CS-184: Computer Graphics

Lecture #10: Scan Conversion

Prof. James O'Brien
University of California, Berkeley

V2014-F-10-1.0

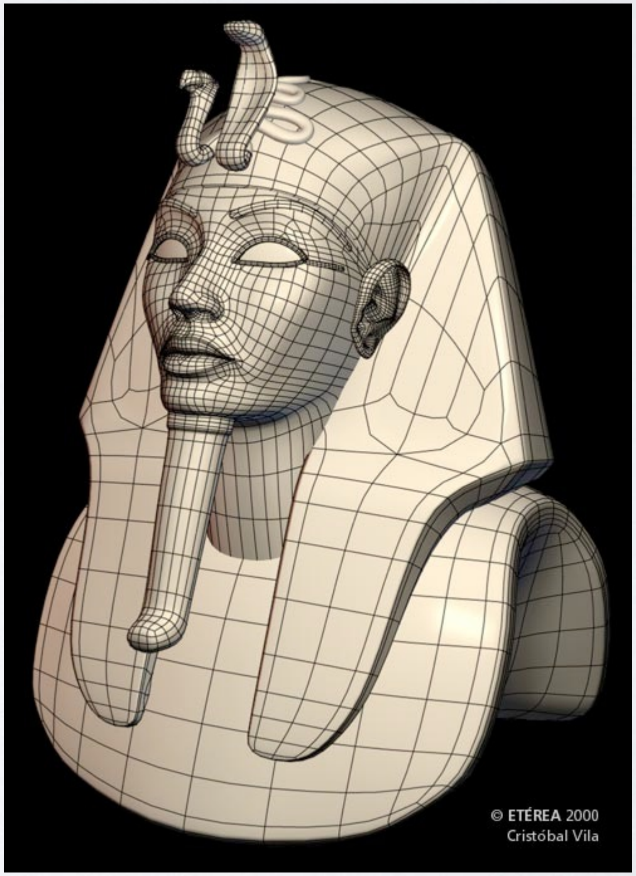
With additional slides based on those of Maneesh Agrawala

Today

- 2D Scan Conversion
  - Drawing Lines
  - Drawing Curves
  - Filled Polygons
  - Filling Algorithms

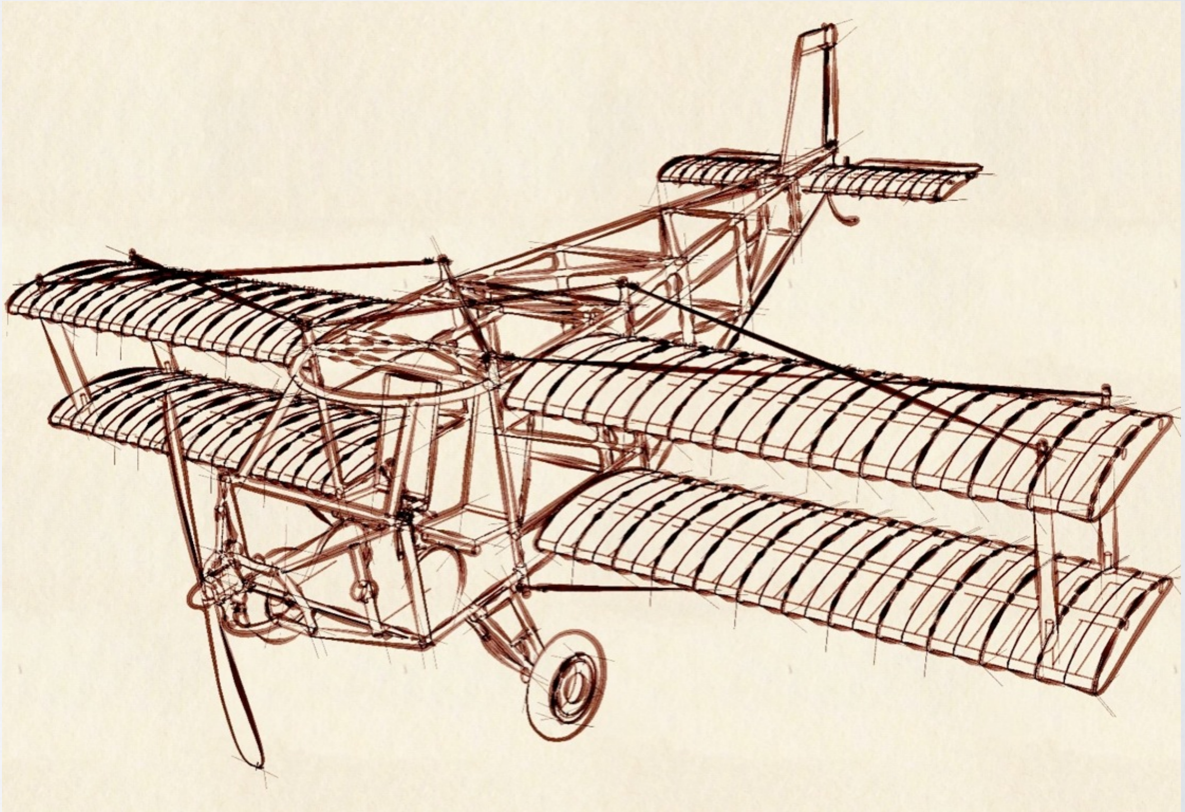
Drawing a Line

- Basically, its easy... but for the details
- Lines are a basic primitive that needs to be done well...



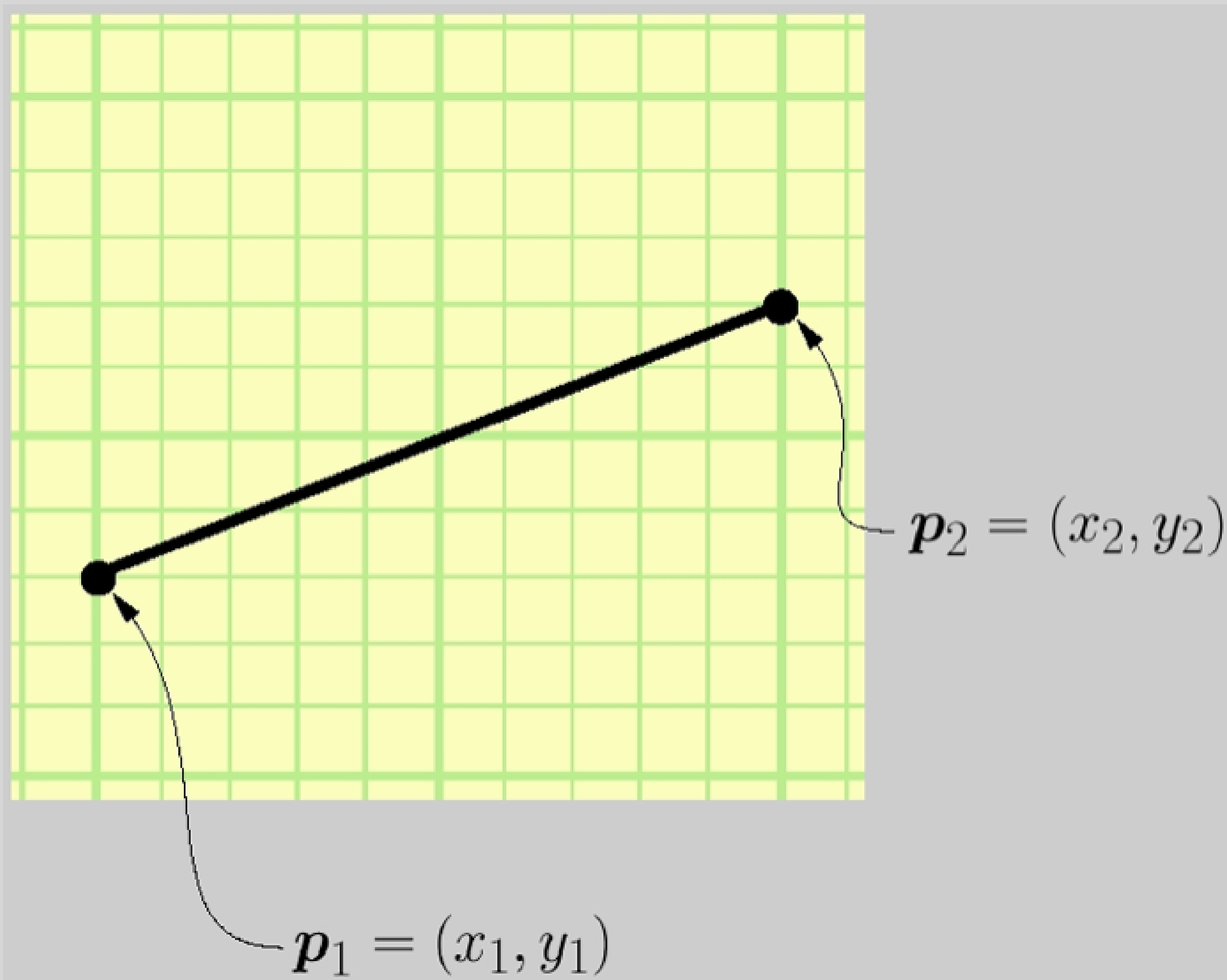
Drawing a Line

- Basically, its easy... but for the details
- Lines are a basic primitive that needs to be done well...

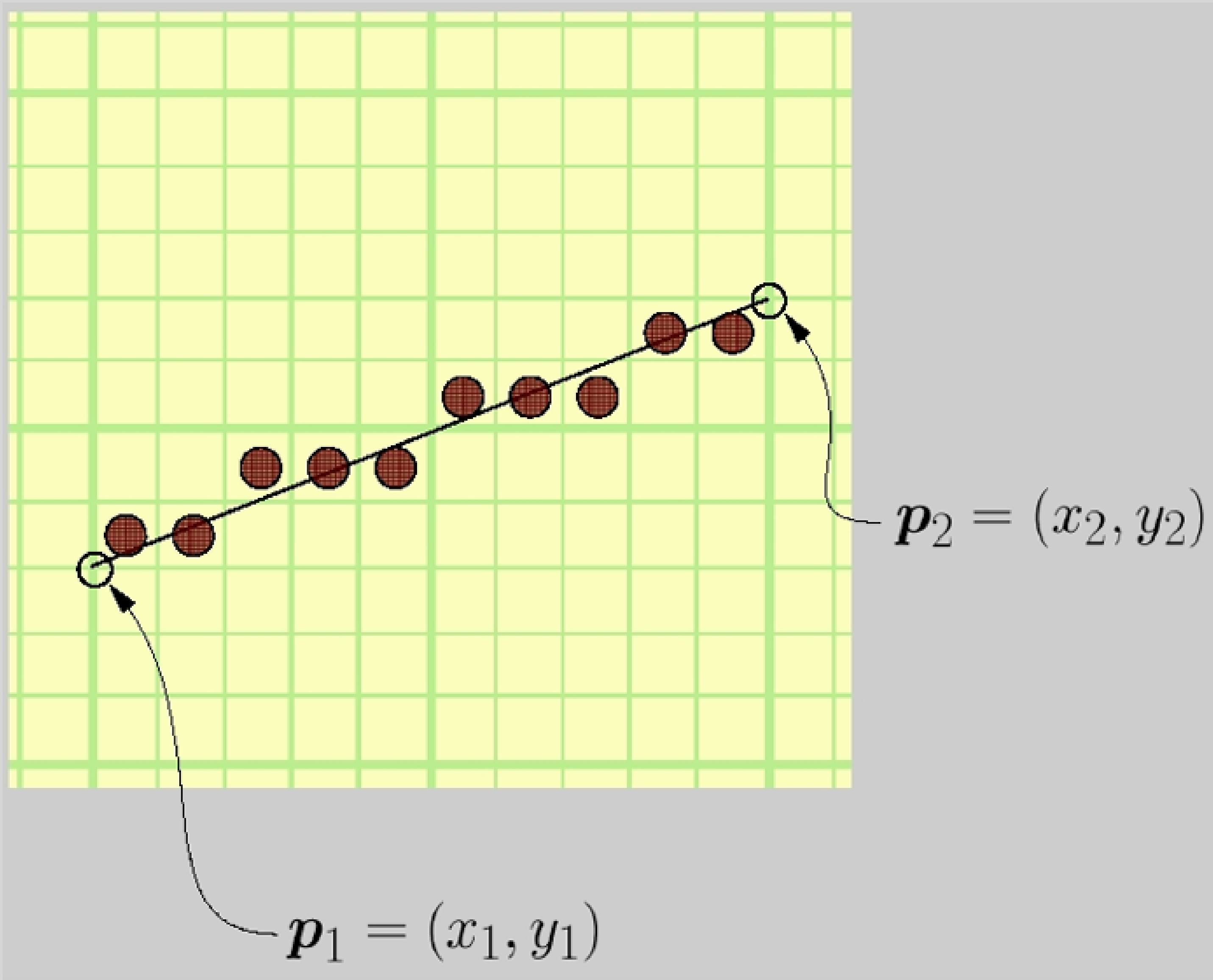


From "A Procedural Approach to Style for NPR Line Drawing from 3D models," by Grabli, Durand, Turquin, Sillion

Drawing a Line



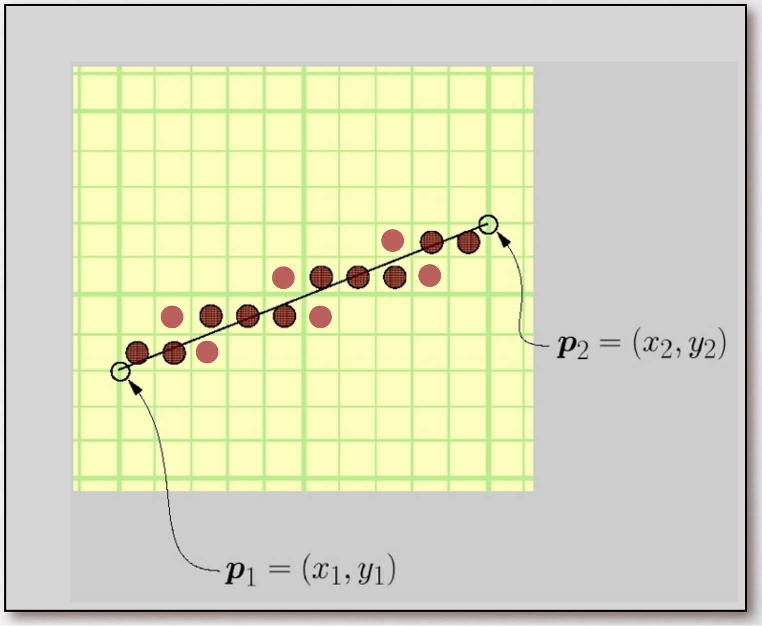
Drawing a Line



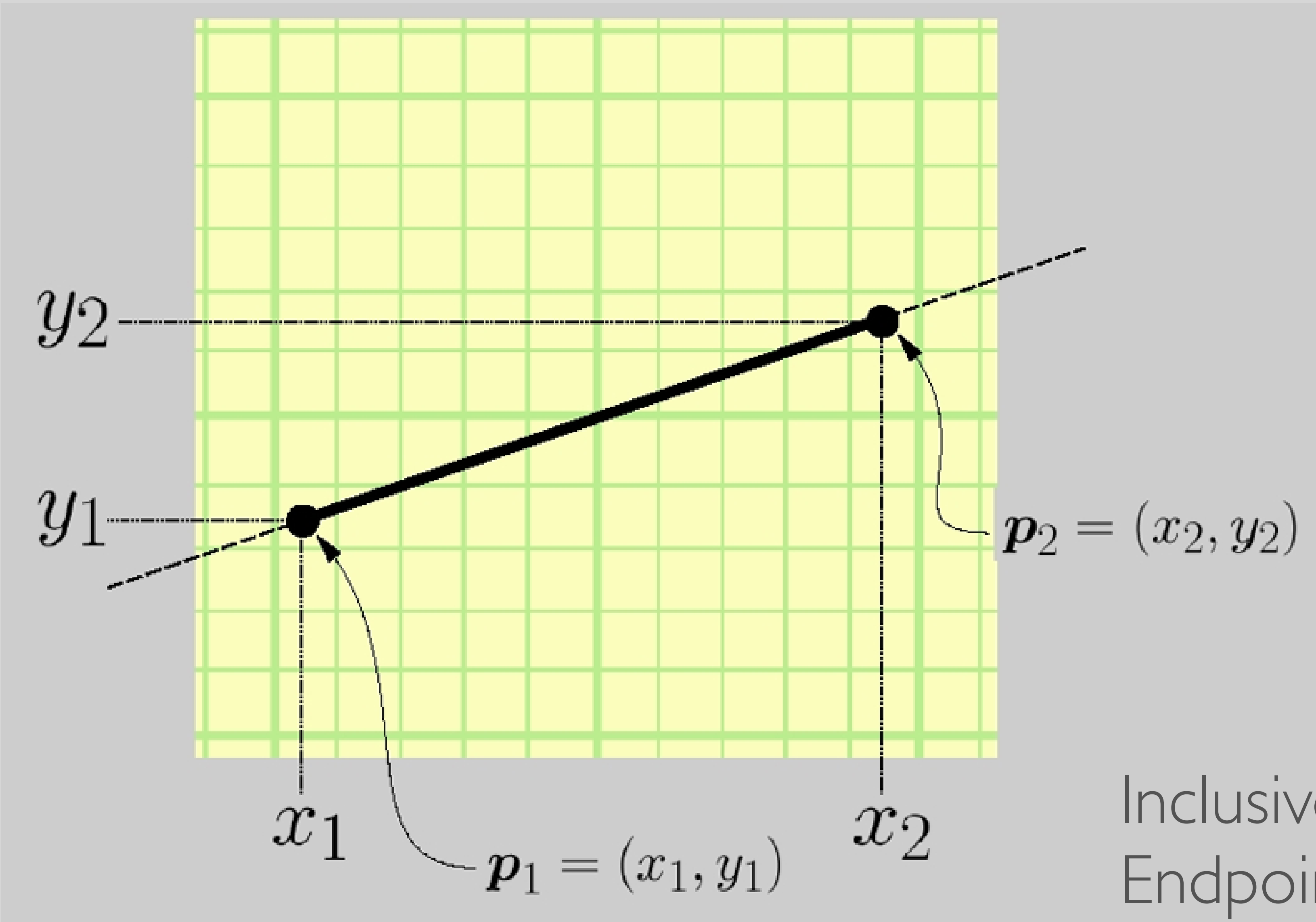
Drawing a Line

- Some things to consider
  - How thick are lines?
  - How should they join up?
  - Which pixels are the right ones?

