Discussion Session 7

Sikun Lin

sikun@ucsb.edu

Last time we covered …

- Rendering Pipeline
  - Modeling matrices
  - Viewing matrix
  - Projection matrix (&transform into canonical view volume)
  - Polygon clipping
  - Depth test

Today’s Topic

- How to draw things onto pixels
  - Line Drawing
  - Scan conversion

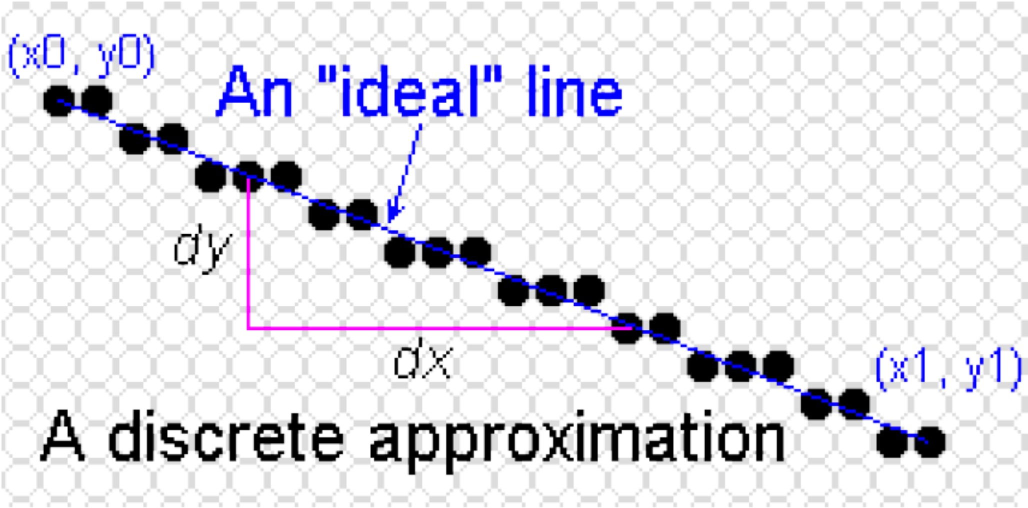
  (both with codes samples)

Line Drawing

- Bresenham’s Line Algorithm
- Bresenham’s Circle Algorithm

Bresenham DDA Algorithm

- Developed by Jack E. Bresenham at IBM
- Ran on Calcomp plotter
- Based on the idea of Digital Difference Analyzer



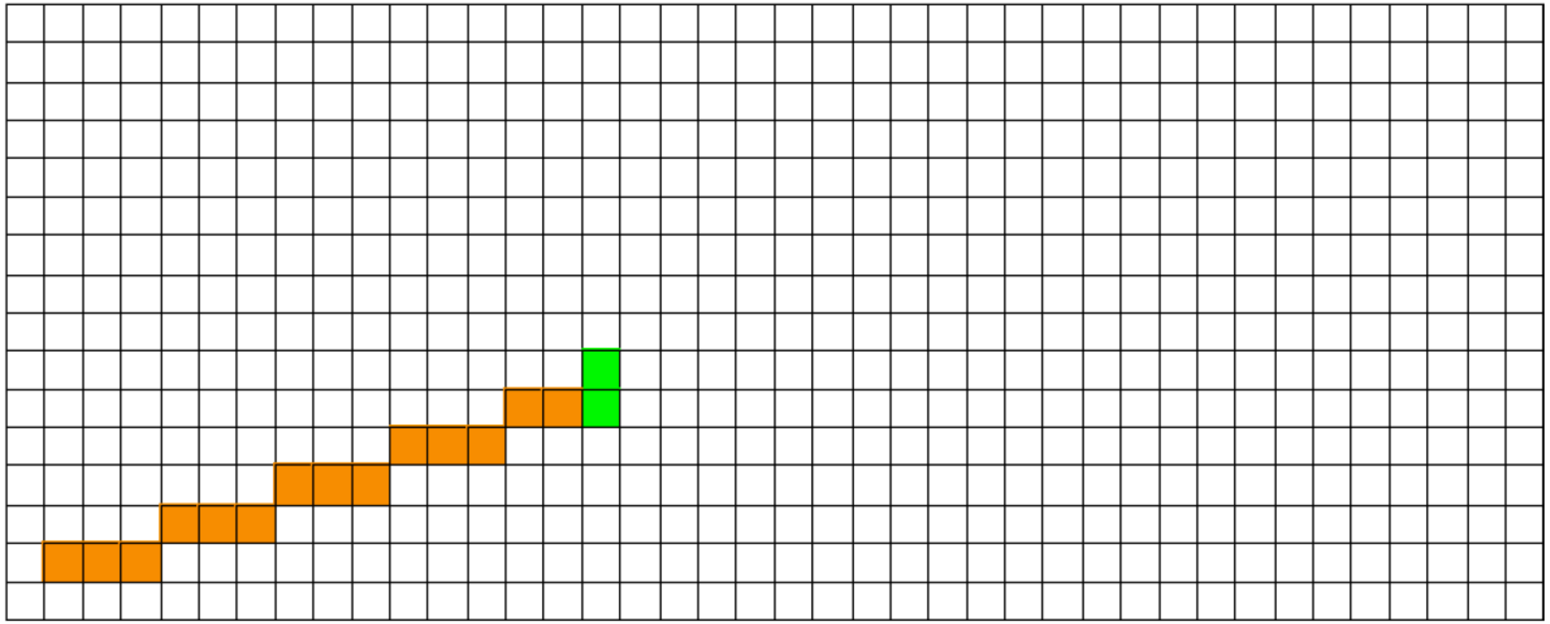
A DDA Line Drawing Function

```
Line(int x1, int y1, int x2, int y2){
   int dx = x1 - x2, dy = y2 - y1;
   int n = max(abs(dx),abs(dy));
   float dt = n, dxdt = dx/dt, dydt = dy/dt;
   float x = x1, y = y1;
   while (n--)
   {
      DrawPoint( round(x), round(y) );
      x += dxdt;
      y += dydt;
   }
}
```

What's bad about this?

We can do better!

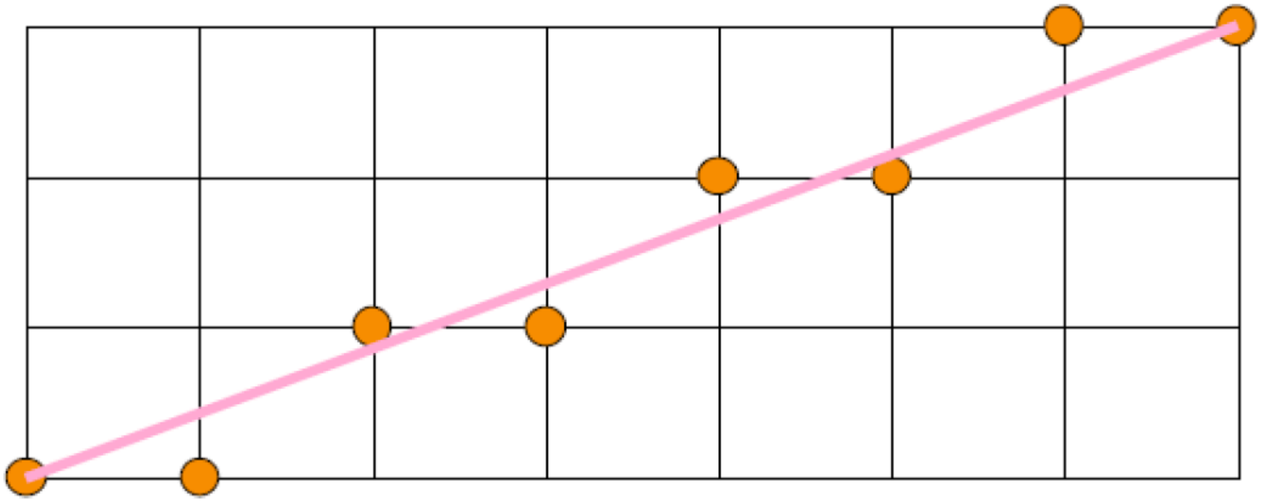
- Get rid of floating point operations
- The idea: chose the rights pixels from those next to the current pixel
  - Assume: dx > dy > 0



- Which of the green pixels is next?

