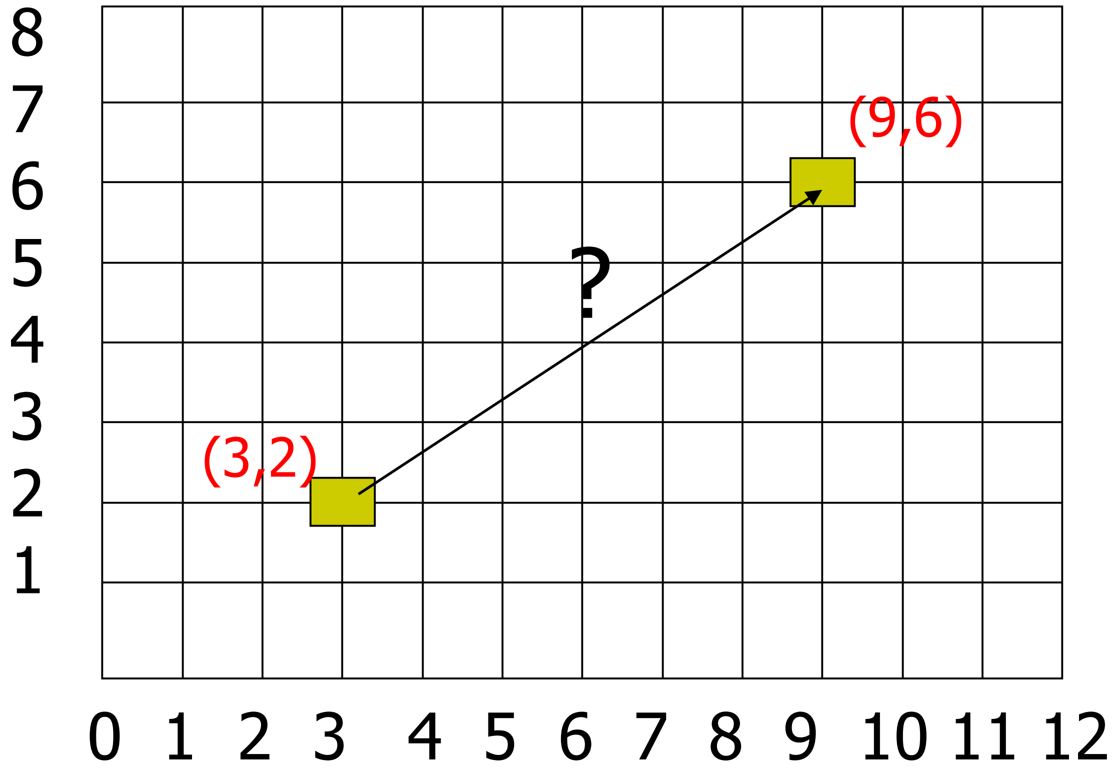
Recall: Line drawing algorithm

- Programmer specifies (x,y) of end pixels
- Need algorithm to determine pixels on line path

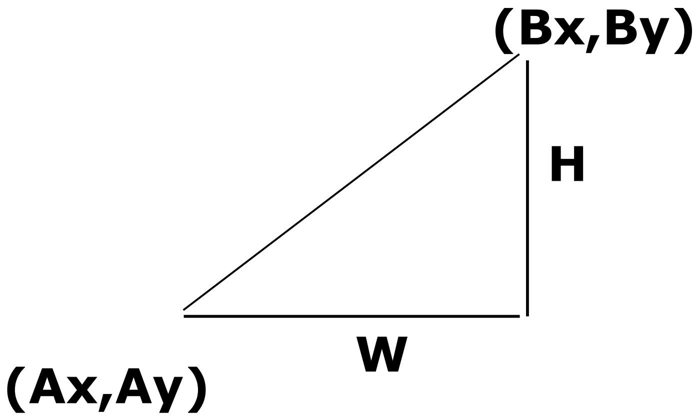


Line: (3,2) -> (9,6)

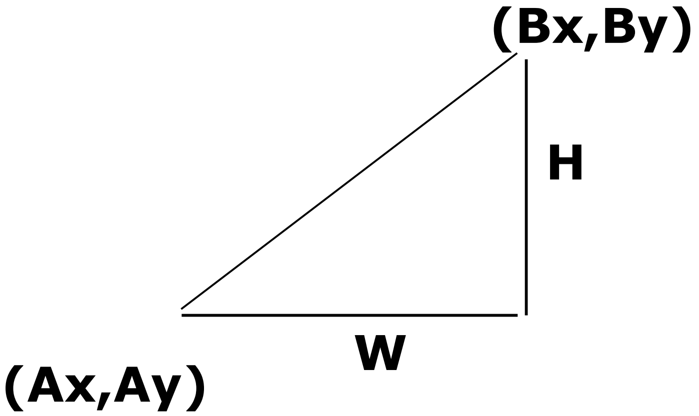
Which intermediate pixels to turn on?

