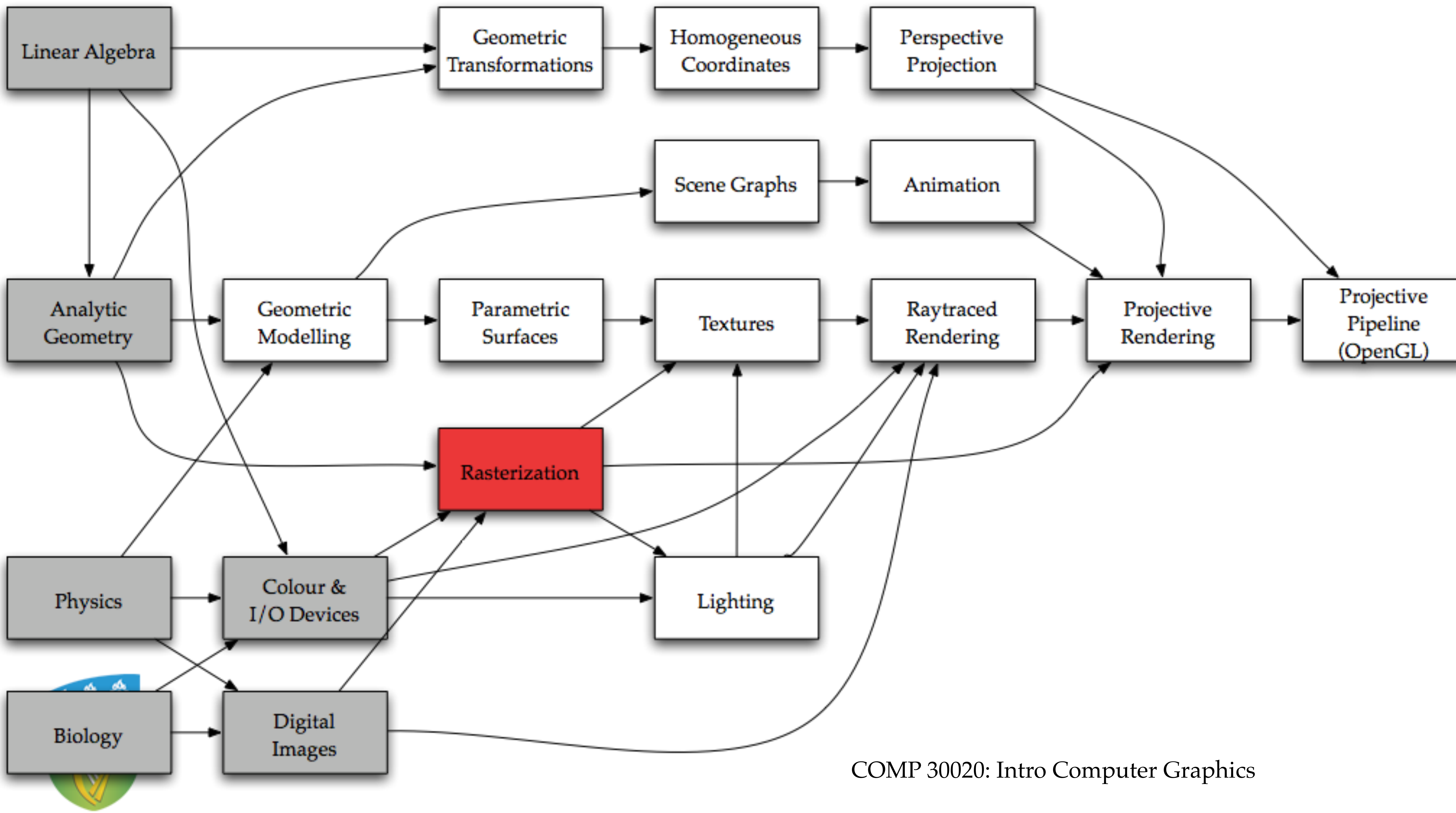
Rasterizing
Lines & Triangles

Where We Are


Linear Algebra
Geometric Transformations
Homogeneous Coordinates
Perspective Projection
Scene Graphs
Animation
Analytic Geometry
Geometric Modelling
Parametric Surfaces
Textures
Raytraced Rendering
Projective Rendering
Projective Pipeline (OpenGL)
Rasterization
Physics
Colour & I/O Devices
Lighting
Biology
Digital Images

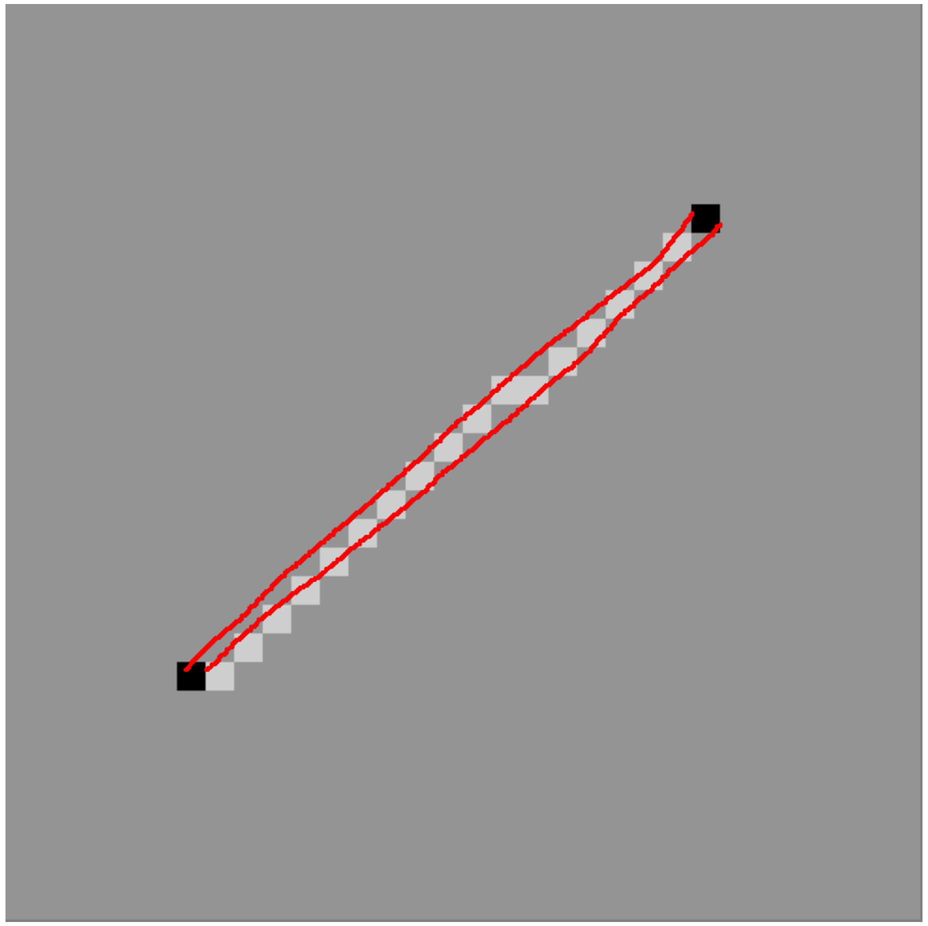
Drawing A Line

- We want to draw a line (segment)
  - from (x_0, y_0) to (x_1, y_1)
  - algorithms exist for:
    - explicit form
    - implicit / normal form
    - parametric form

