Text in Windows

Computer Graphics Attributes

Using Windows Stock Fonts

- Like stock pens, brushes
- Accessed with:

  GetStockObject(font_name);

  - Returns a handle to a font
  - Use by selecting into DC with SelectObject():

  Or --
  CDC::SelectStockObject(font_name);

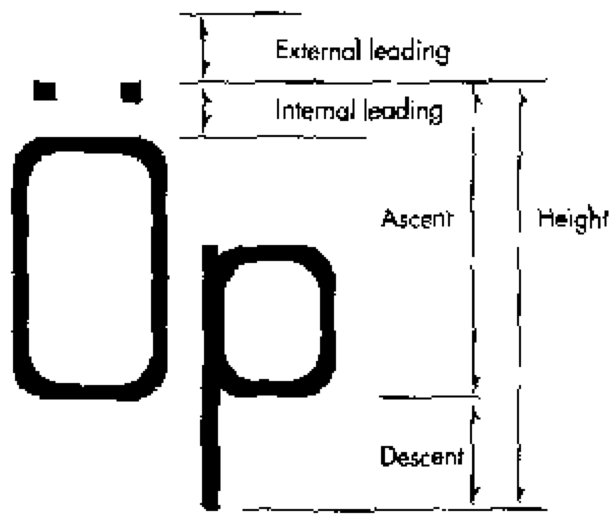
Using Windows Logical Fonts

- Instantiate a CFont object
- Use CFont::CreateFont(14 params!!)
  - Specify characteristics
  - Interpolates data from font file
  - --> new sizes, bold, rotated, etc.
- Select CFont object into the DC
- Called logical since determined by program logic not just file contents
- See online help

Windows Text Metrics

- CreateFont() may not give you exactly what you ask for
- Can use CDC::GetTextMetrics() to find out font details
  - Gives lots of information in a TEXTMETRIC structure
  - Commonly used to determine font size
    - can be used to set line spacing, caret size, sizes of buttons, etc.

Windows Text Metrics



Computer Graphics Attributes

- Line and Text Attributes
  - Fonts in Windows
- Area Fill
  - Boundary/Flood Fill Algorithms
  - Scanline Polygon Fill Algorithm

Attributes

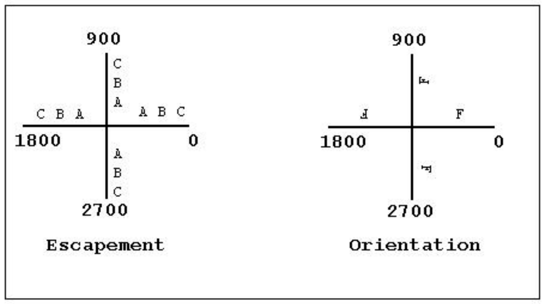
- How primitives are to be displayed
- Most systems use modal attributes
  - Values in effect until changed

Text Attributes

- Font (typeface)
  - Character set with particular design style
- Display style
  - underlined, italic, boldface, outlined, strikeout, spacing, etc.
- Color
- Size (width, height)--specified in points
  - Point = 1/72 inch

Text Attributes, continued

- Orientation--how much character is rotated
- Escapement--orientation of line between first & last character in a string



Character Escapement & Orientation

Line Attributes

- Color
- Width
- Style--solid, dotted, dashed, etc.

  Can be specified by giving a pattern array

  e.g., pat[ ]={1,1,1,1,1,1,0,0}

  Repeat this pattern on entire line:

  i^{th} pixel along line:

  ```
  if (pat[i%8]==1) SetPixel(x,y)
  ```

- In Windows, use a pen (CPen)