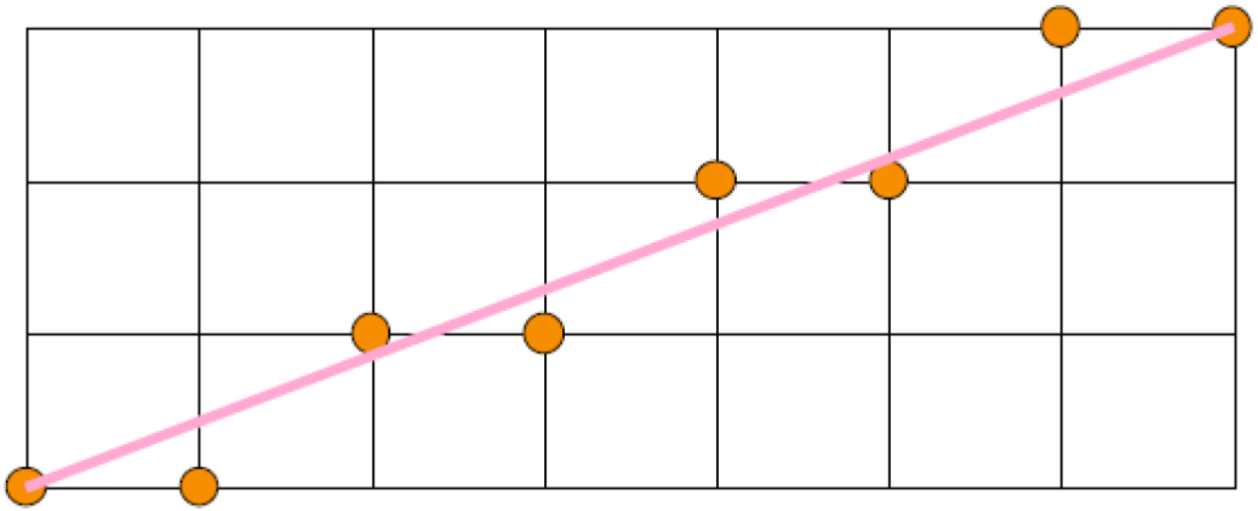
Key Idea

- We only ever go: *right* one pixel, or *up and right* one pixel (0 < slope < 1). Call these choices "E" and "NE"
- Imagine pixels as "lattice points" on a grid
- Given next X coordinate, we only need to chose between y and y+1. (y is the Y coordinate of the last pixel)

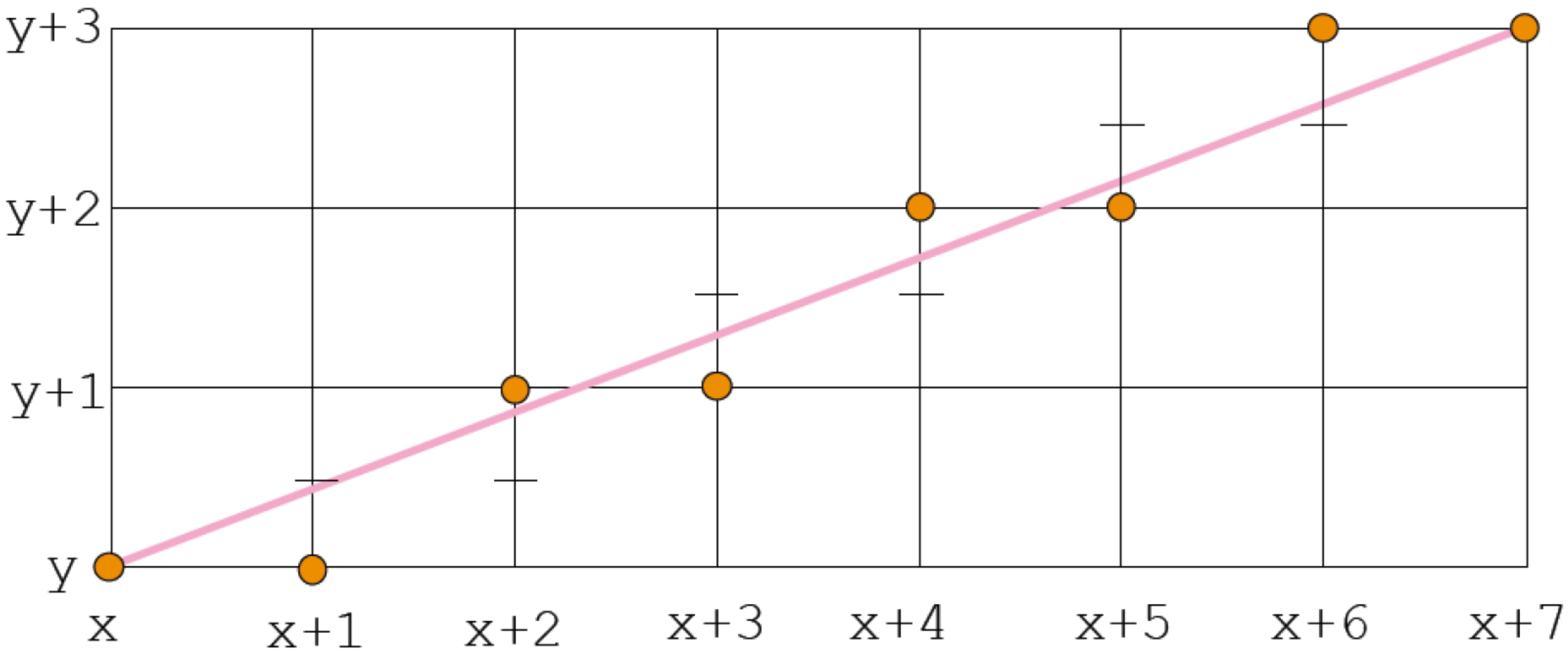


The Midpoint Test

- Look at the vertical grid the line intersects
- On which side of the midpoint ($y+{}^1/_2$) does the intersection lie?
- If it above the midpoint go NE, otherwise go E



Our Example



Implicit Functions

- Normally a line is defined as $y = mx + b$
- Instead, define $F(x,y) = ax+by+c$, and let the line be everywhere $F(x,y) = 0$
- Now if $F(x,y)>0$, we're "above" the line, and if $F(x,y) < 0$, we're "below" the line

So what?

- We can evaluate the implicit line function at the midpoint to determine what to draw next!
- Even better, we can use the last function evaluation to find the next one cheaply!
- For any *x,y*:

$$F(x+1,y) - F(x,y) = a$$

$$F(x+1,y+1) - F(x,y) = a+b$$

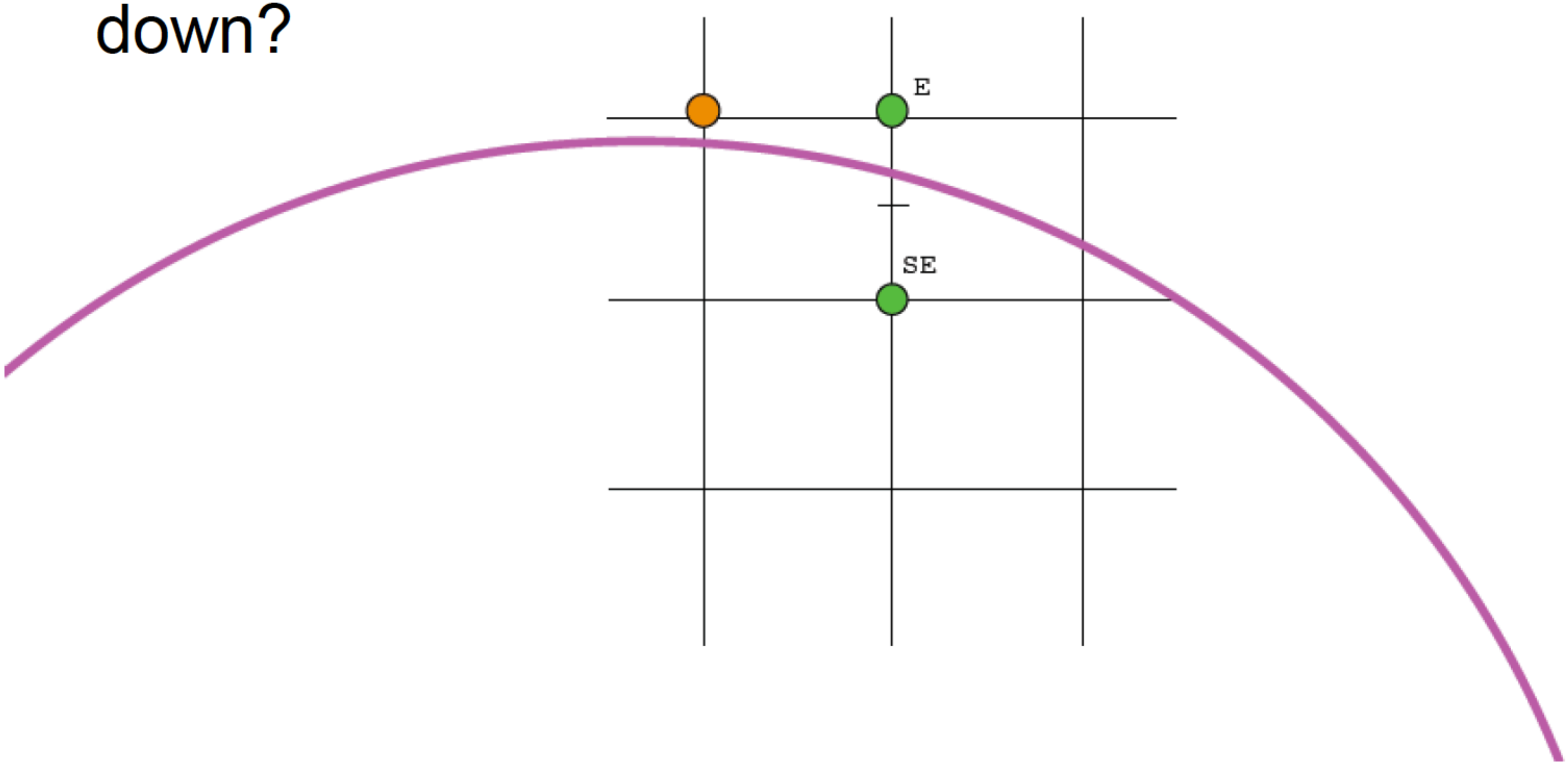
Midpoint Algorithm

```
Line(int x1, int y1, int x2, int y2){

   int dx = x2 - x1, dy = y2 - y1;
   int e = 2*dy - dx;
   int incrE = 2*dy;
   int incrNE = 2*(dy-dx);
   int x = x1, y = y1;

   DrawPoint( x, y );
   while (x < x2){
      x++;
      if ( e <= 0 ) { e += incrE; }
      else { y++; e += incrNE; }
      DrawPoint( x, y );
   }
}
```

