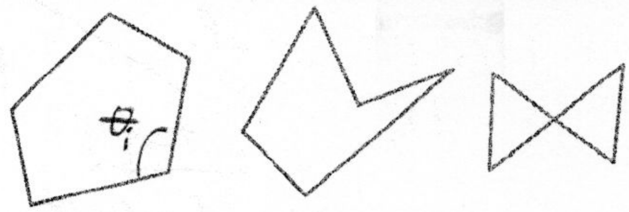
# Polygons

### *Interactive graphics uses polygons*



**simple convex** **simple concave** **non-simple (self-intersection)**

simple: edges do not self-intersect

convex: interior angles, $\theta_i \leq 180^\circ$

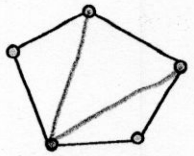
more generally: set $C \subseteq \mathbb{R}^d$ is convex if for any two points $P, Q \in C$ and any $\alpha \in [0,1]$,

$$\alpha P + (1-\alpha) Q \in C.$$

The 2D projections of convex 3D shapes are also convex.

# In practice we use triangles

- why? triangles are always planar, always convex
- simple convex polygons
  - *trivial to break into triangles*
- concave or non-simple polygons
  - *more effort to break into triangles*

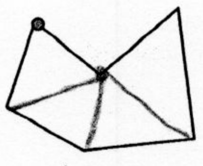


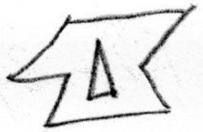
i.e., triangle fan.

simple polygon: $O(n)$

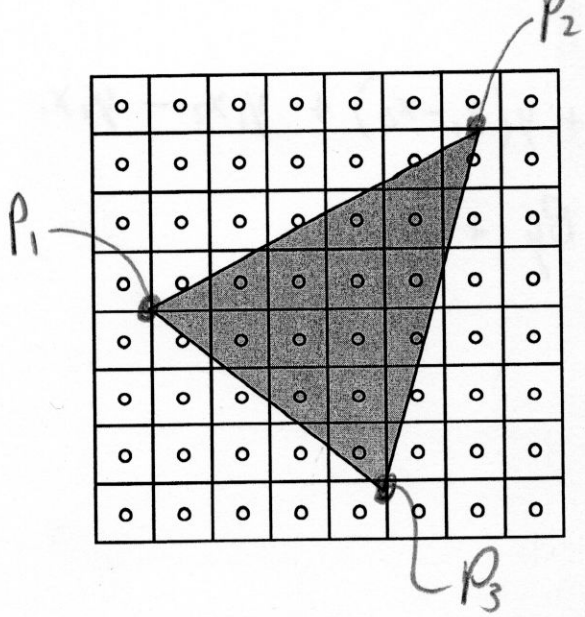
polygon with holes: $O(n \log n)$

n vertices, complex algorithms.

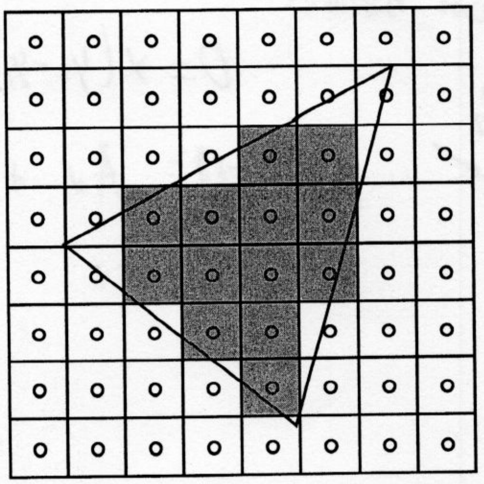




# What is Scan Conversion? (a.k.a. Rasterization)



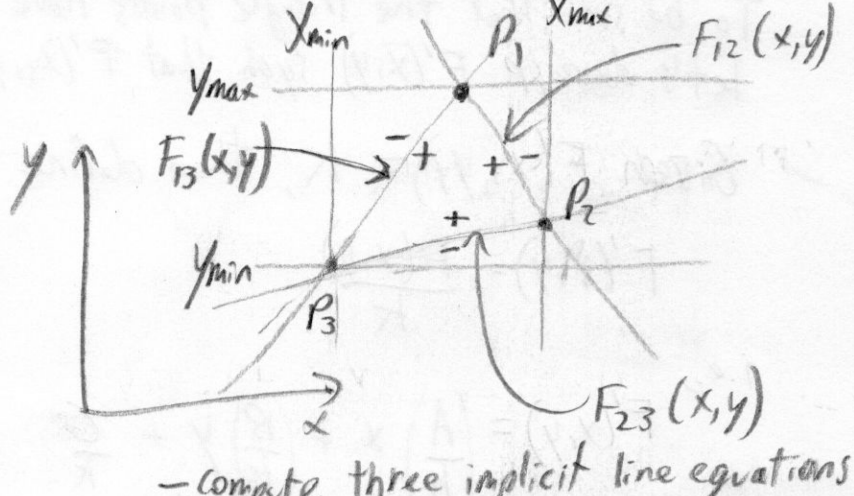
one possible scan conversion



Set all pixels whose center point is "inside" the triangle.