Bresenham's Line-Drawing Algorithm

- **Problem:** Given endpoints (Ax, Ay) and (Bx, By) of line, determine intervening pixels
- First make two simplifying assumptions (remove later):
  - (Ax < Bx) and
  - (0 < m < 1)
- Define
  - Width W = Bx – Ax
  - Height H = By - Ay



Bresenham's Line-Drawing Algorithm



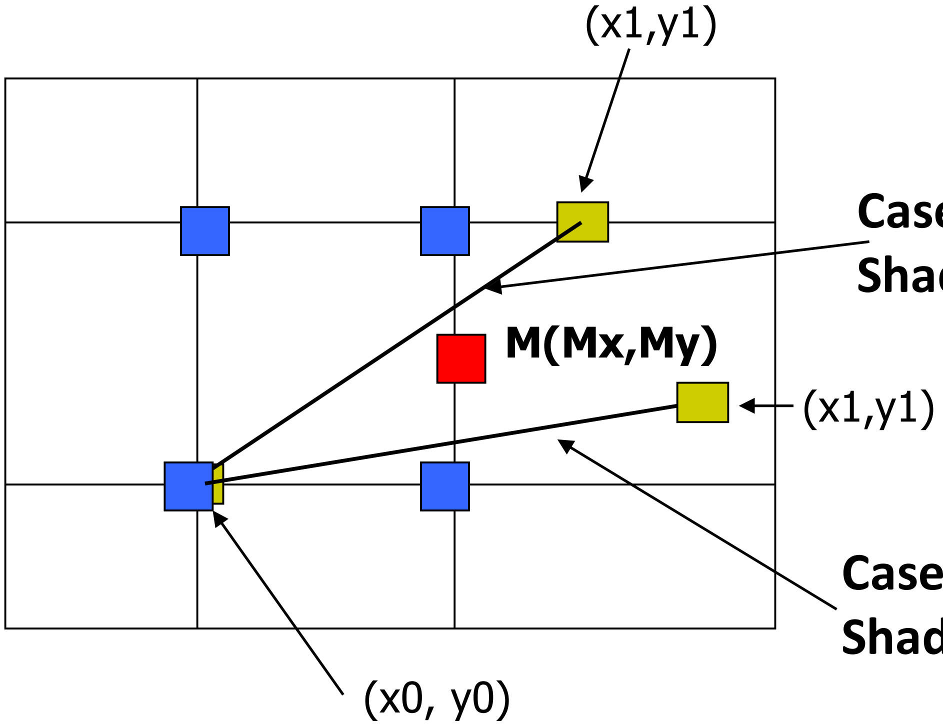
- Based on assumptions (Ax < Bx) and (0 < m < 1)
    - W, H are +ve
    - H < W
- Increment x by +1, y incr by +1 or stays same
- Midpoint algorithm determines which happens

Bresenham's Line-Drawing Algorithm

What Pixels to turn on or off?

Consider pixel midpoint M(Mx, My) = (x + 1, y + ½)

Build equation of actual line, compare to midpoint

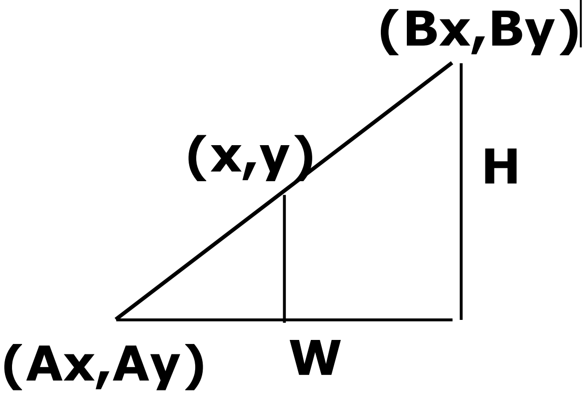


Case a: If midpoint (red dot) is below line,
Shade upper pixel, (x + 1, y + 1)

Case b: If midpoint (red dot) is above line,
Shade lower pixel, (x + 1, y)

Build Equation of the Line

- Using similar triangles:



$$\frac{y - Ay}{x - Ax} = \frac{H}{W}$$

$$H(x - Ax) = W(y - Ay)$$

$$-W(y - Ay) + H(x - Ax) = 0$$

- Above is equation of line from (Ax, Ay) to (Bx, By)
- Thus, any point (x,y) that lies on ideal line makes eqn = 0
- Double expression (to avoid floats later), and call it F(x,y)

$$F(x,y) = -2W(y - Ay) + 2H(x - Ax)$$

Bresenham’s Line-Drawing Algorithm

- So, $F(x,y) = -2W(y - Ay) + 2H(x - Ax)$

- Algorithm, If:
  - $F(x, y) < 0$, (x, y) above line
  - $F(x, y) > 0$, (x, y) below line

- **Hint:** $F(x, y) = 0$ is on line
- Increase y keeping x constant, $F(x, y)$ becomes more negative

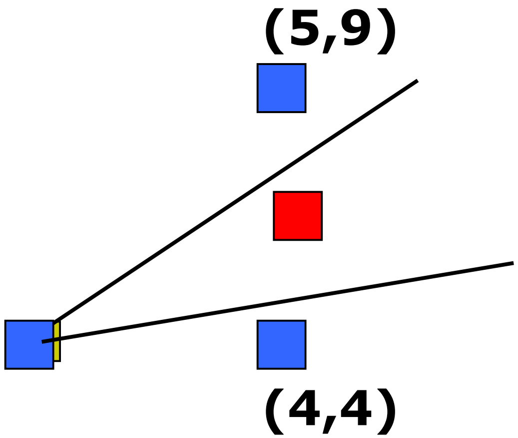
Bresenham's Line-Drawing Algorithm

- **Example:** to find line segment between (3, 7) and (9, 11)