- “e” is the implicit function evaluation at each x value (actually multiplied by 2, but we only need sign)
- Easily extended for lines with arbitrary slopes

Midpoint Algorithm for Circles

- Only consider the second octant (others are symmetric)
- Midpoint test still works: do we go right, or right and down?



More Midpoint Circle Drawing

- The circle's implicit function (with radius r):

$$F(x,y) = x^2 + y^2 - r^2$$

- Once we know the value of the implicit function at one midpoint we get the value at the next with the same differencing technique:
  - If we're going E: $F(x+2,y) - F(x+1,y) = 2x+3$
  - If we're going SE: $F(x+2,y-1) - F(x+1,y) = 2(x-y)+5$

Drawing the 2nd Ocatant

```
void Circle(int cx, int cy, int radius){
   int x = 0, y = radius, e = 1-radius;
   DrawPoint( x + cx, y + cy );
   while (y > x){
      if (e < 0) // Go "east"
      {
        e += 2*x + 3;
      }
      else // Go "south-east"
      {
        e += 2*(x-y) + 5;
        y--;
      }
      x++;
      DrawPoint( x + cx, y + cy );
   }
}
```

- Lots of "expensive" multiplies! Can we get rid of them?
- Perform finite difference in the "e" variable itself
  - If we go E: $e_{new} - e_{old} = 2$
  - If we go SE: $e_{new} - e_{old} = 4$

Final Circle Drawer

```
void Circle(int cx, int cy, int radius){
   int x = 0, y = radius, e = 1-radius;
   int incrE = 3, incrSE = -2*radius + 5
   DrawPoint( x + cx, y + cy );
   while (y > x){
   if (e < 0) // Go “east”
   {
     e += incrE;
     incrE += 2; incrSE += 2;
   } else // Go “south-east”
   {
     e += incrSE;
     incrE += 2; incrSE += 4;
     y--;
   }
   x++;
   DrawPoint( x + cx, y + cy );
   }
}
```

This looks pretty fast!

Scan Conversion

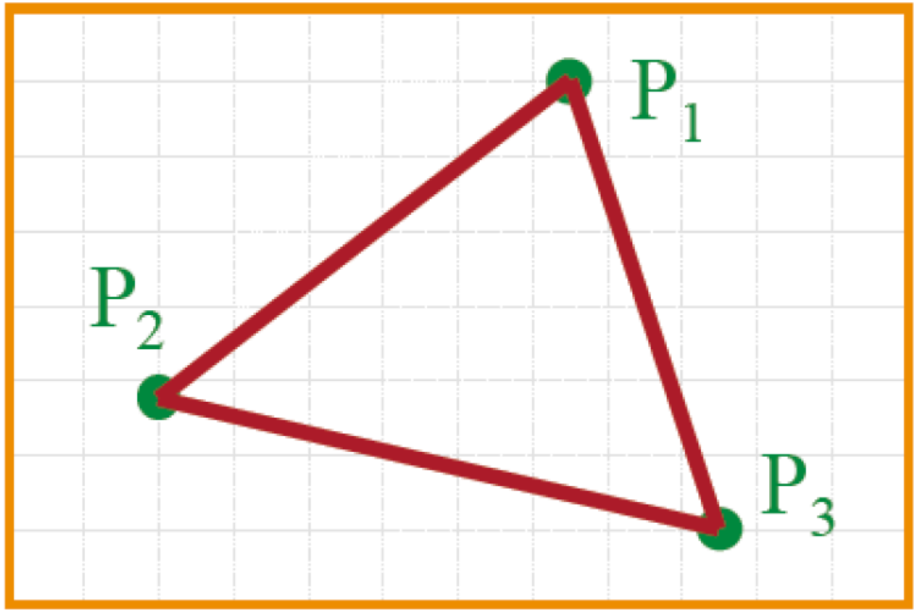
- Figuring out which pixels to turn on

Scan Conversion

- 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

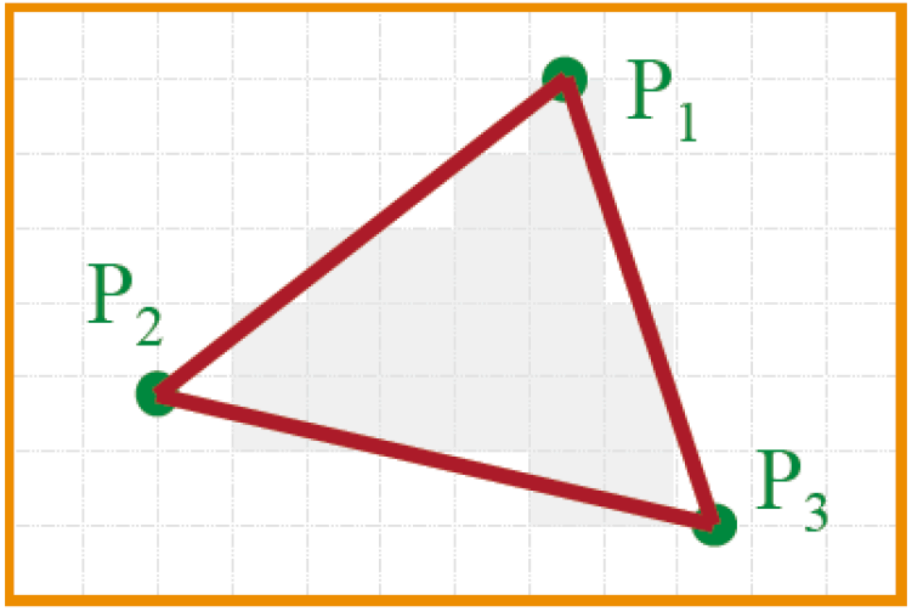


Scan Conversion

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

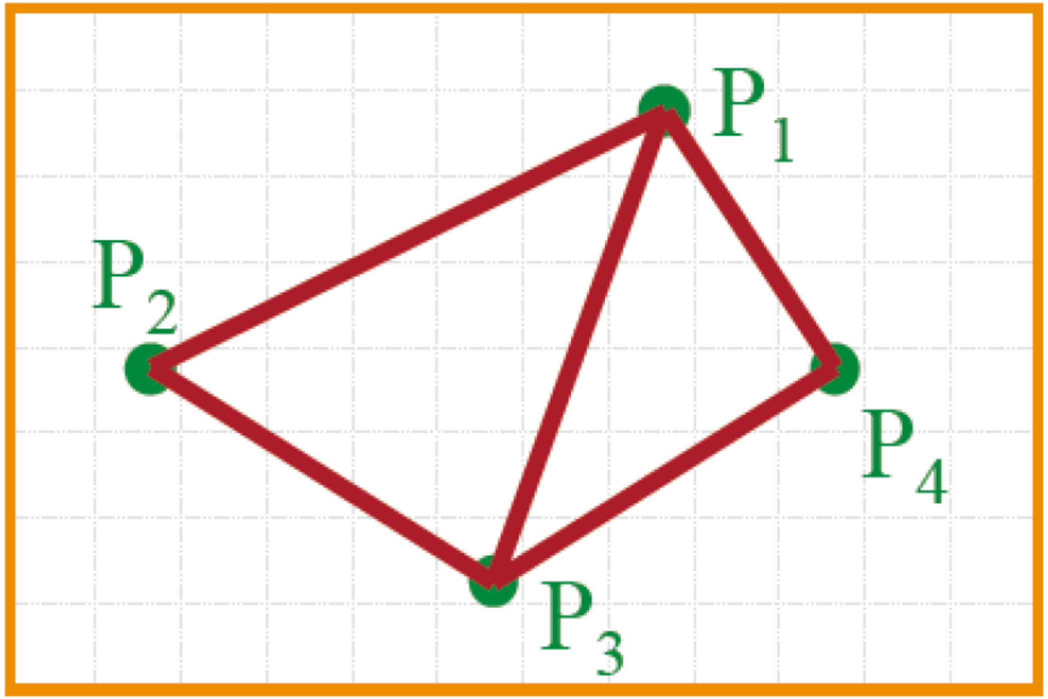
```
void SetPixel(int x, int y, Color rgba)
```

