Rasterization So Far...

- Raster graphics
  - Line-drawing algorithms (DDA, Bresenham's)

- Now
  - Defining and filling Regions
  - Polygon drawing and filling
  - Antialiasing
Defining and Filling Regions of Pixels

- First, understand how to define and fill any defined regions
- Next, how to fill regions bounded by a polygon

Methods of Defining Regions

- Pixel-defined
  - Specifies pixels in color or geometric range

- Symbolic
  - Provides property that pixels in region must have

- Examples of symbolic regions
  - Closeness to some pixel
  - Within circle of radius $R$
  - Within a specified polygon
Pixel-Defined Regions

- **Definition:** Region R is the set of all pixels having color C that are connected to a given pixel S.
- **4-adjacent:** Pixels that lie next to each other horizontally or vertically, NOT diagonally.
- **8-adjacent:** 4-adjacent, plus diagonals.
- **4-connected:** If there is an unbroken path of 4-adjacent pixels.
- **8-connected:** Unbroken path of 8-adjacent pixels.

Recursive Flood-Fill Algorithm

- Recursive algorithm.
- Starts from initial pixel of color \texttt{initColor}.
- Recursively set 4-connected neighbors to \texttt{newColor}.
- **Flood-Fill:** floods region with \texttt{newColor}.
- **Basic idea:**
  - Start at "seed" pixel \((x, y)\).
  - If \((x, y)\) has color \texttt{initColor}, change it to \texttt{newColor}.
  - Do same recursively for all 4 neighbors.
Recursive Flood-Fill Algorithm (cont.)

```c
void floodFill( short x, short y, 
               short initColor ) {
    if( getPixel( x, y ) == initColor ) {
        setPixel( x, y );
        floodFill( x - 1, y, initColor ); // left
        floodFill( x + 1, y, initColor ); // right
        floodFill( x, y + 1, initColor ); // up
        floodFill( x, y - 1, initColor ); // down
    }
}
```

**Note:** `getPixel(x,y)` used to interrogate pixel color at `(x, y)`

---

Okay, now you try it.
Recursive Flood-Fill Algorithm (cont.)

- This version defines region using initColor
- Can also have version defining region by boundary
- Recursive flood-fill is somewhat blind
  - Some pixels may be retested several times before algorithm terminates
- Region coherence is likelihood that an interior pixel will be adjacent to another interior pixel
- Coherence can be used to improve algorithm performance
- A run is a group of adjacent pixels lying on same scan line
- Exploit runs of pixels

Region Filling Using Coherence

- Start at seed s
Region Filling Using Coherence Pseudocode

Push address of seed pixel onto stack
while(stack is not empty)  {
    Pop the stack to provide next seed
    Fill in the run defined by the seed
    In the row above, find the reachable interior runs
    Push the address of their rightmost pixels
    Do the same for row below current run
}

Note: Most efficient if there is span coherence (pixels on scan line have same value) and scan-line coherence (consecutive scan lines are similar)

Filling Polygon-Defined Regions

Problem: Region defined by Polygon P with vertices P_i = (X_i, Y_i), for i = 1...N, specifying sequence of P's vertices
Filling Polygon-Defined Regions (cont.)

- Solution: Progress through frame buffer, scan line by scan line, filling in appropriate portions of each line.
- Filled portions defined by intersection of scan line and polygon edges.
- Runs lying between edges inside P are filled.

Filling Polygon-Defined Regions Pseudocode

```plaintext
for (each scan Line L) {
    Find intersections of L with all edges of P
    Sort the intersections by increasing x-value
    Fill pixel runs between all pairs of intersections
}
```
Filling Polygon-Defined Regions (cont.)

- **Example:** scan line \( y = 3 \) intersects 4 edges \( e_3, e_4, e_5, e_6 \)
- Sort x values of intersections and fill runs in pairs
- **Note:** At each intersection, use inside-outside (parity), or vice versa

![Diagram](image1)

What if two polygons A, B share an edge?

- Algorithm behavior could result in
  - Setting edge first in one color and then another
  - Drawing edge twice too bright
- **Make Rule:** When two polygons share edge, each polygon owns its left and bottom edges
- E.g., below draw shared edge with color of polygon B

![Diagram](image2)

*Read:* Hill: pg 481
Filling Polygon-Defined Regions (cont.)

- How do we handle cases where scan line intersects with polygon endpoints?
- Solution: Discard intersections with horizontal edges, and with upper endpoint of any edge

Antialiasing

- Raster displays have pixels as rectangles
- Aliasing: Discrete nature of pixels introduces "jaggies"
Antialiasing (cont.)

- Aliasing effects
  - Distant objects may disappear entirely
  - Objects can blink on and off in animations

- Antialiasing techniques involve some form of *blurring* to reduce contrast, smooth image

- Three main antialiasing techniques
  - Prefiltering
  - Supersampling
  - Postfiltering

Prefiltering

- Basic idea
  - Compute area of polygon coverage
  - Use proportional intensity value

- Example: if polygon covers 1/2 of the pixel
  - use 1/2 polygon color
  - add it to 1/2 of adjacent region color

- Cons: computing pixel coverage can be time consuming
Supersampling

- Useful if we can compute color of any (x,y) value on the screen
- Increase frequency of sampling
- Instead of (x,y) samples in increments of 1, sample (x,y) in fractional (e.g., 1/2) increments
- Find average of samples
- Example: Triple sampling = increments of 1/2 = 9 color values averaged for each pixel

Postfiltering

- Supersampling uses average
- Gives all samples equal importance
- Postfiltering: use weighting (different levels of importance)
  - Compute pixel value as weighted average
  - Samples close to pixel center given more weight

Sample weighting

<table>
<thead>
<tr>
<th></th>
<th>1/16</th>
<th>1/16</th>
<th>1/16</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1/16</td>
<td>1/2</td>
<td>1/16</td>
</tr>
<tr>
<td></td>
<td>1/16</td>
<td>1/16</td>
<td>1/16</td>
</tr>
</tbody>
</table>
Antialiasing in OpenGL

- Many alternatives
- Simplest: accumulation buffer
  - Extra storage, similar to frame buffer
- Samples are accumulated
- When all slightly perturbed samples are done, copy results to frame buffer and draw

```c
// First initialize
glutInitDisplayMode( GLUT_SINGLE | GLUT_RGB | GLUT_ACCUM | GLUT_DEPTH );

// Zero out accumulation buffer
glClear( GLUT_ACCUM_BUFFER_BIT );

// Add samples to accumulation buffer using glAccum( )
```
OpenGL Antialiasing Sample Code

```c
void antialias(void)
{
    glClear( GL_ACCUM_BUFFER_BIT );
    for( int i = 0; i < 8; i++ )  {
        cam.slide(f*jitter[i].x, f*jitter[i].y, 0);
        display( );
        glAccum( GL_ACCUM, 1/8.0 );
    }
    glAccum( GL_RETURN, 1.0 );
}
```

- `jitter[]` stores randomized slight displacements of camera,
- `Factor, f` controls amount of overall sliding

Antialiasing Example
Antialiasing Example

References

- Hill, Chapter 9