$$F(x,y) = -2W(y - Ay) + 2H(x - Ax)$$

$$= (-12)(y - 7) + (8)(x - 3)$$

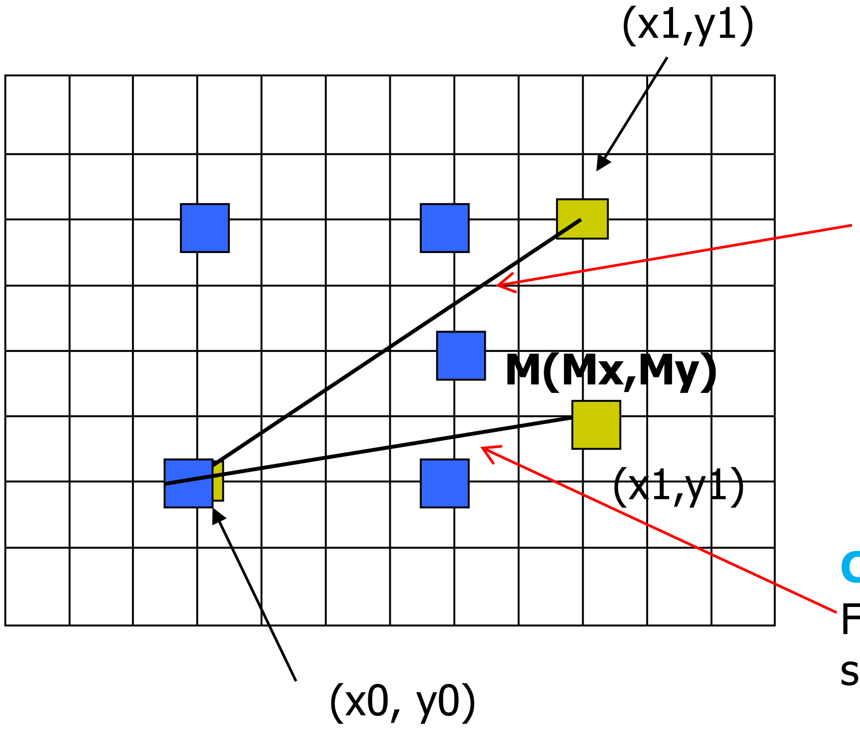
- For points on line. E.g. (7, 29/3), F(x, y) = 0
- A = (4, 4) lies below line since F = 44
- B = (5, 9) lies above line since F = -8



Bresenham’s Line-Drawing Algorithm

What Pixels to turn on or off?

Consider pixel midpoint M(Mx, My) = (x0 + 1, Y0 + ½)



Case a: If M below actual line
F(Mx, My) < 0
shade upper pixel (x + 1, y + 1)

Case b: If M above actual line
F(Mx,My) > 0
shade lower pixel (x + 1, y)

Can compute F(x,y) incrementally

Initially, midpoint M = (Ax + 1, Ay + ½)

F(Mx, My) = -2W(y – Ay) + 2H(x – Ax)

i.e. F(Ax + 1, Ay + ½) = 2H – W

Can compute F(x,y) for next midpoint incrementally

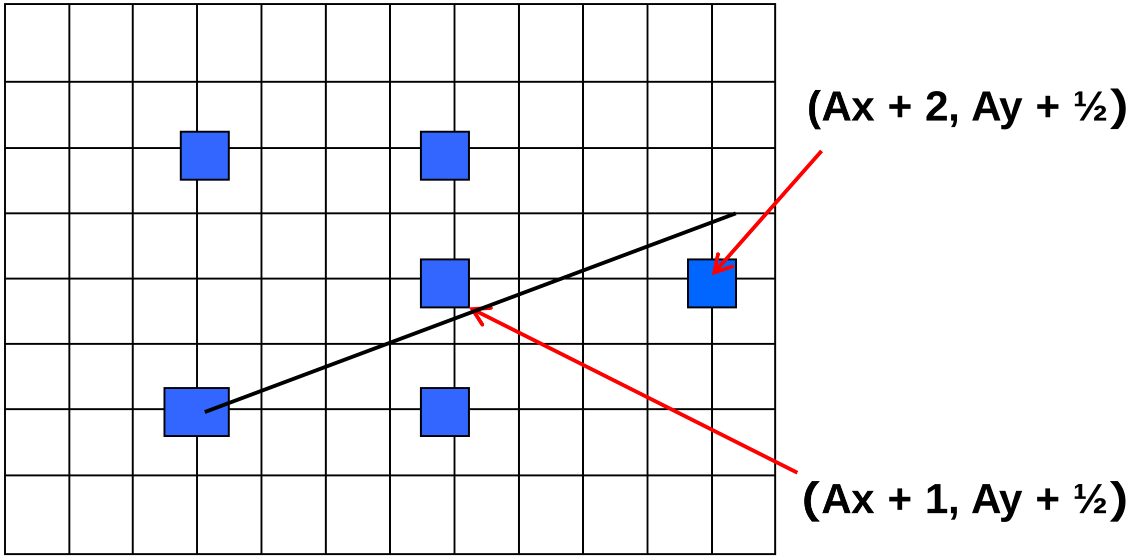
If we increment to (x + 1, y), compute new F(Mx,My)

F(Mx, My) += 2H

i.e. F(Ax + 2, Ay + ½)

- F(Ax + 1, Ay + ½)