- Example: Filling the inside of a triangle



Triangle Scan Conversion

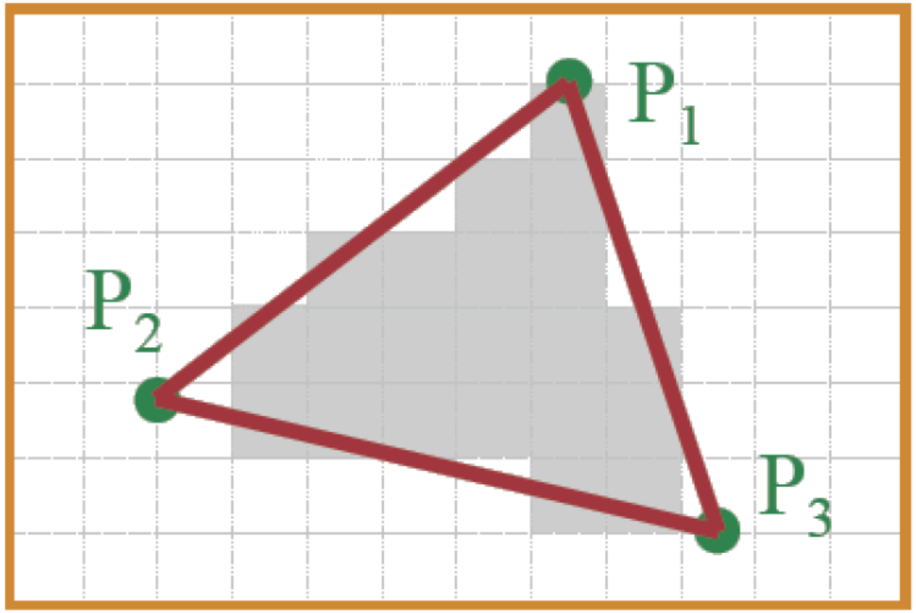
- Properties of a good algorithm
  - Symmetric
  - Straight edges
  - Antialiased edges
  - No cracks between adjacent primitives
  - MUST BE FAST!



Simple Algorithm

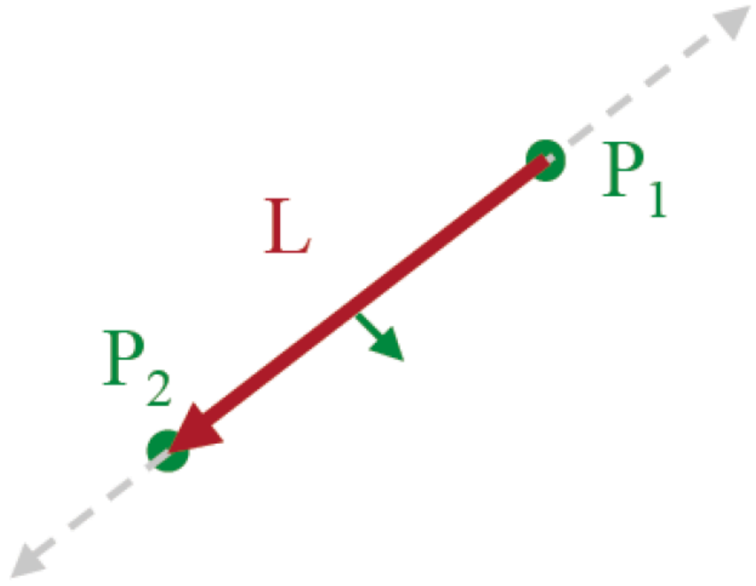
- Color all pixels inside triangle

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



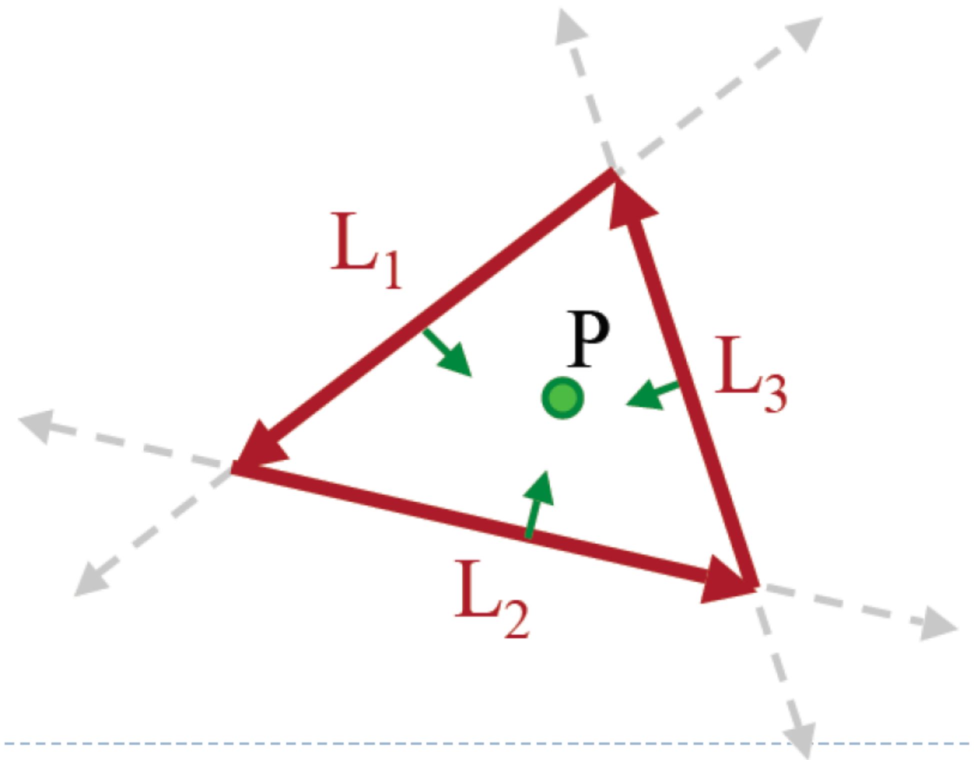
Line defines 2 half spaces

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



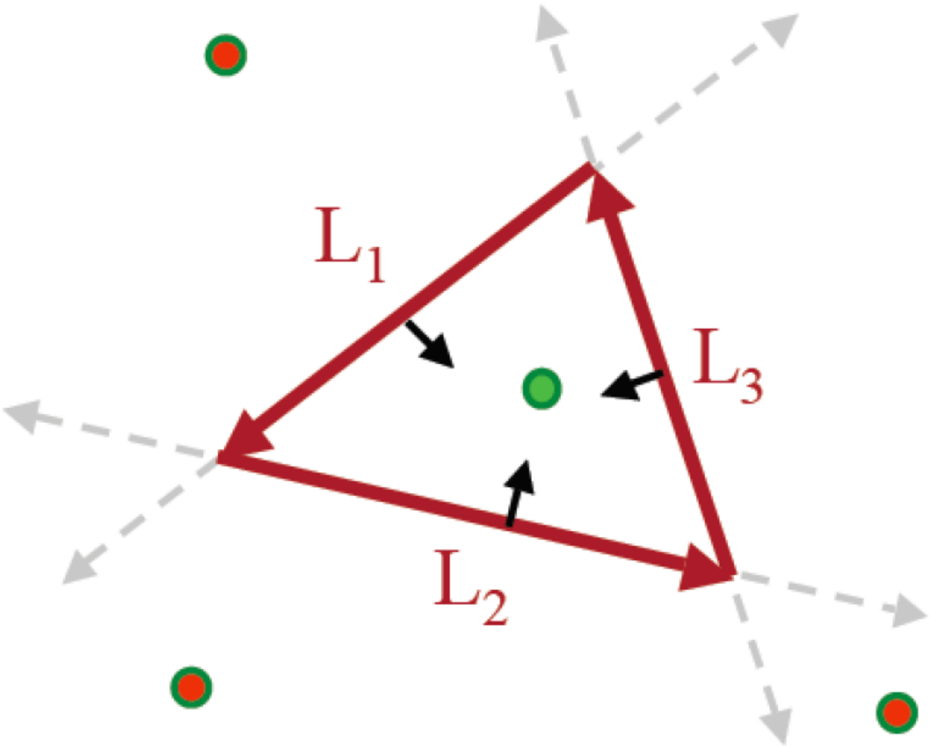
Inside Triangle Test

- A point is inside a triangle if it is in the 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

