CS 4731/543: Computer Graphics
Lecture 7 (Part I): Raster Graphics: Polygons & Antialiasing

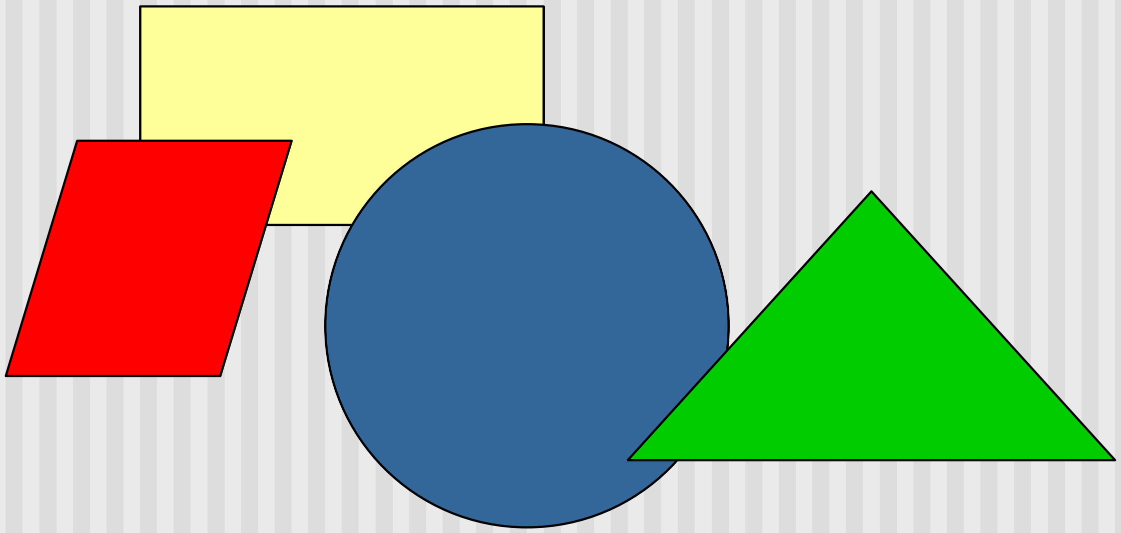
Emmanuel Agu

So Far…

- Raster graphics:
  - Line drawing algorithms (simple, Bresenham’s)
- Today:
  - 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



Defining and Filling Regions of Pixels

- 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 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:** pixels that lie next to each other horizontally, vertically OR diagonally
- **4-connected:** if there is unbroken path of 4-adjacent pixels connecting them
- **8-connected:** unbroken path of 8-adjacent pixels connecting them

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 (x, y) has color intColor, change it to newColor
  - Do same recursively for all 4 neighbors