= 2H

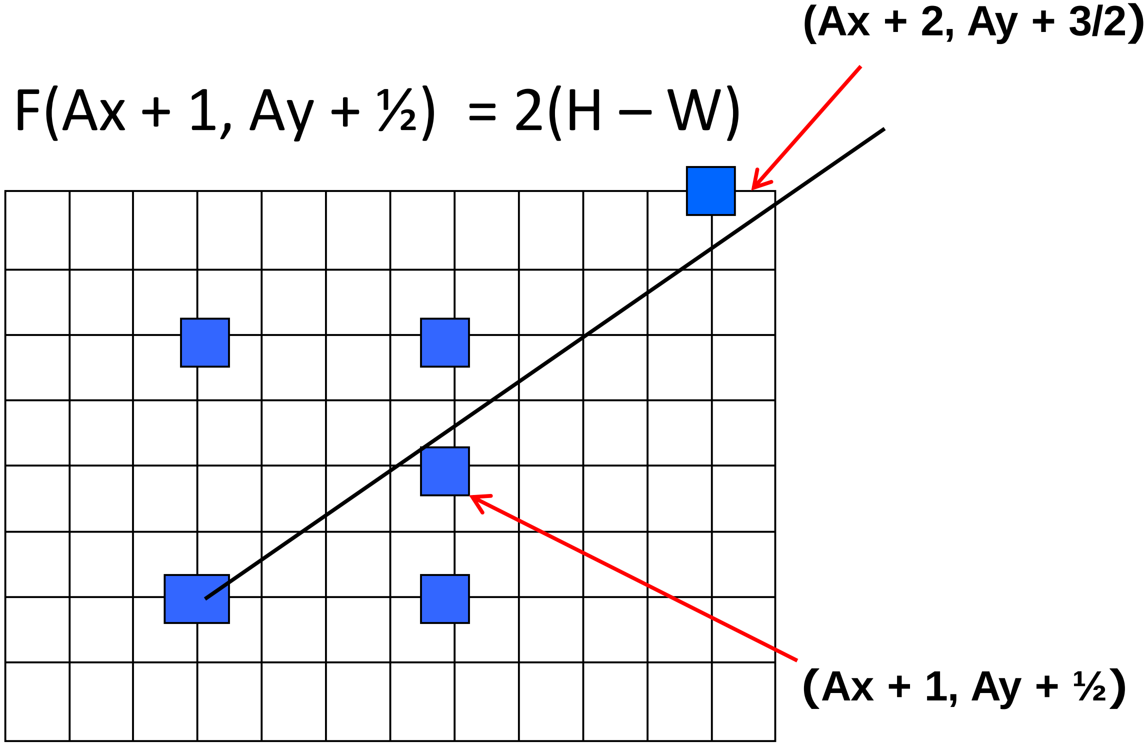


Can compute F(x,y) incrementally

If we increment to (x +1, y + 1)

F(Mx, My) += 2(H – W)

i.e. F(Ax + 2, Ay + 3/2) - F(Ax + 1, Ay + ½) = 2(H – W)



Bresenham's Line-Drawing Algorithm

```
Bresenham(IntPoint a, InPoint b)
{ // restriction: a.x < b.x and 0 < H/W < 1
    int y = a.y, W = b.x – a.x, H = b.y – a.y;
    int F = 2 * H – W;   // current error term
    for(int x = a.x;   x <= b.x;   x++)
    {
     setpixel at (x, y);  // to desired color value
        if F < 0           // y stays same
           F = F + 2H;
        else{
           Y++,  F = F  + 2(H – W)     // increment y
        }
    }
}
```

- Recall: F is equation of line

Bresenham's Line-Drawing Algorithm

- Final words: we developed algorithm with restrictions $0 < m < 1$ and $Ax < Bx$

- Can add code to remove restrictions
  - When $Ax > Bx$ (swap and draw)
  - Lines having $m > 1$ (interchange x with y)
  - Lines with $m < 0$ (step x++, decrement y not incr)
  - Horizontal and vertical lines (pretest a.x = b.x and skip tests)

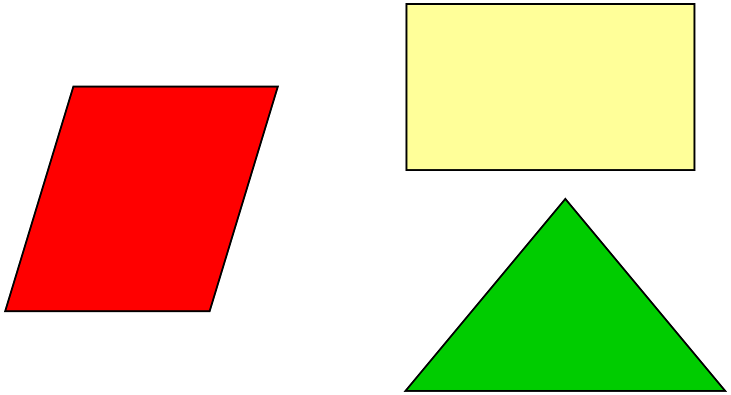
Computer Graphics
CS 4731 Lecture 23
Polygon Filling & Antialiasing

Prof Emmanuel Agu

Computer Science Dept.
Worcester Polytechnic Institute (WPI)

