Computer Graphics
CS 543 Lecture 13a
Polygon Filling & Antialiasing

Prof Emmanuel Agu

Computer Science Dept.

Worcester Polytechnic Institute (WPI)

Announcement

- Course Evaluations
  - Now online
  - Will be available from Friday, Dec 7
  - Will also allow students to do it in class, first 15 mins of next week's class
- Also, Exam 3 next week

Defining and Filling Regions of Pixels

Ref: Hill and Kelley, Computer Graphics Using OpenGL (3rd edition), pgs 476-487

- Methods of defining region
  - **Pixel-defined:** specifies pixels in color or geometric range
  - **Symbolic:** provides property pixels in region must have
  - Examples of symbolic:
    - Closeness to some pixel
    - Within circle of radius R
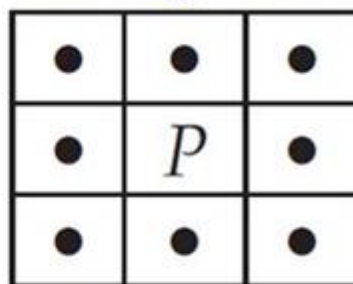   - Within a specified polygon
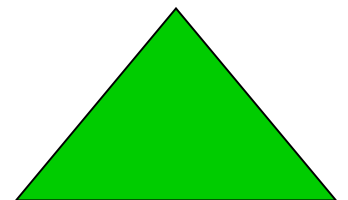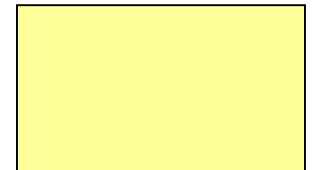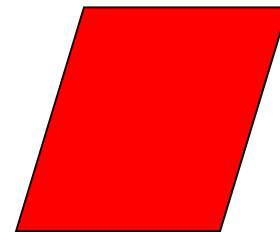
Pixel-Defined Regions

- **Definition:** Region R is 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:** pixels that lie next to each other horizontally, vertically OR diagonally
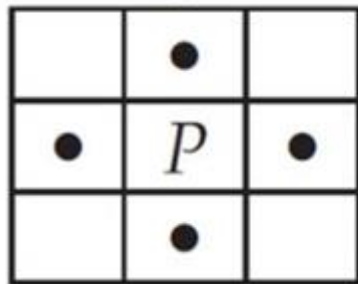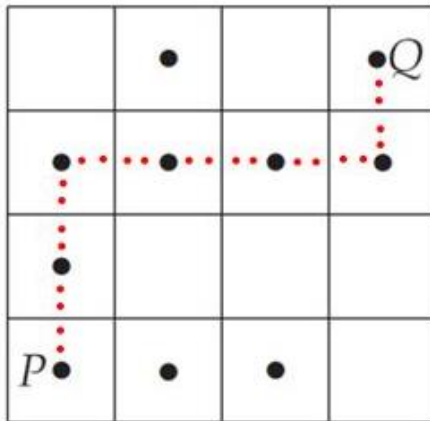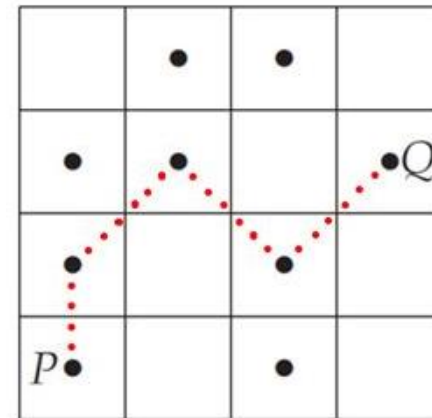
4-adjacent   8-adjacent

Pixel-Defined Regions

- **Definition:** Region R is set of all pixels having color C that are connected to a given pixel S

- **4-connected:** if there is unbroken path of 4-adjacent pixels connecting them

