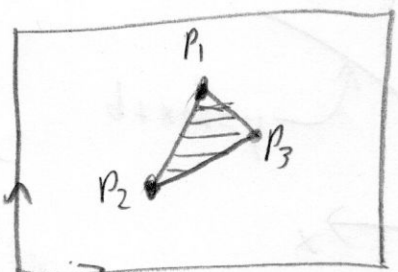
University of British Columbia

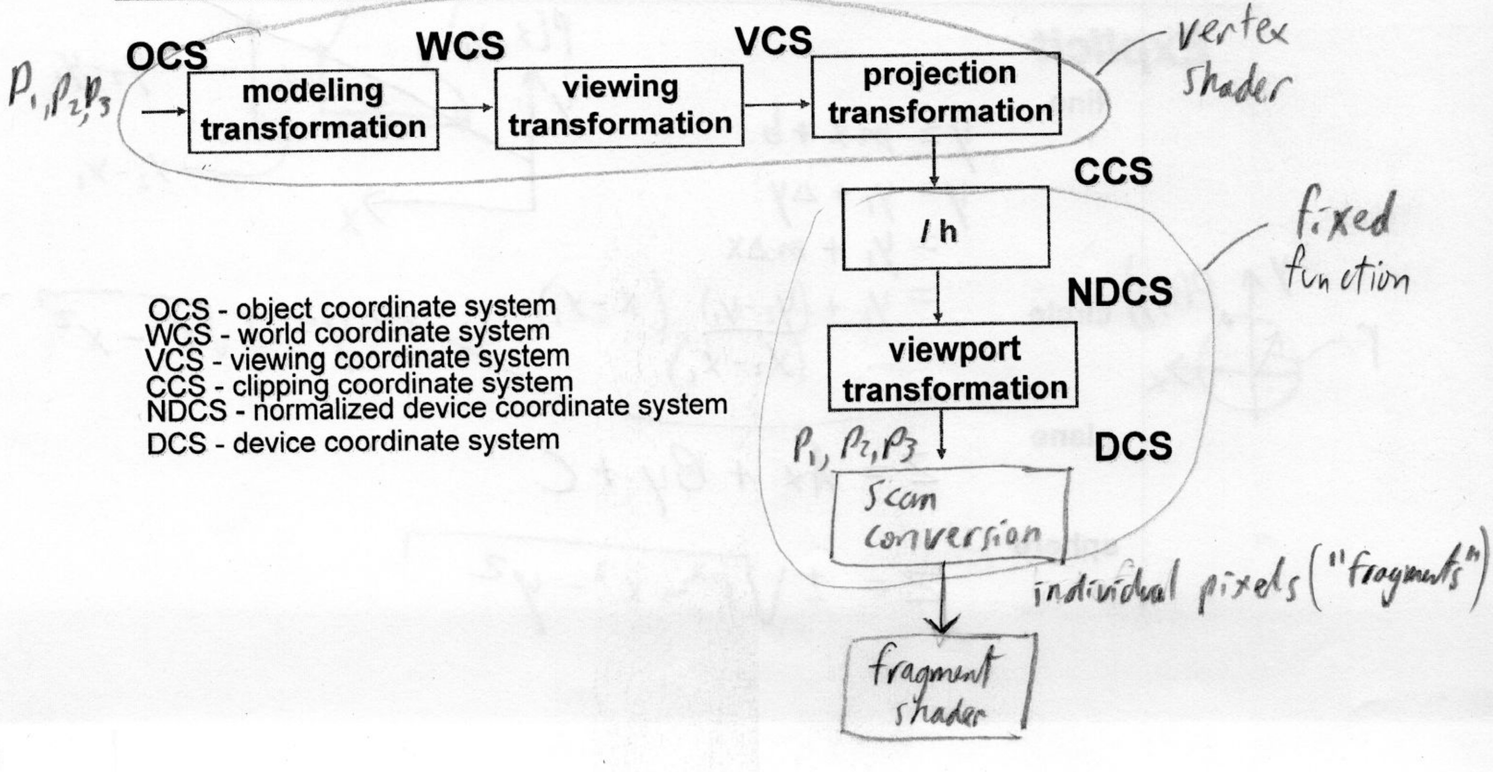
Scan Conversion

The process of drawing all the pixels ("fragments") that form a polygon. (In practice, we nearly always use triangles



DCS
device coordinate system

Scan Conversion

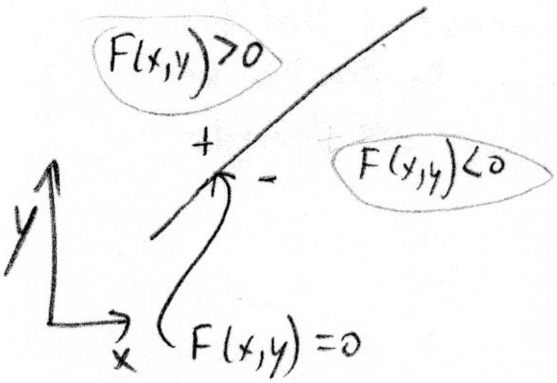


OCS - object coordinate system
WCS - world coordinate system
VCS - viewing coordinate system
CCS - clipping coordinate system
NDCS - normalized device coordinate system
DCS - device coordinate system