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); // down pixel
        floodFill(x, y – 1, intColor); // fill up
   }
}
```

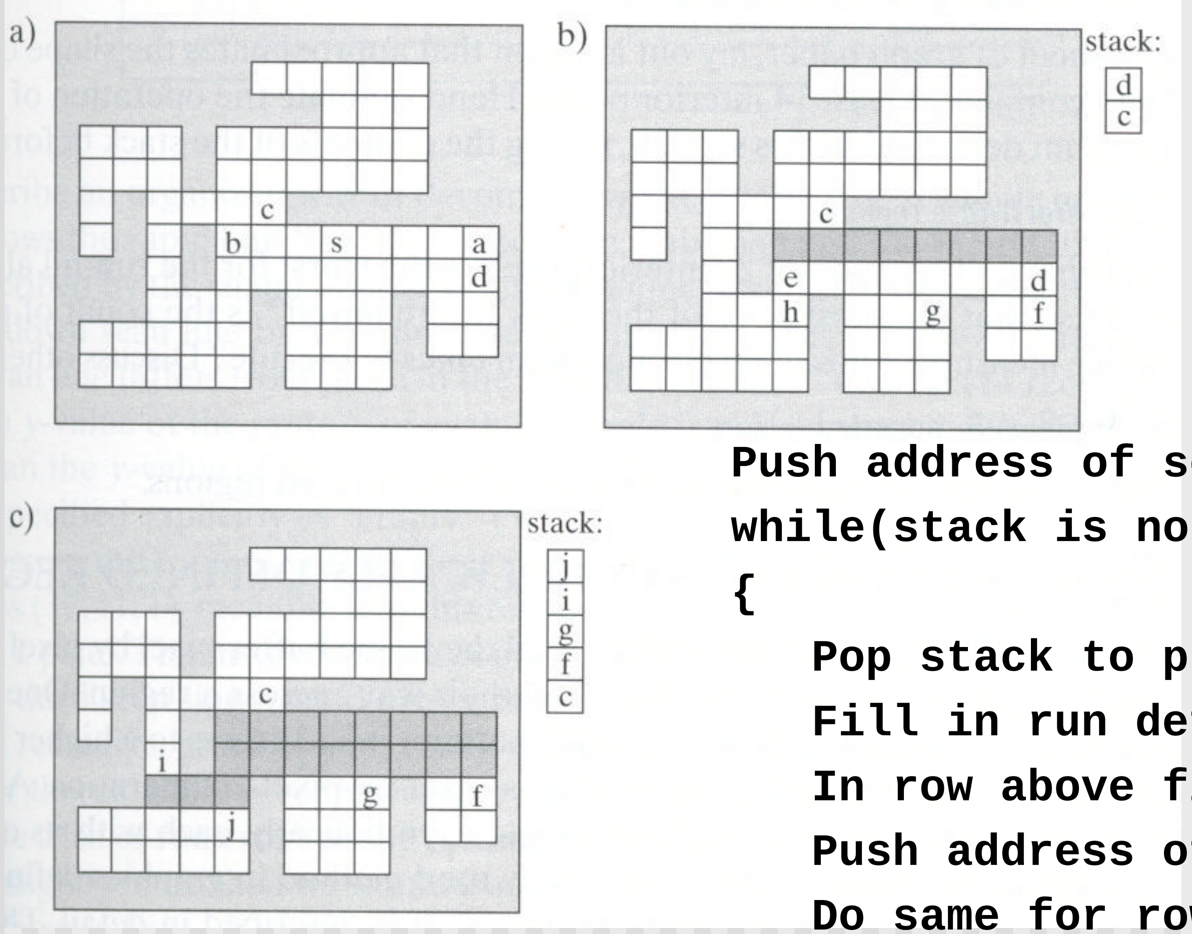
Recursive Flood-Fill Algorithm

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

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

Region Filling Using Coherence

- Example: start at s, initial seed

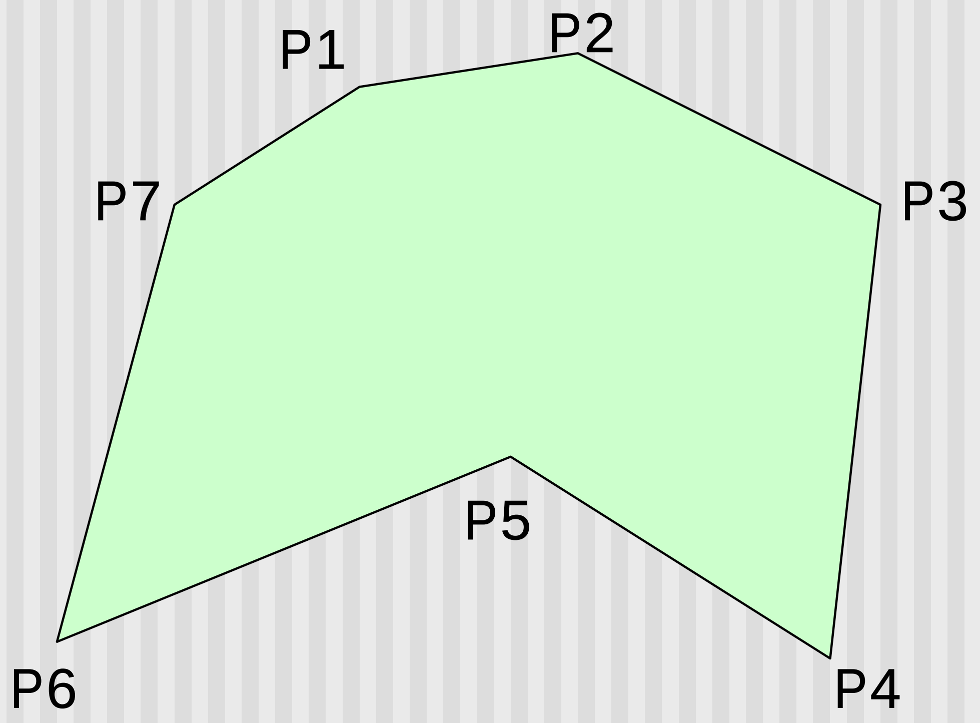


Pseudocode:

```
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
}
```

Filling Polygon-Defined Regions

- **Problem:** Region defined by Polygon P with vertices Pi = (Xi, Yi), for i – 1…N, specifying sequence of P's vertices



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

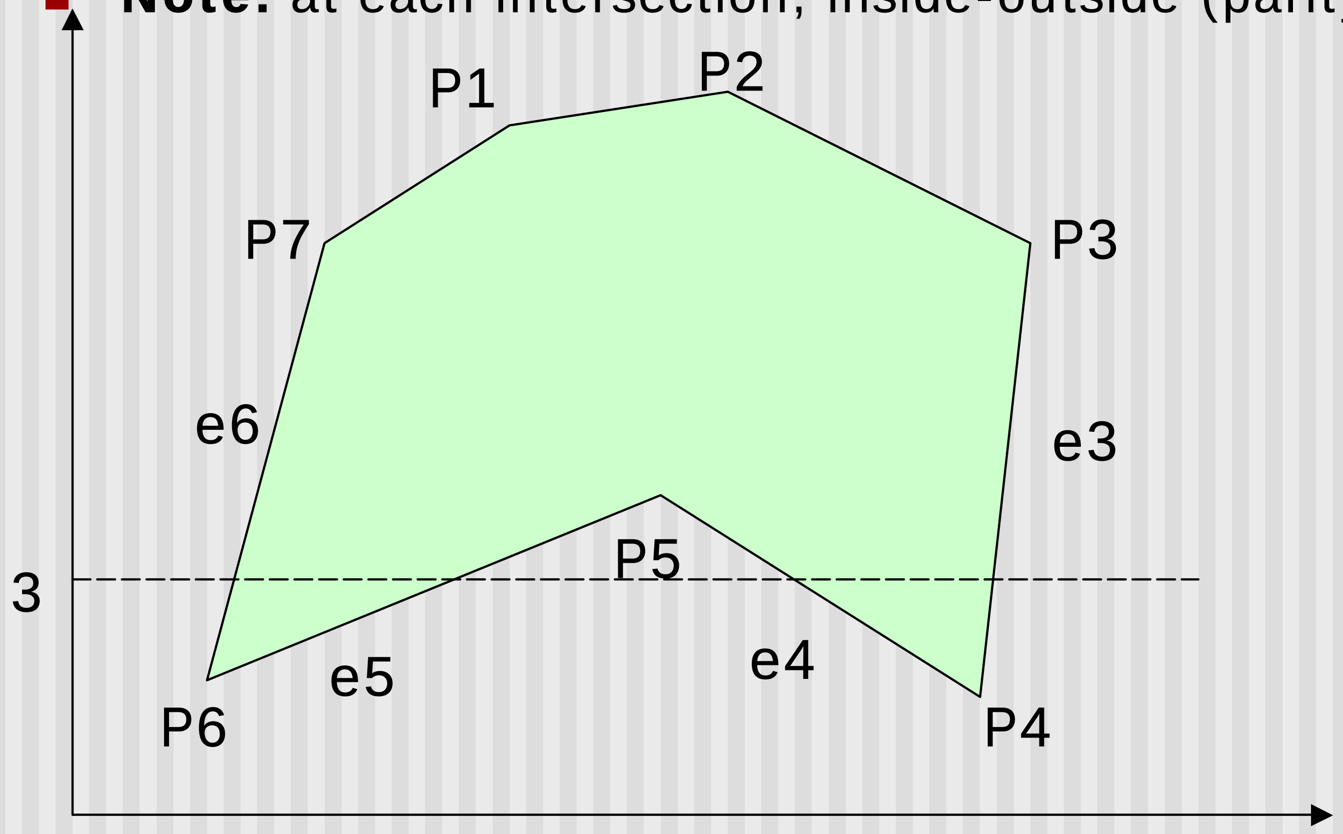
Filling Polygon-Defined Regions

- **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
}
```