- **8-connected:** unbroken path of 8-adjacent pixels connecting them
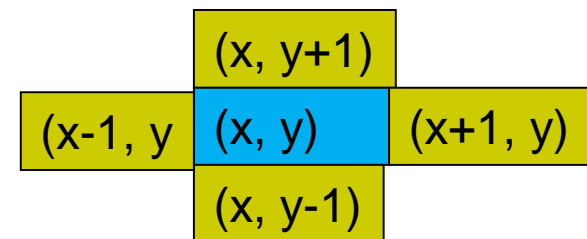
4-connected

8-connected

Recursive Flood-Fill Algorithm

- Recursive algorithm

- Starts from initial pixel of color, `intColor`

- Recursively set 4-connected neighbors to `newColor`

- **Flood-Fill**: floods region with `newColor`

- **Basic idea:**

  - start at "seed" pixel (x, y)

  - If pixel (x, y) has color **intColor**, change it to **newColor**

  - Do same recursively for all 4 neighbors

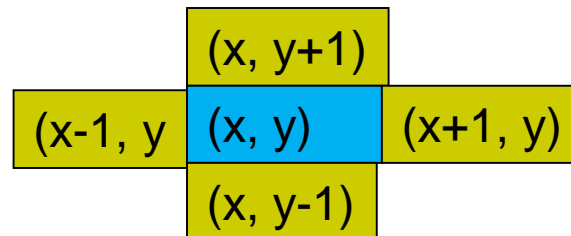|	(x, y+1)	
(x-1, y)	(x, y)	(x+1, y)
	(x, y-1)	

Recursive Flood-Fill Algorithm

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

```
void floodFill(short x, short y, short intColor)
{
    if(getPixel(x, y) == intColor)
    {
        setPixel(x, y);
        floodFill(x – 1, y, intColor); // left pixel
        floodFill(x + 1, y, intColor); // right pixel
        floodFill(x, y + 1, intColor); // upper pixel
        floodFill(x, y – 1, intColor); // lower pixel
    }
}
```

|	(x, y+1)	
(x-1, y	(x, y)	(x+1, y)
	(x, y-1)	

Recursive Flood-Fill Algorithm

- Recursive flood-fill is blind

- Some pixels retested several times

- **Coherence** can be used to improve algorithm performance (i.e. usually **groups of pixels** need coloring)

- **Region coherence** is likelihood that an interior pixel is most likely adjacent to another interior pixel

- **A run:** group of adjacent pixels lying on same scanline

- Fill runs(adjacent, on same scan line) of pixels

Region Filling Using Coherence

- Example: start at s, initial seed
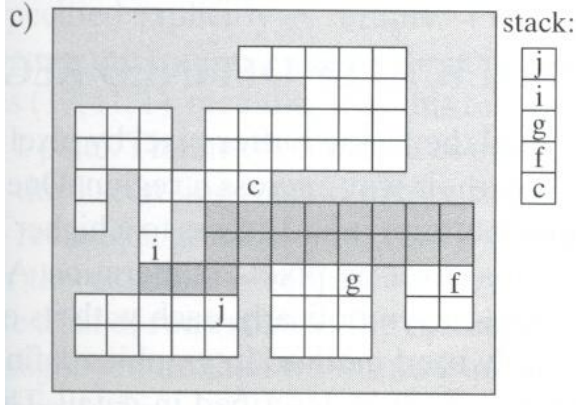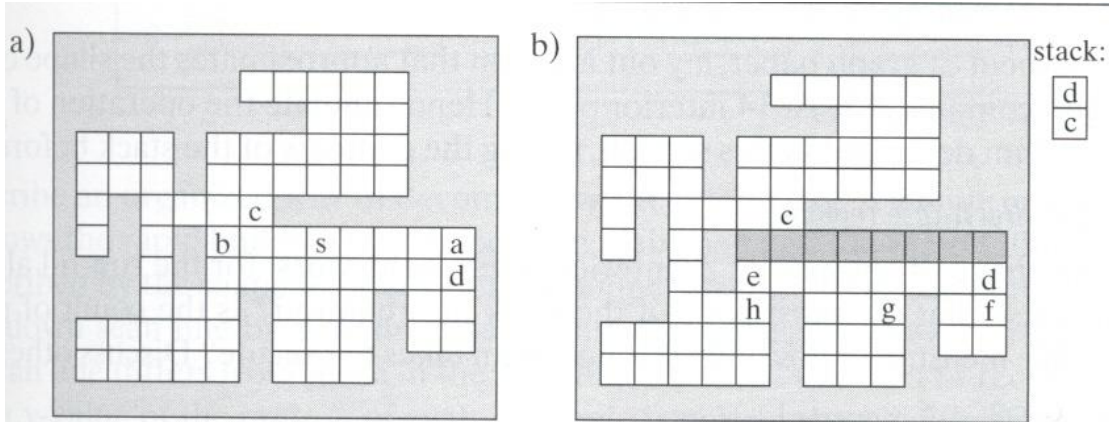
Pseudocode:

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

Note: algorithm most efficient if there is **span coherence** (pixels on scanline have same value) and **scan-line coherence** (consecutive scanlines similar)

Filling Polygon-Defined Regions

- **Problem:** Region defined polygon with vertices

  Pi = (Xi, Yi), for i = 1…N, specifying sequence of P's vertices

P1 P2

P7 P3

P5

P6 P4
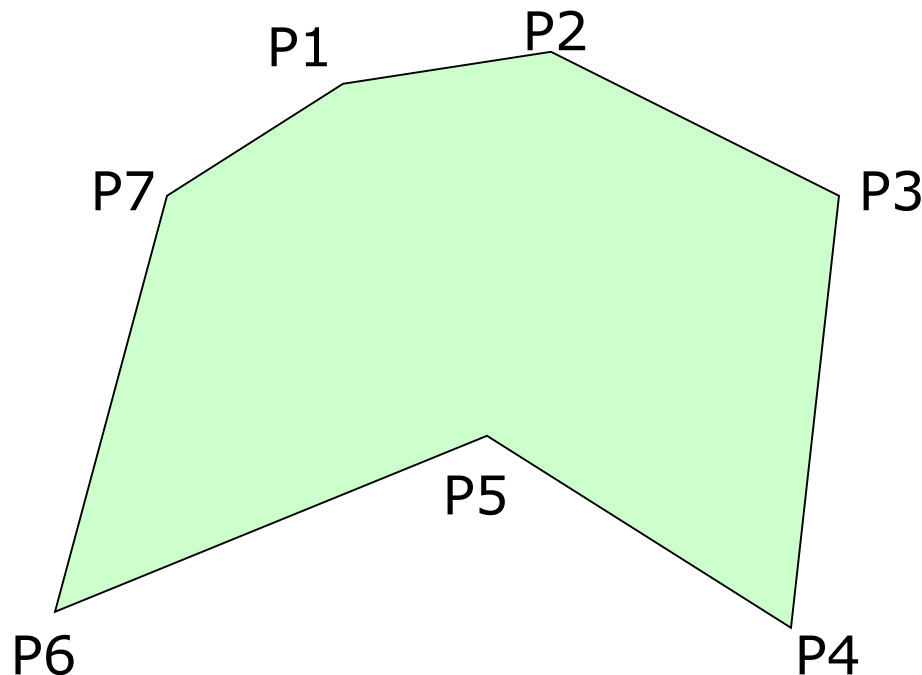
Filling Polygon-Defined Regions

- **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

- **Pseudocode:**

```
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
}
```
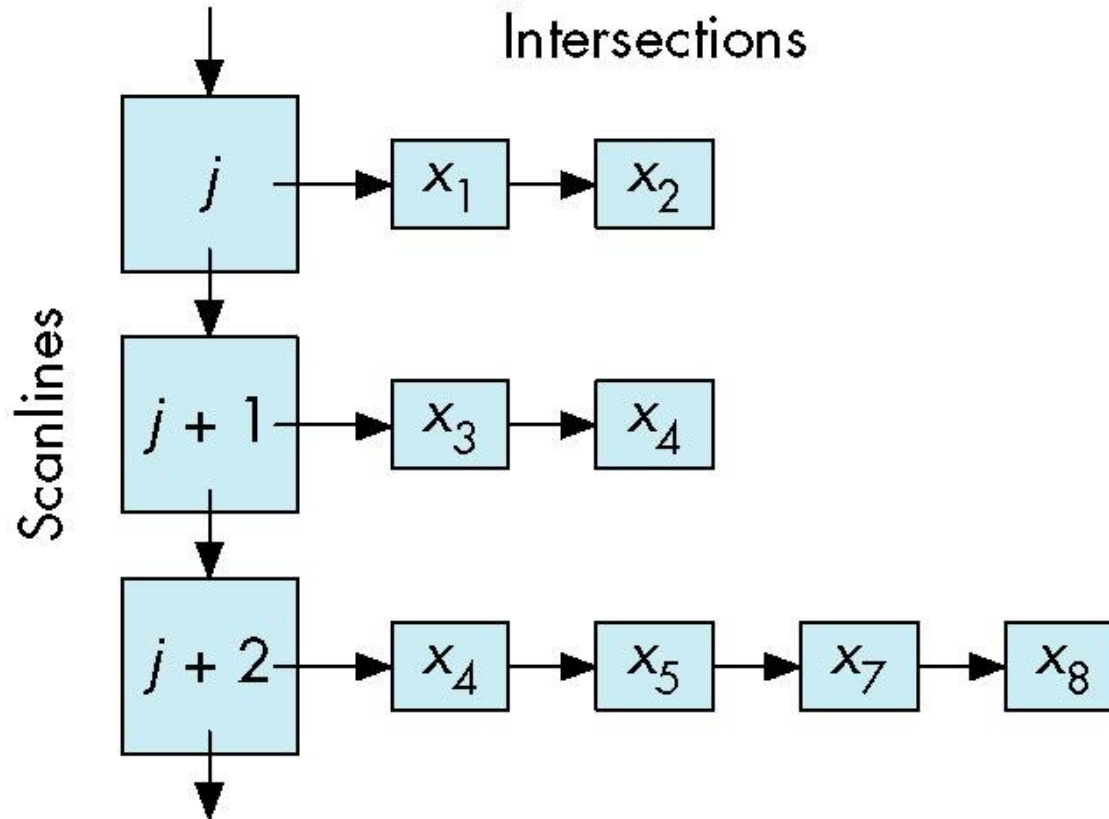
Filling Polygon-Defined Regions

