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

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

In Windows, use a pen (CPen)
Area Fill

- 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:

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 int. pt. calculations are very simple
  - Use vertical and horizontal coherence to get new intersection points from old rapidly

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() 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

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
    - Linker System
      - Stack Reserve
        - Reserve: perhaps 10000000
        - Commit: 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

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 = \( y/\tilde{x} \), so \( \tilde{x} = (1/m) \cdot y \)
  - But \( \tilde{y} = 1 \), so:
    - \( x <-- x + 1/m \)
Preprocessing

A polygon edge intersected by the current scanline
As polygon is scanned, edges will become active and inactive.
Criterion for activating an edge:
\[ y_{sl} = y_{min} \] of the edge
(Here \( y_{sl} = y \) of current scanline)
Criterion for deactivating an edge:
\[ y_{sl} = y_{max} \] 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:
\[ y_{new} = y_{old} + 1 \]
\[ x_{new} = x_{old} + \frac{1}{m} \]
\( \frac{1}{m} \) = inverse slope of edge

Scanline Polygon Fill

Algorithm Input

List of polygon vertices \((x_i, y_i)\)

Algorithm Data Structures

1. Edge table:
   - For each edge: edge #, y_{min}, y_{max}, x, 1/m
2. Activation Table:
   - \( (y, \text{edge number activated at } y) \)
     - 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
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

Scanline Poly Fill Alg. (with example Data)

Video of BALSA Scanline Poly Fill Algorithm Animator

Demo of Scanline Polygon Fill Algorithm vs. Boundary Fill Algorithm