Explicit Form

- Use equation of line to connect points:

```
for (x = x0; x < x1; x++)
        {
        y = mx + c;
        setPixel(x, y);
        }
```



Problems

- Doesn't work for slope > 1
  - Solution: use x = ny+d instead
- Doesn't work when $x_1 < x_0$ or $y_1 < y_0$
  - Solution: use descending loop
- These add complexity to the algorithm

Bresenham's Algorithm

```
rise = y1 - y0; run = x1 - x0;
if (run > 0)
    if (rise > 0)
        if (run > rise)
            for (x = x0; x < x1; x++)   {
                 y = (rise/run) x + c;
                 setPixel(x, y);         }
        else
            for (y = y0; y < y1; y++)   {
                 x = (run/rise) y + b;
                 setPixel(x, y);         }
    else
        if (run > -rise)
            for (x = x0; x < x1; x++)   {
                 y = (rise/run) x + c;
                 setPixel(x, y);         }
        else
            for (y = y0; y < y1; y++)   {
                 x = (run/rise) y + b;
                 setPixel(x, y);         }
&c., &c., &c.
```

Implicit Form

- We want to draw a line 1 pixel wide
  - all pixels within 0.5 pixels of line
  - we know how to measure distance
- But this draws a line, not a segment

```
for (x = xMin; x < xMax; x++)
    for (y = yMin; y < yMax; y++)
        if (abs(distance((x, y), (x0, y0), (x1, y1))) < 0.5)
            setPixel(x, y);
```

Parametric Form

- This is the easiest one yet!
- Walk along the line one step at a time:

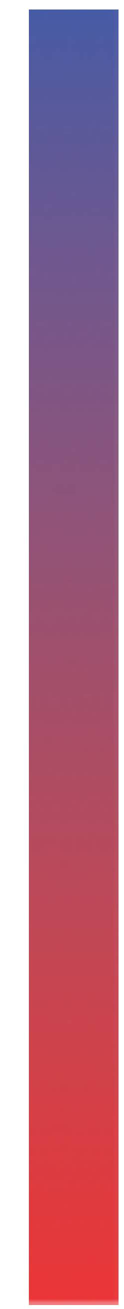
```
for (t = 0.0; t <= 1.0; t += 0.001)
    {
    point_r = point_p + (point_q - point_p) * t;
    setPixel(round(r_x), round(r_y));
    }
```

Comparison

- Explicit: conceptually easy, but messy
- Implicit: easy, but lines not segments
- Parametric: easy to code
- Which is most efficient?

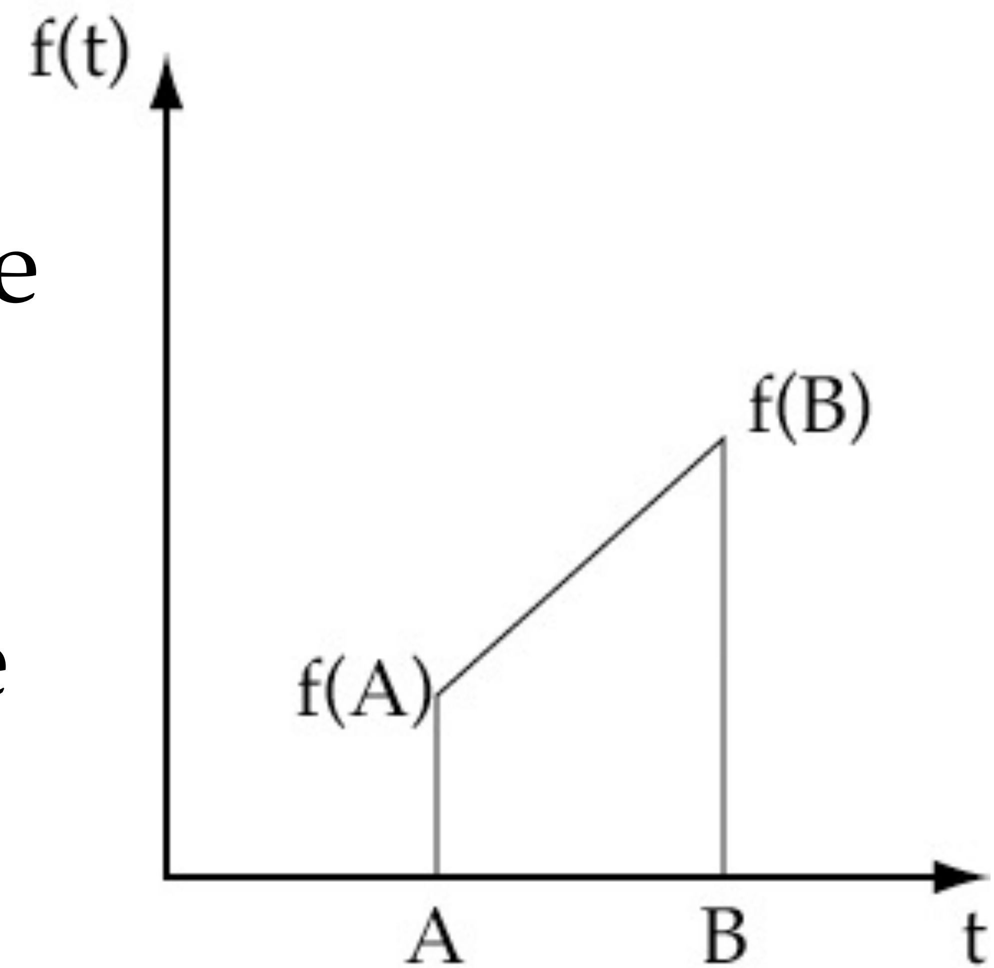
Colour Interpolation

- What if we want a coloured gradient?
  - At p, the line is 100% red, 0% blue
  - At q, the line is 0% red, 100% blue
  - In between, it varies smoothly
- This process is called interpolation



Linear Interpolation

- Assume parametric line
- Let f(t) be the colour
    - we use a straight line
    - f changes linearly:

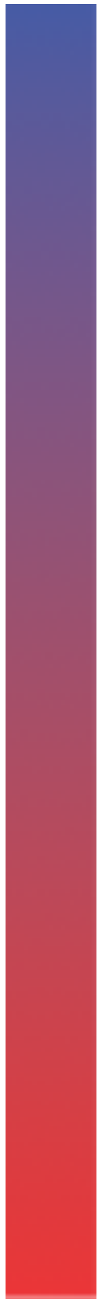


$$f(t) = f(A) + \left(\frac{t-A}{B-A}\right)\left(f(B) - f(A)\right)$$

Interpolating Colour

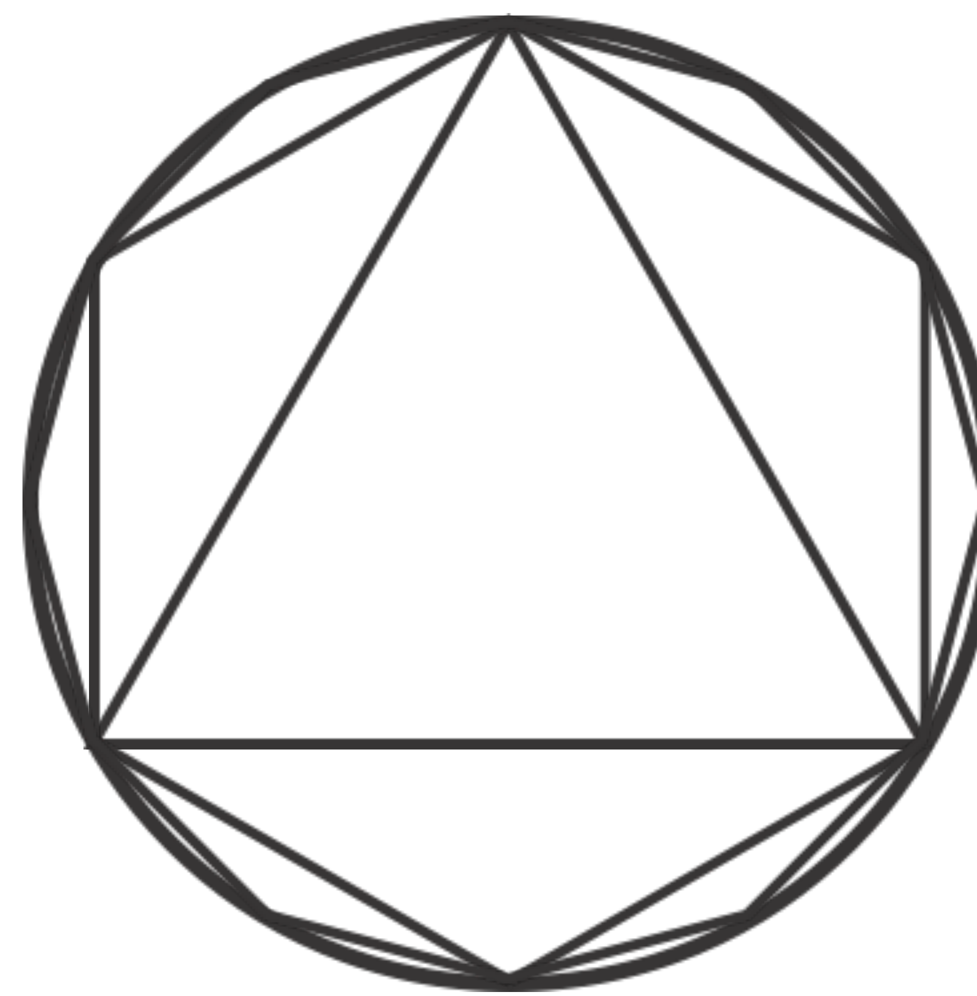
- Easiest in parametric form:

```
for (t = 0.0; t <= 1.0; t += 0.001)
    {
    point_r = point_p + (point_q - point_p) * t;
    colour = colour_p + (colour_q - colour_p) * t;
    setColour(colour);
    setPixel(round(r_x), round(r_y));
    }
```

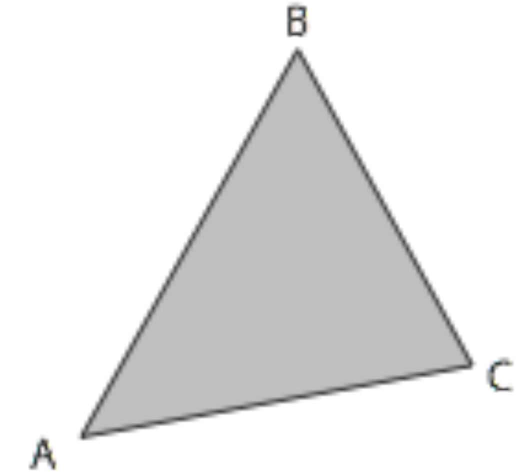


Lines & Curves

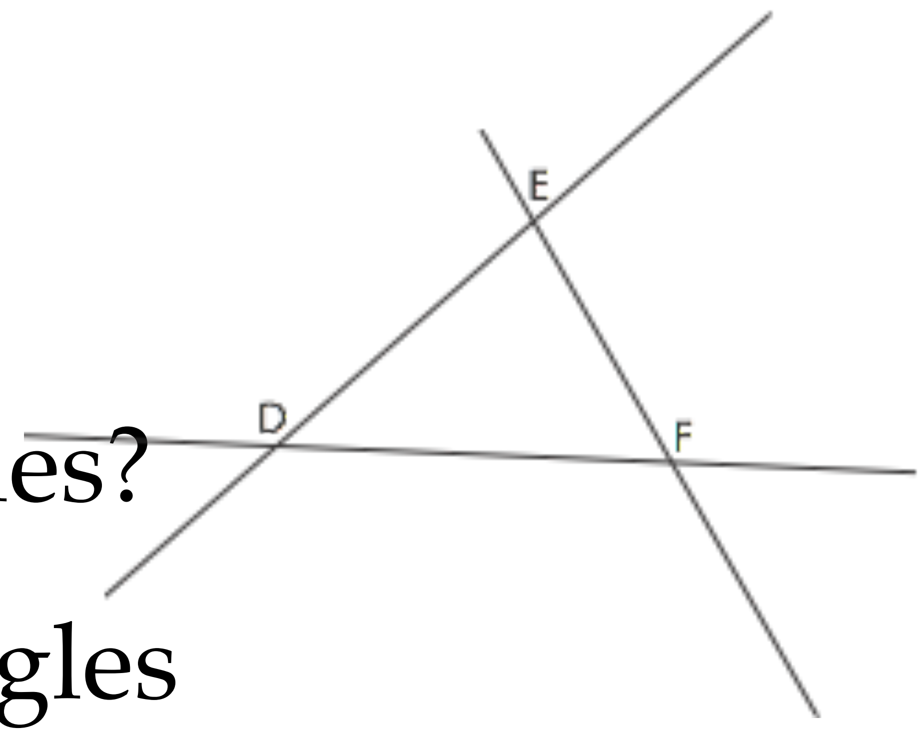
- We approximate curves
  - with many short lines
- Not always the best way!
  - we'll come back to this