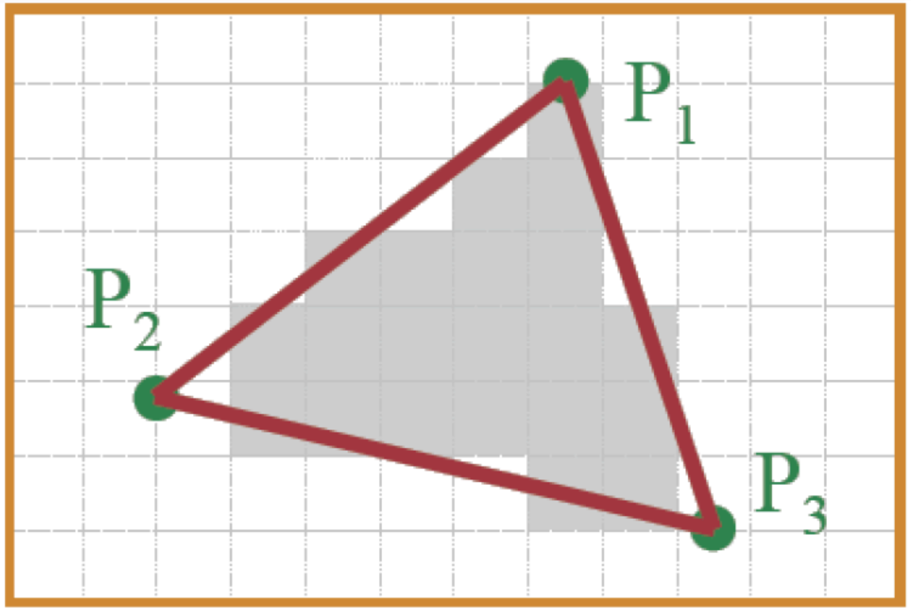
```
Boolean Inside(Triangle T, Point P)
{
   for each boundary line L of T {
      Scalar d = L.a*P.x + L.b*P.y + L.c;
      if (d < 0.0) return FALSE;
   }
   return TRUE;
}
```



Simple Algorithm

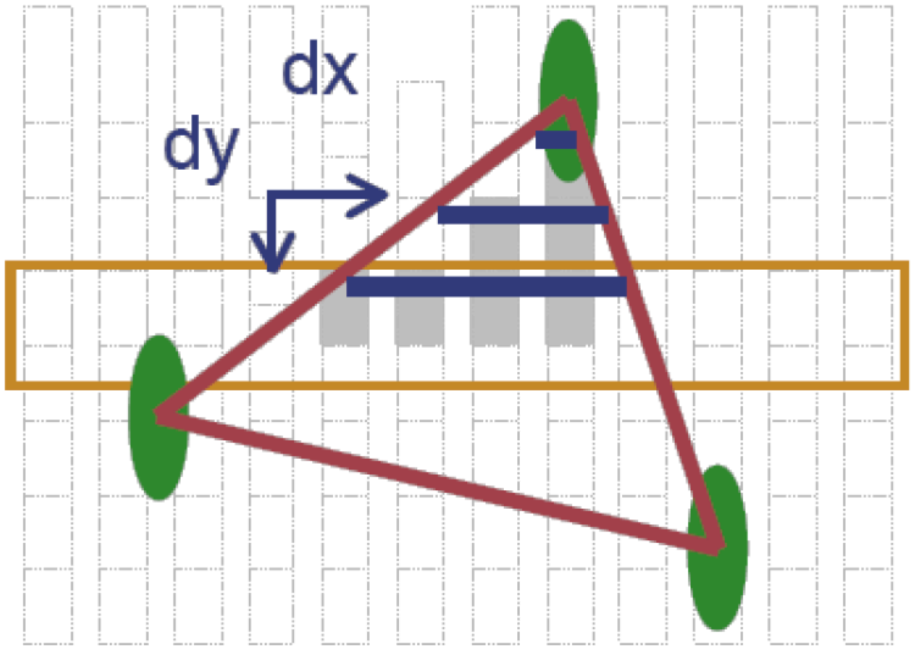
- What's bad about this algorithm?