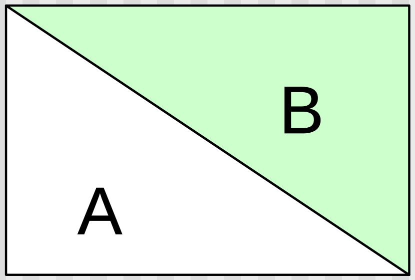
Filling Polygon-Defined Regions

- **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



Filling Polygon-Defined Regions

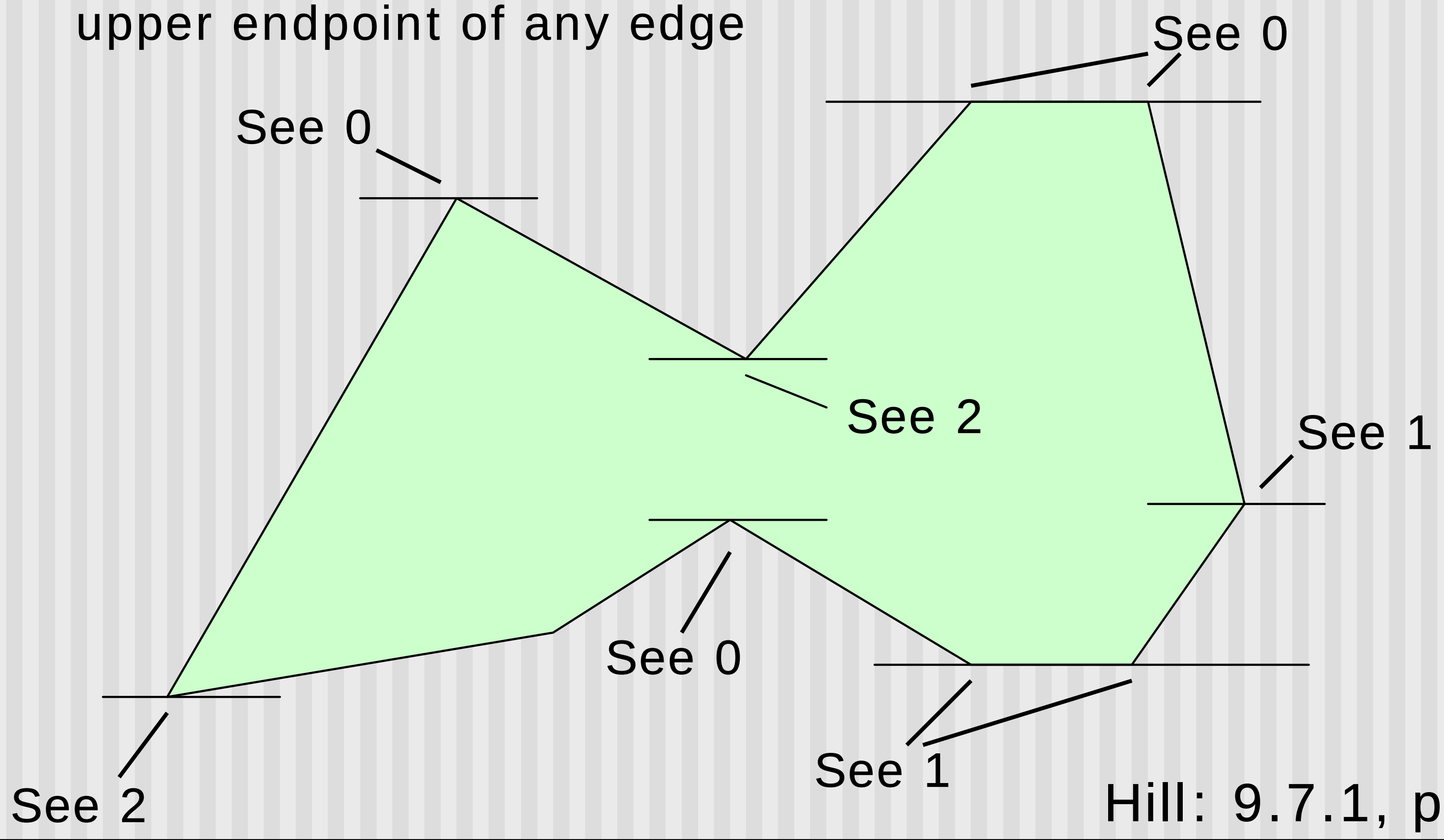
- What if two polygons A, B share an edge?
- Algorithm behavior could result in:
  - setting edge first in one color and the another
  - Drawing edge twice too bright
- **Make Rule:** when two polygons share edge, each polygon owns its left and bottom edges
- E.g. below draw shared edge with color of polygon **B**