For example:



Drawing a Line



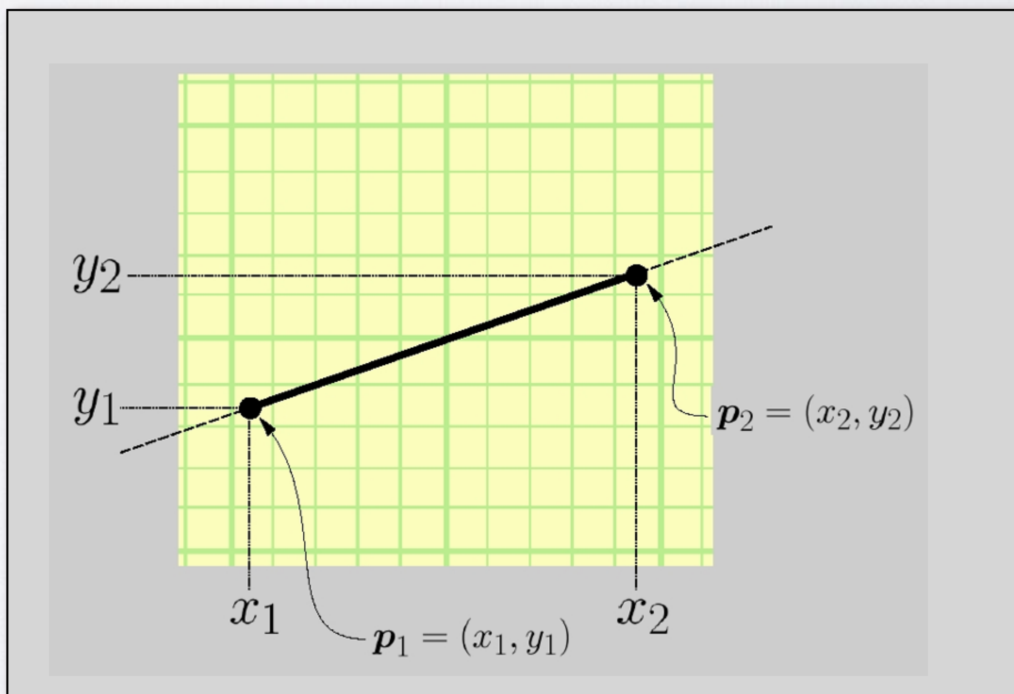
Inclusive
Endpoints

Drawing a Line

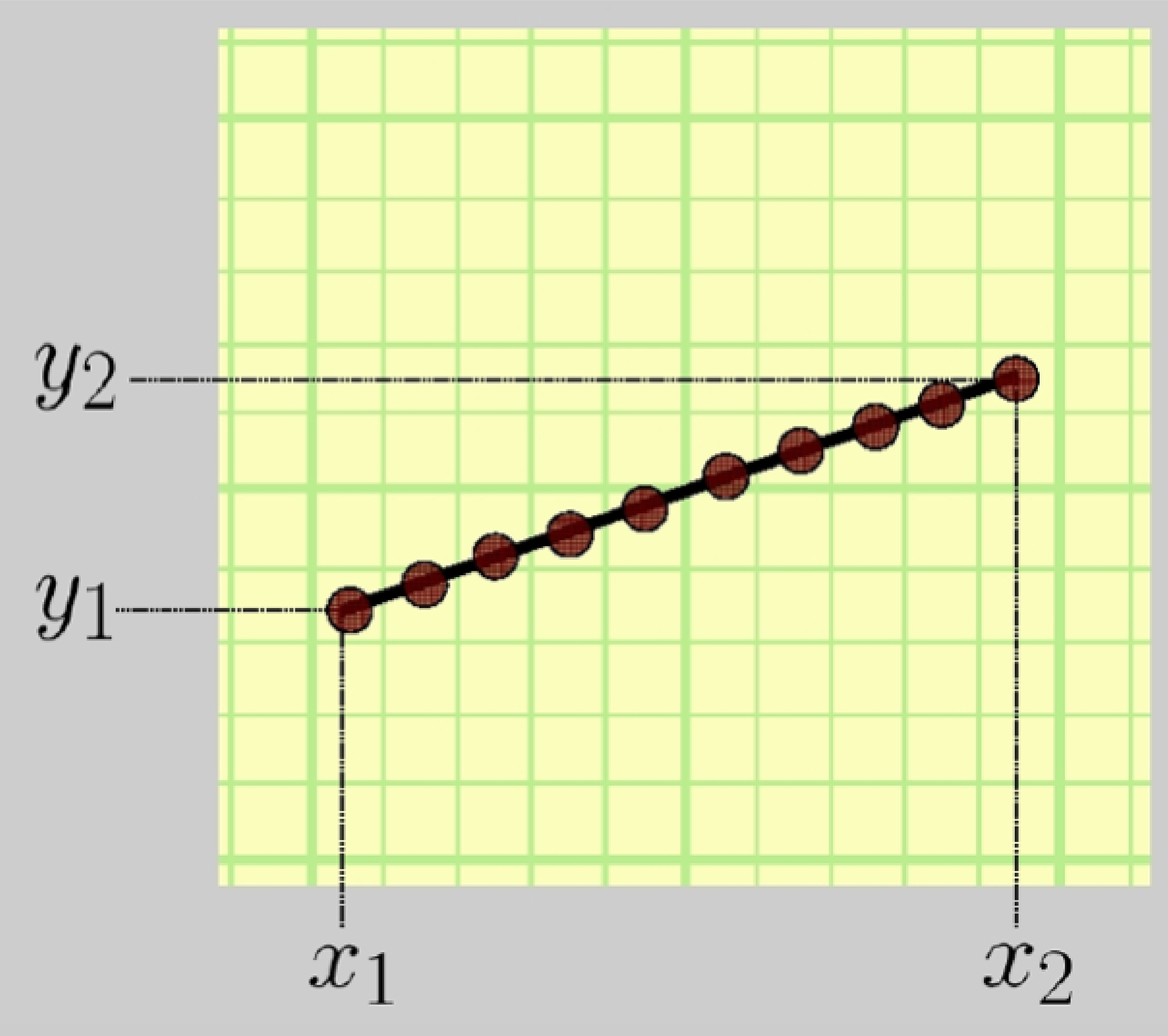
$$y = m \cdot x + b, x \in [x_1, x_2]$$

$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

$$b = y1 - m \cdot x_1$$



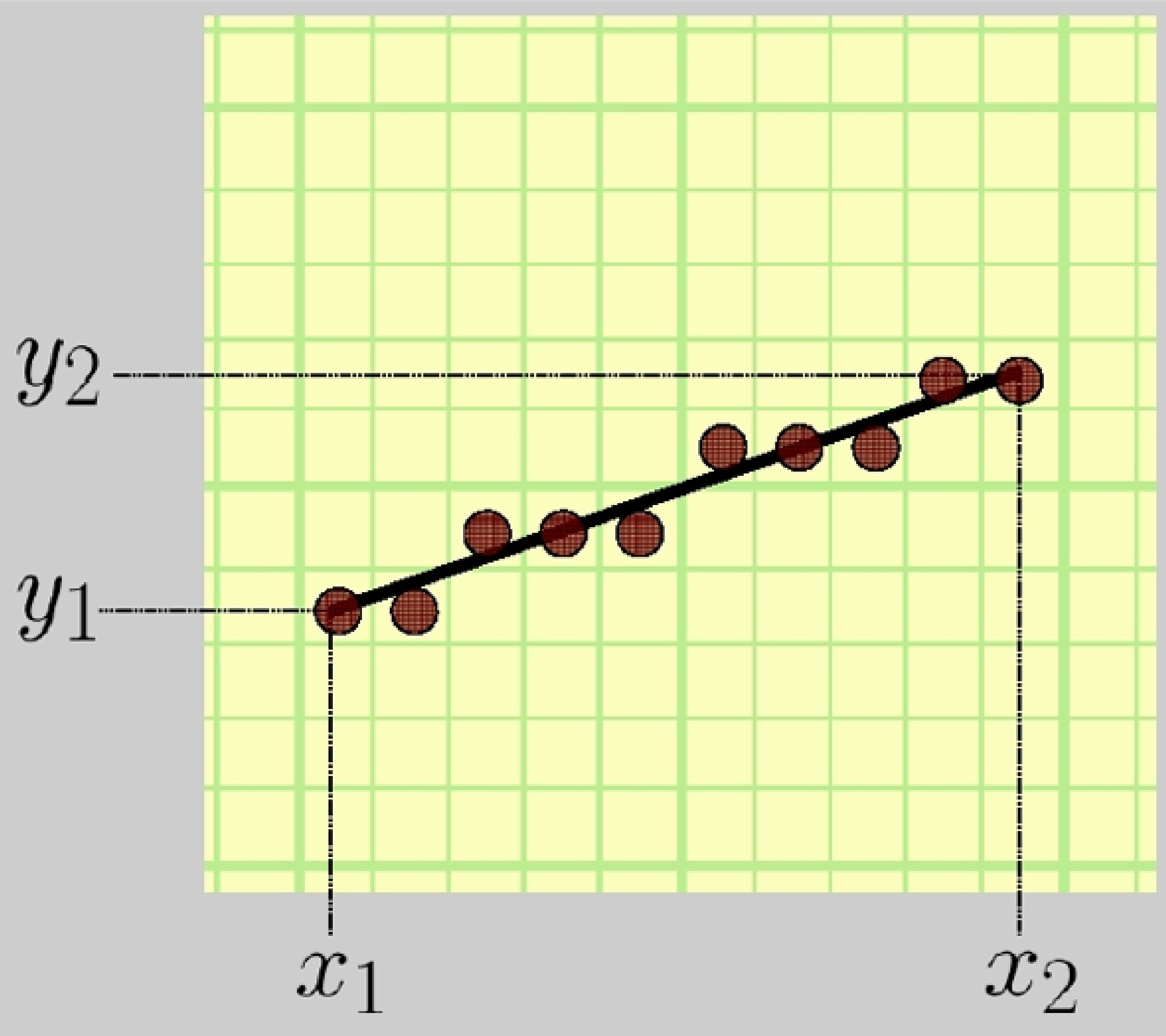
Drawing a Line



$$\Delta x = 1$$
$$\Delta y = m \cdot \Delta x$$

```
x=x1
y=y1
while(x<=x2)
   plot(x,y)
   x++
   y+=Dy
```

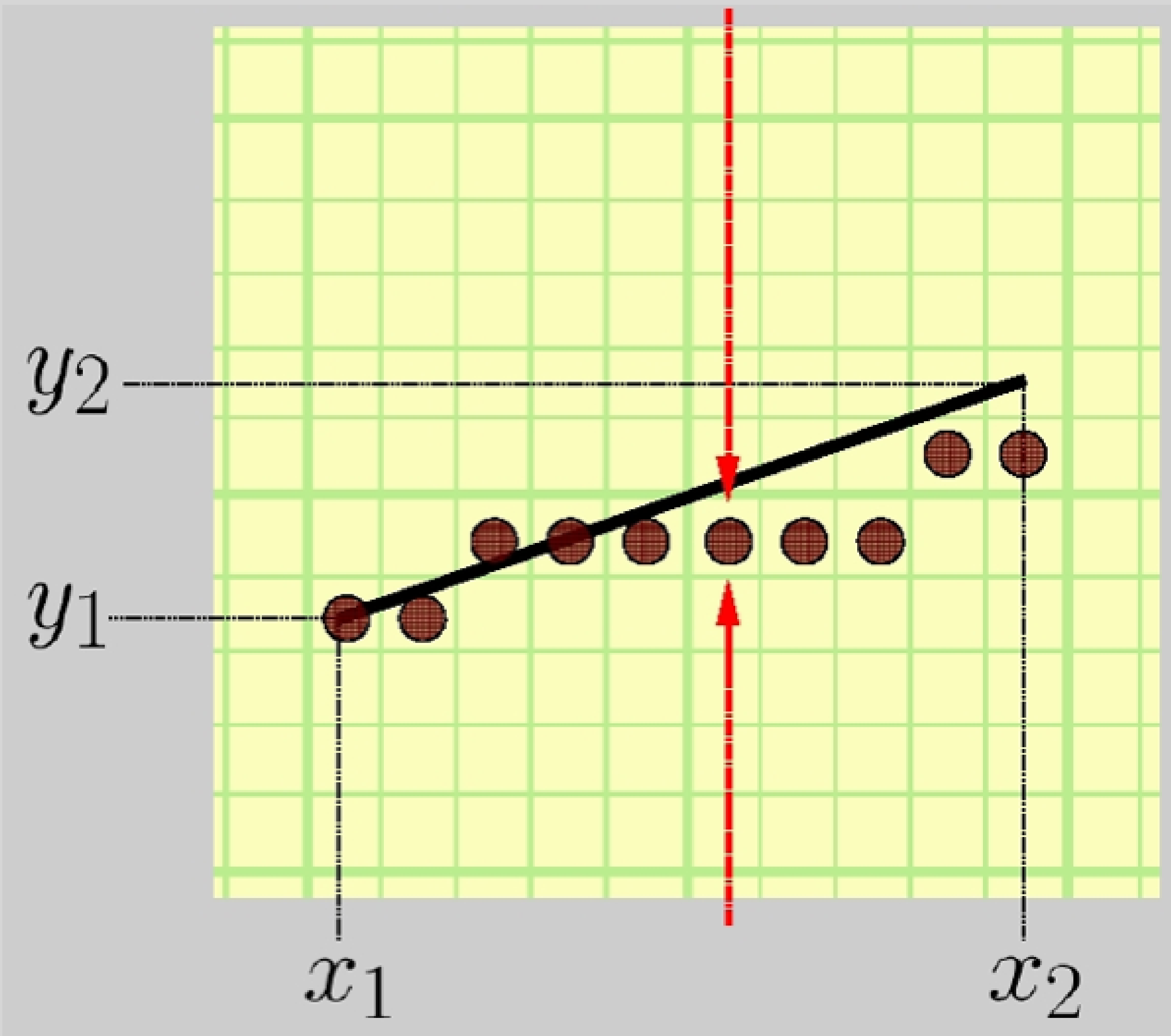
Drawing a Line



$$\Delta x = 1$$
$$\Delta y = m \cdot \Delta x$$

After rounding

Drawing a Line

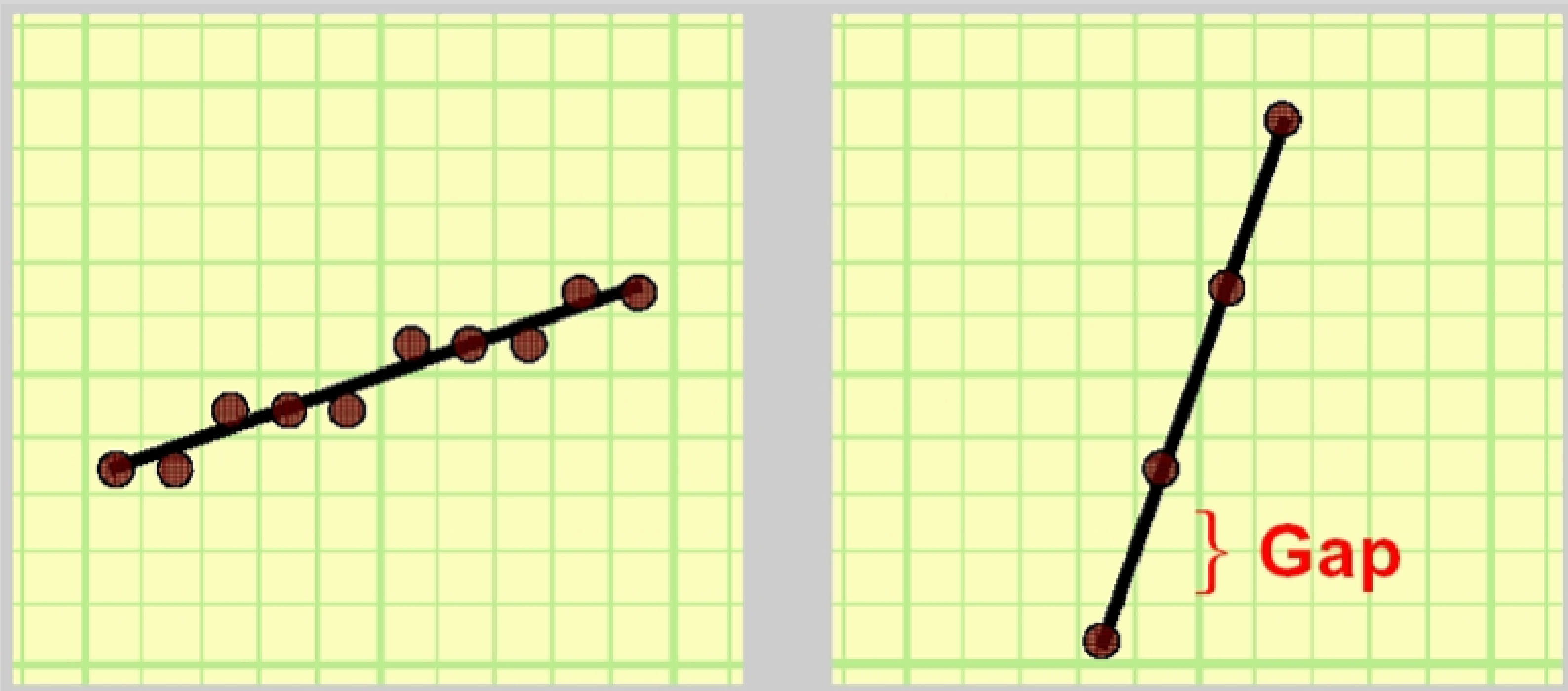


$$\Delta x = 1$$
$$\Delta y = m \cdot \Delta x$$
$$y += \Delta y$$

Accumulation of roundoff errors

How slow is float-to-int conversion?