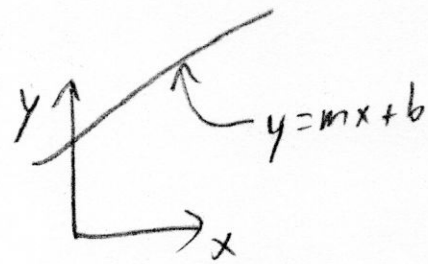
Implicit, Explicit, and Parametric equations for defining geometry

Three ways to write a line equation:

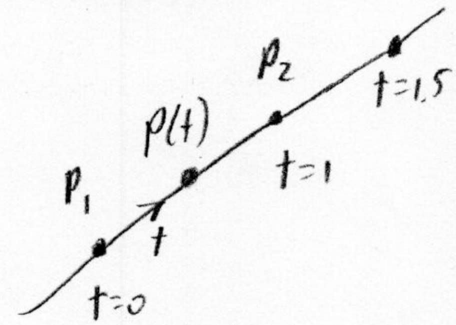
① Implicit



② Explicit

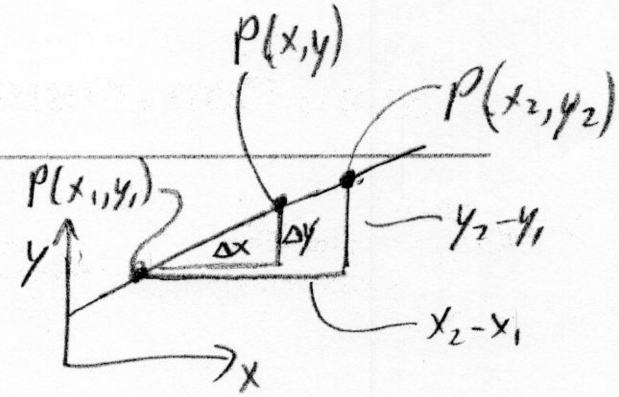


③ Parametric



The point is a function of some underlying parameter, t.

Lines and Curves



Explicit

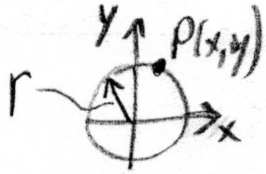
line

$$y = mx + b$$

$$y = y_1 + \Delta y$$

$$= y_1 + m\Delta x$$

$$= y_1 + \frac{(y_2 - y_1)}{(x_2 - x_1)}(x - x_1)$$



circle

$$y = \pm\sqrt{r^2 - x^2}$$

plane

$$z = Ax + By + C$$

sphere

$$z = \pm\sqrt{r^2 - x^2 - y^2}$$

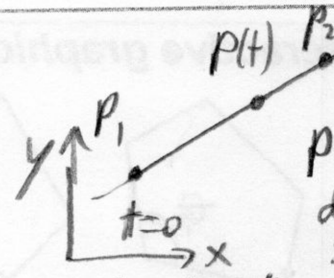
Lines and Curves

Parametric

line

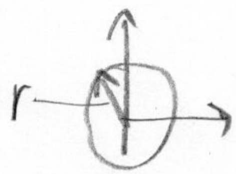
$$P(t) = P_1 + t(P_2 - P_1)$$
$$= \underbrace{(1-t)}P_1 + \underbrace{t}P_2$$

"basis functions"



$p(t),\ t \in [0,1]$ defines the line segment $\overline{P_1P_2}$.

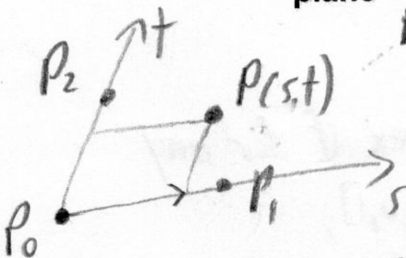
circle



$$x(t) = r\cos(t) \qquad t \in [0, 2\pi]$$
$$y(t) = r\sin(t)$$

plane

$$P(s,t) = P_0 + s(P_1 - P_0) + t(P_2 - P_0)$$



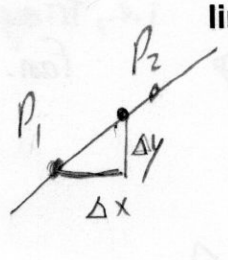
$$\begin{bmatrix} x(s,t) \\ y(s,t) \\ z(s,t) \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} + s\begin{bmatrix} x_1 - x_0 \\ y_1 - y_0 \\ z_1 - z_0 \end{bmatrix} + t\begin{bmatrix} x_2 - x_0 \\ y_2 - y_0 \\ z_2 - z_0 \end{bmatrix}$$

Three points uniquely define a 3D-plane.