- **Example:** scan line y = 3 intersects 4 edges e3, e4, e5, e6
- Sort x values of intersections and fill runs in pairs
- **Note:** at each intersection, inside-outside (parity), or vice versa
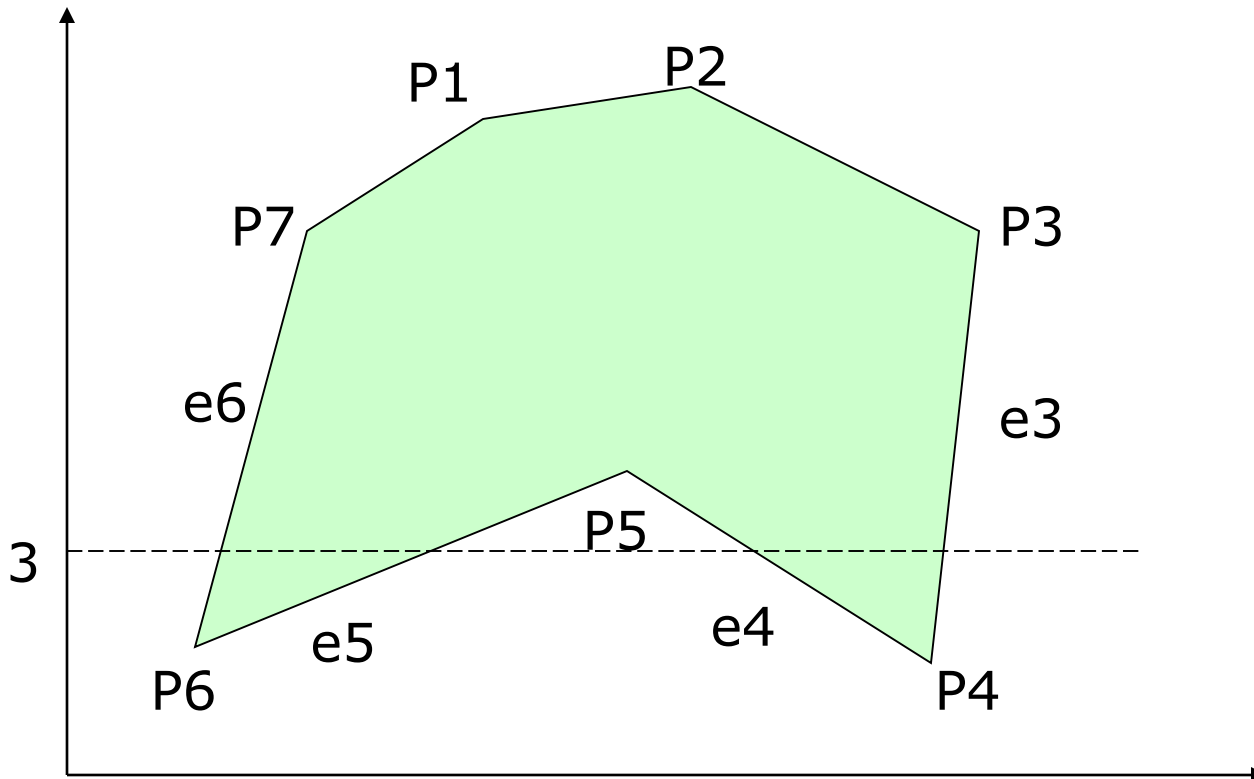
Data Structure
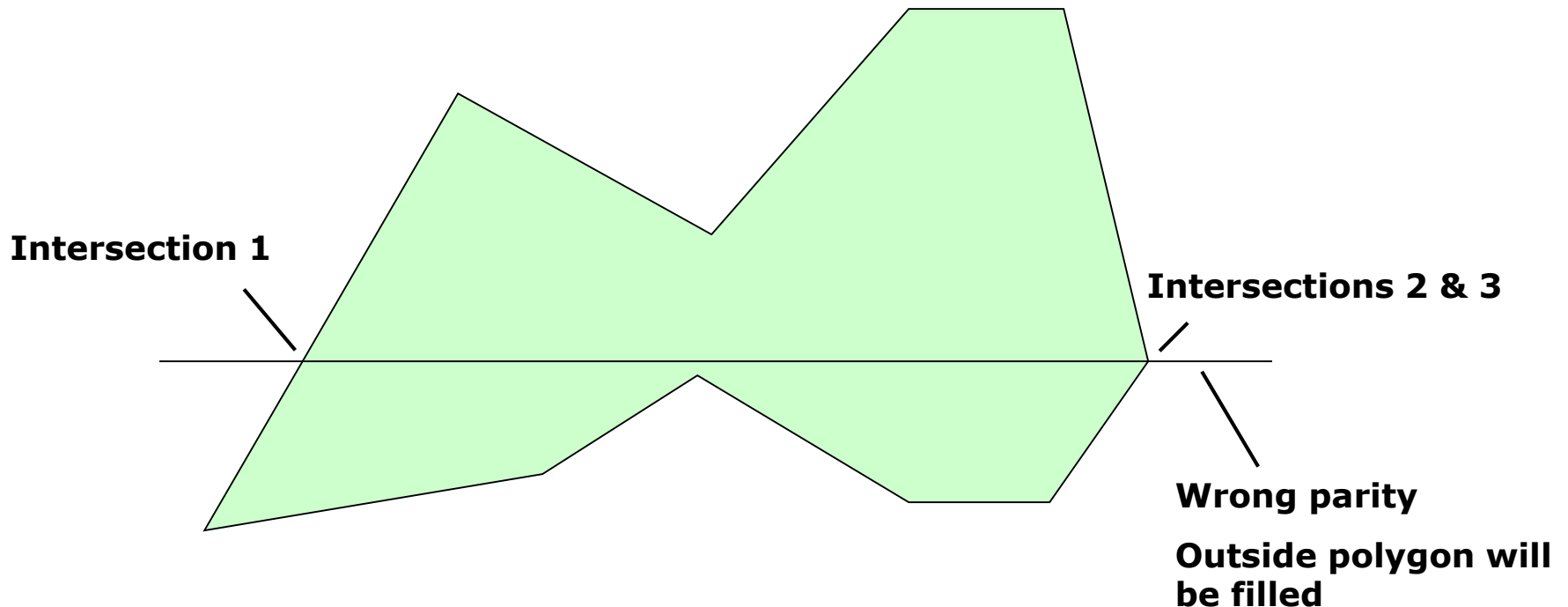
Filling Polygon-Defined Regions