Drawing a Line



$|m| \leq 1$ $\qquad$ $|m| > 1$

Drawing a Line

```
void drawLine-Error1(int x1,x2, int y1,y2)

  float m = float(y2-y1)/(x2-x1)
  int x = x1
  float y = y1

  while (x <= x2)

    setPixel(x,round(y),PIXEL_ON)

    x += 1
    y += m
```

Not exact math

Accumulates errors

Drawing a Line

```
void drawLine-Error2(int x1,x2, int y1,y2)

  float m = float(y2-y1)/(x2-x1)
  int x = x1
  int y = y1
  float e = 0.0

  while (x <= x2)

    setPixel(x,y,PIXEL_ON)

    x += 1
    e += m
    if (e >= 0.5)
      y+=1
      e-=1.0
```

No more rounding

Drawing a Line

```
void drawLine-Error3(int x1,x2, int y1,y2)

  int x = x1
  int y = y1
  float e = -0.5

  while (x <= x2)

    setPixel(x,y,PIXEL_ON)

    x += 1
    e += float(y2-y1)/(x2-x1)
    if (e >= 0.0)
      y+=1
      e-=1.0
```

Drawing a Line

```
void drawLine-Error4(int x1,x2, int y1,y2)

  int x = x1
  int y = y1
  float e = -0.5*(x2-x1)              // was -0.5

  while (x <= x2)

    setPixel(x,y,PIXEL_ON)

    x += 1
    e += y2-y1                        // was /(x2-x1)
    if (e >= 0.0)                     // no change
      y+=1
      e-=(x2-x1)                      // was 1.0
```

Drawing a Line

```
void drawLine-Error5(int x1,x2, int y1,y2)

  int x = x1
  int y = y1
  int e = -(x2-x1)                    // removed *0.5

  while (x <= x2)

    setPixel(x,y,PIXEL_ON)

    x += 1
    e += 2*(y2-y1)                    // added 2*
    if (e >= 0.0)                     // no change
      y+=1
      e-=2*(x2-x1)                    // added 2*
```

Drawing a Line

```
void drawLine-Bresenham(int x1,x2, int y1,y2)

  int x = x1
  int y = y1
  int e = -(x2-x1)

  while (x <= x2)

    setPixel(x,y,PIXEL_ON)

    x += 1
    e += 2*(y2-y1)
    if (e >= 0.0)
      y+=1
      e-=2*(x2-x1)
```

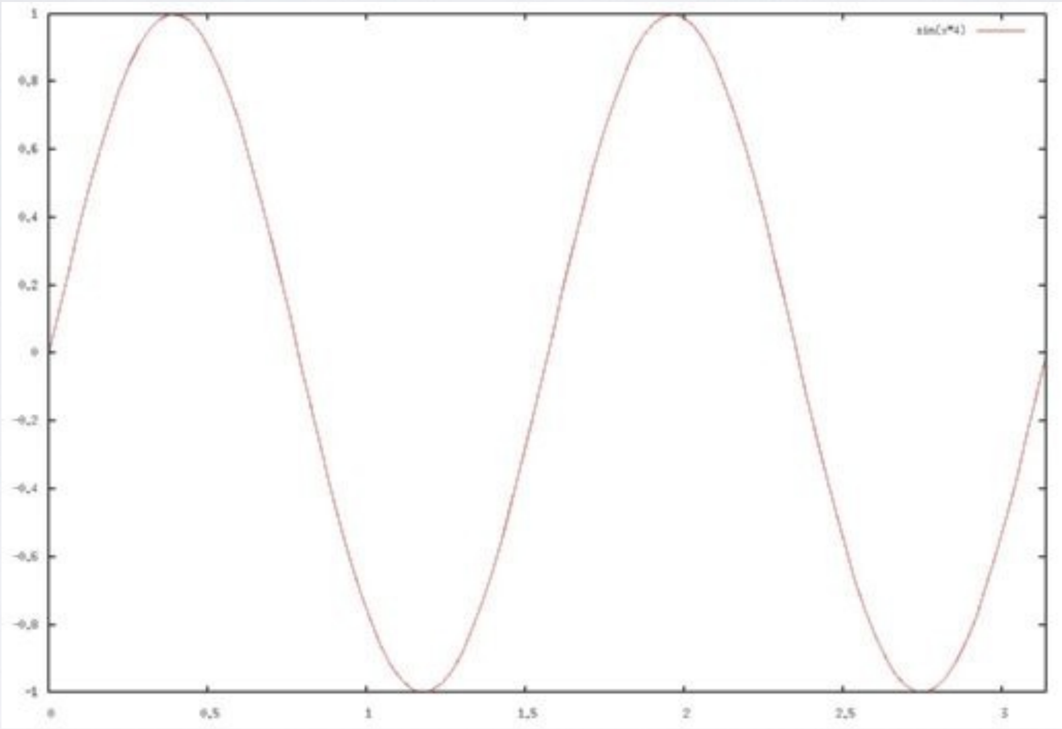
Faster

Not wrong

$$0 \leq m \leq 1$$

$$x_1 \leq x_2$$

Drawing Curves



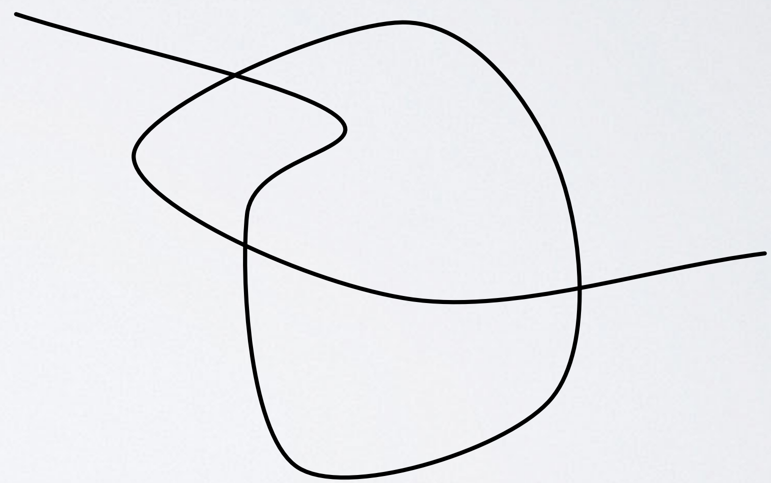
$$y = f(x)$$

Only one value of y for each value of x...

Drawing Curves

- Parametric curves
  - Both x and y are a function of some third parameter

$$x = f(u)$$
$$y = f(u)$$



$$\mathbf{x} = \mathbf{f}(u)$$

$$u \in [u_0 \ldots u_1]$$

Drawing Curves

$$\mathbf{x} = \mathbf{f}(u) \qquad u \in [u_0 \ldots u_1]$$

Drawing Curves

- Draw curves by drawing line segments
  - Must take care in computing end points for lines
  - How long should each line segment be?

$$\mathbf{x} = \mathbf{f}(u) \qquad u \in [u_0 \ldots u_1]$$

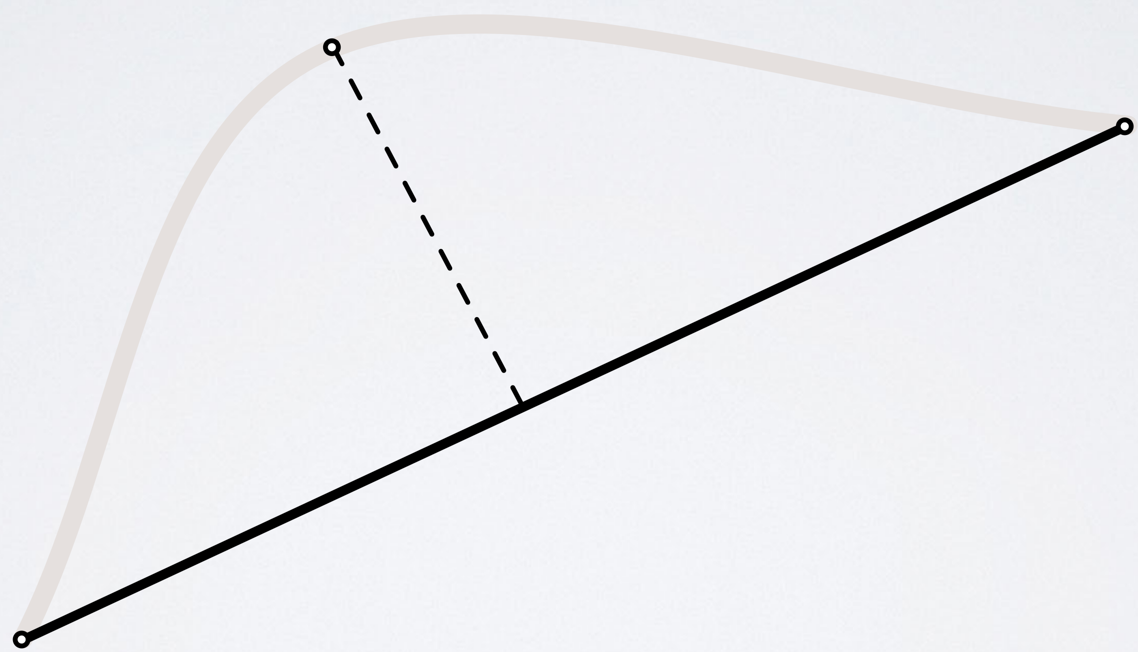
Drawing Curves

- Draw curves by drawing line segments
  - Must take care in computing end points for lines
  - How long should each line segment be?
  - Variable spaced points

$$\mathbf{x} = \mathbf{f}(u) \qquad u \in [u_0 \ldots u_1]$$

Drawing Curves

- Midpoint-test subdivision



$$|\mathbf{f}(u_{mid}) - \mathbf{l}(0.5)|$$

Drawing Curves

- Midpoint-test subdivision

$$|\mathbf{f}(u_{mid}) - \mathbf{l}(0.5)|$$

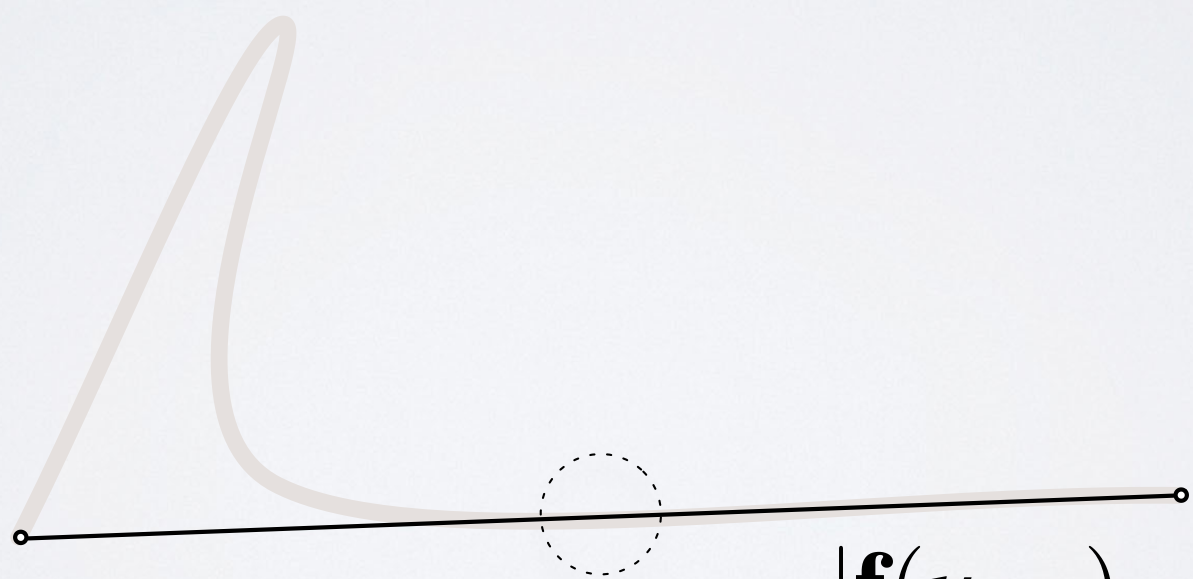
Drawing Curves

- Midpoint-test subdivision

$$|\mathbf{f}(u_{mid}) - \mathbf{l}(0.5)|$$

Drawing Curves

- Midpoint-test subdivision
  - Not perfect
  - We need more information for a guarantee...



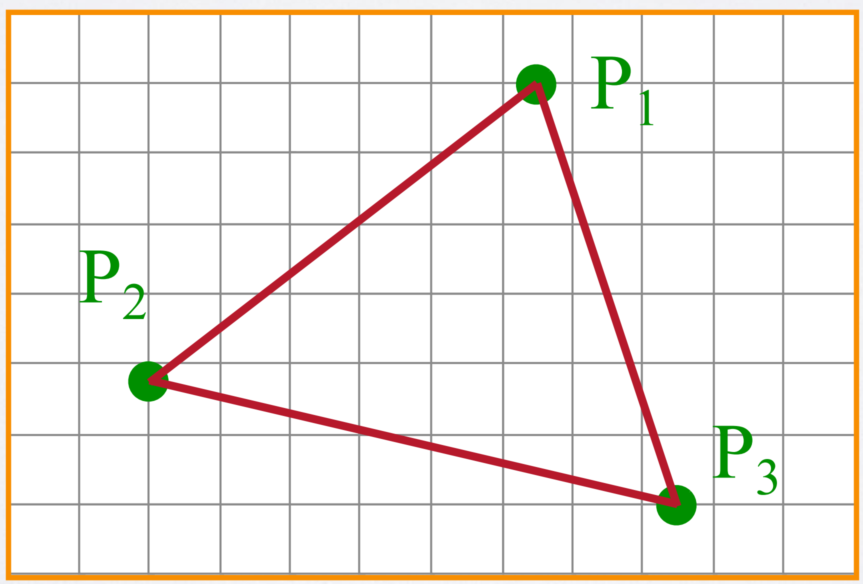
$$|\mathbf{f}(u_{mid}) - \mathbf{l}(0.5)|$$

Filling Triangles

- Render an image of a geometric primitive by setting pixel colors

```
void SetPixel(int x, int y, Color rgba)
```

- Example: Filling the inside of a triangle

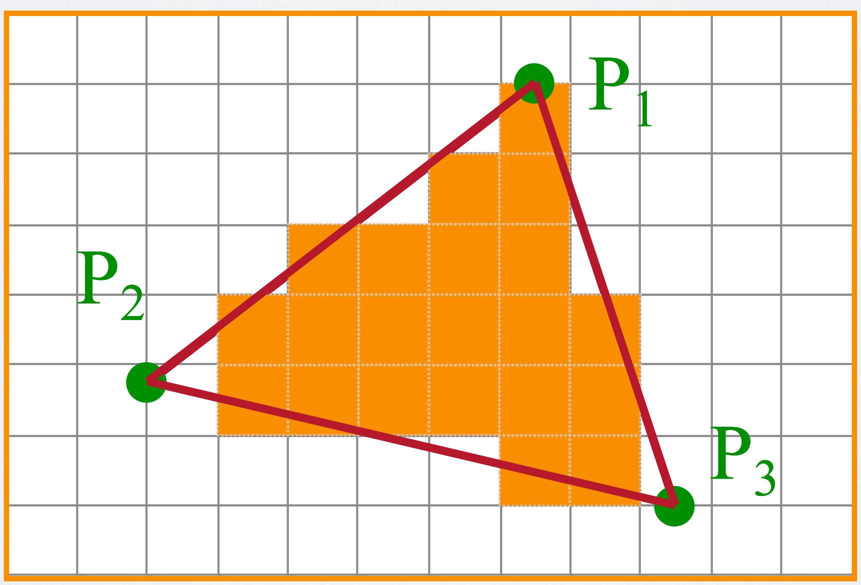


Filling Triangles

- Render an image of a geometric primitive by setting pixel colors

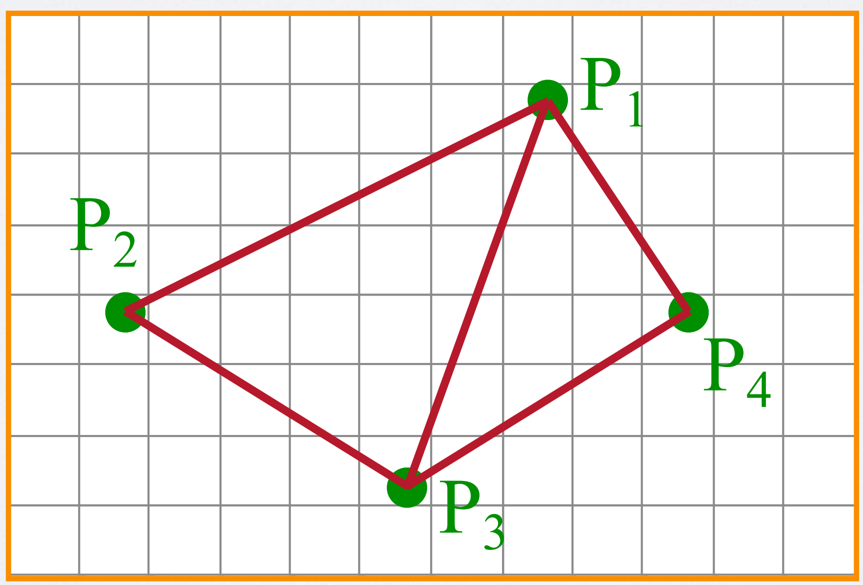
```
void SetPixel(int x, int y, Color rgba)
```

- Example: Filling the inside of a triangle



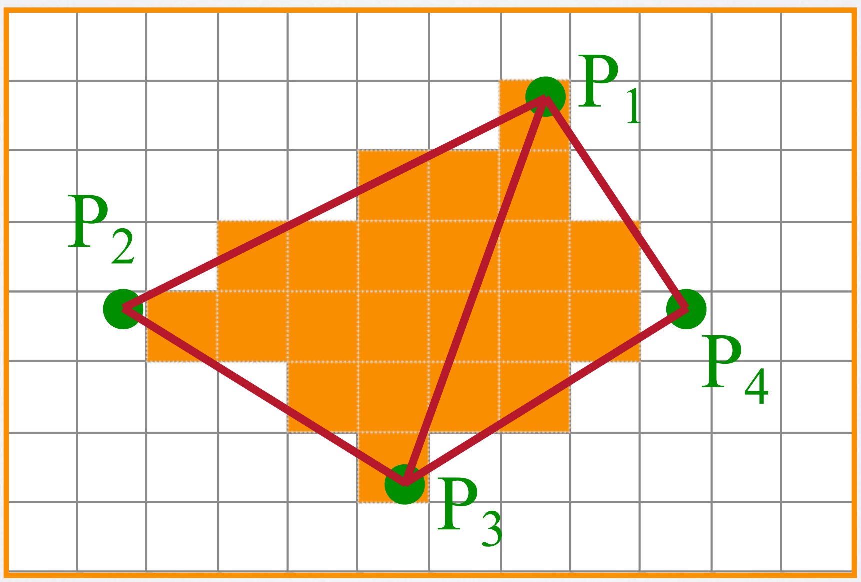
Triangle Scan Conversion

- Properties of a good algorithm
  - Symmetric
  - Straight edges
  - Antialiased edges
  - No cracks between adjacent primitives
  - MUST BE FAST!



Triangle Scan Conversion

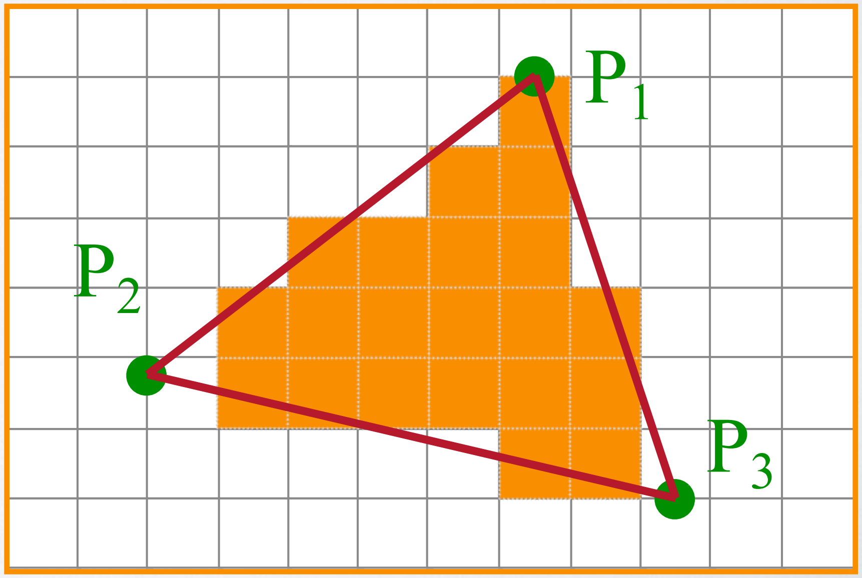
- Properties of a good algorithm
  - Symmetric
  - Straight edges
  - Antialiased edges
  - No cracks between adjacent primitives
  - MUST BE FAST!