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

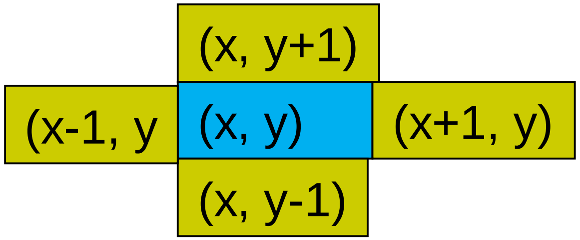


Pixel-Defined Regions

- **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

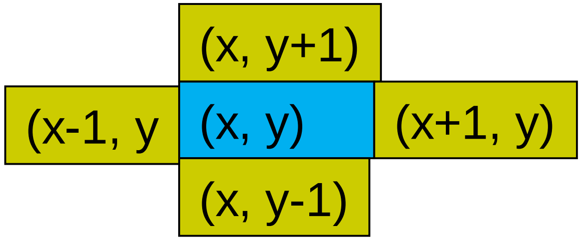
- Recursive algorithm
- Starts from initial pixel of color, `intColor`
- Recursively set 4-connected neighbors to `newColor`
- **Flood-Fill**: floods region with `newColor`
- **Basic idea:**
  - start at “seed” pixel (x, y)
  - If (x, y) has color **intColor**, change it to **newColor**
  - Do same recursively 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); // up pixel
   }
}
```

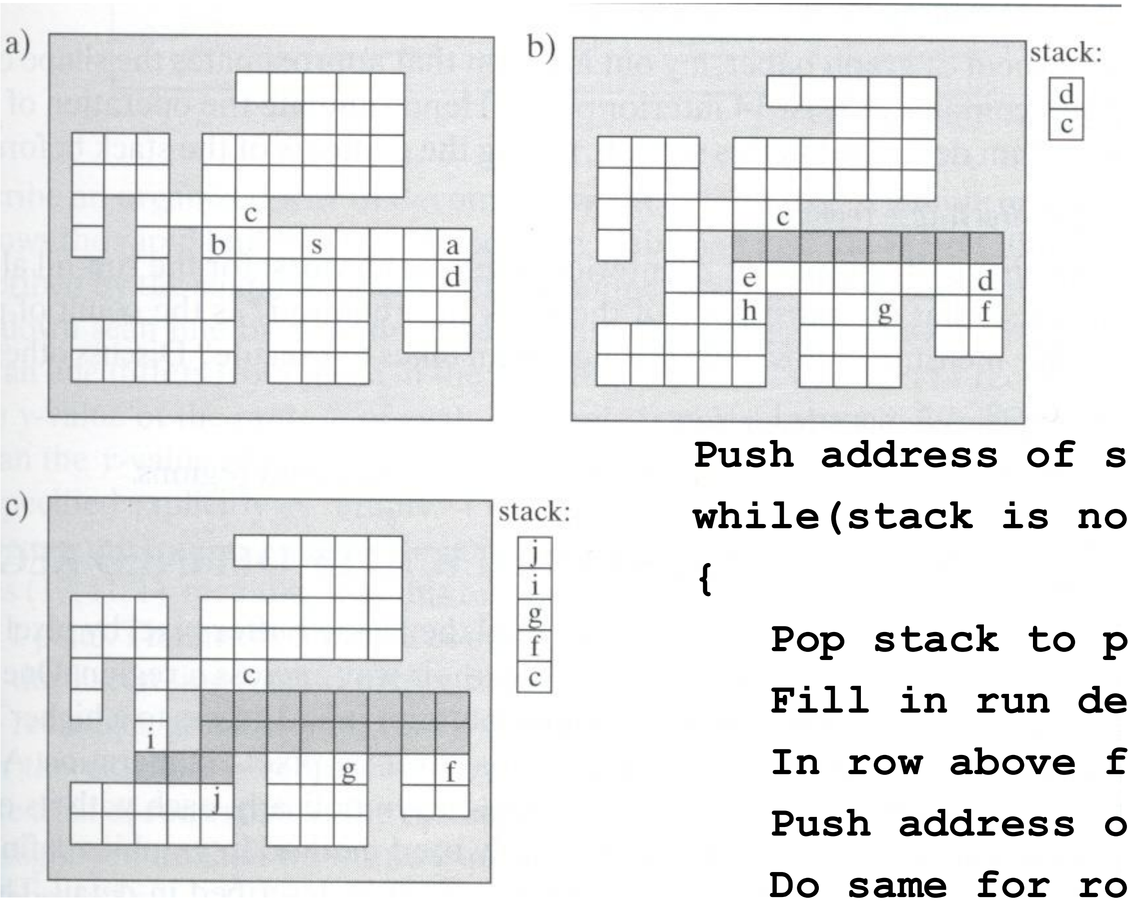


Recursive Flood-Fill Algorithm