Triangles



- Defined by 3 points:
  - Or by 3 lines
- Drawing three lines is easy
- But what about filled triangles?
- Start with equations of triangles



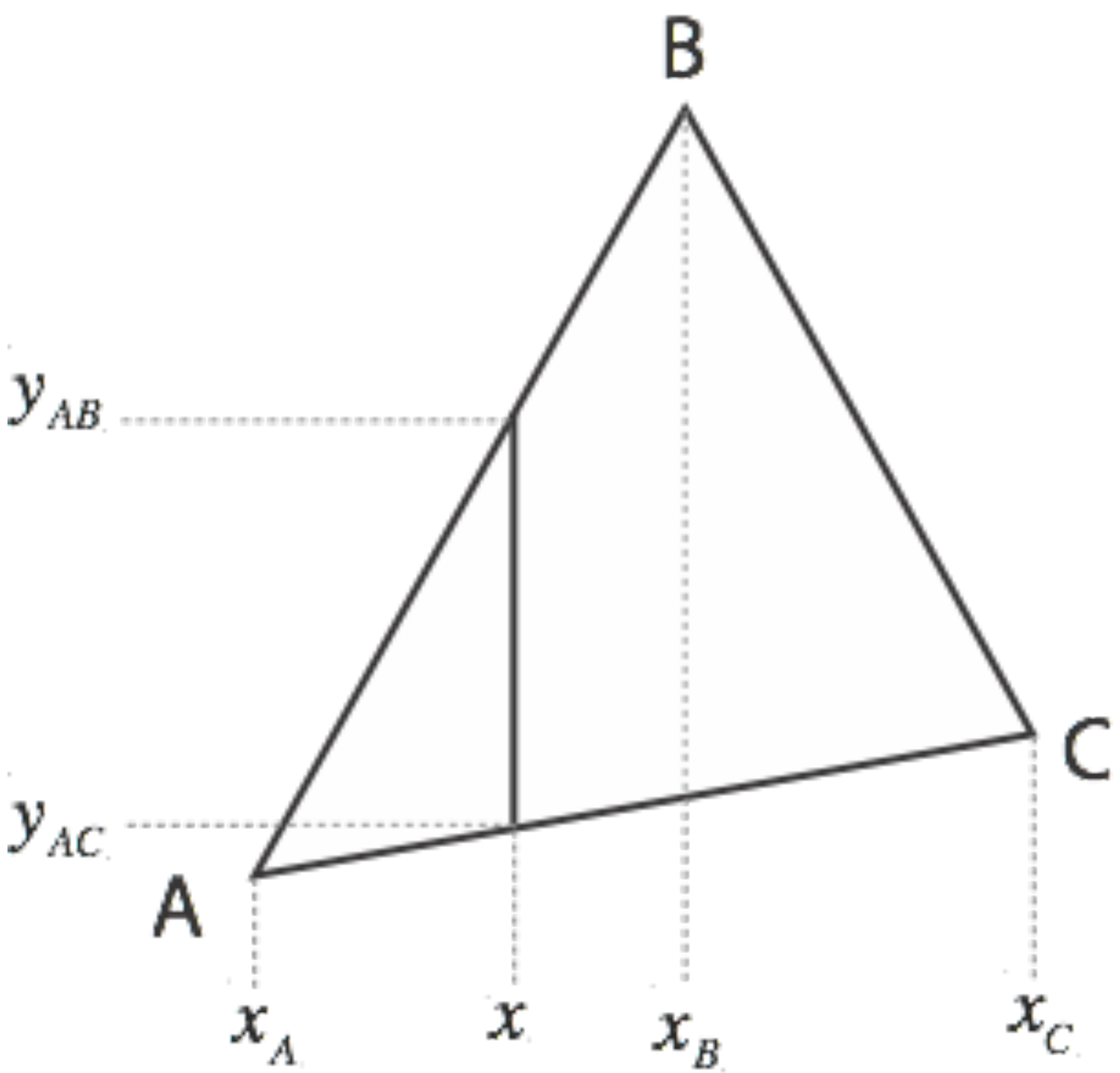
Explicit Form

- For any x, specify valid y

$$y_{AC} \le y \le y_{AB} \quad \text{if} \quad x_A \le x \le x_B$$

$$y_{AC} \le y \le y_{BC} \quad \text{if} \quad x_B \le x \le x_C$$

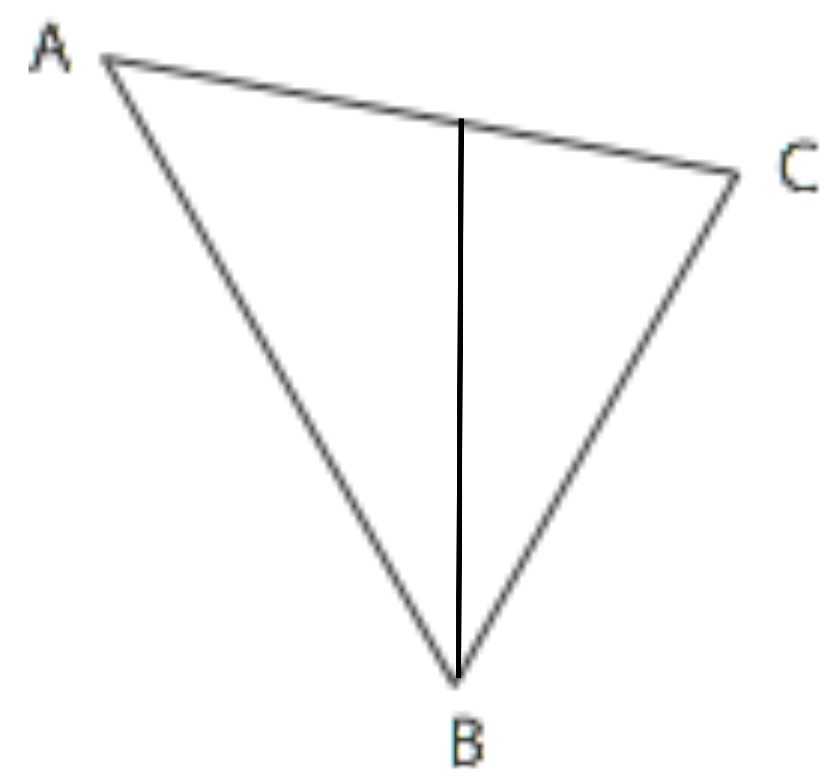