Simple Algorithm

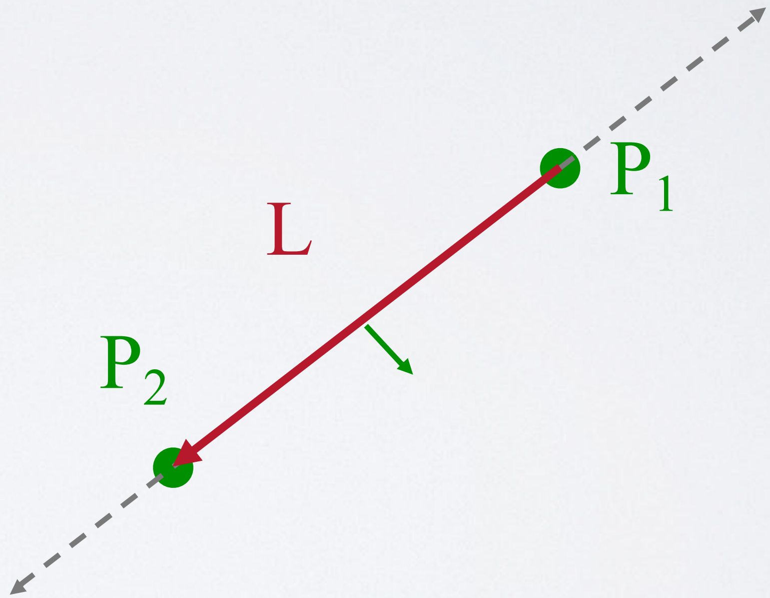
- Color all pixels inside triangle

```
void ScanTriangle(Triangle T, Color rgba){
   for each pixel P at (x,y){
       if (Inside(T, P))
           SetPixel(x, y, rgba);
   }
}
```



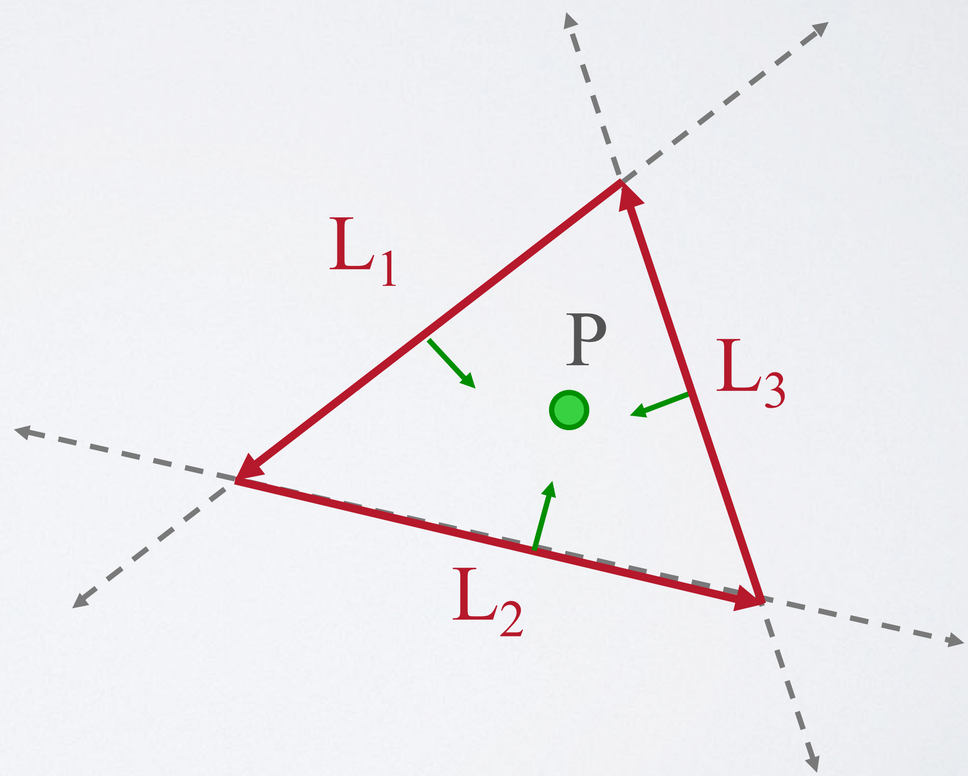
Line Defines Two Halfspaces

- Implicit equation for a line
  - On line: $ax + by + c = 0$
  - On right: $ax + by + c < 0$
  - On left: $ax + by + c > 0$



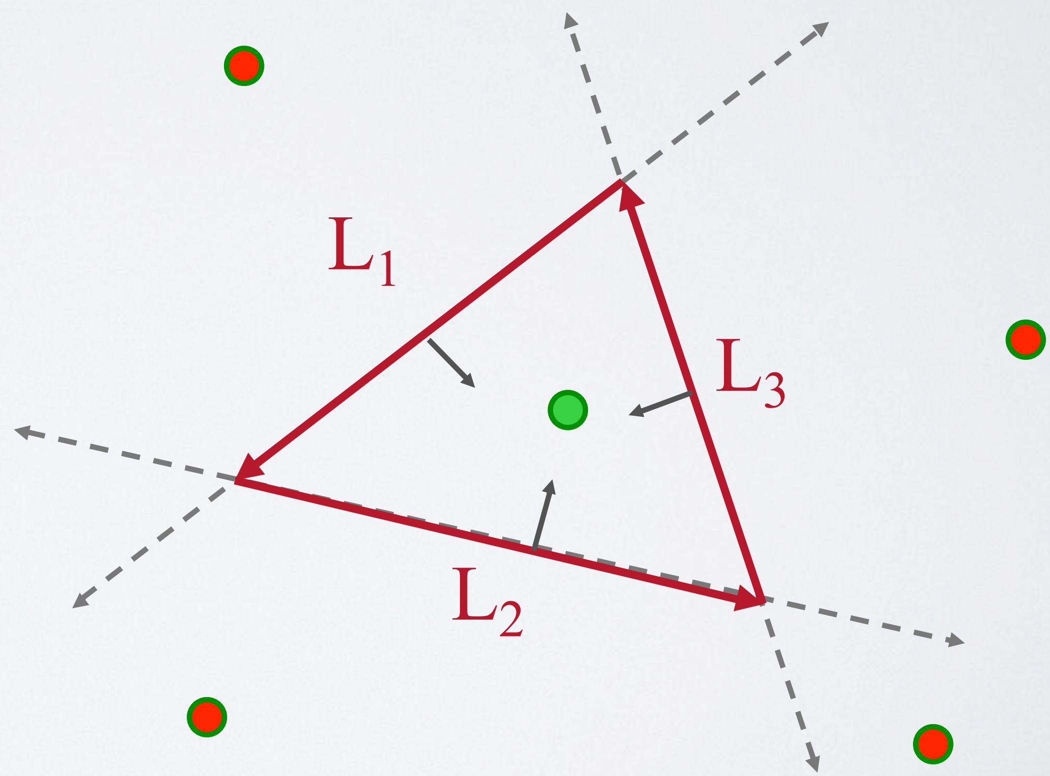
Inside Triangle Test

- Point is inside triangle if it is in positive halfspace of all three boundary lines
  - Triangle vertices are ordered counter-clockwise
  - Point must be on the left side of every boundary line



Inside Triangle Test

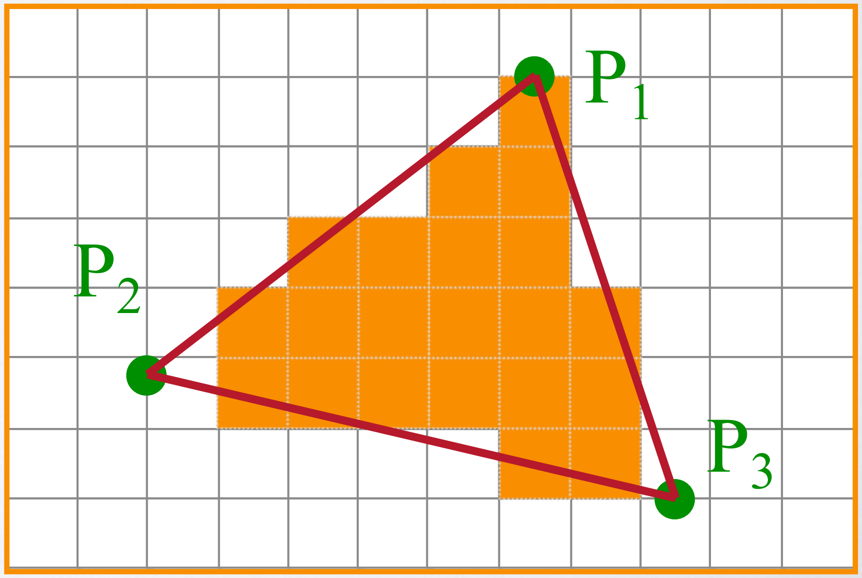
```
Boolean Inside(Triangle T, Point P)
{
   for each boundary line L of T {
      Scalar d = L.a*P.x + L.b*P.y + L.c;
      if (d < 0.0) return FALSE;
   }
   return TRUE;
}
```



Simple Algorithm

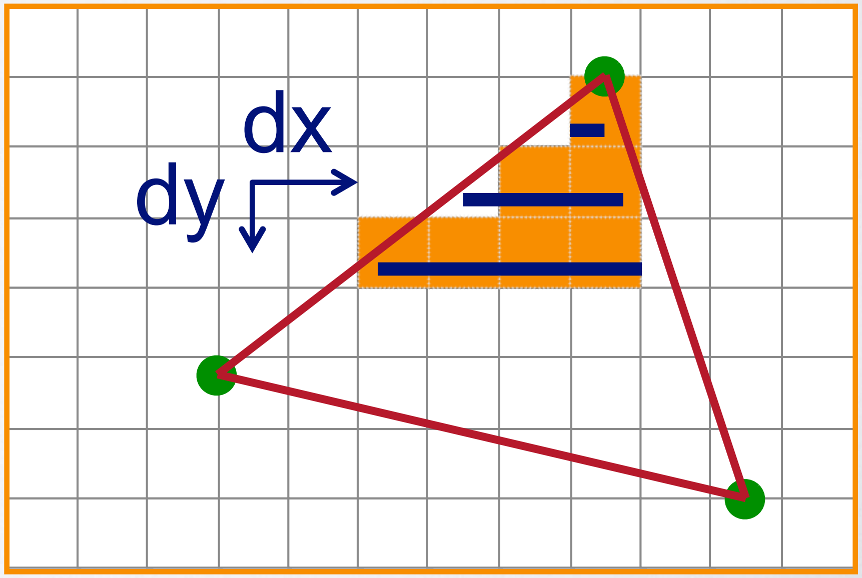
- What is bad about this algorithm?

```
void ScanTriangle(Triangle T, Color rgba){
   for each pixel P at (x,y){
       if (Inside(T, P))
           SetPixel(x, y, rgba);
   }
}
```



Triangle Sweep-Line Algorithm

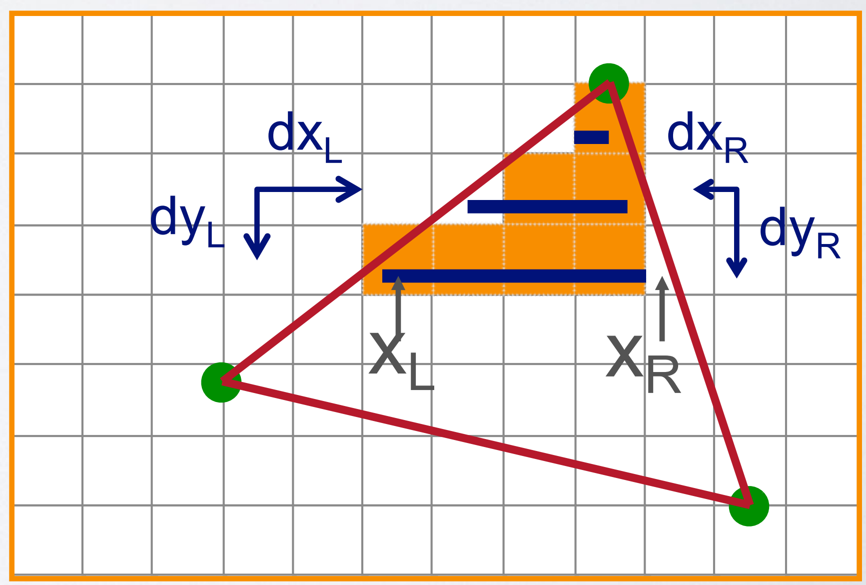
- Take advantage of spatial coherence
  - Compute which pixels are inside using horizontal spans
  - Process horizontal spans in scan-line order
- Take advantage of edge linearity
  - Use edge slopes to update coordinates incrementally



Triangle Sweep-Line Algorithm

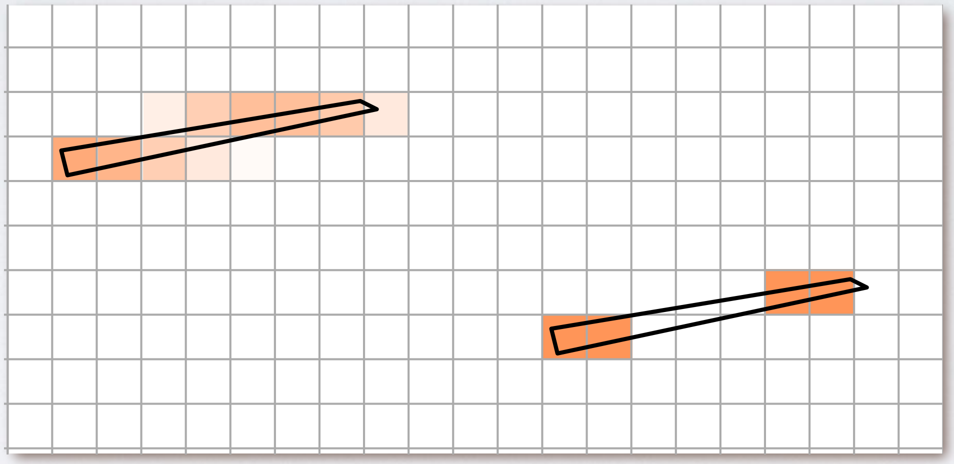
```
void ScanTriangle(Triangle T, Color rgba){
    for each edge pair {
        initialize x_L, x_R;
        compute dx_L/dy_L and dx_R/dy_R;
        for each scanline at y
            for (int x = ceil(x_L); x <= x_R; x++)
                SetPixel(x, y, rgba);
        x_L += dx_L/dy_L;
        x_R += dx_R/dy_R;
    }
}
```

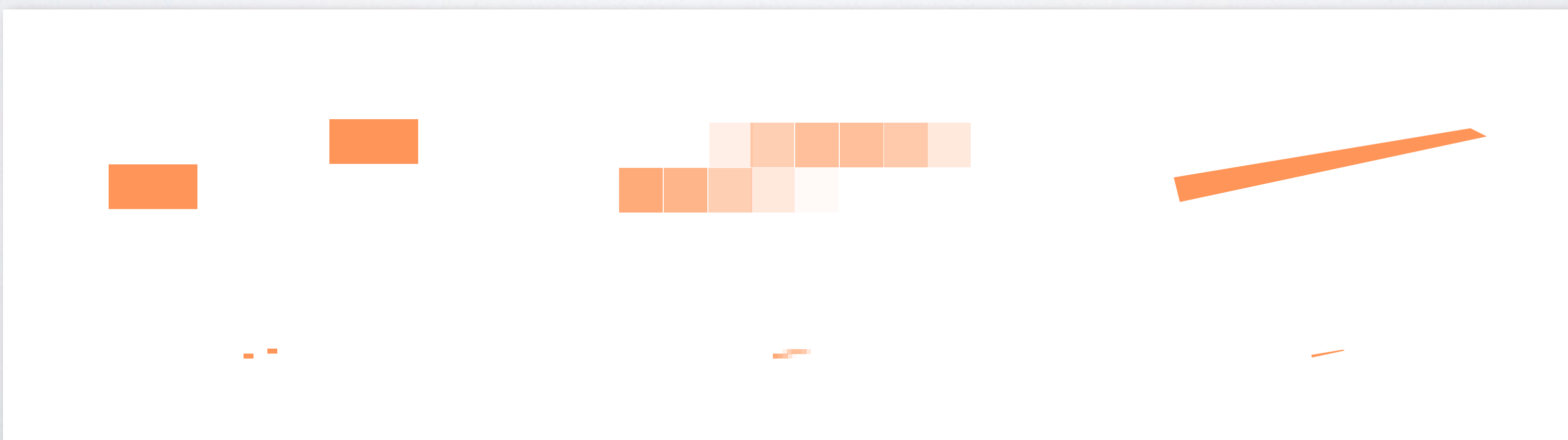
Bresenham's algorithm works the same way, but uses only integer operations!