- Recursive flood-fill is blind
- Some pixels retested several times
- **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:** 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:

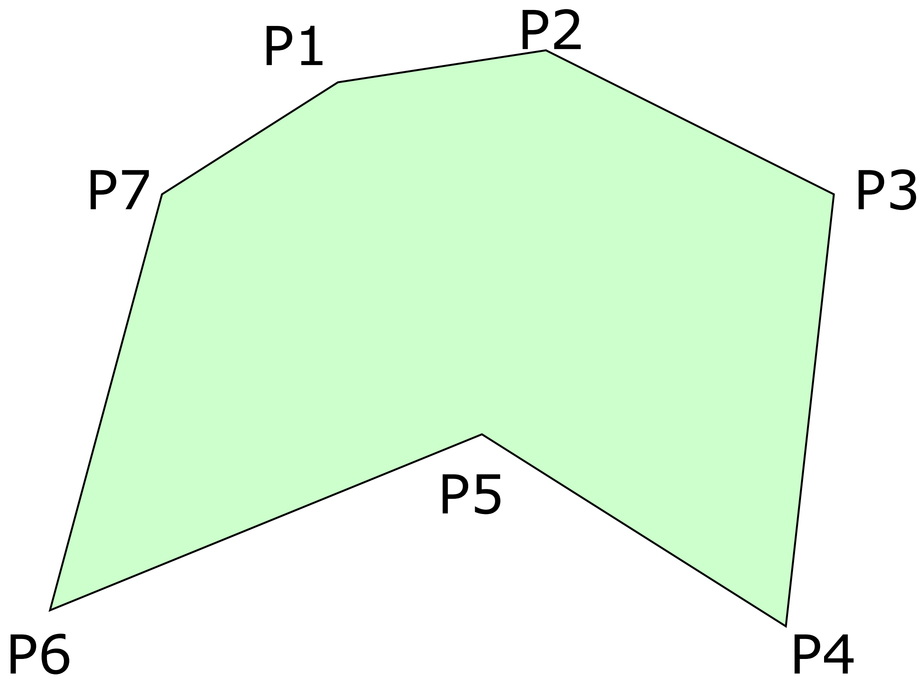
```
Push address of seed pixel onto stack
while(stack is not empty)
{
   Pop stack to provide next seed
   Fill in run defined by seed
   In row above find reachable interior runs
   Push address of their rightmost pixels
   Do same for row below current run
}
```

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



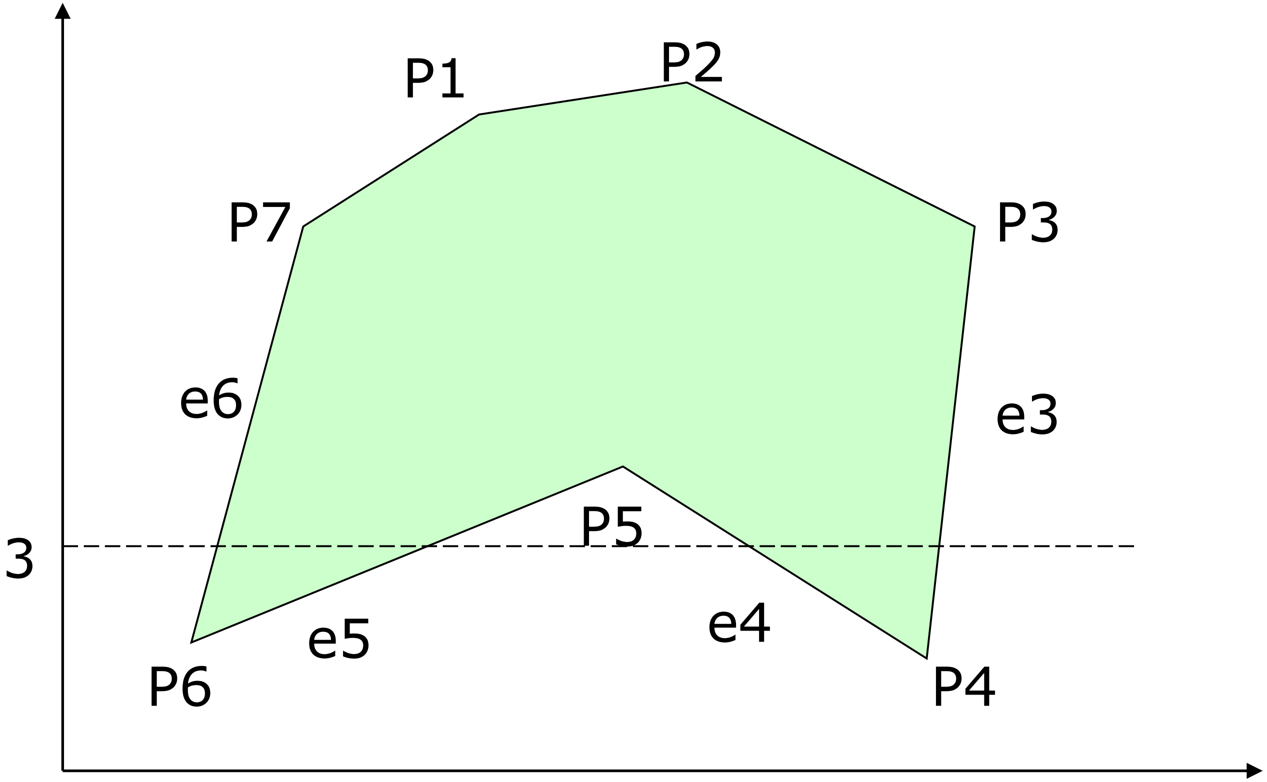
Filling Polygon-Defined Regions

- **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:**