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



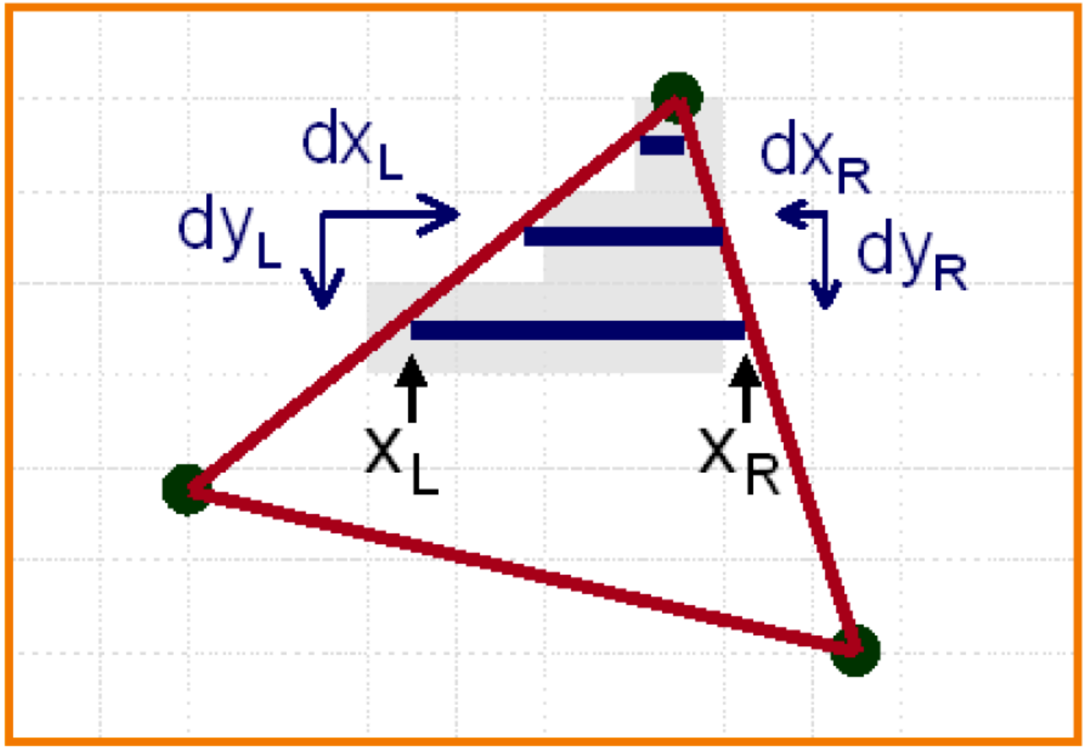
Triangle Sweep-Line Algorithm

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



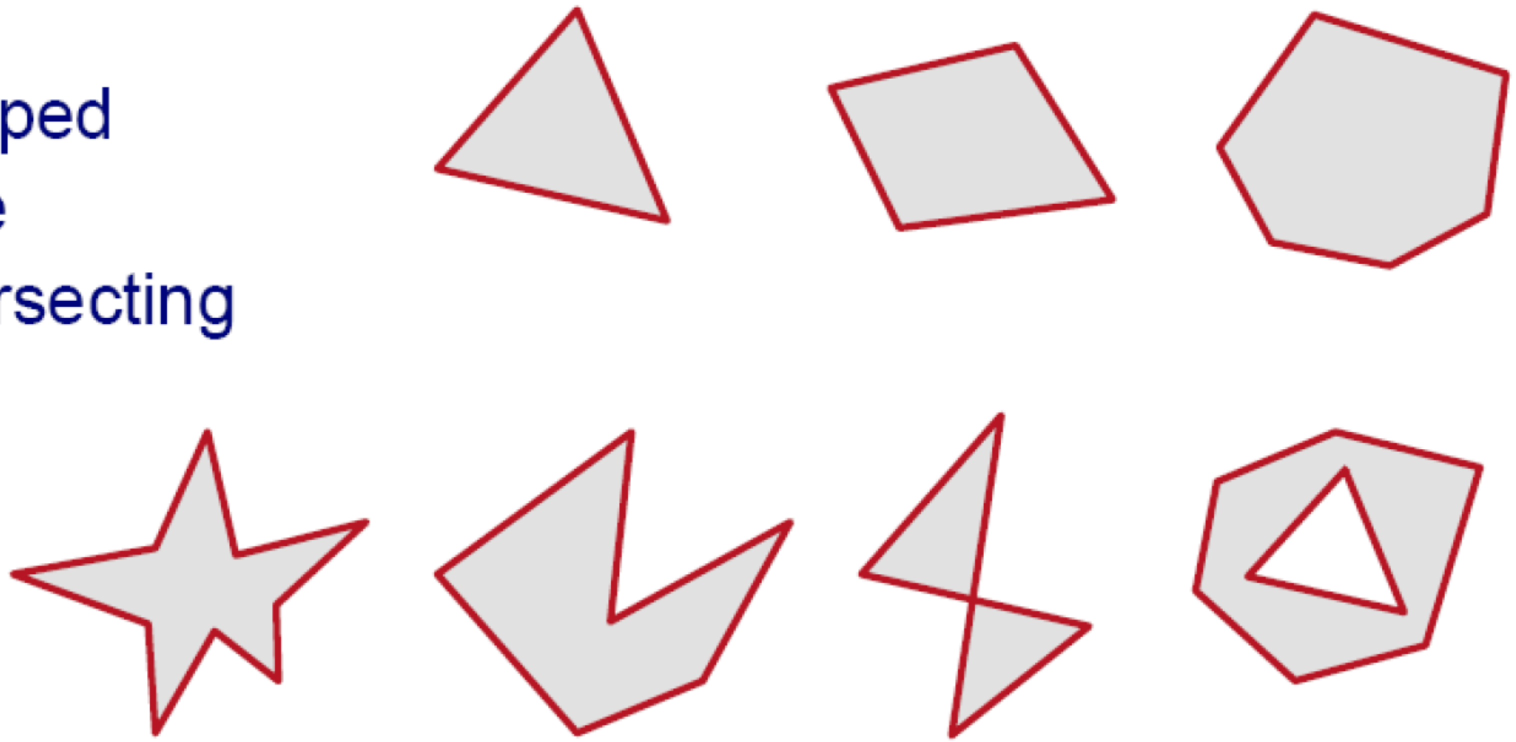
Triangle Sweep-Line Algorithm

```
void ScanTriangle(Triangle T, Color rgba){
    for each edge pair {
        initialize xL, xR;
        compute dxL/dyL and dxR/dyR;
        for each scanline at y
        for (int x = xL; x <= xR; x++)
                SetPixel(x, y, rgba);
        xL += dxL/dyL;
        xR += dxR/dyR;
    }
}
```



Polygon Scan Conversion

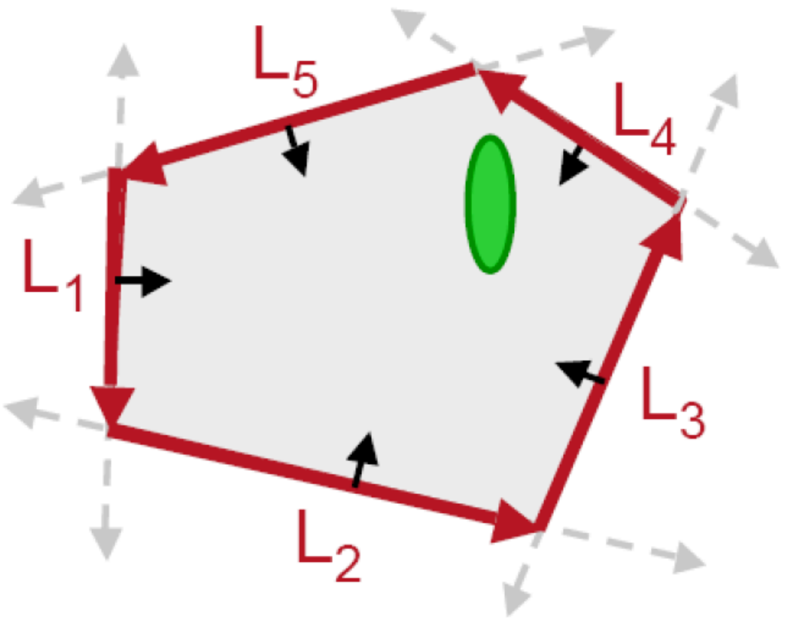
- Fill pixels inside a polygon
  - Triangle
  - Quadrilateral
  - Convex
  - Star-shaped
  - Concave
  - Self-intersecting
  - Holes



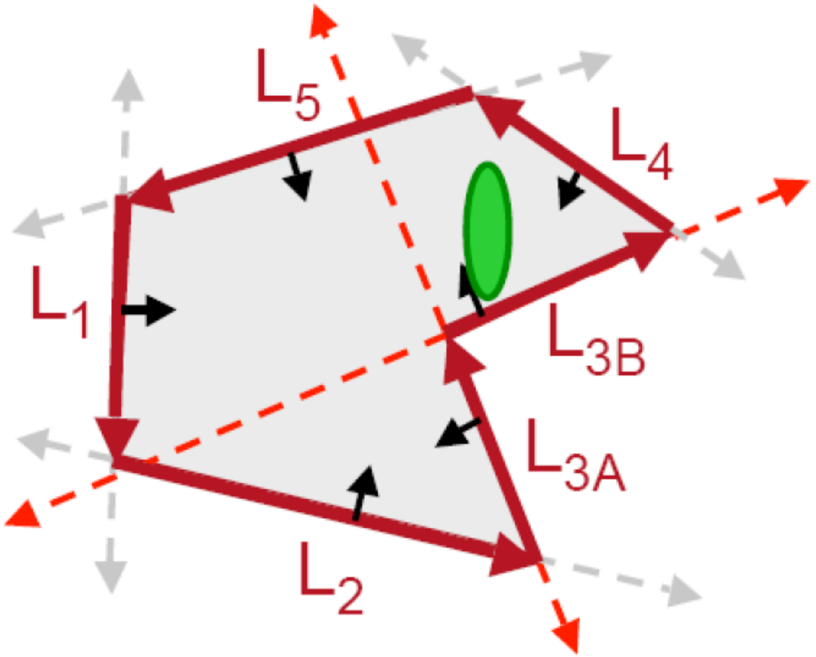
What problems do we encounter with arbitrary polygons?