Antialiasing

Desired solution of an integral over pixel

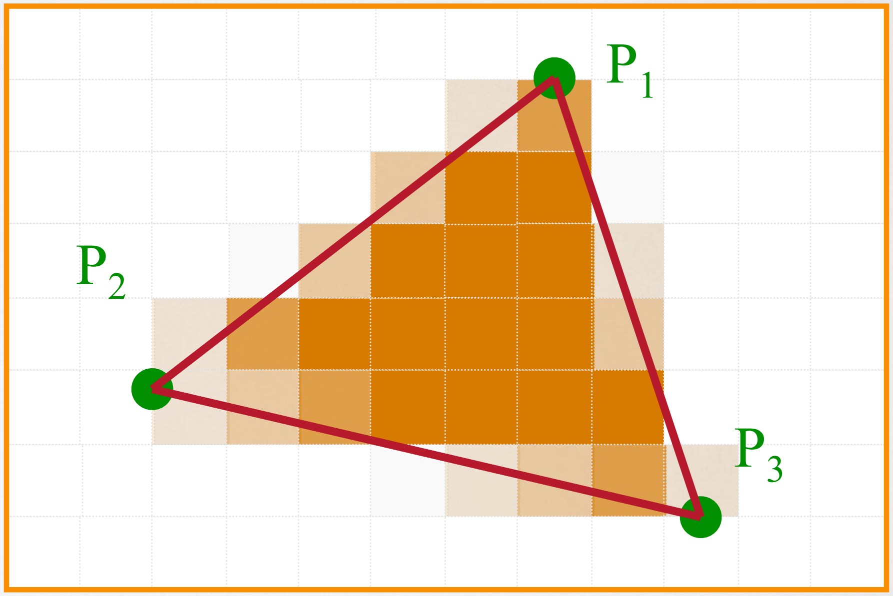




Hardware Antialiasing

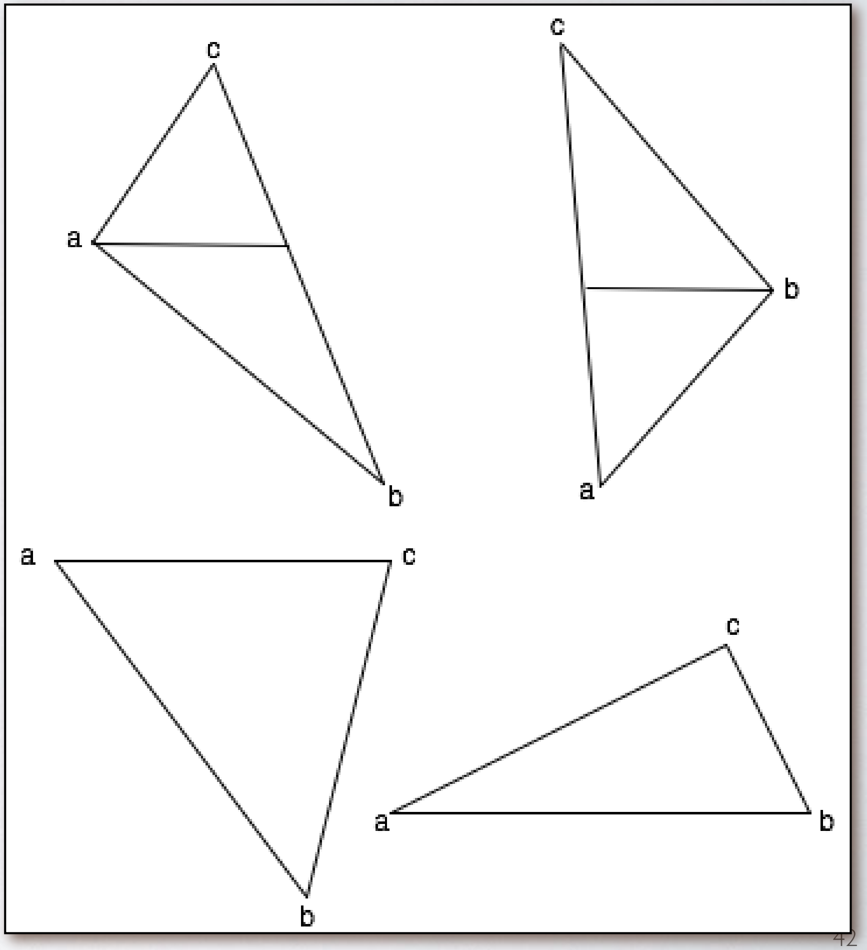
Supersample pixels

- Multiple samples per pixel
- Average subpixel intensities (box filter)
- Trades intensity resolution for spatial resolution



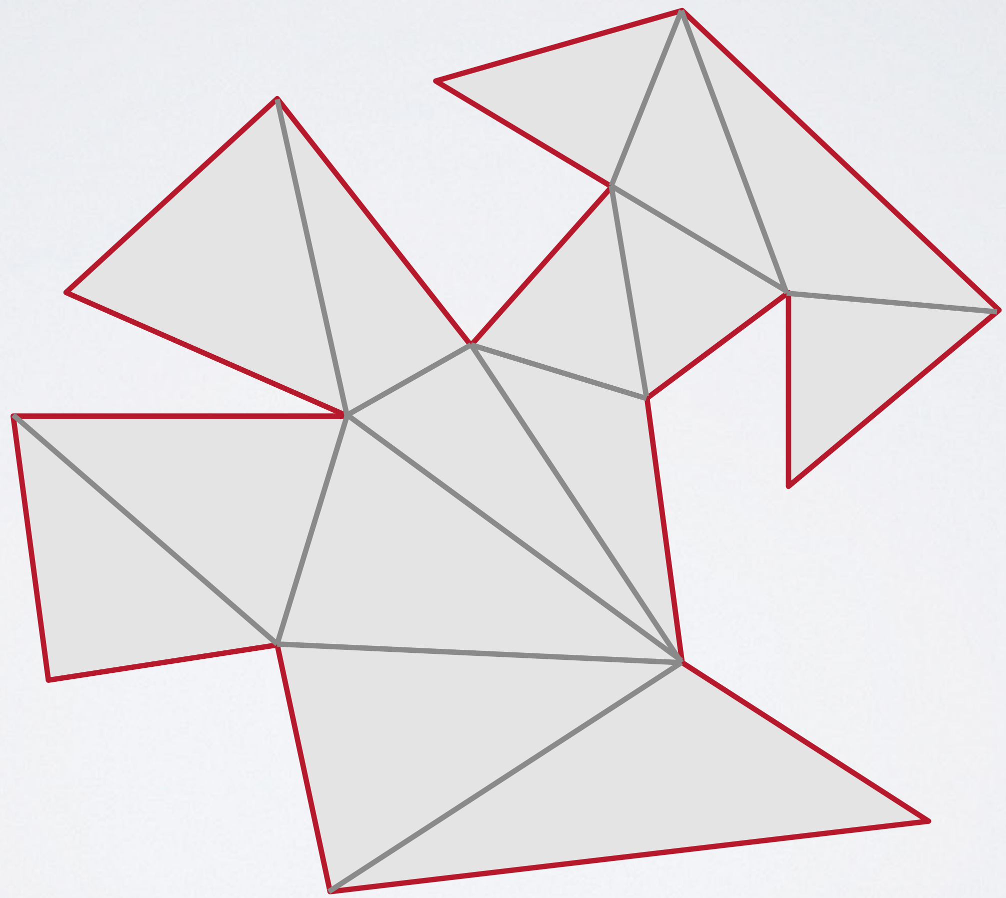
Optimize for Triangles

- Spilt triangle into two parts
  - Two edges per part
  - Y-span is monotonic
- For each row
  - Interpolate span
- Interpolate barycentric coordinates



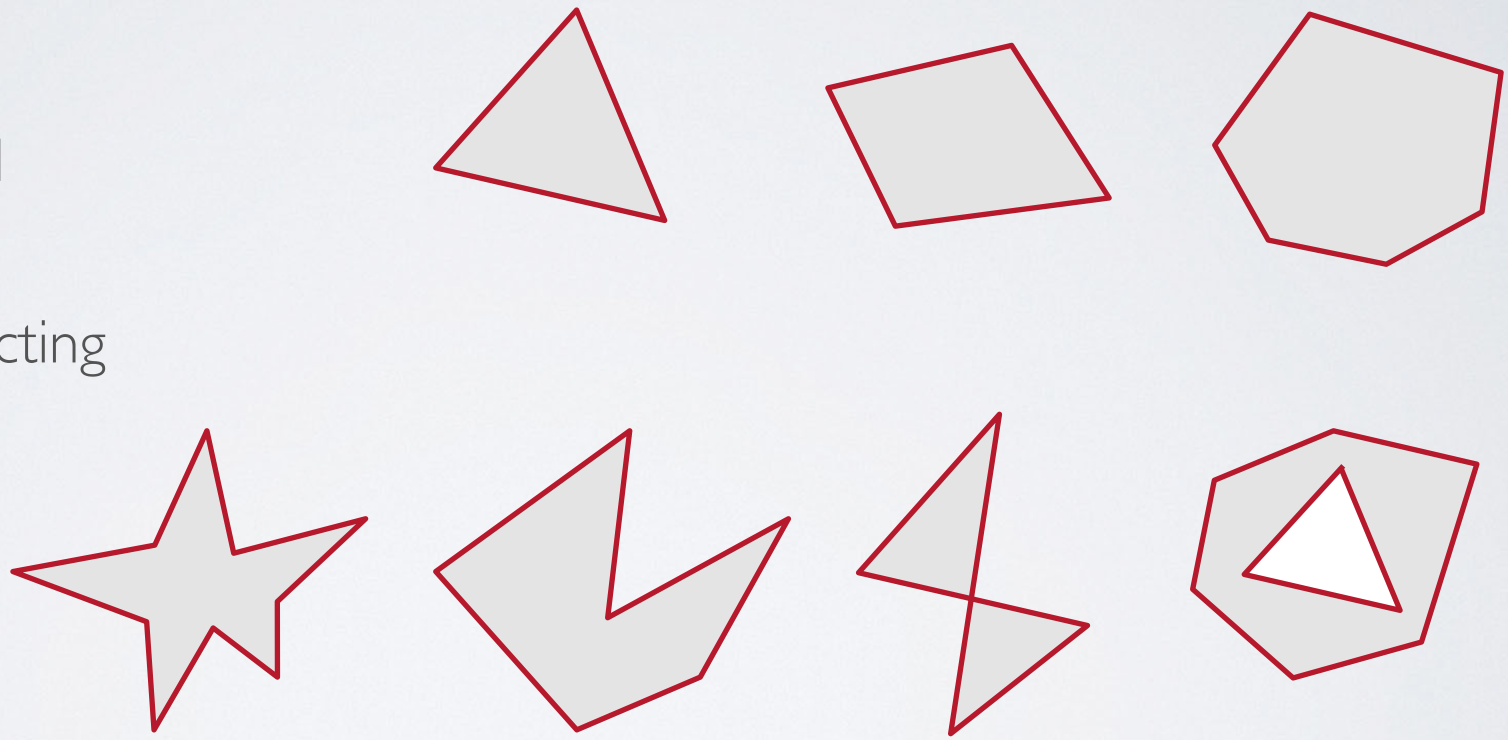
Hardware Scan Conversion

- Convert everything into triangles
  - Scan convert the triangles



Polygon Scan Conversion

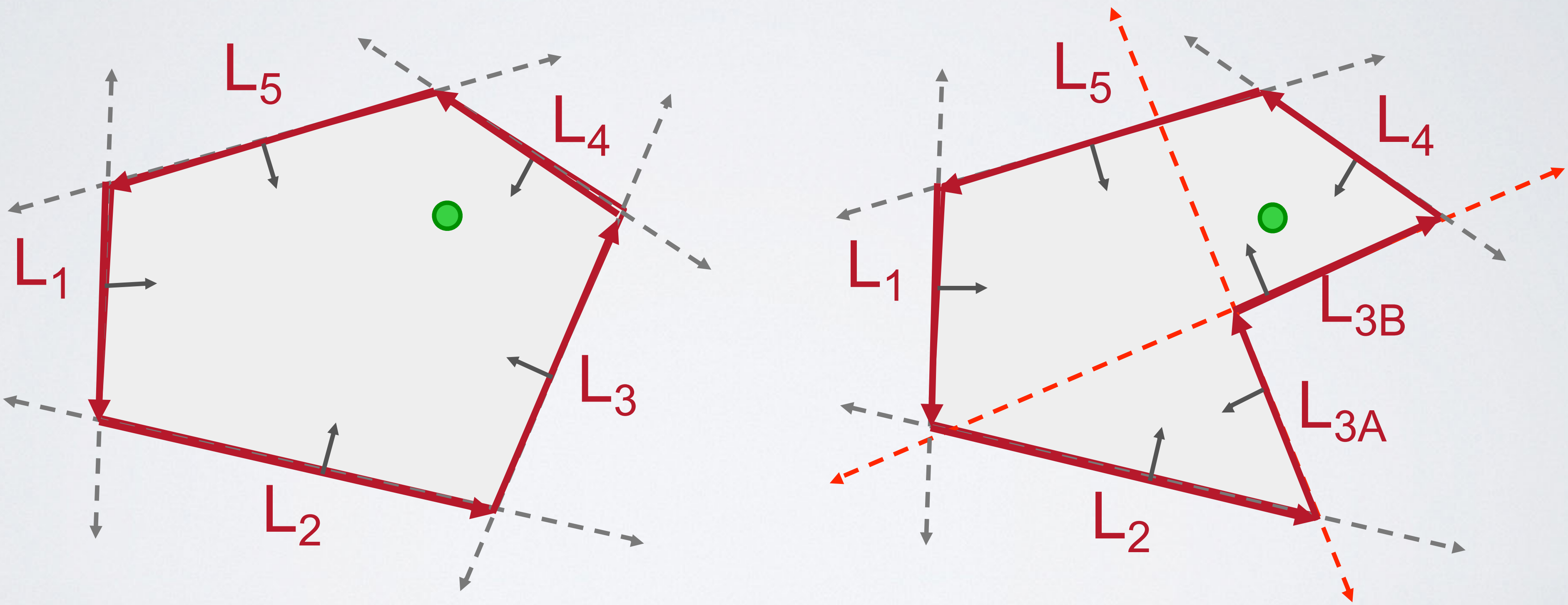
- Fill pixels inside a polygon
  - Triangle
  - Quadrilateral
  - Convex
  - Star-shaped
  - Concave
  - Self-intersecting
  - Holes



What problems do we encounter with arbitrary polygons?

Polygon Scan Conversion

- Need better test for points inside polygon
  - Triangle method works only for convex polygons

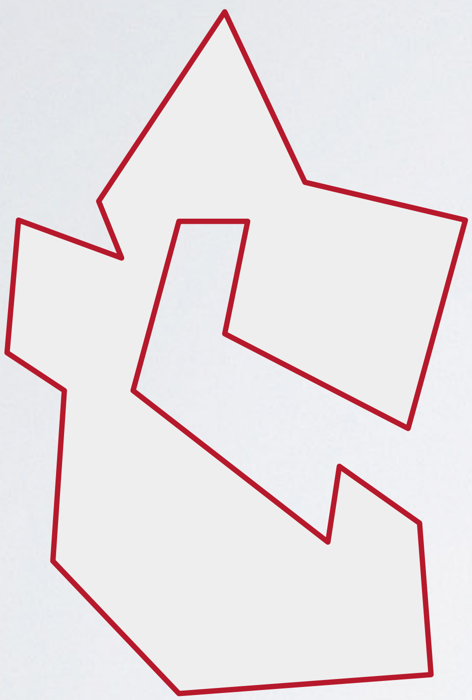


Convex Polygon

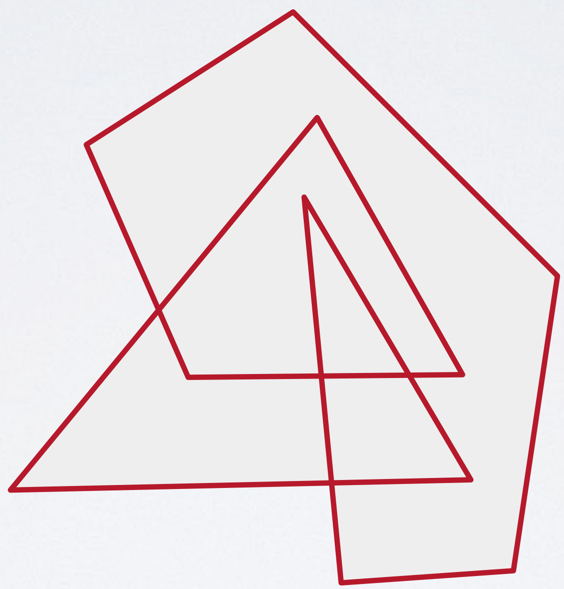
Concave Polygon

Inside Polygon Rule

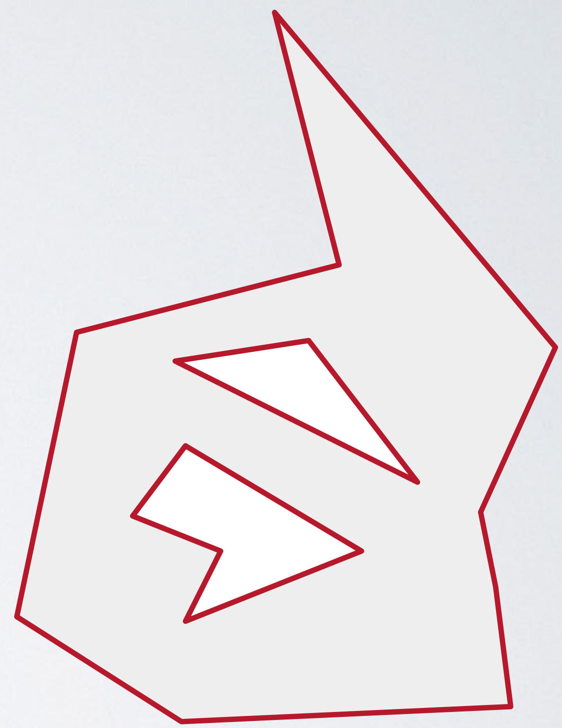
- What is a good rule for which pixels are inside?