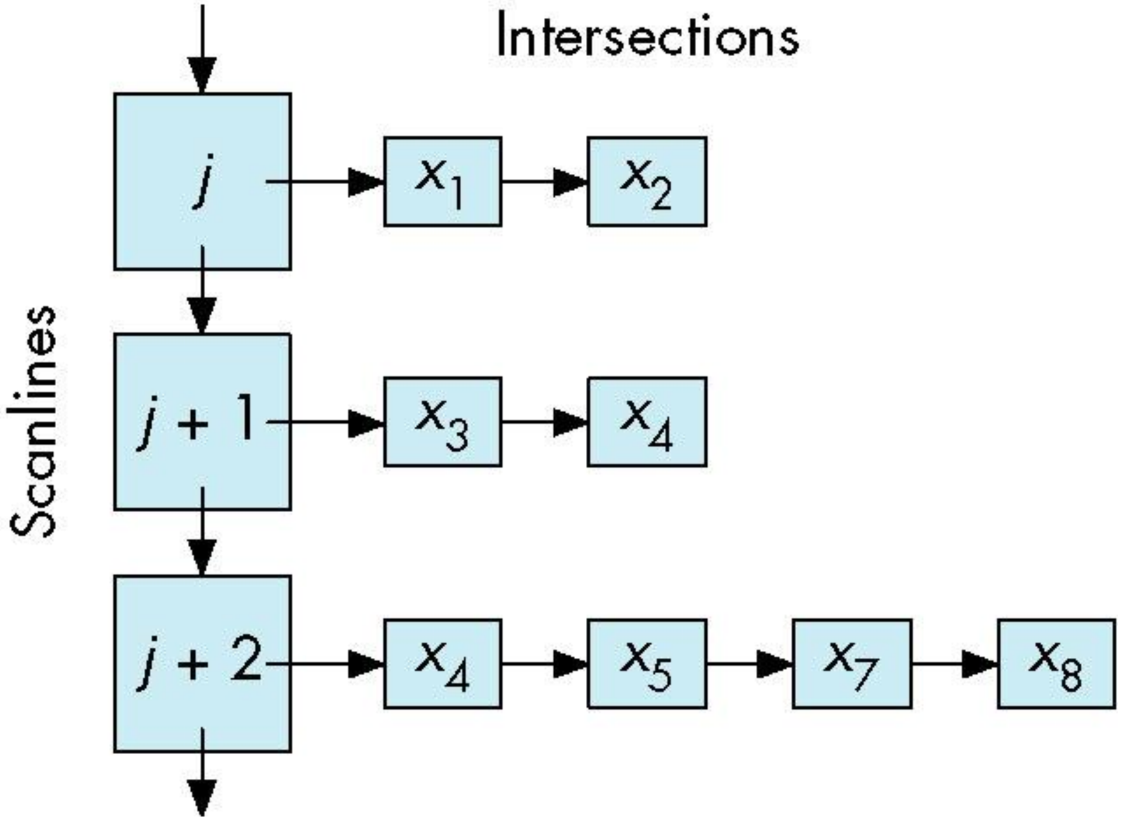
```
for(each scan Line L)
{
   Find intersections of L with all edges of P
   Sort the intersections by increasing x-value
   Fill pixel runs between all pairs of intersections
}
```

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

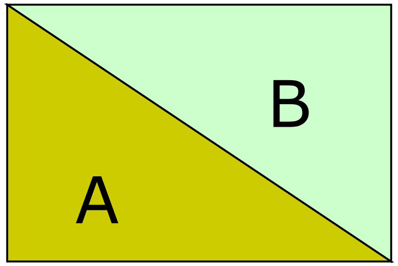


Data Structure



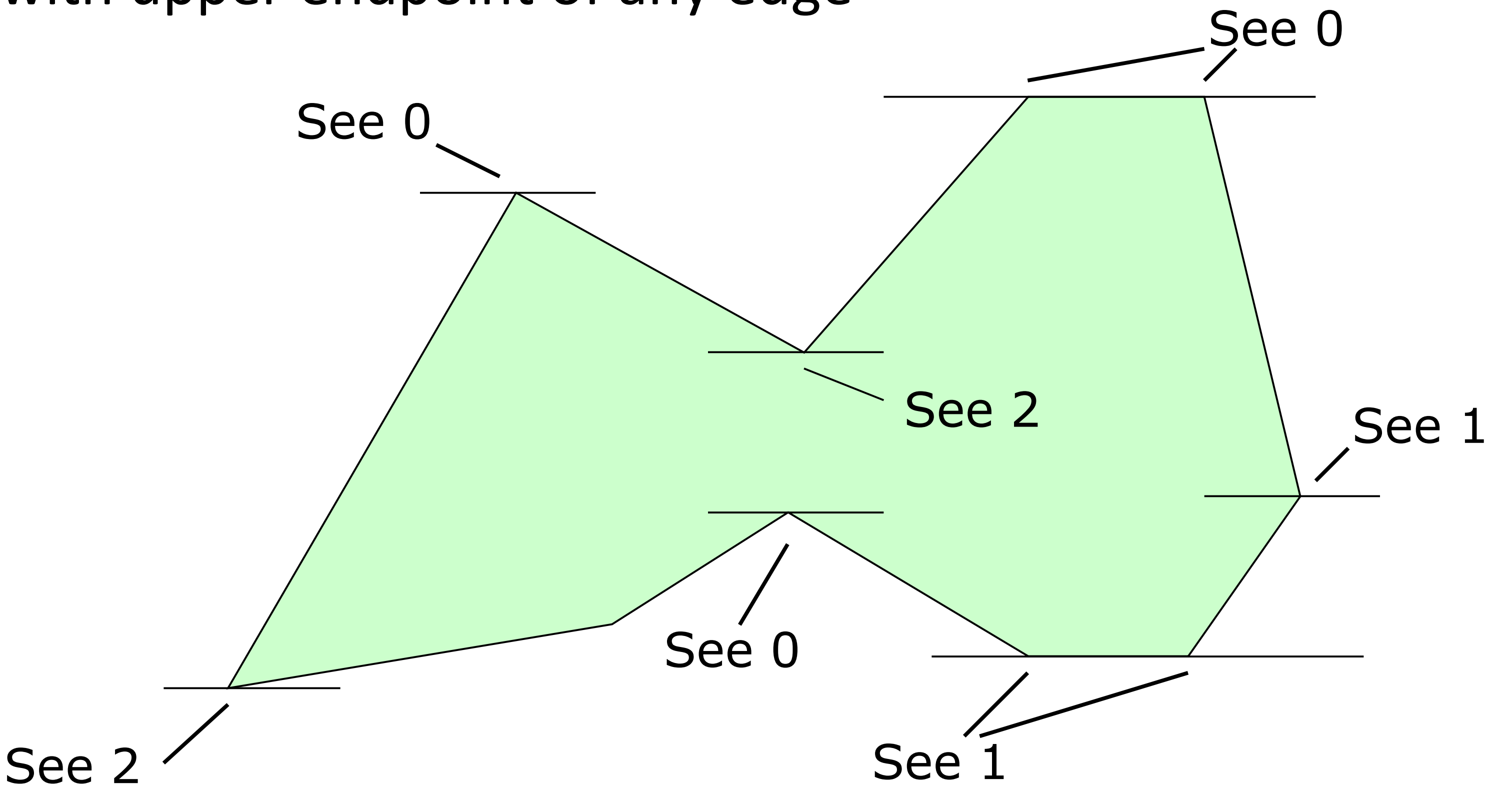
Filling Polygon-Defined Regions

- **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**



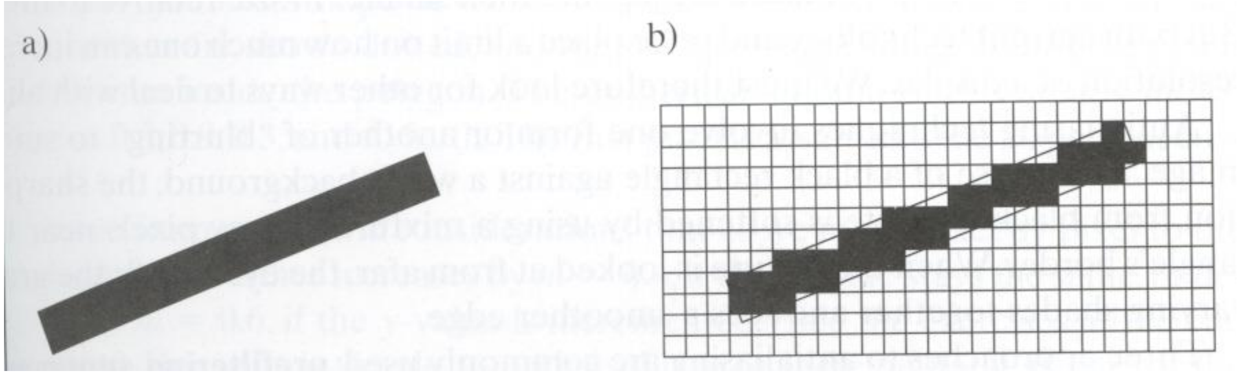
Filling Polygon-Defined Regions

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

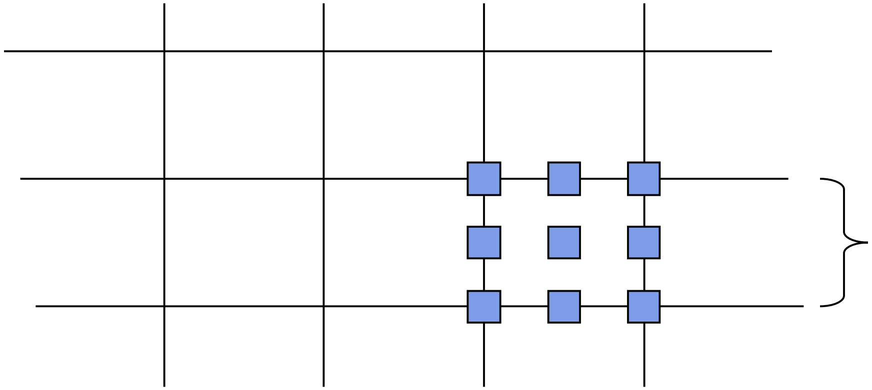
- Aliasing effects:
  - Distant objects may disappear entirely