- Assumes B is above AC



Explicit Form, II

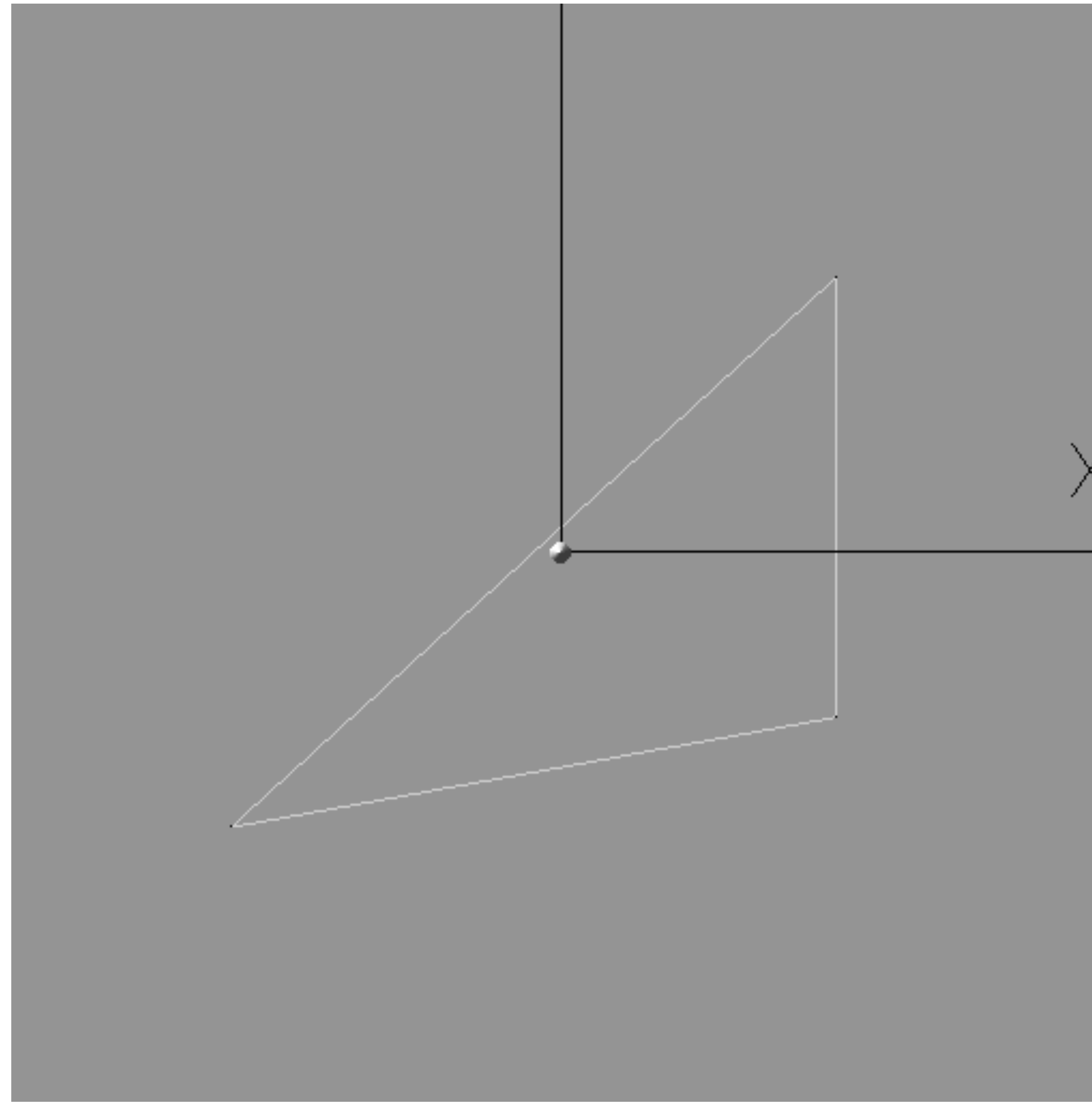
- If B is below AC:

$$y_{AB} \le y \le y_{AC} \quad if \;\; x_A \le x \le x_B$$
$$y_{BC} \le y \le y_{AC} \quad if \;\; x_B \le x \le x_C$$



Raster Scan Algorithm

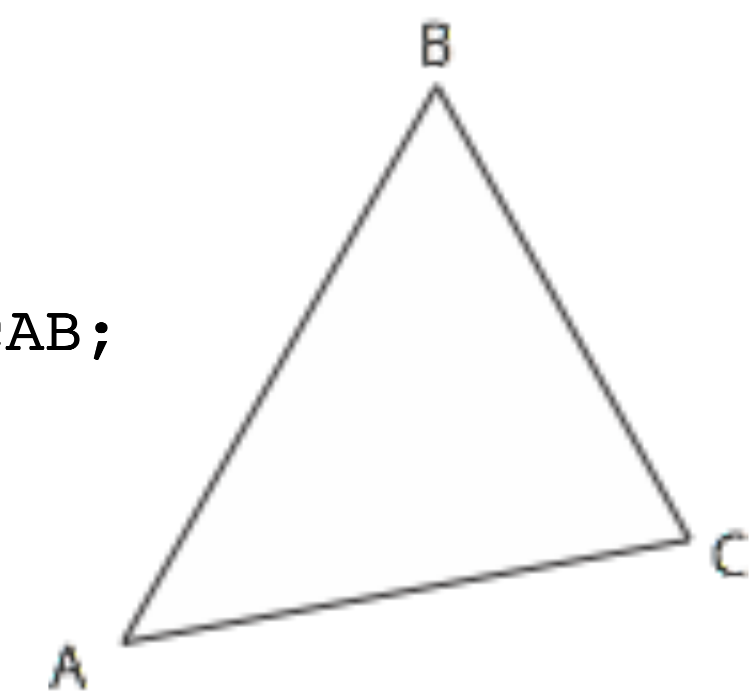
- Algorithm scans one line at a time
  - raster scan (raster means a rake)
  - scan conversion of triangles to pixels