Area Fill

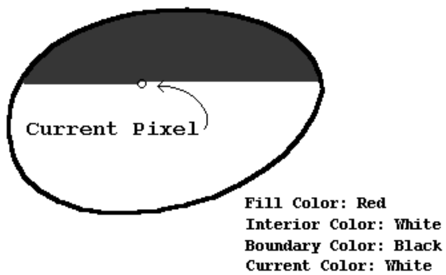
- Important for any closed output primitive
  - Polygons, Circles, Ellipses, etc.
- Attributes:
  - fill color
  - fill pattern
- 2 Types of area fill algorithms:
  - Boundary/Flood Fill Algorithms
  - Scanline Algorithms

Area Fill Algorithms

- See CS-460/560 Notes Web Page
- Link to:
  - Week 5-BC: Area Fill Algorithms
- URL:
  - http://www.cs.binghamton.edu/~reckert/460/fillalgs.htm

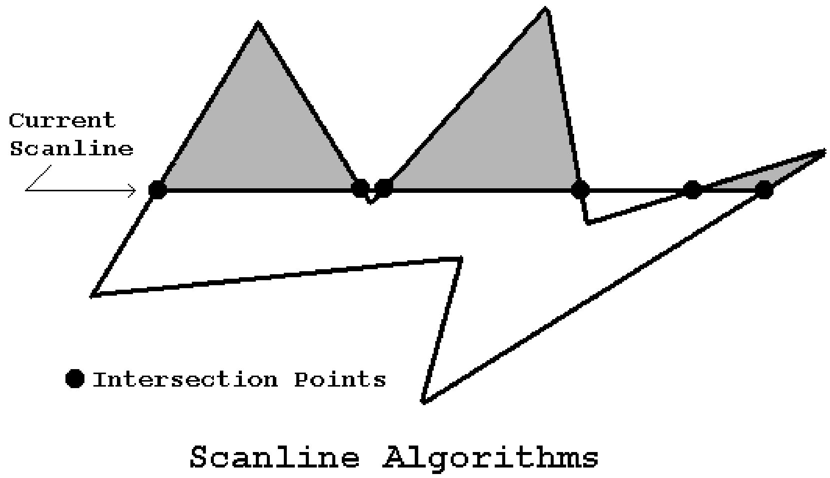
Boundary/Flood Fill Algorithms

- Determine which points are inside from pixel color information
  - e.g., interior color, boundary color, fill color, current pixel color
  - Color the ones that are inside.



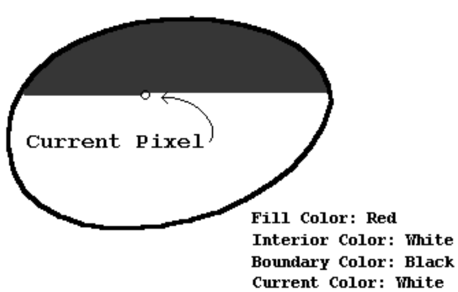
Scanline Algorithms

- Examine horizontal scanlines spanning area
- Find intersection points between current scanline and borders
- Color pixels along the scanline between alternate pairs of intersection points
- Especially useful for filling polygons
  - polygon intersection point calculations are very simple and fast
  - Use vertical and horizontal coherence to get new intersection points from old



Boundary/Flood Fill Algorithms

- Determine which points are inside from pixel color information
  - e.g., interior color, boundary color, fill color, current pixel color
  - Color the ones that are inside.



Connected Area Boundary Fill Algorithm

- For arbitrary closed areas
- Input:
  - Boundary Color (BC), Fill Color (FC)
  - (x,y) coordinates of seed point known to be inside
- Define a recursive BndFill(x,y,BC,FC) function:

  If pixel(x,y) not set to BC or FC, then set to FC

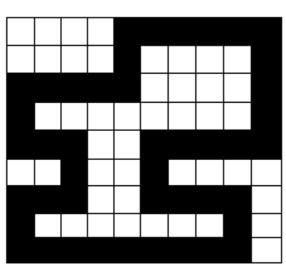
  Call BndFill() recursively for neighboring points

- To be able to implement this, need an inquire function
- e.g., Windows GetPixel(x,y)
  - returns color of pixel at (x,y)