Concave



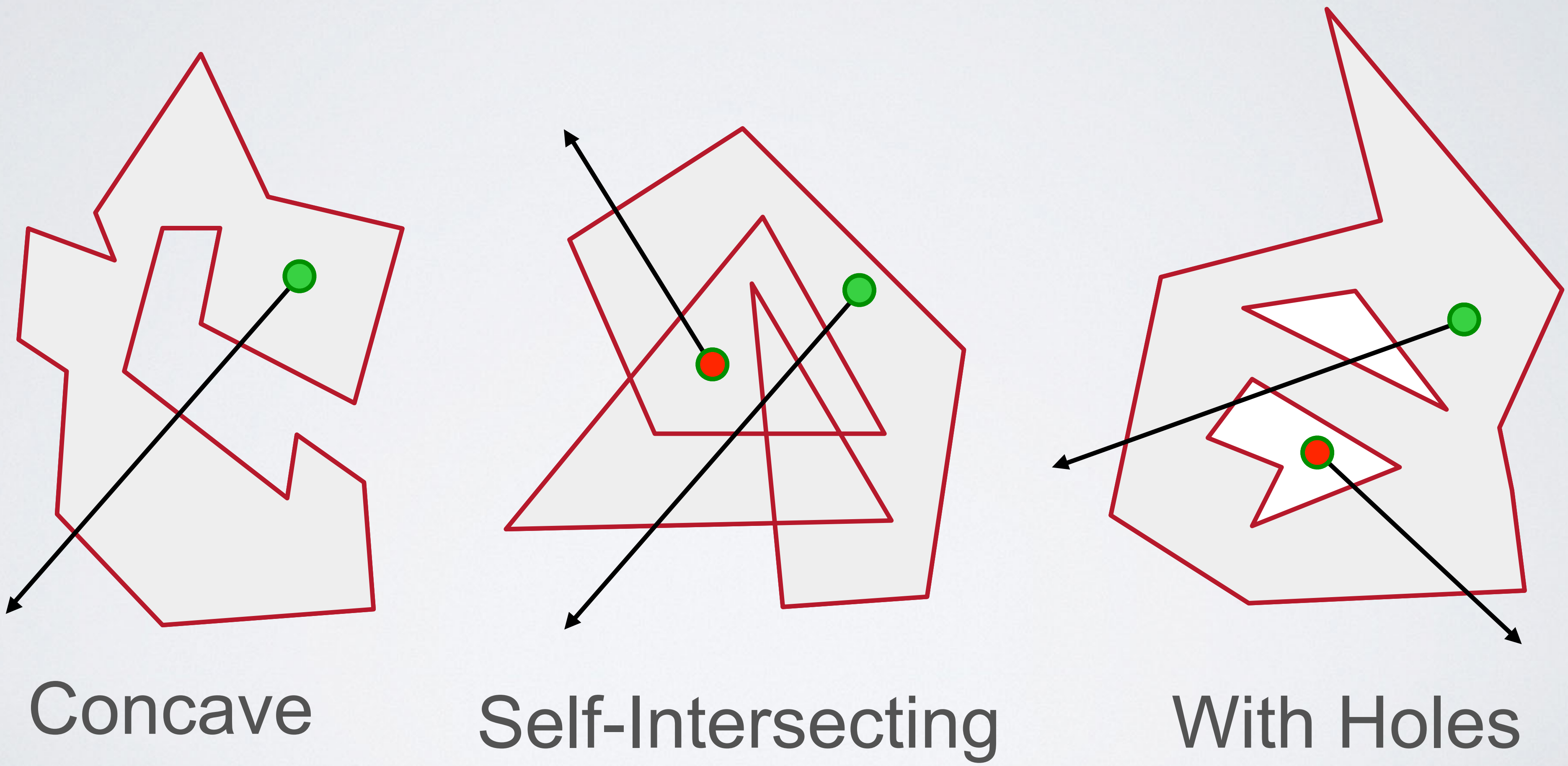
Self-Intersecting



With Holes

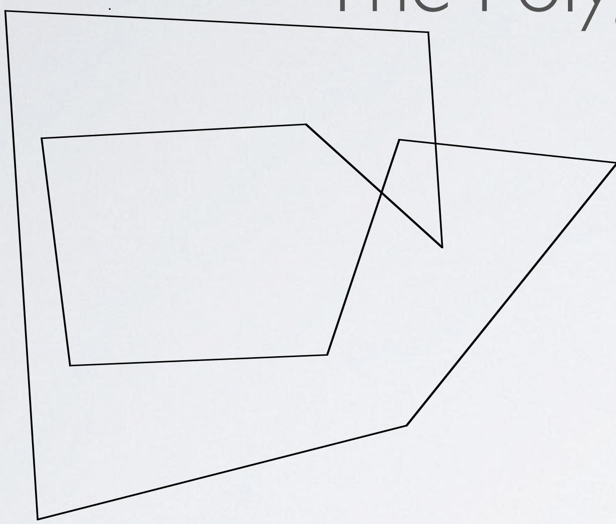
Inside Polygon Rule

- Odd-parity rule
  - Any ray from P to infinity crosses odd number of edges



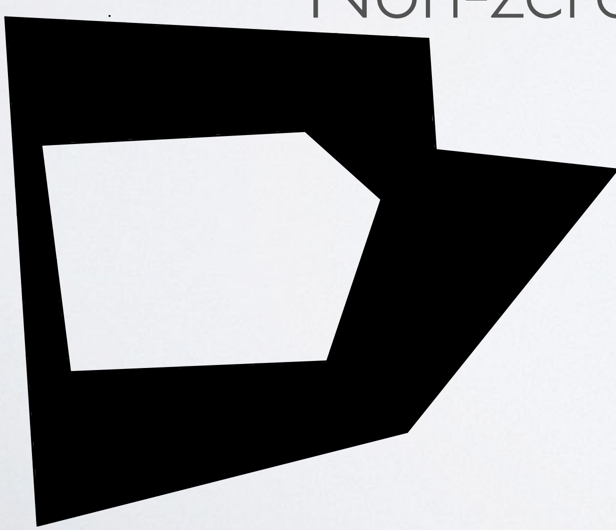
Inside/Outside Testing

The Polygon



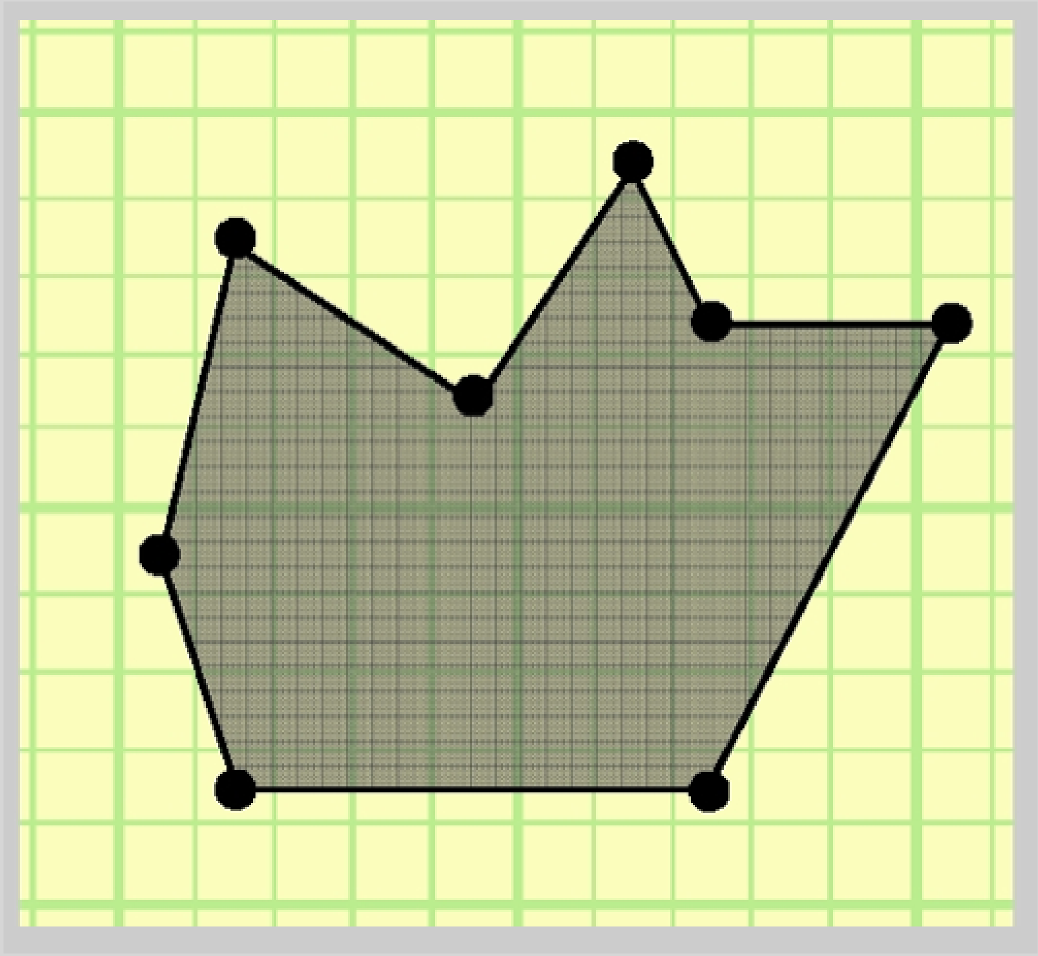
Non-exterior

Non-zero winding

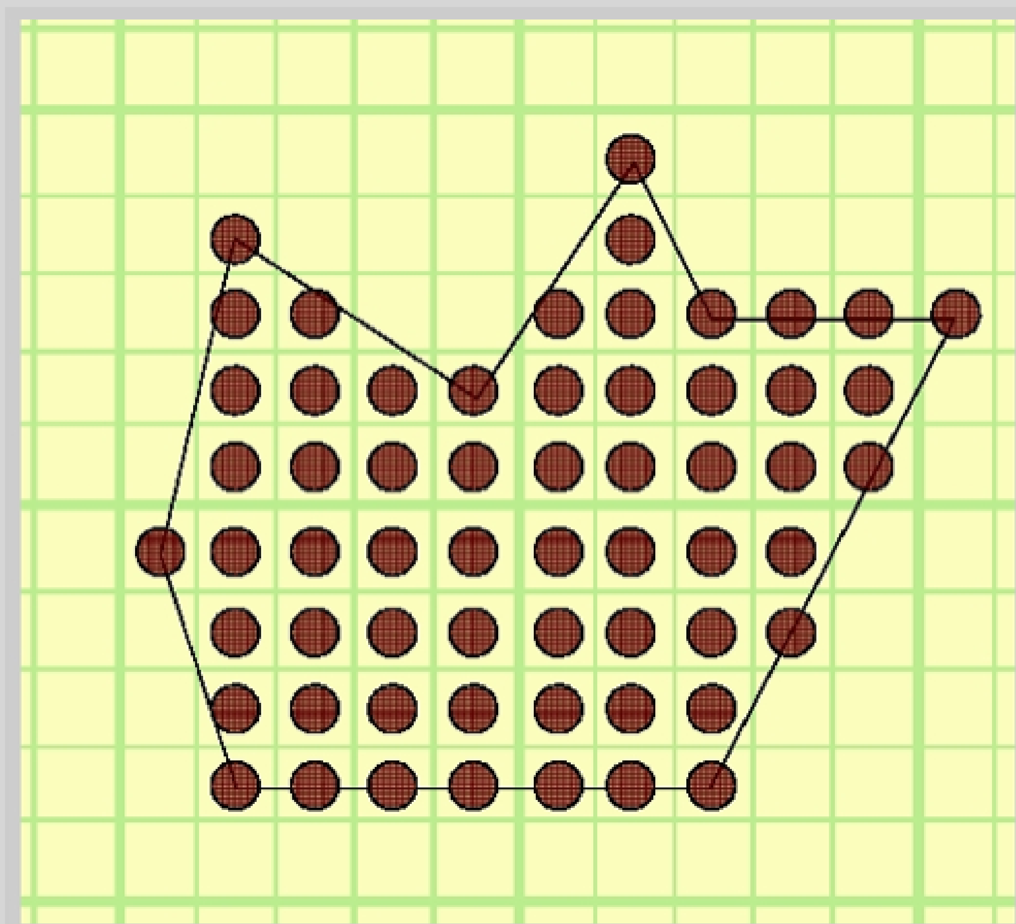


Parity

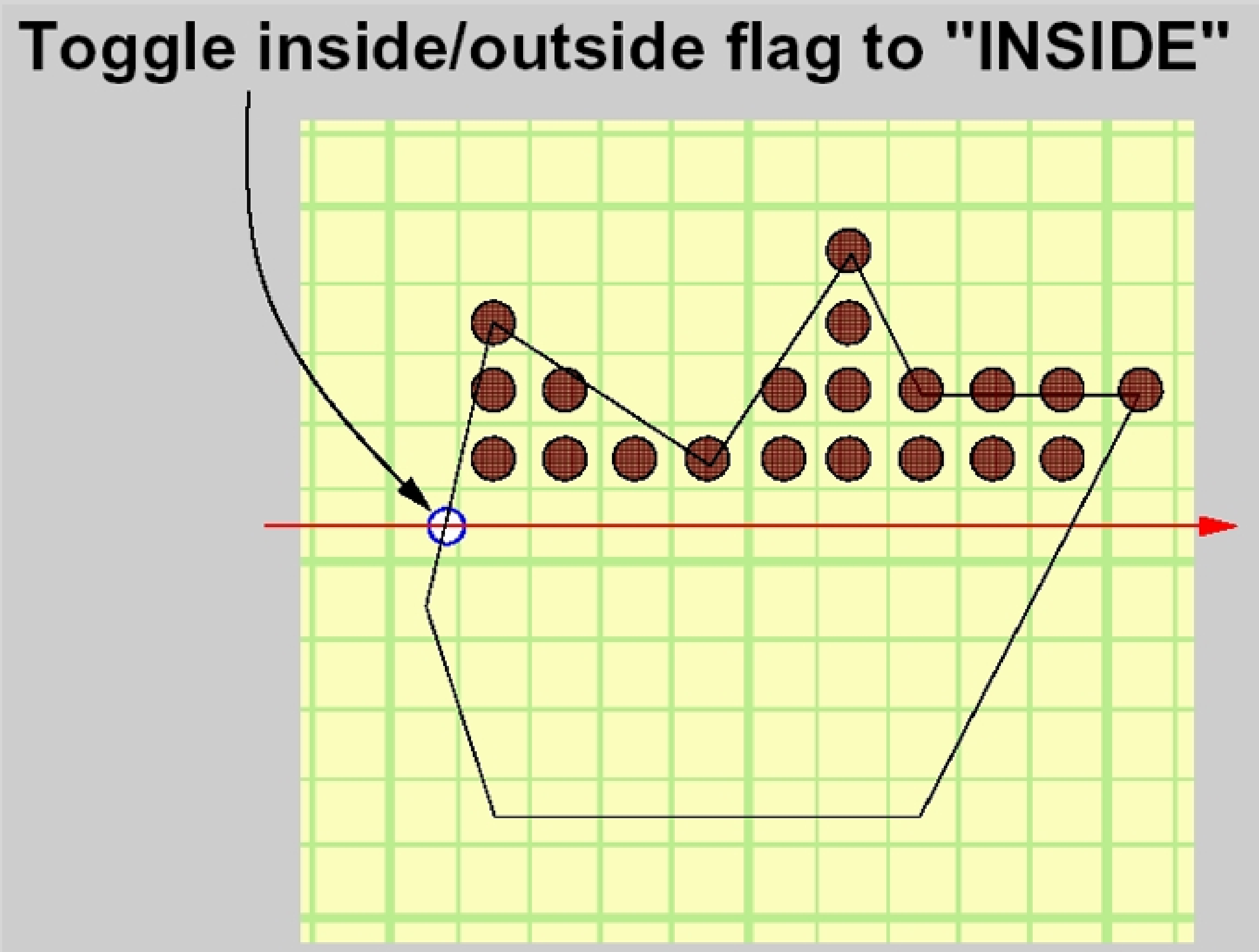