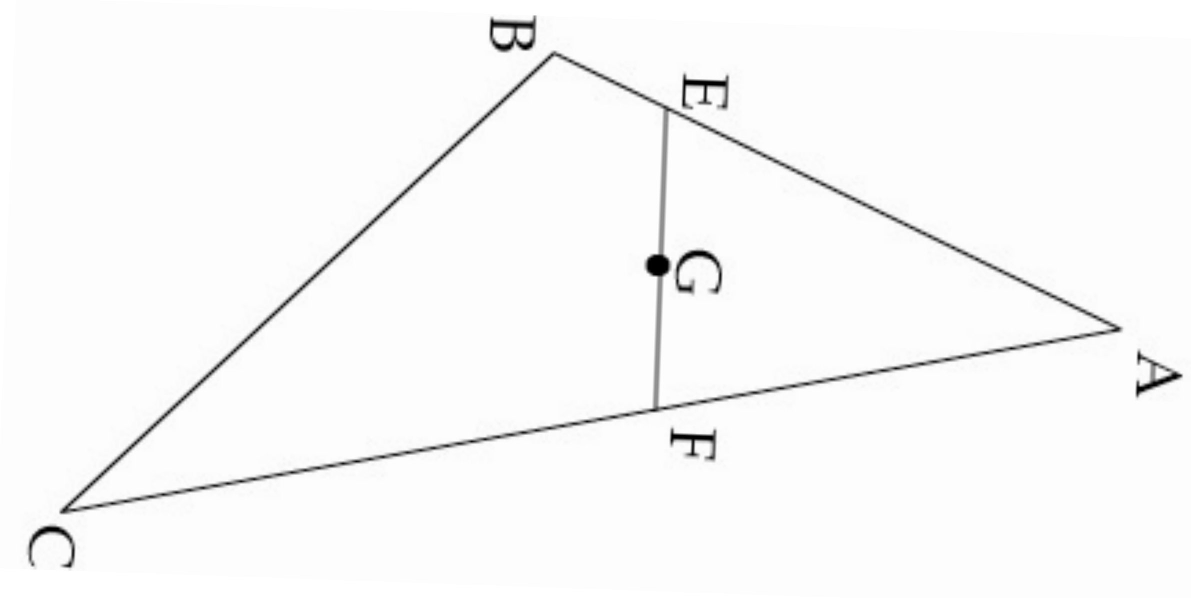
Explicit Algorithm

- Also called linewise scan
- Usually loops horizontally, not vertically

```
Sort A, B, C so Ax < Bx < Cx
Find slopes mAB, mAC, mBC,
Find y-intercepts cAB, cAC, cBC
for (x = Ax; x <= Bx; x++)
  { // for each column
  yMin = mAC * x + cAC; yMax = mAB * x + cAB;
  if (yMin < yMax)
    swap(yMin, yMax);
  for (y = yMin; y <= yMax; y++)
      setPixel(x,y);
  } // for each column
```



Linewise Interpolation



- To compute f(G):
  - Interpolate f(E) from f(A), f(B)
  - Interpolate f(F) from f(A), f(C)
  - Interpolate f(G) from f(E), f(F)
- Perform for each of R,G,B

Implicit / Normal Form

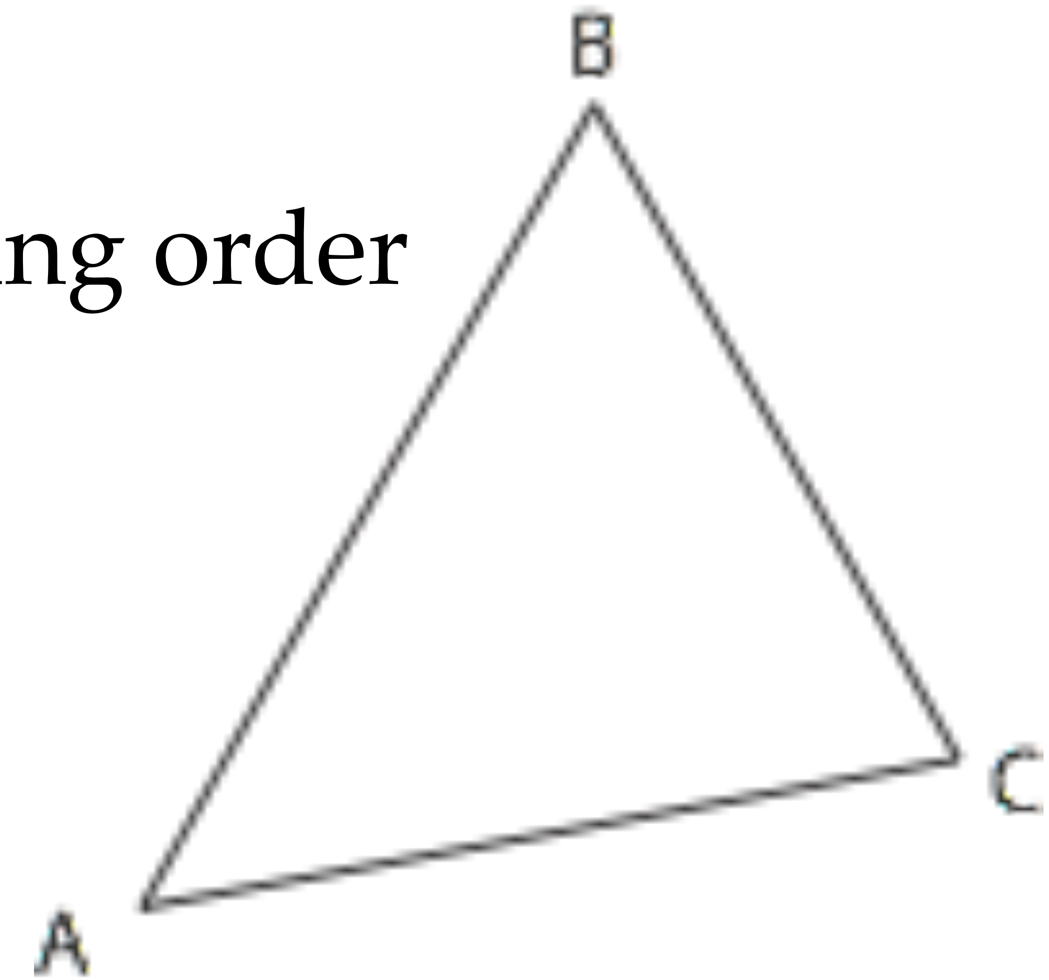
- Based on normal form of lines:

$$\vec{n} \cdot p - c = \begin{cases} - & \text{to } \textit{left} \text{ of line} \\ 0 & \textit{on} \text{ line} \\ + & \text{to } \textit{right} \text{ of line} \end{cases}$$