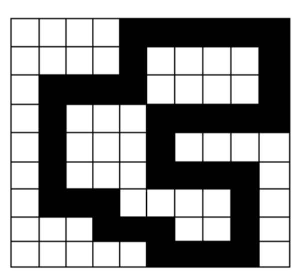
The BndFill() Function

```
BndFill(x,y,BC,FC)
{
   color = GetPixel(x,y)
   if ( (color != BC) && (color != FC) )
   {
      SetPixel(x,y,FC);
      BndFill(x+1,y,BC,FC);   BndFill(x,y+1,BC,FC);
      BndFill(x-1,y,BC,FC);  BndFill(x,y-1,BC,FC);
   }
}
```

- This would be called by code like:
  BndFill(50,100,5,8); // 5,8 are colors
  - Windows GDI: colors are COLORREFs
  - RGB() macro could be used
- As given, only works with 4-connected regions
- Boundary must be of a single color
  - Could have multiple interior colors



A 4-connected Region



An 8-connected Region

Flood Fill Algorithm

- A variation Boundary Fill
- Fill area identified by the interior color
  - instead of boundary color
- Good for single colored area with multicolor border

Ups & Downs of Boundary / Flood Fill

- Big Up: Can be used for arbitrary areas!
- BUT-- Deep Recursion so:
  - Uses enormous amounts of stack space
    - (Adjust stack size before building in Windows!)
- Also very slow since:
  - Extensive pushing/popping of stack
  - Pixels may be visited more than once
  - GetPixel() & SetPixel() called for each pixel
    - 2 accesses to frame buffer for each pixel plotted

Adjusting Stack Size in VC++

- ‘Project’ on Main Menu
  - Properties
    - Configuration Properties
      - Linker

        System

        Stack Reserve Size:

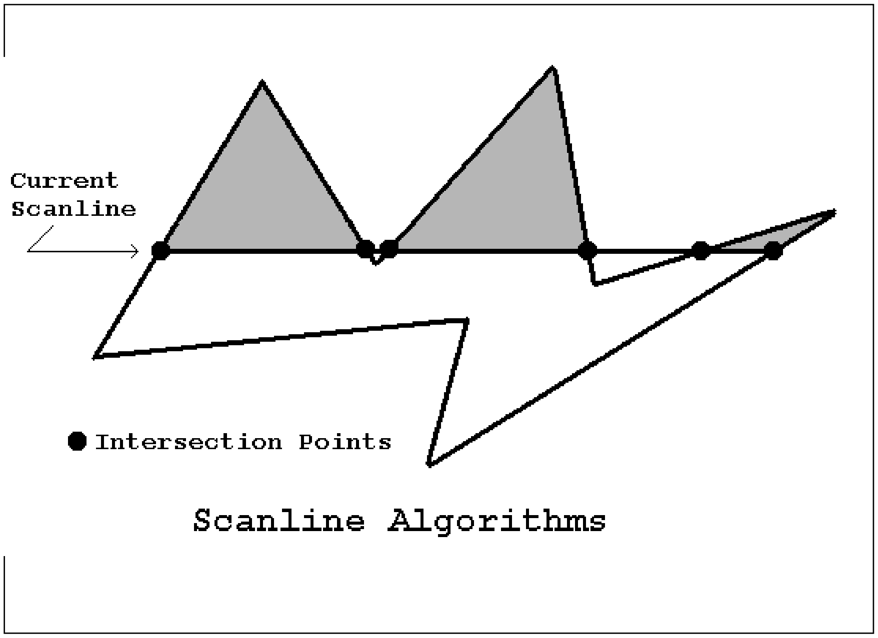
        perhaps 10000000

        Stack Commit Size:

        perhaps 8000000

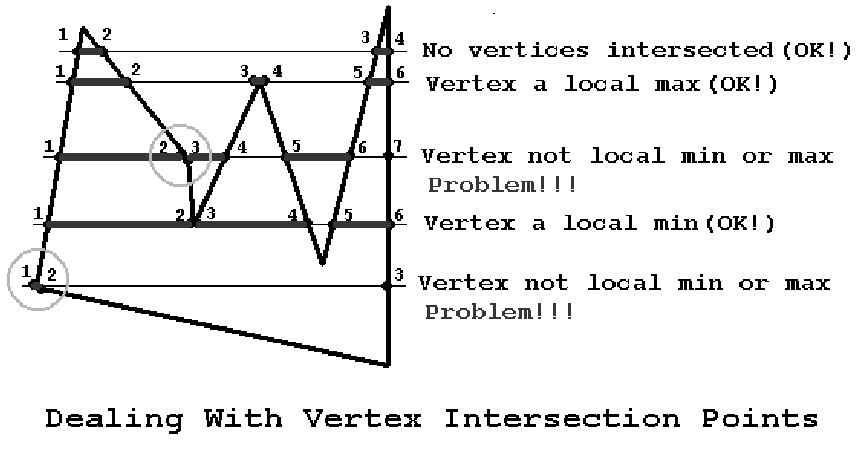
Scanline Polygon Fill Algorithm

- Look at individual scan lines
- Compute intersection points with polygon edges
- Fill between alternate pairs of intersection points



More specifically:

- For each scanline spanning the polygon:
  - Find intersection points with all edges the current scanline cuts
  - Sort intersection points by increasing x
  - Turn on all pixels between alternate pairs of intersection points
- But--
  - There may be a problem with intersection points that are polygon vertices



Vertex intersection points that are not local max or min must be preprocessed!

Preprocessing non-max/min intersection points

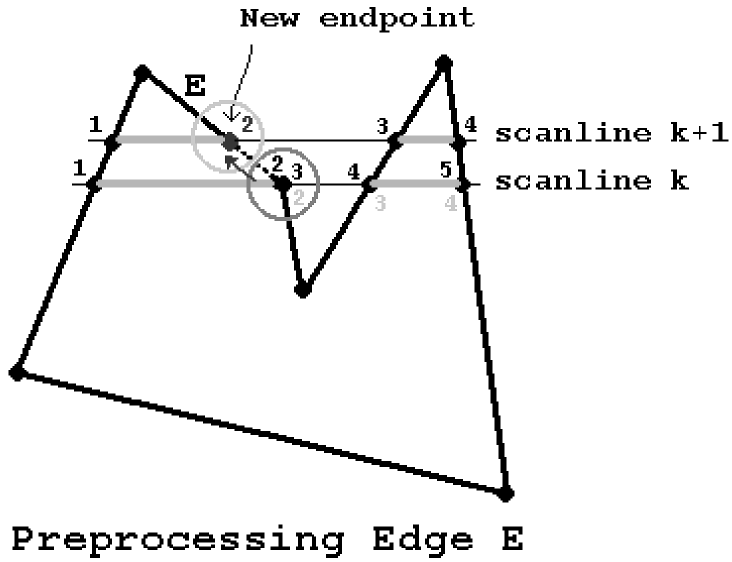
- Move lower endpoint of upper edge up by one pixel
- i.e., y <-- y + 1
- What about x?

  m = Δy/Δx, so Δx = (1/m) * Δy

  But Δy = 1, so:

  x <-- x + 1/m

Preprocessing



Active Edge

- A polygon edge intersected by the current scanline
- As polygon is scanned, edges will become active and inactive.
- Criterion for activating an edge:

  ysl = ymin of the edge

  (Here ysl = y of current scanline)
- Criterion for deactivating an edge:
- ysl = ymax of the edge

Vertical & Horizontal Coherence

- Moving from one scanline to next:
- y = y + 1
- If edge remains active, new intersection point coordinates will be:

  ynew = yold + 1

  xnew = xold + 1/m

  (1/m = inverse slope of edge)