Read: Hill: 9.7.1, pg 481

Filling Polygon-Defined Regions

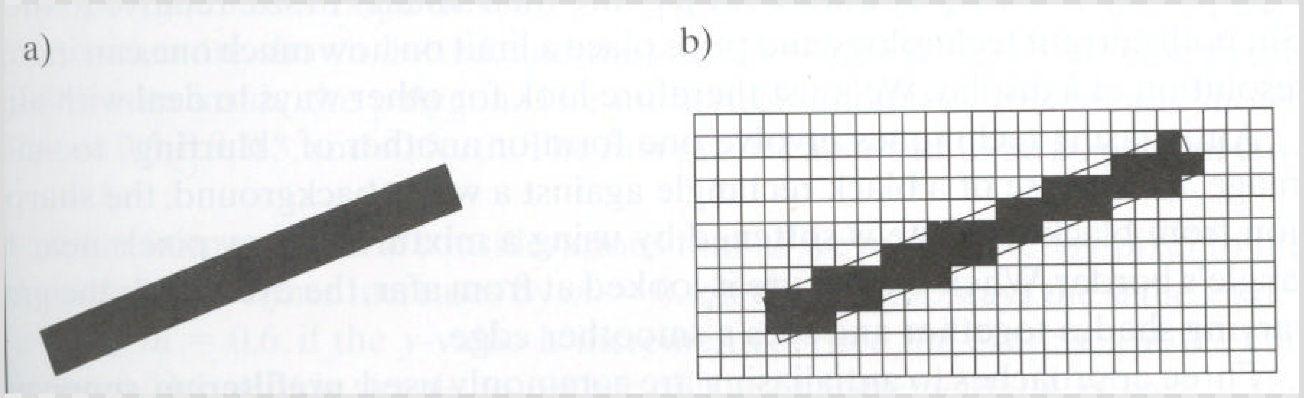
- How to handle cases where scan line intersects with polygon endpoints to avoid wrong parity?
- Solution: Discard intersections with horizontal edges and with upper endpoint of any edge



