-1, interior, new);
   End; 
End Label;

Elegance versus Efficiency?
Scan Conversion of Filled Polygons

Uses spatial coherence

For each scan line crossing a polygon:
   Locate intersections of scan lines with polygon edges
   Sort intersections by x
   Fill horizontal regions using pairs of intersections
   (see Figure 1)

Scan lines passing through vertices
   add the intersection twice
   (see Figure 2)
   okay for y', not for y
   so add the intersection twice only if end points of two
   edges don't monotonically increase or decrease
   (add both only if a local extrema)
   now okay for both

OR

Shorten one edge at vertex where not extrema
   If monotonically decreasing, then shorten top
   of next segment such that y = y-1
   If monotonically increasing, then shorten
   top of current segment such that y - y-1
   (see Figure 3)

Use scan coherence to calculate intersections
   express slope in terms of coordinates of scan line intersection points
   \[ m = \frac{(y_{k+1} - y_k)}{(x_{k-1} - x_k)} \]
   \[ m = \frac{1}{(x_{k+1} - x_k)} \]
   \[ x_{k+1} = x_k + \frac{1}{m} \]
to avoid using fractions
\[ x_{k+1} = x_k + \frac{\Delta x}{\Delta y} \]

now initialize counter to 0
increment counter by $\Delta x$ as go up a scan line
when counter $\geq \Delta y$, then increment current $x$ intersection
and decrease counter by $\Delta y$

Example:
slope = $\frac{7}{3}$
counter inc = 3
counter = 0
$x = x_0 = 5$
y = $y_0 = 4$

intercepts counter
5,4 0
5,5 3
5,6 6
6,7 9 reset to 2
6,8 5
7,9 8 reset to 1

etc. This in essence truncates instead of rounding

Can increment by $2\Delta x$, and decrement by $2\Delta y$ and compare
to $\Delta y$
This rounds
How avoid multiplication?

Using a sorted edge table
Going clockwise around polygon, use bucket sort to store edges
sorted by smallest $y$ value of edge, and $x$ intercept of low pt
Don't store horizontal edges
Do edge shortening at monotonic vertices
Each entry:
max $y$ value of edge
$x$ intercept for lower vertex of edge
inverse of the slope of edge
($\Delta x$ and $\Delta y$) (see Figure)
Start at scan line at bottom of polygon and generate active edge list of all lines crossed by the scanline.

- Add new edges from sorted edge table
- Remove edges if $y > y_{\text{max}}$ of edge

If new entry in active edge list then compute scanline intersection else incrementally compute scanline intersection.

Store intersections in same sorted order.
Read off pairs of intersections and fill between them.

**Example:**

<table>
<thead>
<tr>
<th>ScanLine</th>
<th>Active List</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>AE, AB</td>
</tr>
<tr>
<td>2</td>
<td>AE, AB</td>
</tr>
<tr>
<td>3</td>
<td>CD, DE, AE, AB</td>
</tr>
<tr>
<td>4</td>
<td>CD, AB</td>
</tr>
<tr>
<td>5</td>
<td>CD, AB</td>
</tr>
<tr>
<td>6</td>
<td>CB, AB</td>
</tr>
<tr>
<td>7</td>
<td>CB, AB</td>
</tr>
<tr>
<td>8</td>
<td>CB, AB</td>
</tr>
<tr>
<td>9</td>
<td>none</td>
</tr>
</tbody>
</table>

**Inside/Outside Tests**

- Different Filling Rules
  - Odd-even rule
  - Nonzero winding rule