- **Problem:** What if two polygons A, B share an edge?
- Algorithm behavior could result in:
  - setting edge first in one color and the 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**
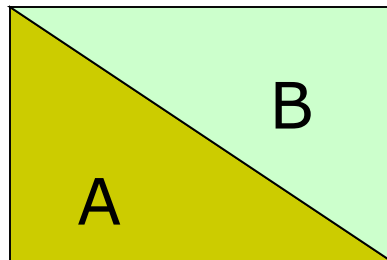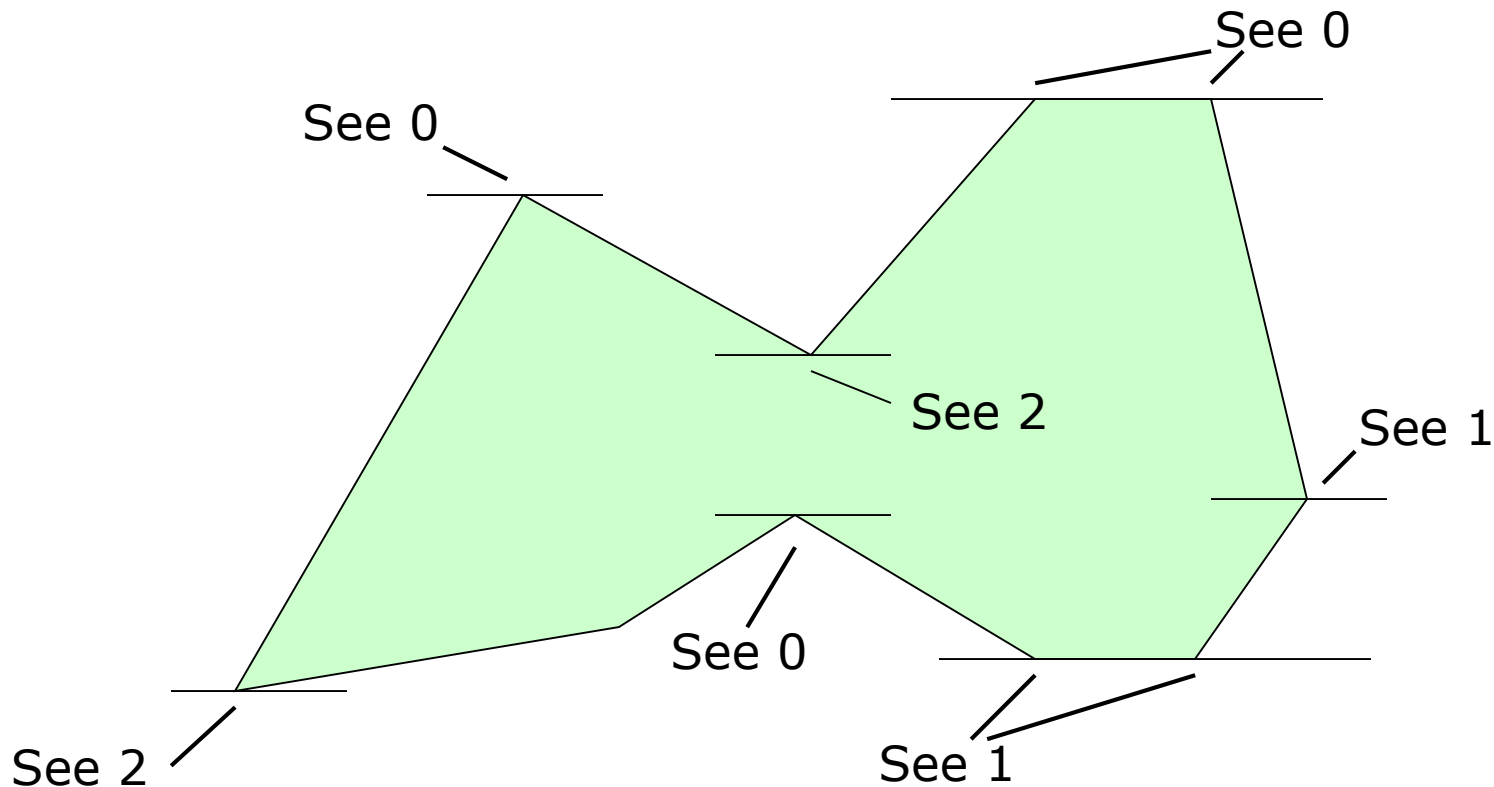
Filling Polygon-Defined Regions