Hill: 9.7.1, pg. 482

Antialiasing

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



Antialiasing

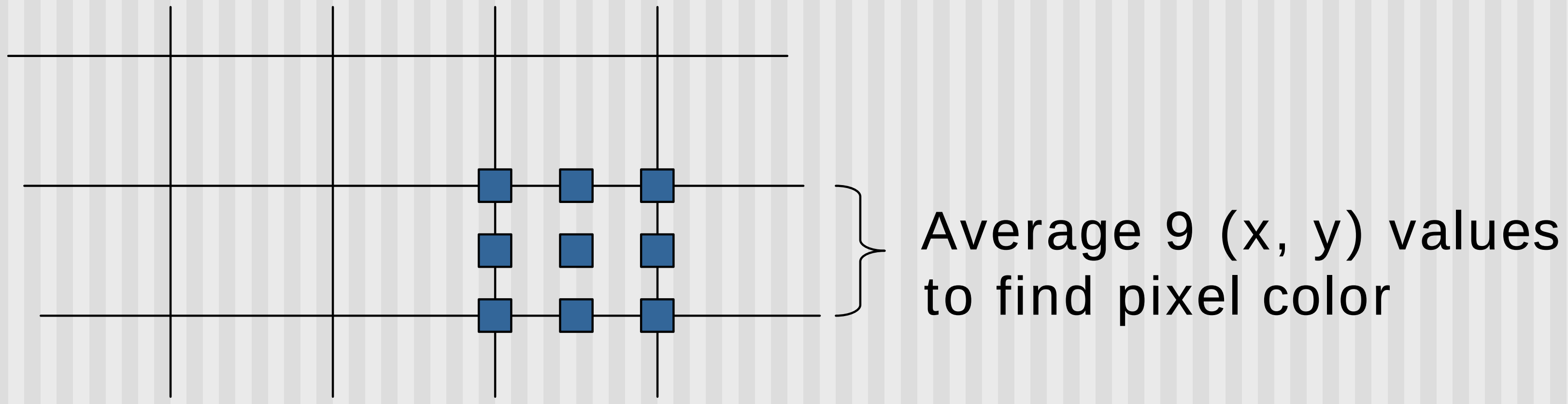
- Aliasing effects:
  - Distant objects may disappear entirely
  - Objects can blink on and off in animations
- Antialiasing techniques involve some form of blurring to reduce contrast, smoothen image
- Three antialiasing techniques:
  - Prefiltering
  - Postfiltering
  - Supersampling

Prefiltering

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

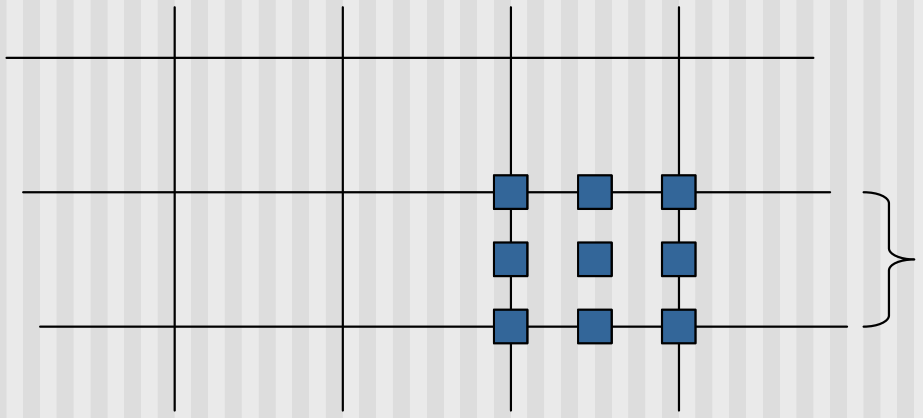
Supersampling

- 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. ½) increments
- Find average of samples
- Example: Double sampling = increments of ½ = 9 color values averaged for each pixel



Postfiltering

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



Sample weighting

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

Antialiasing in OpenGL

- Many alternatives
- 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[] stores randomized slight 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], 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

- Hill, chapter 9