# Modern Rasterization

***Define a triangle as follows:***

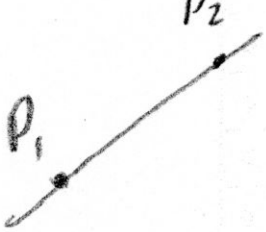


- compute three implicit line equations: $F_{12}$, $F_{13}$, $F_{23}$
- compute $X_{min}$, $X_{max}$, $y_{min}$, $y_{max}$
- for each pixel, set pixel if $F_{12}(x,y) \geq 0$, $F_{13}(x,y) > 0$, $F_{23}(x,y) > 0$

## Implicit Line Equation

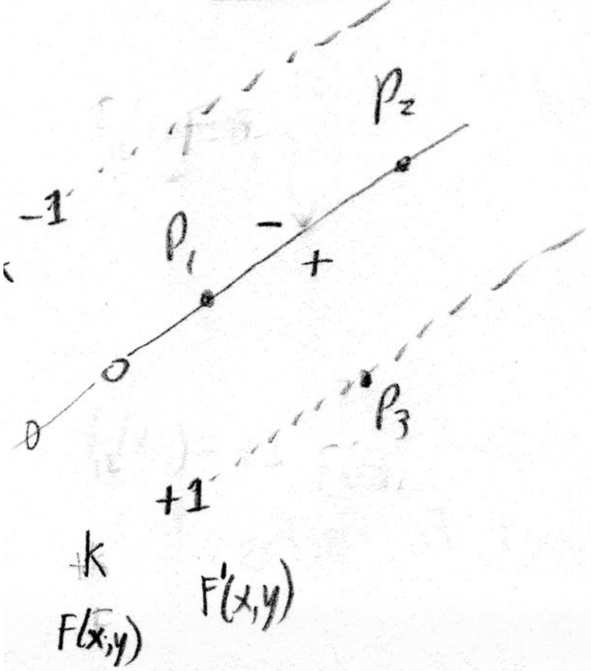
From before:

$$0 = x(y_2 - y_1) + y(x_1 - x_2) + y_1 x_2 - y_2 x_1$$



$$F(x,y) = 0 = Ax + By + C$$

## Scaled Implicit Line Equation



To be sure that the triangle points have $F(x,y) > 0$, let's develop $F'(x,y)$ such that $F'(x_3,y_3) = +1$.

Given $F(x_3,y_3) = k$, then define

$$F'(x,y) = \frac{F(x,y)}{k}$$

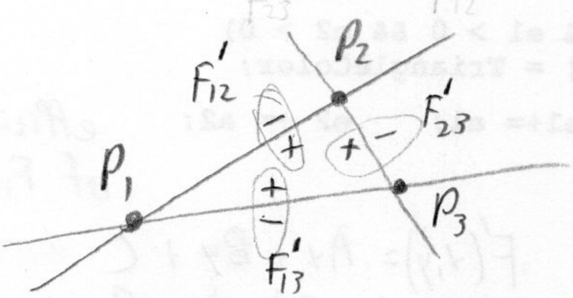
i.e.,

$$F'(x,y) = \left(\frac{A}{k}\right)x + \left(\frac{B}{k}\right)y + \frac{C}{k}$$

# Edge Equations: Code

***Basic structure of code:***

- Setup: compute edge equations, bounding box
- (Outer loop) For each scanline in bounding box...
- (Inner loop) …check each pixel on scanline, evaluating edge equations and drawing the pixel if all three are positive



# Edge Equations: Code

```
findBoundingBox(&xmin, &xmax, &ymin, &ymax);
setupEdges (&a0,&b0,&c0,&a1,&b1,&c1,&a2,&b2,&c2);

for (int y = yMin; y <= yMax; y++) {
      for (int x = xMin; x <= xMax; x++) {
            float e0 = a0*x + b0*y + c0;
            float e1 = a1*x + b1*y + c1;
            float e2 = a2*x + b2*y + c2;
            if (e0 > 0 && e1 > 0 && e2 > 0)
                        Image[x][y] = TriangleColor;

      }
}
```

$= F'_{12}(x,y)$

$= F'_{23}(x,y)$

$= F'_{13}(x,y)$

"inside" wrt all edges?

# Edge Equations: Code

```
  // more efficient inner loop
for (int y = yMin; y <= yMax; y++) {
      float e0 = a0*xMin + b0*y + c0;
      float e1 = a1*xMin + b1*y + c1;
      float e2 = a2*xMin + b2*y + c2;
      for (int x = xMin; x <= xMax; x++) {
        if (e0 > 0 && e1 > 0 && e2 > 0)
          Image[x][y] = TriangleColor;
        e0 += a0;   e1+= a1;    e2 += a2;
      }
}
```

setup

efficient update of $F_{12}, F_{13}, F_{23}$:

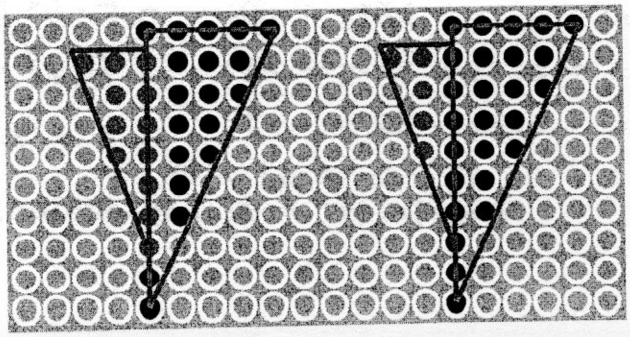
$$F'(x,y) = Ax + By + C$$
$$F'(x+1,y) = A(x+1) + By + C$$
$$\Delta F = A$$

# Triangle Rasterization Issues

Exactly which pixels should be lit?

***A: Those pixels inside the triangle edges***

What about pixels exactly on the edge?



Choices:

① Draw them. ⟹: result depends on triangle order

② Don't draw them ⟹ gap

③ Use a consistent-but-arbitrary rule

e.g: draw pixels on left or top boundaries.