  - Objects can blink on and off in animations
- Antialiasing techniques involve some form of blurring to reduce contrast, smoothen image
- Three antialiasing techniques:
  - Prefiltering
  - Postfiltering
  - Supersampling

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

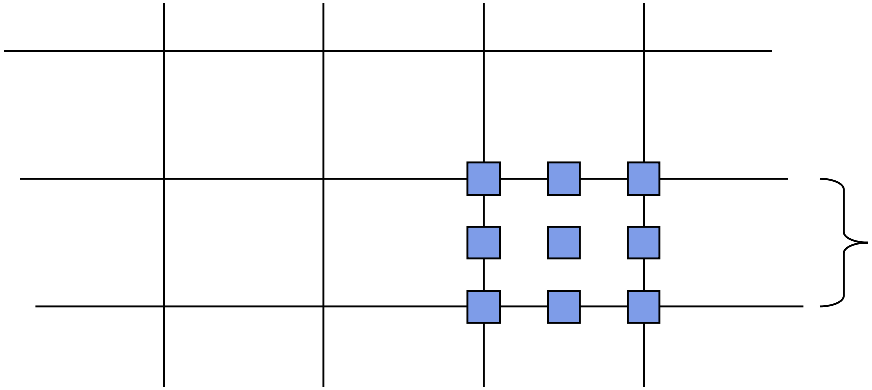
- Assumes we can compute color of any location (x,y) on screen
- Sample (x,y) in fractional (e.g. ½) increments, average samples
- Example: Double sampling = increments of ½ = 9 color values averaged for each pixel



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



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].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