- **Problem:** How to handle cases where scan line intersects with polygon endpoints to avoid wrong parity?
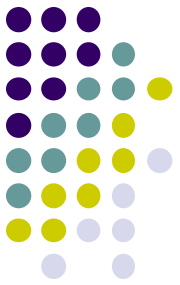
- Example

Filling Polygon-Defined Regions

- **Solution:** Discard intersections with horizontal edges and with upper endpoint of any edge
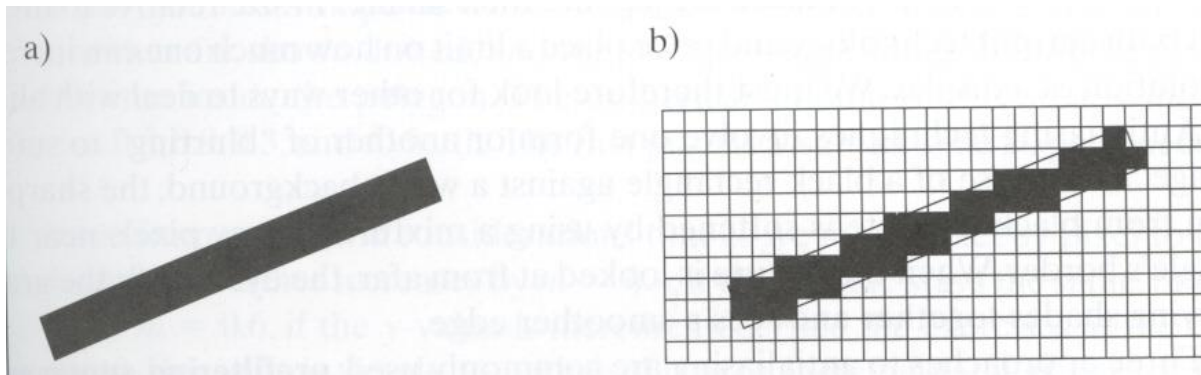
- Result after fix