Filled Polygons



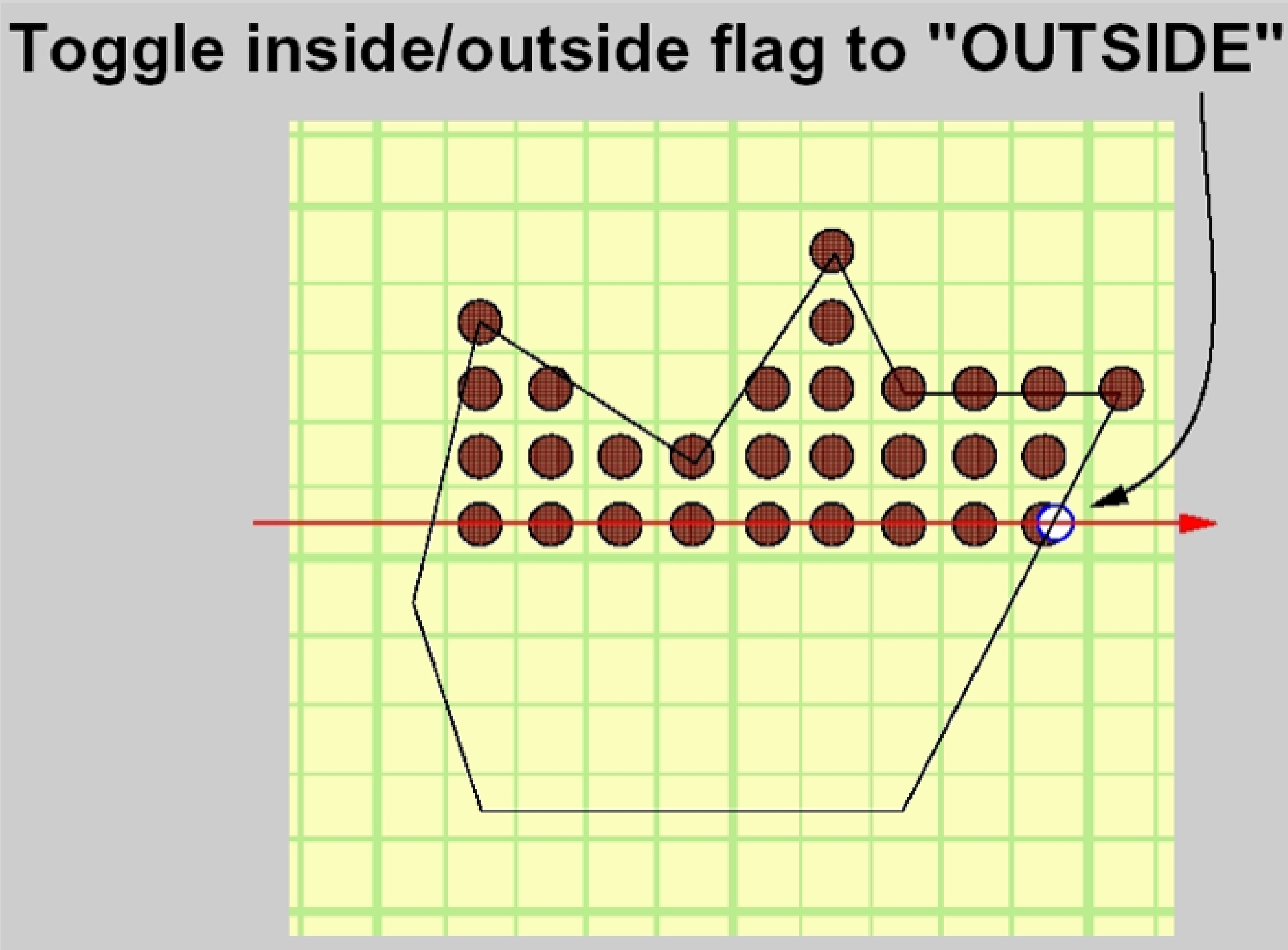
Filled Polygons



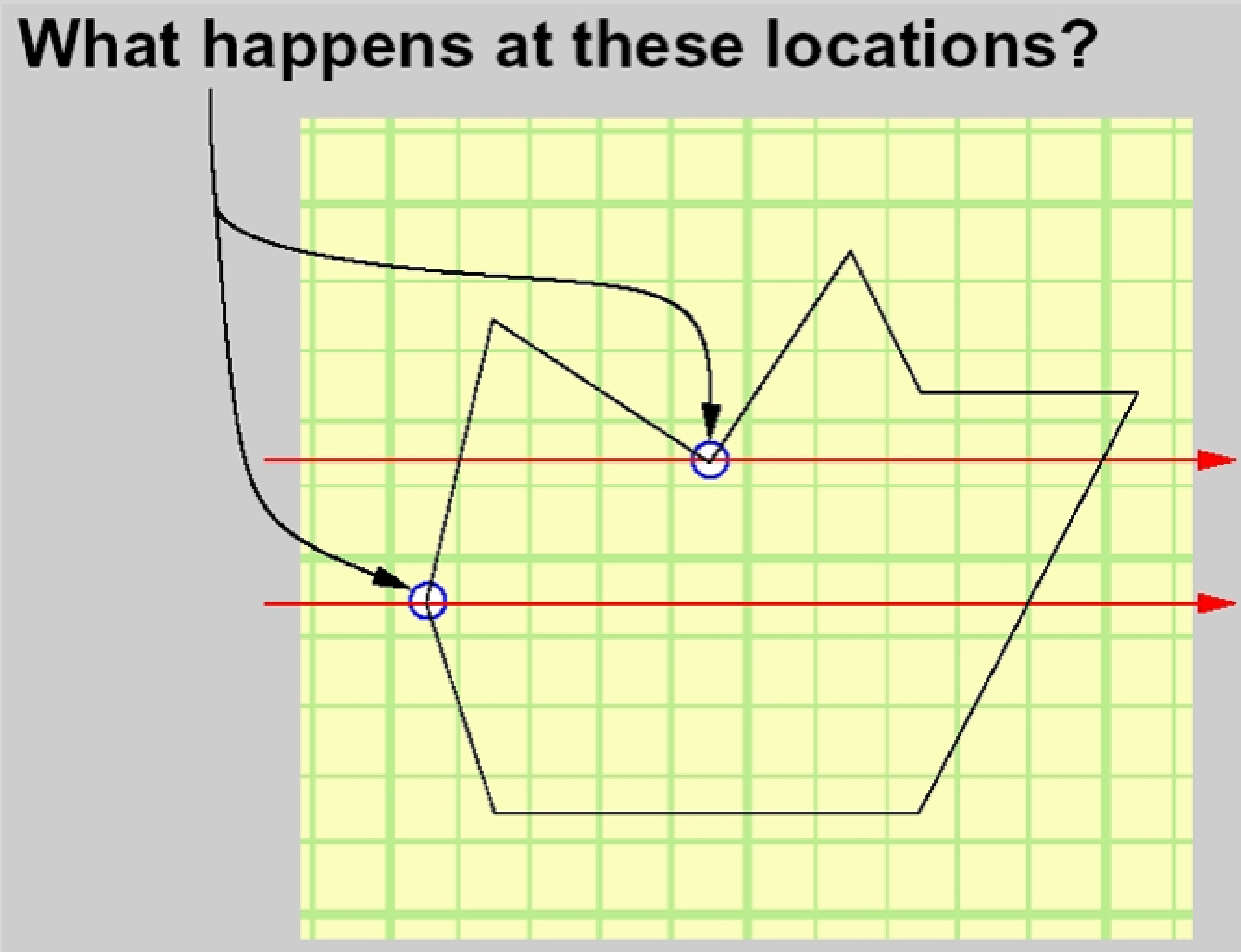
Filled Polygons



Filled Polygons

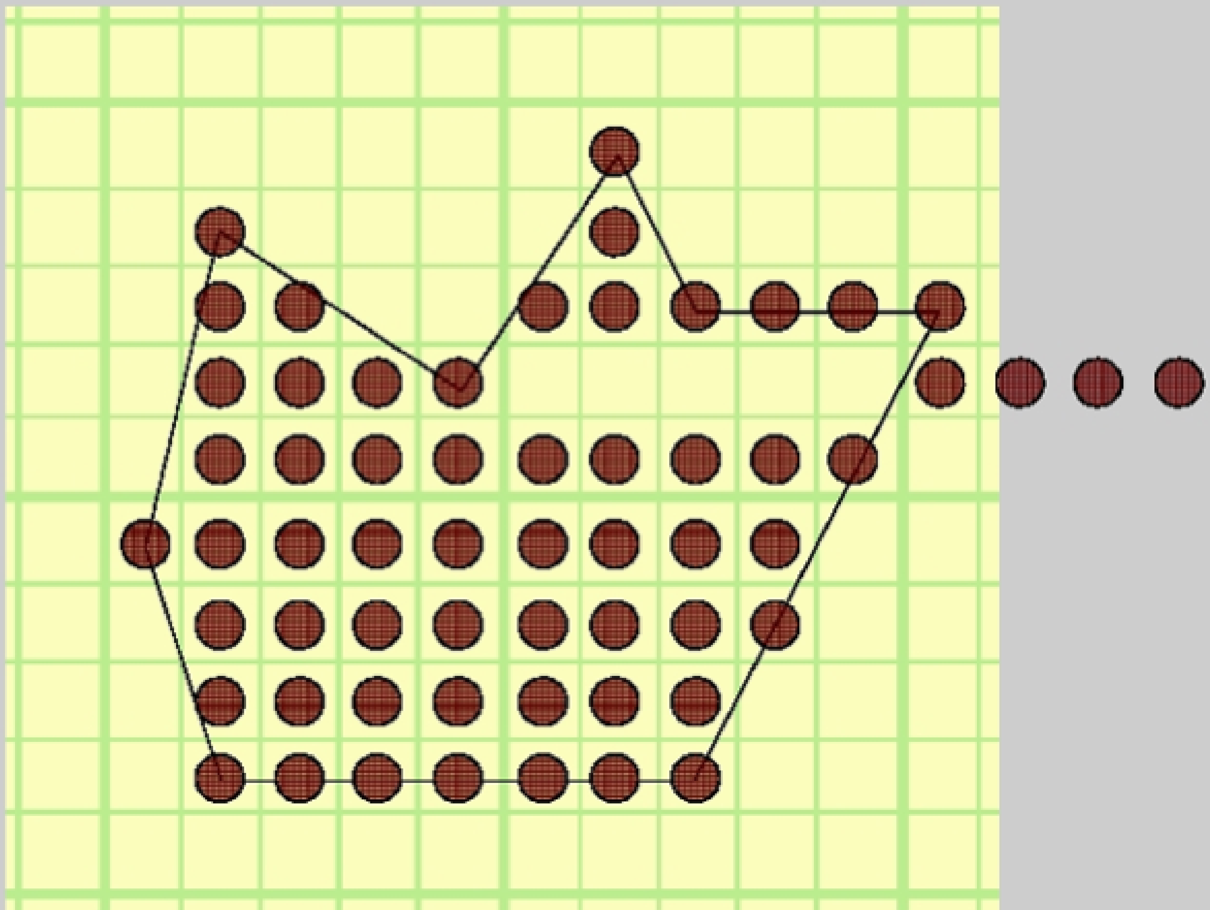


Filled Polygons

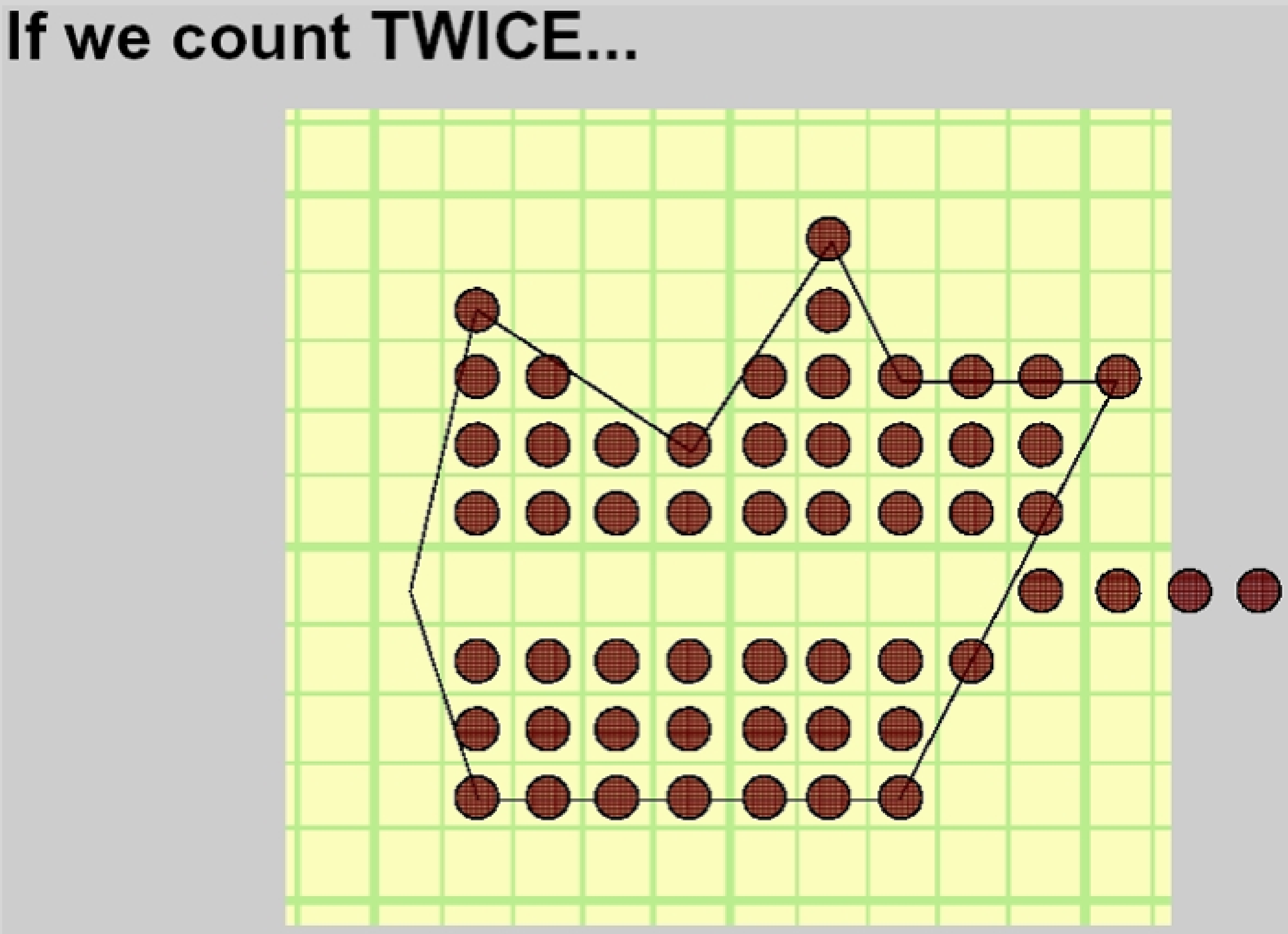


Filled Polygons

If we count ONCE...

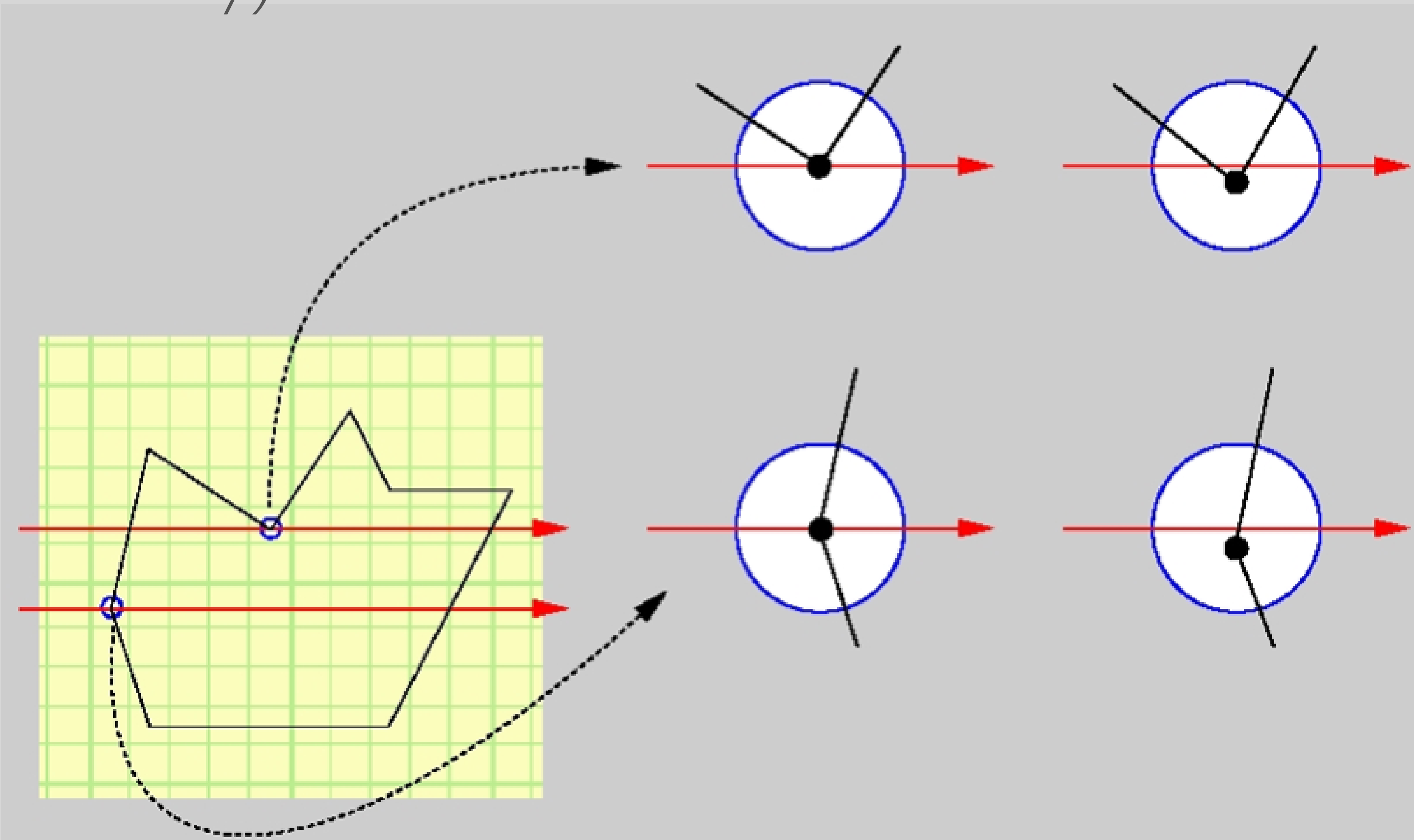


Filled Polygons



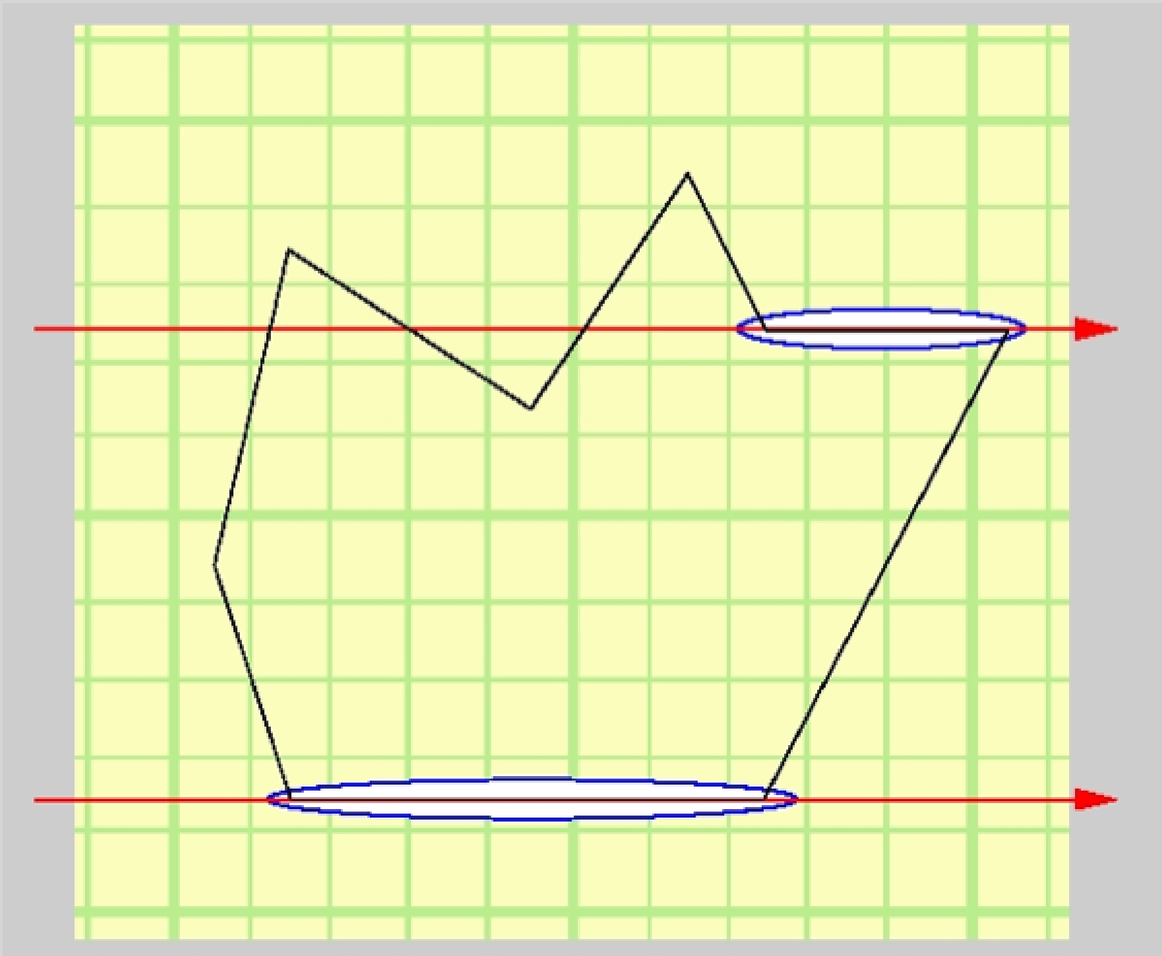
Filled Polygons

Treat (scan y = vertex y) as (scan y > vertex y)