- Also known as the half-plane test

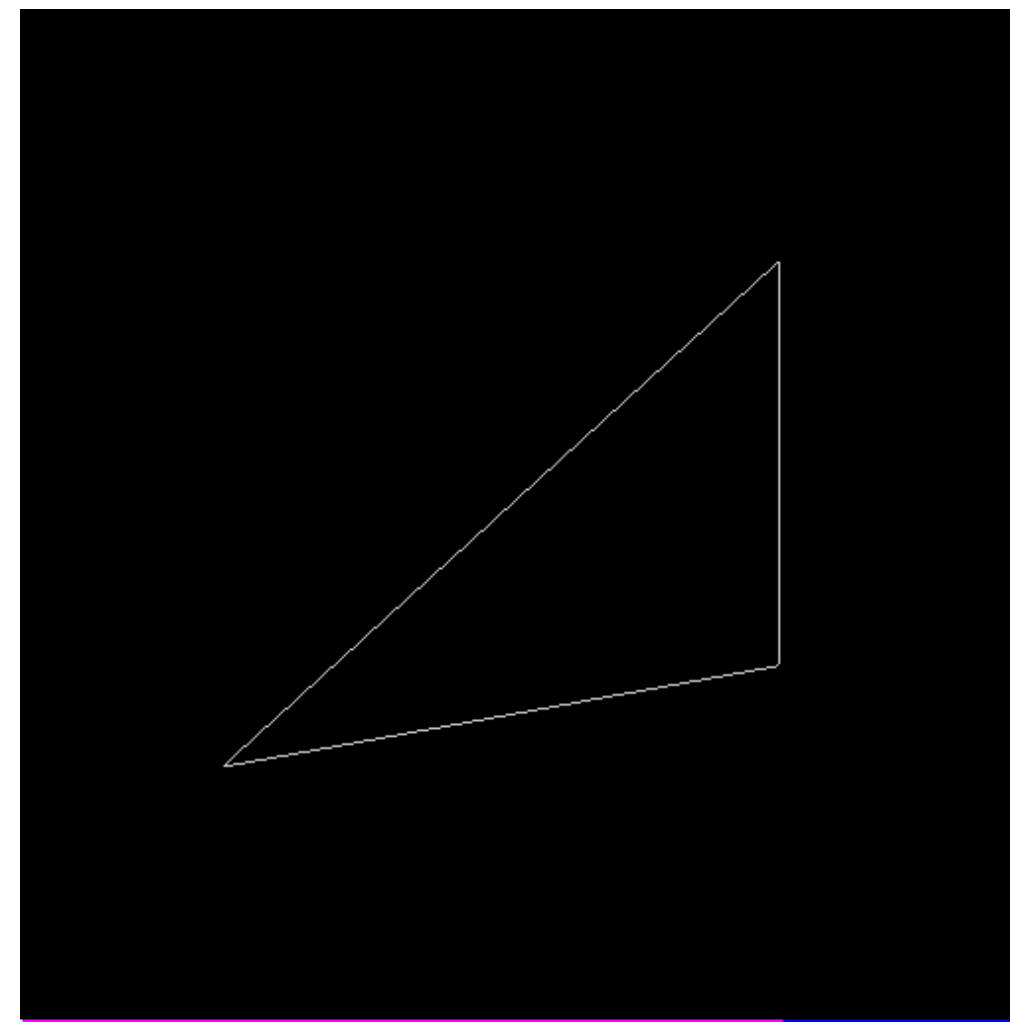
Winding Order

- Inside depends on the winding order
  - which direction we wind
  - ABC is clockwise (CW)
    - inside on right
  - ACB is counterclockwise (CCW)
    - inside on left



Half-Plane Test

- Each test divides plane in half:
  - Red vs. Not-Red
  - Green vs. Not-Green
  - Blue vs. Not-Blue
- Triangle is inside each



Implicit Algorithm

- Assume CCW winding order (left is inside)

```
for (x = xMin; x < xMax; x++)
    for (y = yMin; y < yMax; y++)
        if (  (x,y) leftOf (A,B) &&
              (x,y) leftOf (B,C) &&
              (x,y) leftOf (C,A))
            setPixel(x,y);
```

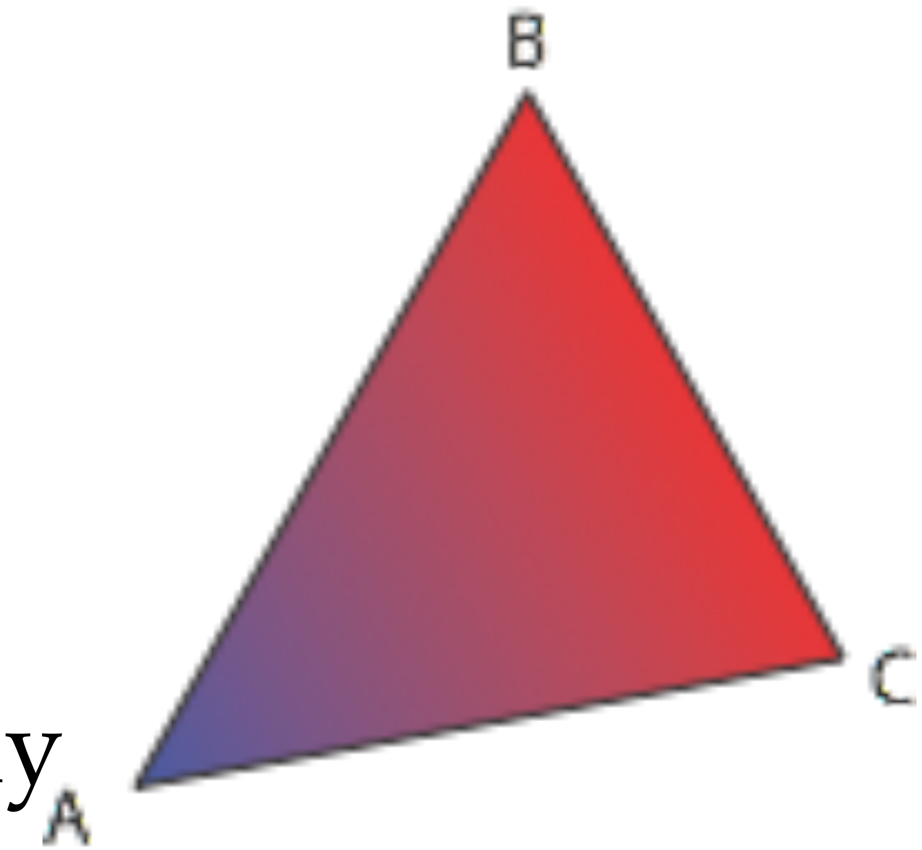
- But what about colour interpolation?
  - As with lines, we need parametric form

Parametric Form

- For a line pq, t = 0.0 at p, t = 1.0 at q
- How can we parameterize a triangle?
- We need at least two parameters
  - Start with one parameter
  - Use it to interpolate colour as well