Polygon Scan Conversion

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



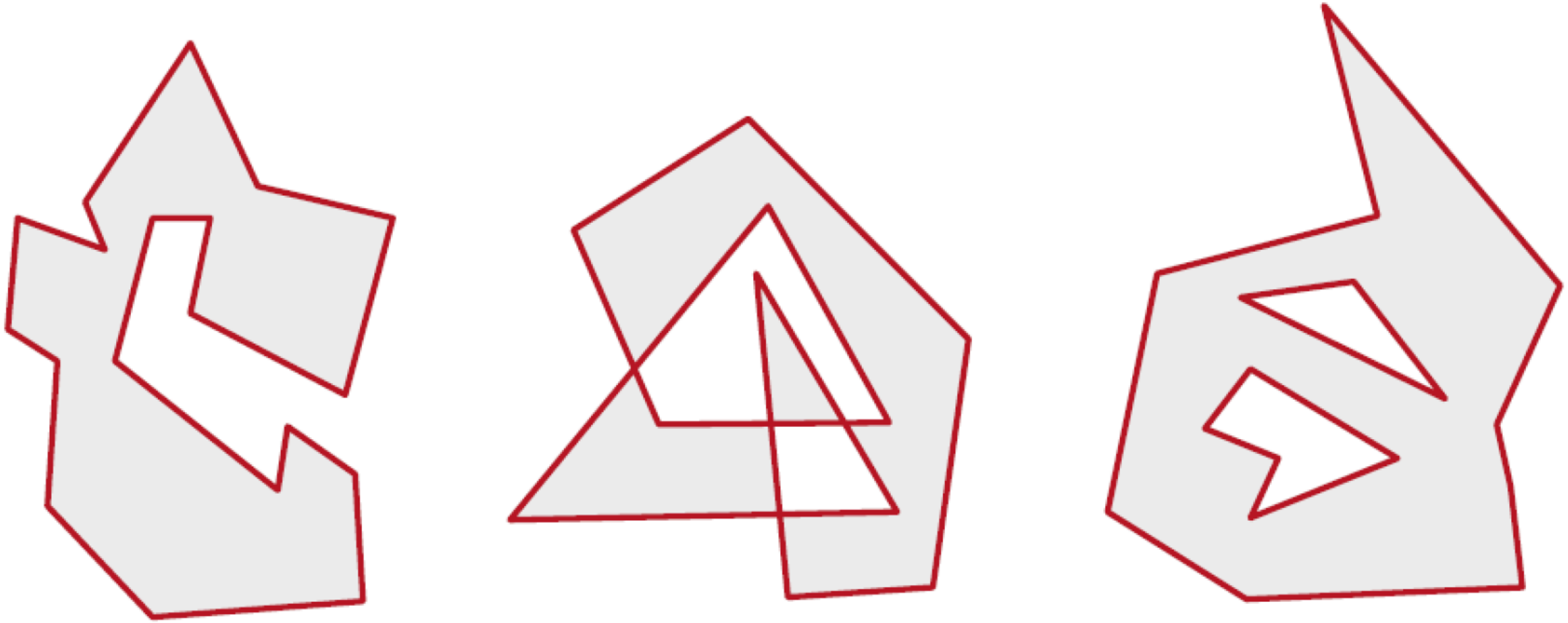
Convex Polygon



Concave Polygon

Inside Polygon Rule

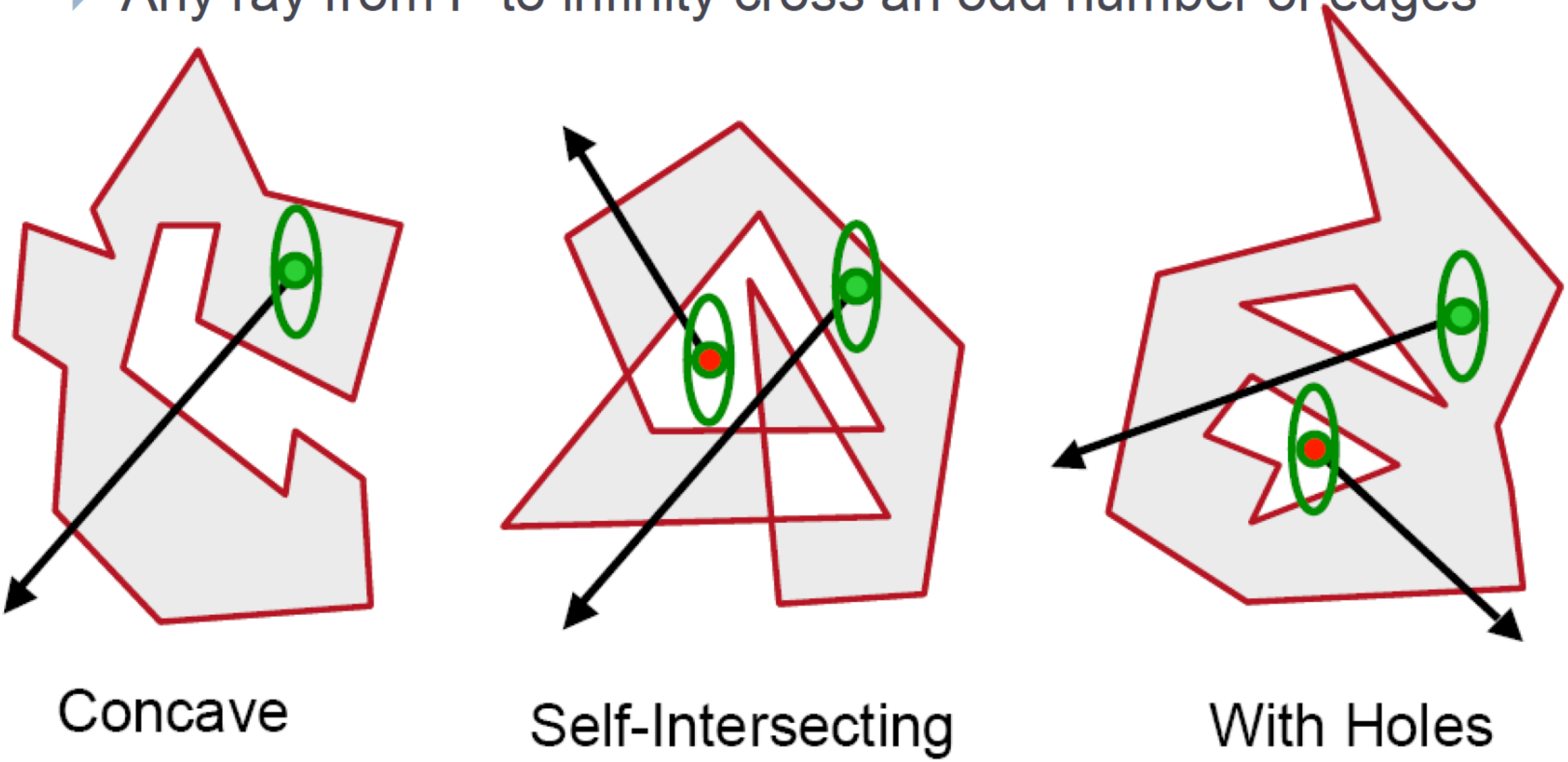
- What's a good rule for which pixels are inside?



Concave Self-Intersecting With Holes

Inside Polygon Rule

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

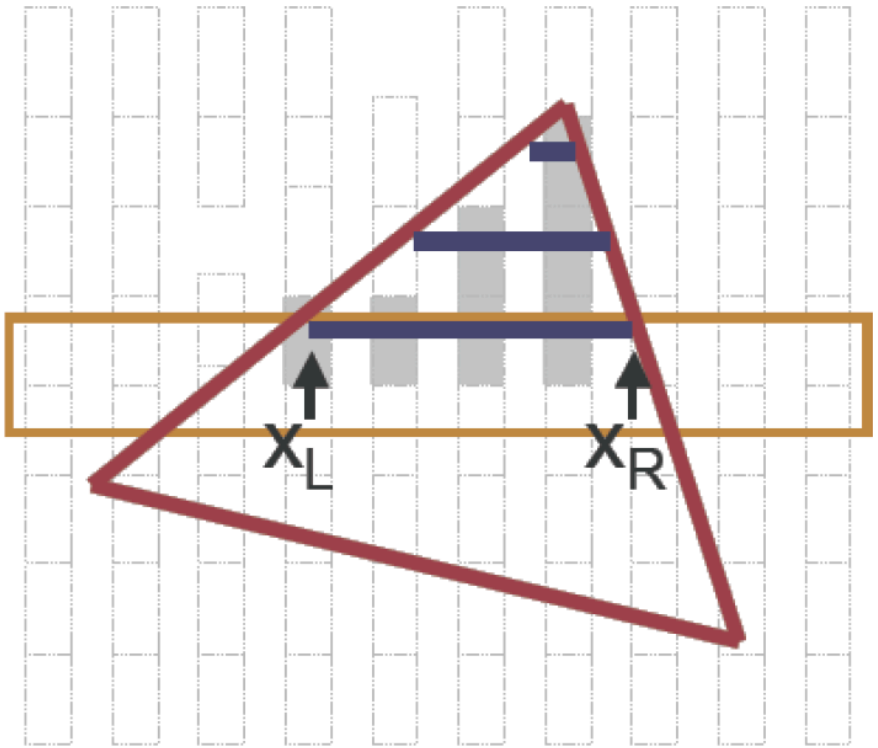


That’s what we discussed last time!

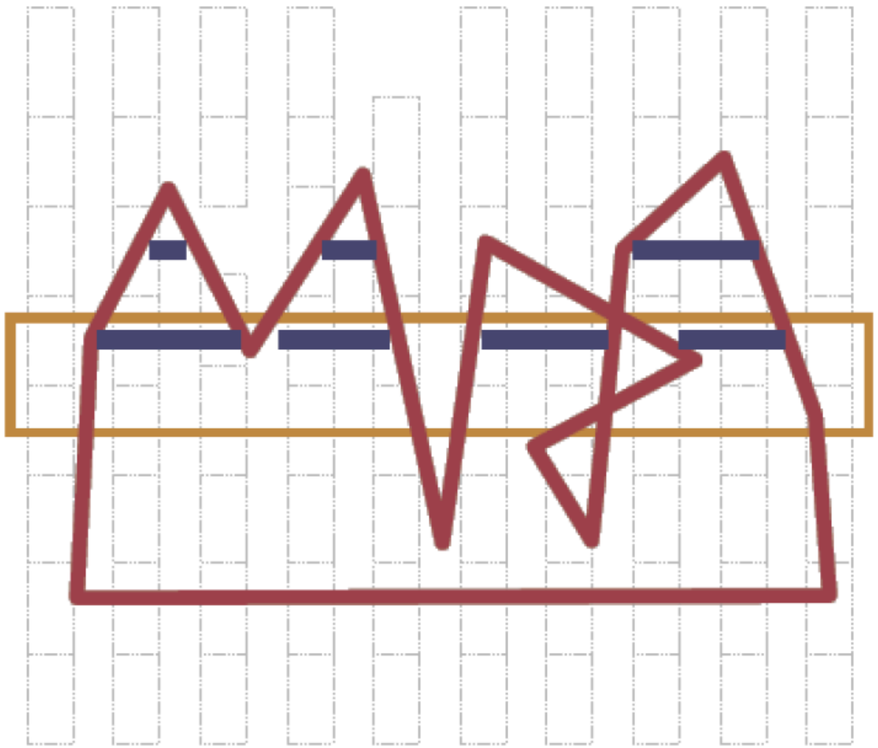
- Scanline algorithm

Polygon Sweep-Line Algorithm

- Incremental algorithm to find spans, and determine insideness with odd parity rule
  - Takes advantage of scanline coherence