Filled Polygons

Horizontal edges

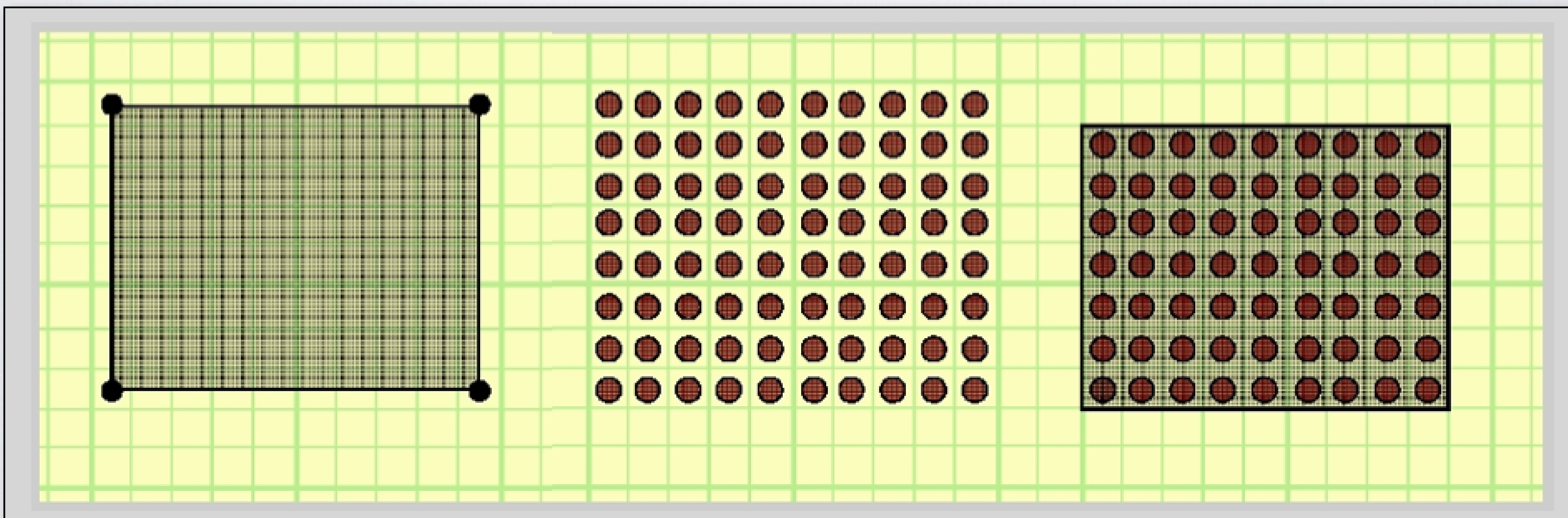


Filled Polygons

Horizontal edges

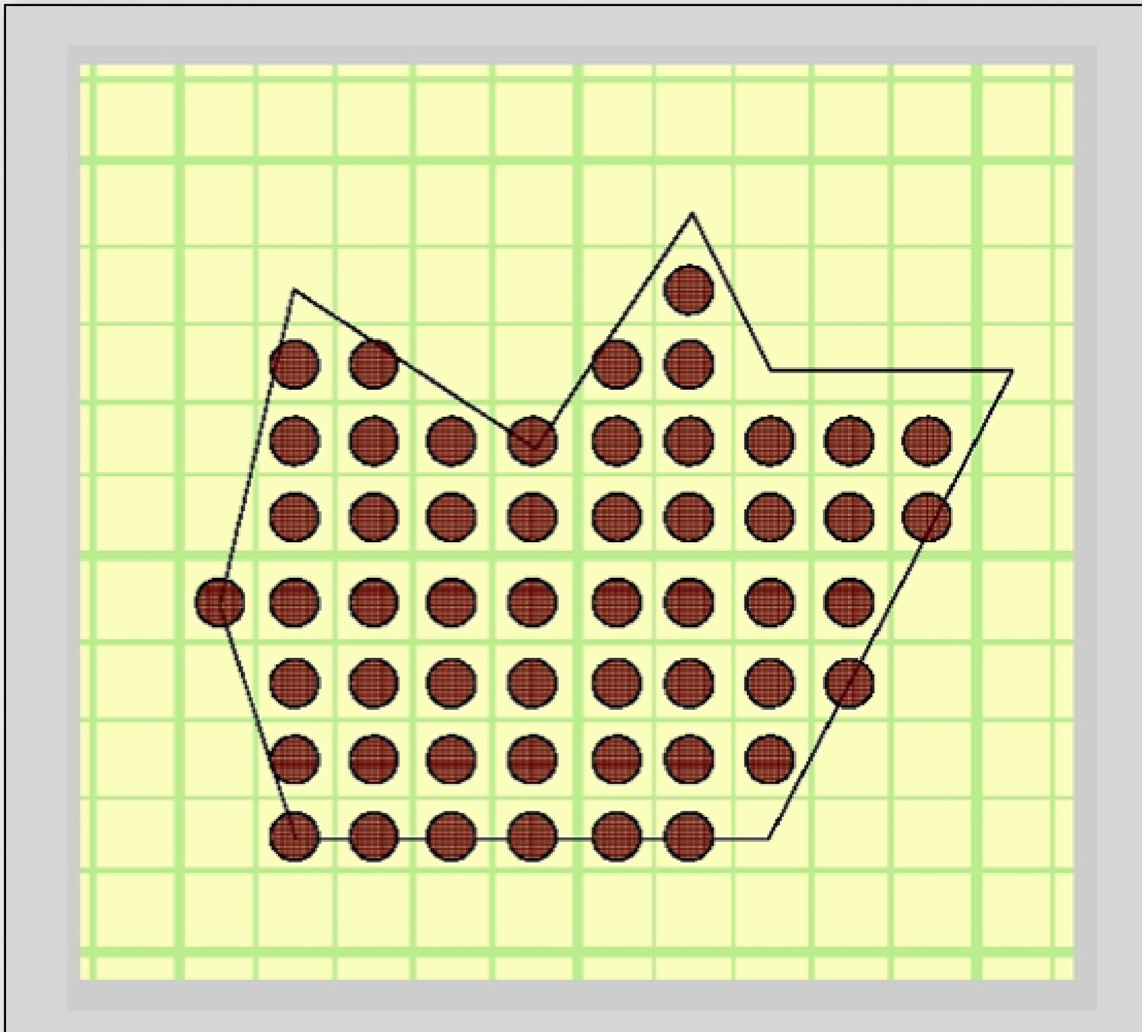
Filled Polygons

- "Equality Removal" applies to all vertices
- Both x and y coordinates



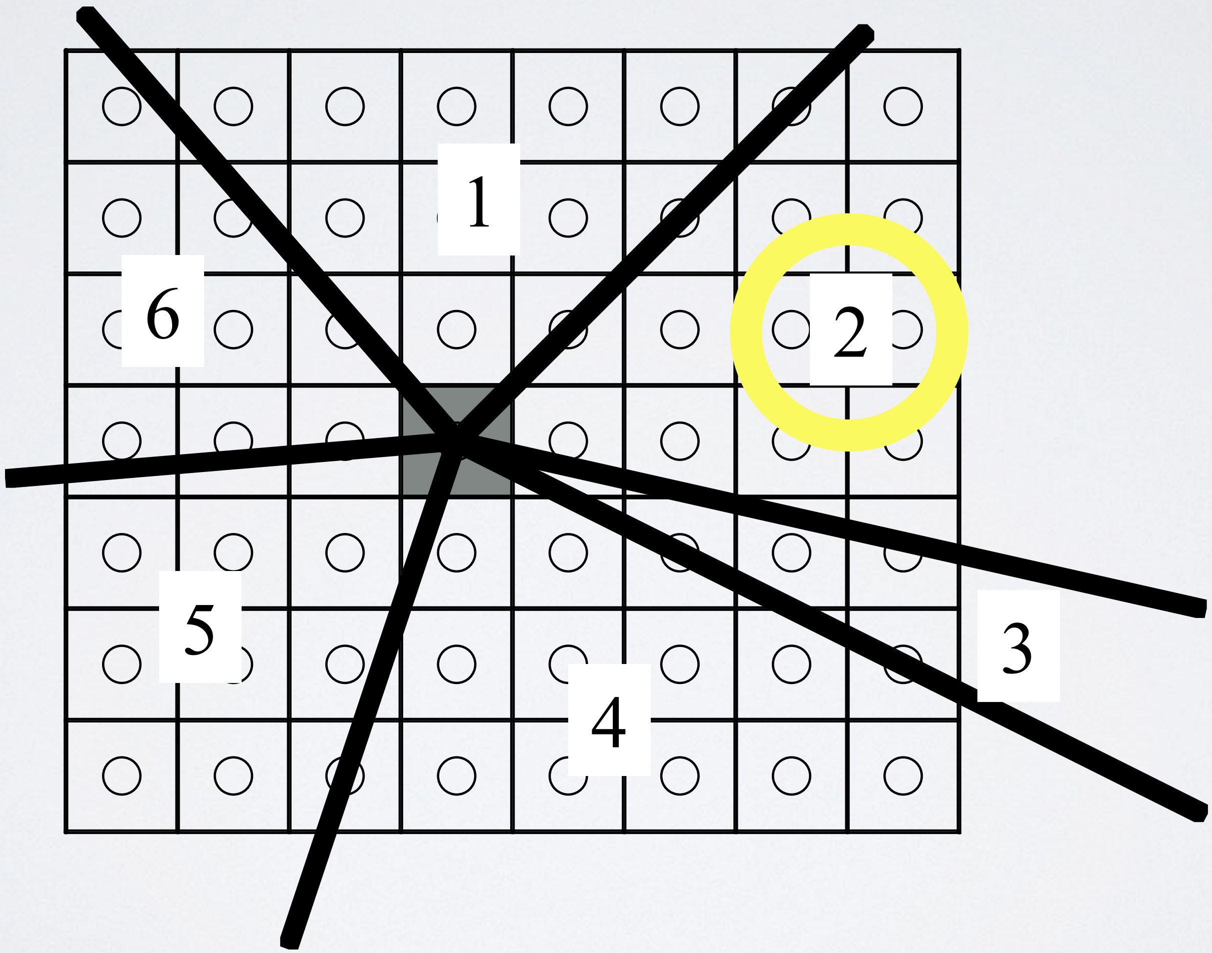
Filled Polygons

- Final result:



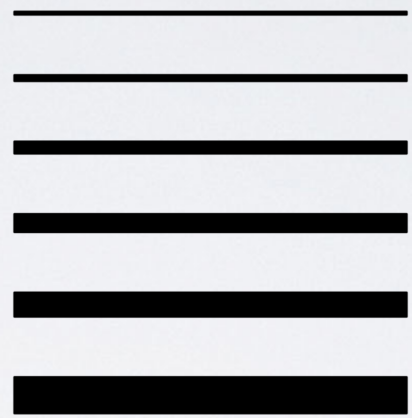
Filled Polygons

- Who does this pixel belong to?



Drawing a Line

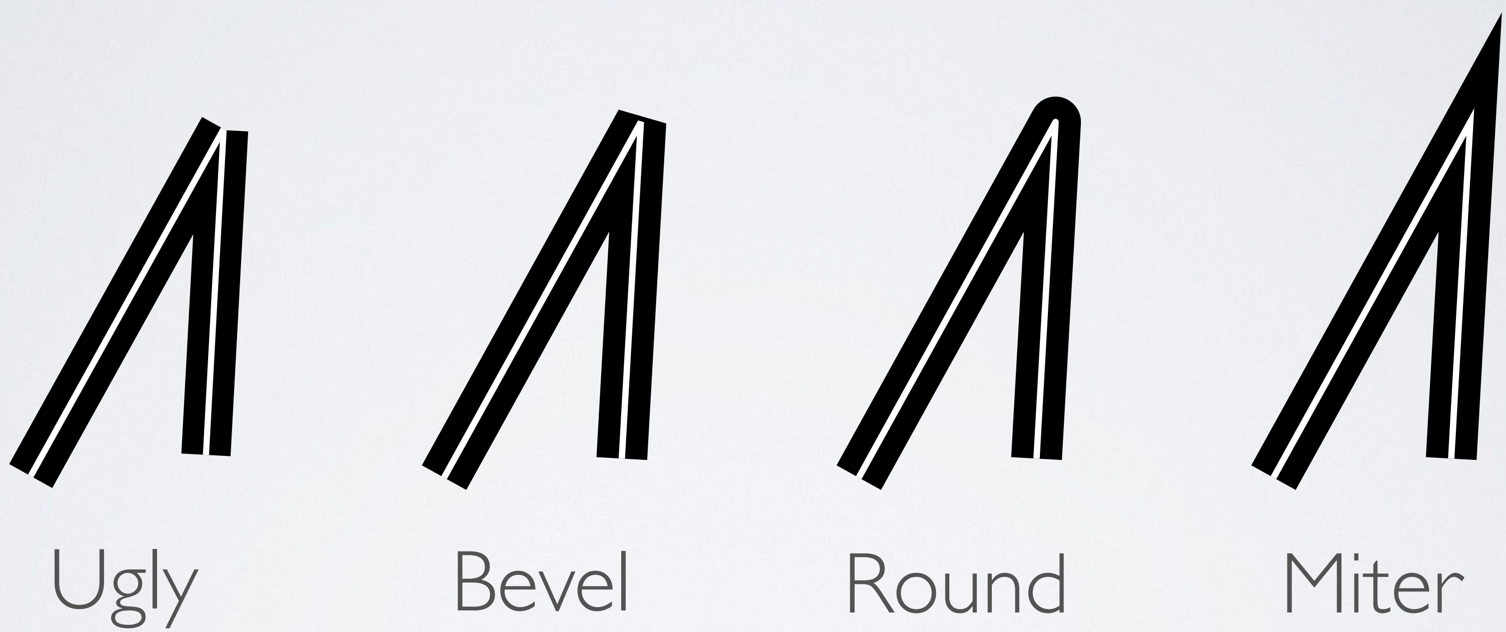
- How thick?



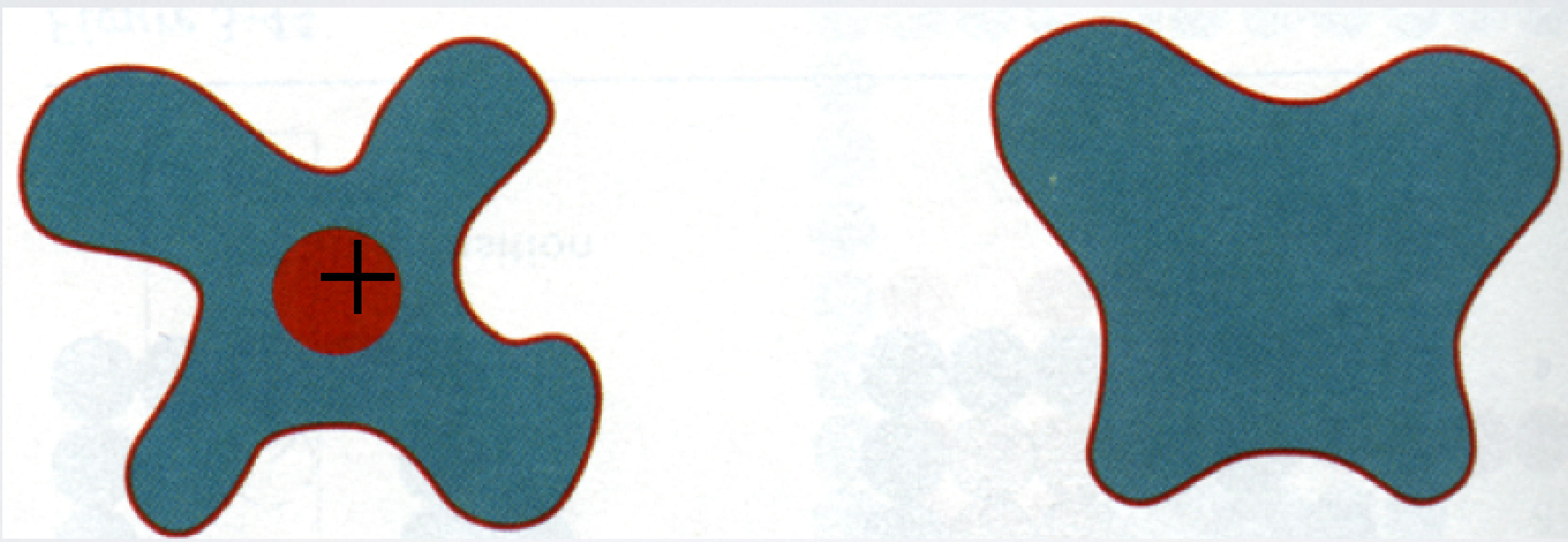
- Ends?

Drawing a Line

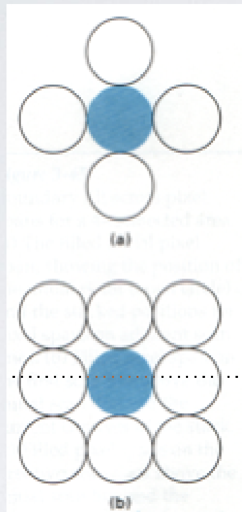
- Joining?

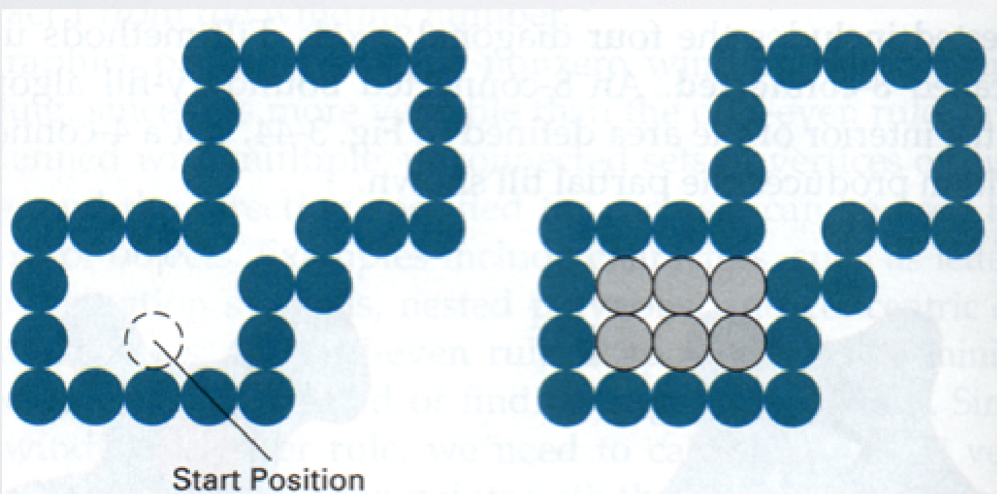


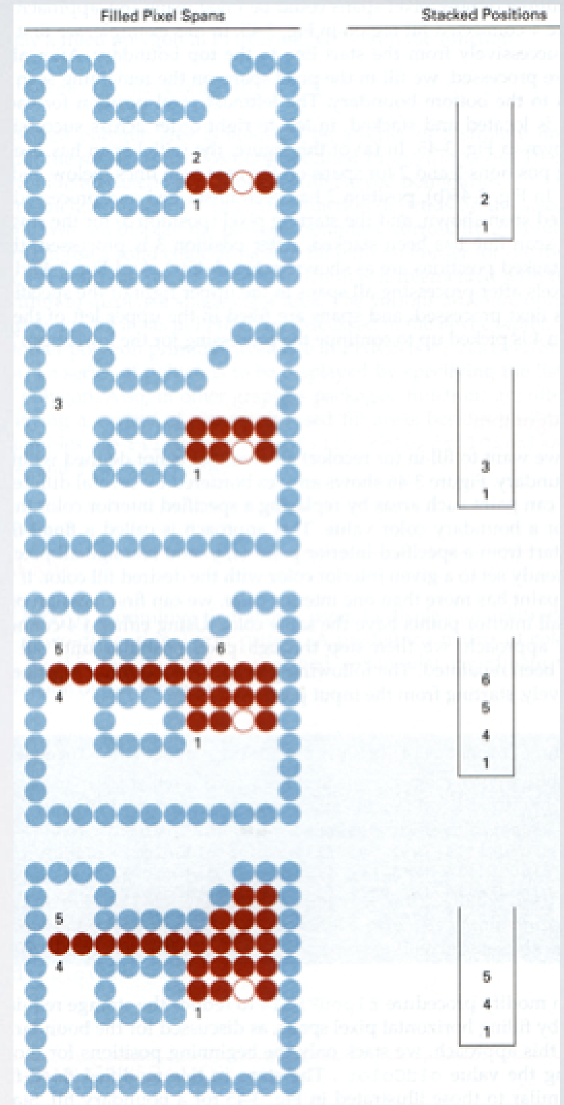
Flood Fill



Flood Fill

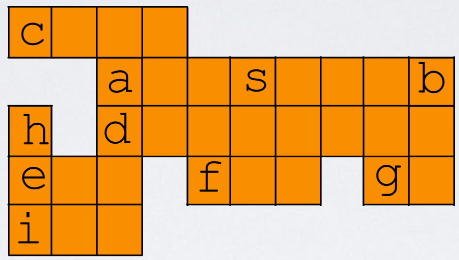






Span-Based Algorithm

Definition: a ***run*** is a horizontal span of identically colored pixels



1. Start at pixel “s”, the seed.
2. Find the run containing “s” (“b” to “a”).
3. Fill that run with the new color.
4. Search every pixel above run, looking for pixels of interior color
5. For each one found,
6. Find left side of that run (“c”), and push that on a stack.
7. Repeat lines 4-7 for the pixels below (“d”).
8. Pop stack and repeat procedure with the new seed

The algorithm finds runs ending at “e”, “f”, “g”, “h”, and “i”