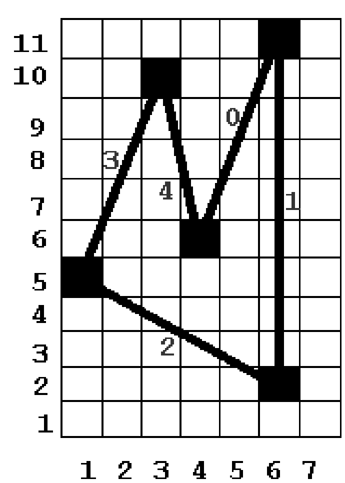
Scanline Polygon Fill Algorithm Input

- List of polygon vertices (xi,yi)

Scanline Polygon Fill Algorithm Data Structures

1. Edge table:
   – For each edge: edge #, ymin, ymax, x, 1/m
2. Activation Table:
   – (y, edge number activated at y)
     • Provides edge(s) activated for each new scanline
     • Constructed by doing a "bin" or "bucket" sort
3. Active Edge List (AEL):
   – Active edge numbers sorted on x
     • A dynamic data structure

Bin Sort for Activation Table



Edge Table

e	ymin	ymax
0	6	11
1	2	11
2	2	5
3	5	10
4	6	10

Activation Table

y	activated edge
2	1 2
3	2
4	
5	3
6	0 4

0

Scanline Polygon Fill Algorithm

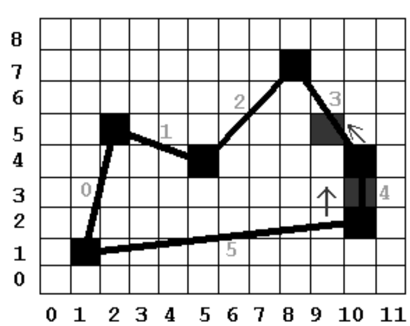
1. Set up edge table from vertex list; determine range of scanlines spanning polygon (miny, maxy)
2. Preprocess edges with nonlocal max/min endpoints
3. Set up activation table (bin sort)
4. For each scanline spanned by polygon:
   - Add new active edges to AEL using activation table
   - Sort active edge list on x
   - Fill between alternate pairs of points (x,y) in order of sorted active edges
   - For each edge e in active edge list:

     If (y != ymax[e]) Compute & store new x (x+=1/m)

     Else Delete edge e from the active edge list

Scanline Polygon Fill Algorithm Example

```
poly={1,1, 2,5, 5,4, 8,7, 10,4, 10,2, 1,1}
```



edge	x1	y1	x2	y2	sgn(Dy)
0	1	1	2	5	+
1	2	5	5	4	−
2	5	4	8	7	+
(3)	8	7	10	4	− ⟵
(4)	10	4	10	2	− ⟵
5	10	2	1	1	−
0	1	1	2	5	+

Activation Table

y	1	2	3	4	5	6	7
activated edge #s	0 5		4	1 2	3		

Edge Table

edge	1/m	ymin	x	ymax
0	1/4	1	1	5
1	-3	4	5	5
2	1	4	5	7
3	-2/3	4→5	10→9 1/3	7
4	0	2→3	10→10	4
5	9	1	1	2

Scanline Poly Fill Alg. (with example Data)

Edge Table (As Algorithm Executes)

Edge	1/m	ymax	ymin	x
0	1/4	5	1	1, 1.25, 1.5, 1.75, 2
1	-3	5	4	5, 2
2	1	7	4	5, 6, 7, 8
3	-2/3	7	5	9.33, 8.67, 8
4	0	4	3	10, 10
5	9	2	1	1, 10

Active Edge List (As it develops)

y	1	2	3	4	5	6	7
Active Edges	0,5	0,5	0,4	0,1,2,4	0,1,2,3	2,3	2,3
Fill between	1-1	1-10	2-10	2-5,5-10	2-2,6-9	7-9	8-8