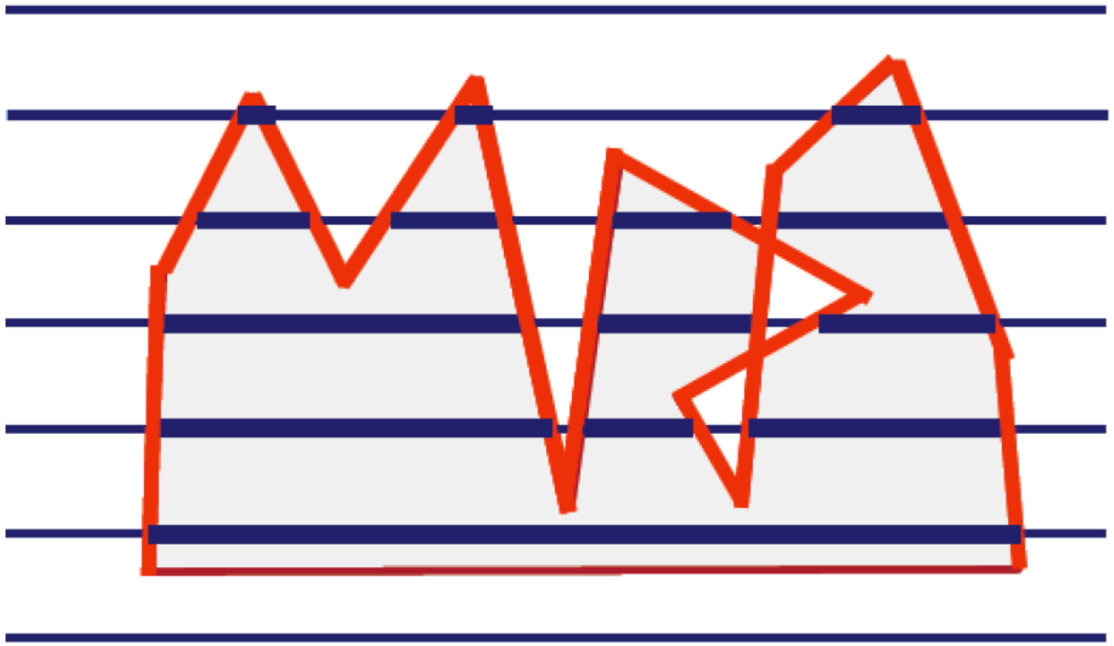
Triangle



Polygon

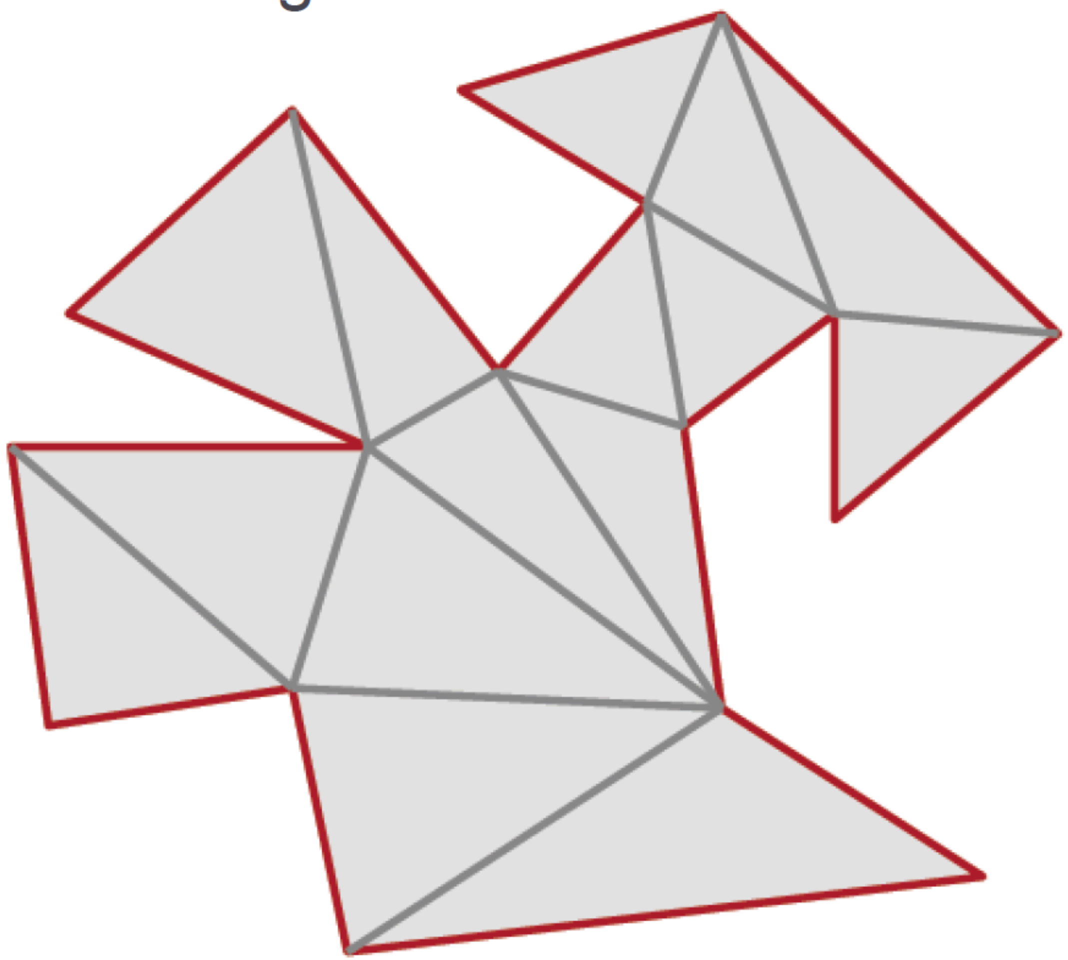
Polygon Sweep-Line Algorithm

```
void ScanPolygon(Triangle T, Color rgba){
   sort edges by maxy
   make empty "active edge list"
   for each scanline (top-to-bottom) {
      insert/remove edges from "active edge list"
      update x coordinate of every active edge
      sort active edges by x coordinate
      for each pair of active edges (left-to-right)
         SetPixels(x_i, x_i+1, y, rgba);
      }
}
```



Hardware Scan Convert

- Turn *everything* into triangles!
  - Scan convert Triangles



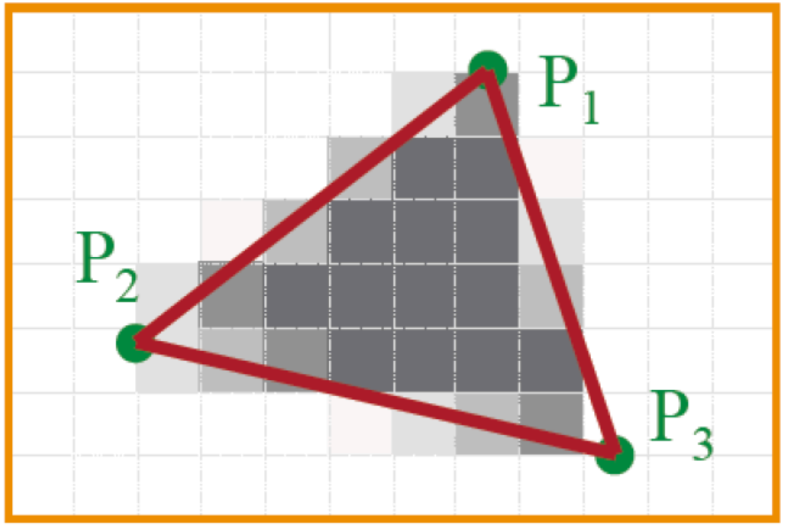
Hardware Antialiasing

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

Pixel with sampling positions

Sampled colours

Average = displayed colour



Q&A

ref: http://comp575.web.unc.edu/files/2010/10/14_ScanConversion.pdf