Antialiasing

- Raster displays have pixels as rectangles
- Aliasing: Discrete nature of pixels introduces "jaggies"

Antialiasing

- 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, smoothen image
- Three antialiasing techniques:
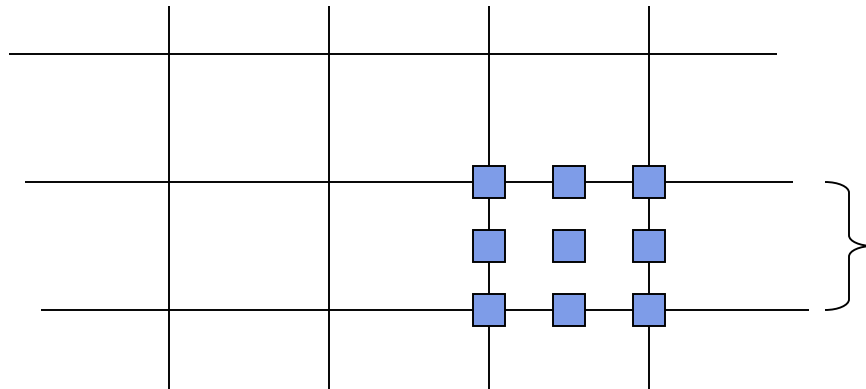 - Prefiltering
  - Postfiltering
  - Supersampling

Prefiltering

- Basic idea:
  - compute area of polygon coverage
  - use proportional intensity value
- Example: if polygon covers ¼ of the pixel
  - Pixel color = ¼ polygon color + ¾ adjacent region color
- Cons: computing polygon coverage can be time consuming

Supersampling

- Assumes we can compute color of any location (x,y) on screen
- Sample (x,y) in fractional (e.g. ½) increments, average samples
- Example: Double sampling = increments of ½ = 9 color values averaged for each pixel
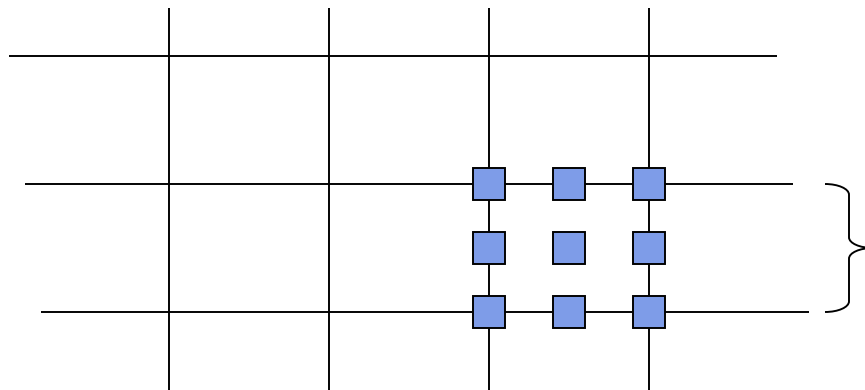
Average 9 (x, y) values to find pixel color

Postfiltering

- Supersampling weights all samples equally
- Post-filtering: use unequal weighting of samples
- Compute pixel value as weighted average
- Samples close to pixel center given more weight

Example weighting

1/16	1/16	1/16
1/16	1/2	1/16
1/16	1/16	1/16

Antialiasing in OpenGL

- Many alternative antialiasing implementation
- Simplest: accumulation buffer
- **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

Antialiasing in OpenGL

- 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(  )
    ```

Antialiasing in OpenGL

- Sample code
- Jitter.h stores slight random, displacements of camera,
- factor, f controls amount of overall sliding

```
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.h
-0.3348, 0.4353
0.2864, -0.3934
……
```

References

- Angel and Shreiner, Interactive Computer Graphics, 6th edition
- Hill and Kelley, Computer Graphics using OpenGL, 3rd edition, Chapter 9