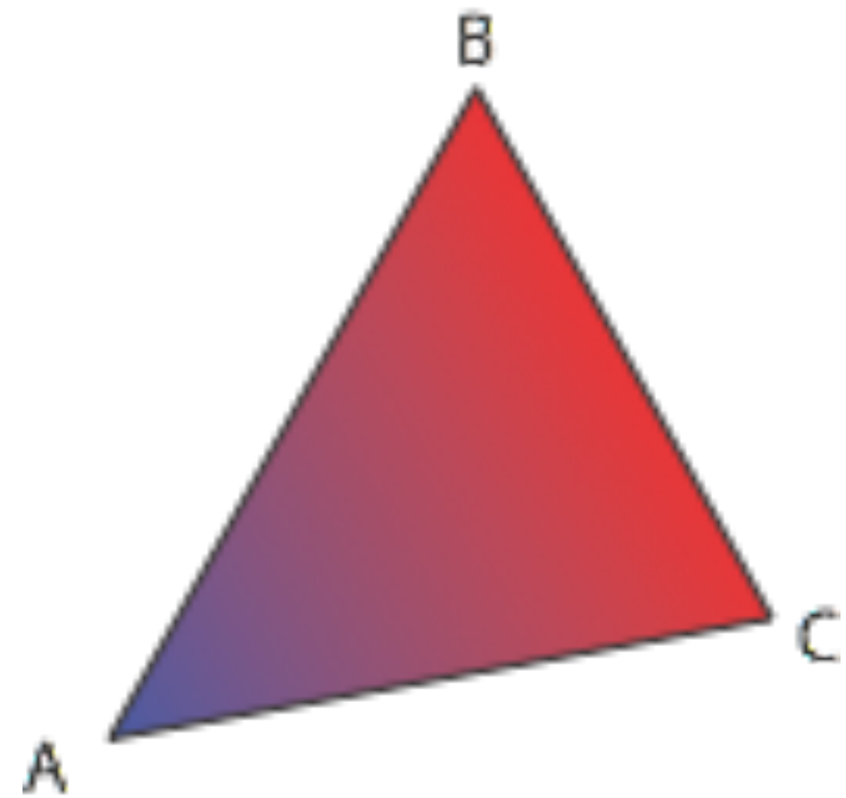
Triangle Interpolation

- Pick a vertex A
  - Set 100% blue at A
  - Set 0% blue at CB
- In between, varies linearly
  - perpendicular to CB



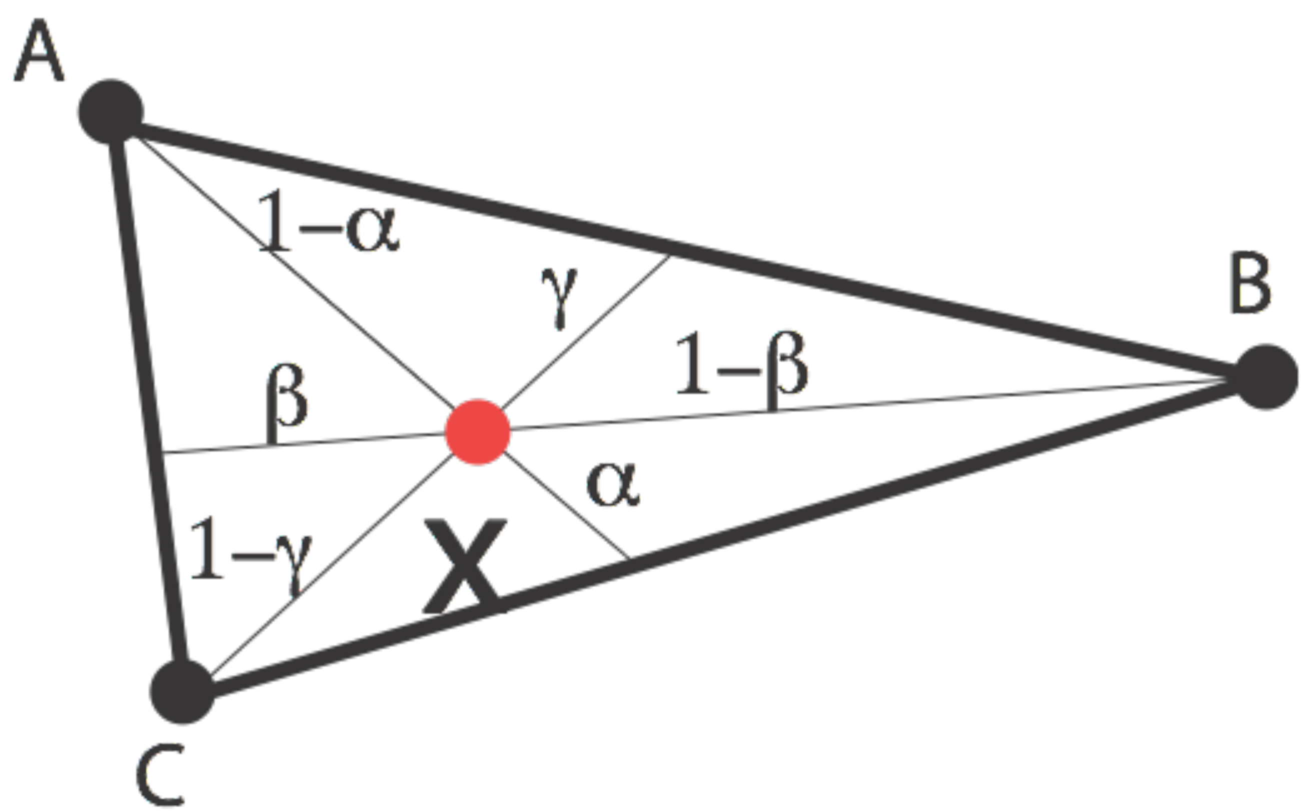
The Parameter α

- Colour depends on distance from CB
- Call this distance α
  - Parametrize so that:
    - $\alpha = 1.0$ at A
    - $\alpha = 0.0$ at BC



$$\alpha = \frac{dist(P,CB)}{dist(A,CB)}$$

Do it Three Times



$$\alpha = \frac{dist(P,CB)}{dist(A,CB)}$$

$$\beta = \frac{dist(P,AC)}{dist(B,AC)}$$

$$\gamma = \frac{dist(P,BA)}{dist(C,BA)}$$

$$f(x) = \alpha f(A) + \beta f(B) + \gamma f(C)$$

$$\alpha + \beta + \gamma = 1$$

Barycentric Coordinates

- α,β,γ are called barycentric coordinates
- Conveniently, $\alpha + \beta + \gamma = 1.0$
- So we really only have two parameters
- But we have three weights
  - This lets us interpolate from three vertices
    - to get colour, normals, textures, &c.

Parametric Algorithm

```
for (x = xMin; x < xMax; x++)
    for (y = yMin; y < yMax; y++)
        alpha = distance((x,y), BC) / distance(A, BC);
        gamma = distance((x,y), AB) / distance(C, AB);
        beta = 1 – alpha – gamma;
if ((alpha < 0.0) || (beta < 0.0) || (gamma < 0.0))
            continue;
        colour = alpha * colour(A) + beta * colour(B)
                  + gamma * colour(C);
        setColour(colour);
        setPixel(x,y);
```

Comparison

- Which is best?
- Explicit form is easiest to understand
- Half-plane (implicit) is easiest to code
- Barycentric (parametric) also computes weights for colour interpolation