Lines and Curves

Implicit $F(x,y) = 0, \quad F(x,y,z) = 0$

line



$$y = y_1 + \Delta y$$
$$= y_1 + m\Delta x$$
$$y = y_1 + \left(\frac{y_2 - y_1}{x_2 - x_1}\right)(x - x_1)$$

$$0 = (y_1 - y) + \left(\frac{y_2 - y_1}{x_2 - x_1}\right)(x - x_1)$$

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

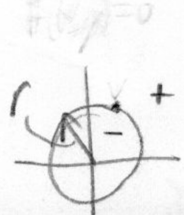
$$0 = x(y_2 - y_1) + y(x_1 - x_2)$$
$$+ \underbrace{y_1x_2 - y_1x_1 - y_2x_1 + y_1x_1}_{}$$

$$\underbrace{-y_1x_1 + y_1x_1}_{0}$$

$$0 = x\underbrace{(y_2 - y_1)}_{A} + y\underbrace{(x_1 - x_2)}_{B} + \underbrace{y_1x_2 - y_2x_1}_{C}$$

$$0 = Ax + By + C$$

circle

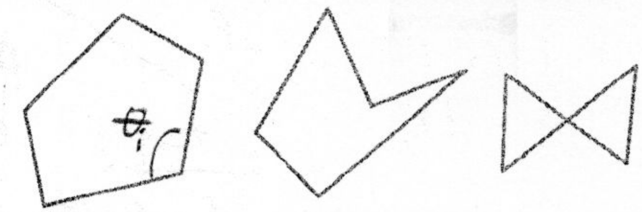


$$r^2 = x^2 + y^2$$
$$0 = x^2 + y^2 - r^2$$

$F(x,y) = 0$: on circle
< 0 : inside
> 0 : outside

Polygons

Interactive graphics uses polygons



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

simple: edges do not self-intersect

convex: interior angles, $\theta_i \leq 180°$

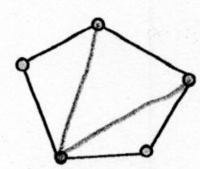
More generally: set $C \subseteq \mathbb{R}^d$ is convex if for any two points $P, Q \in C$ and any $\alpha \in [0,1]$,

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

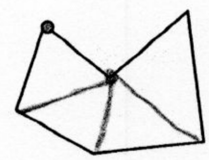
The 2D projections of convex 3D shapes are also convex.

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*



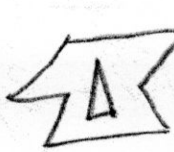
i.e., triangle fan.



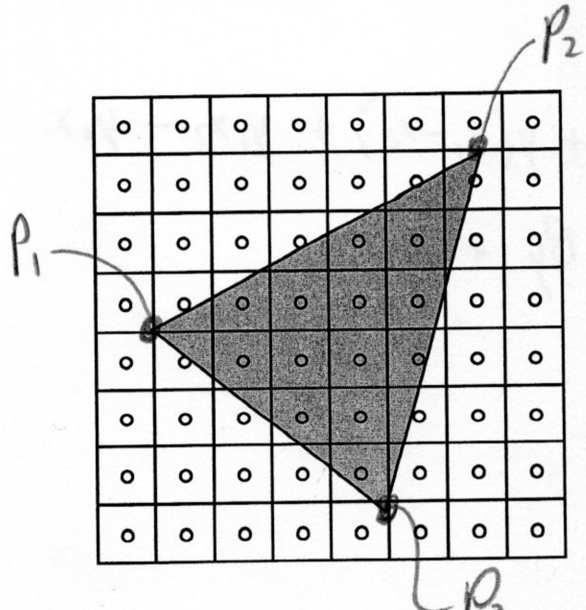
simple polygon: $O(n)$

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

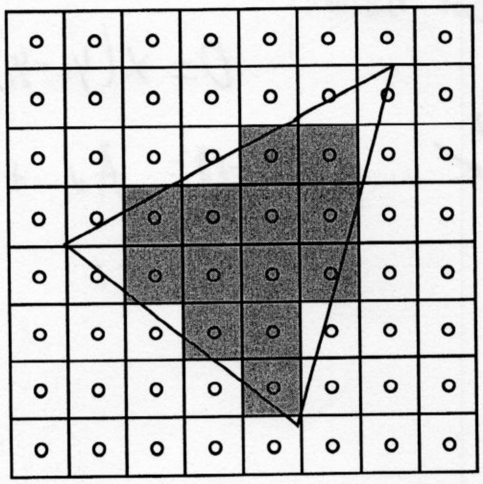
n vertices, complex algorithms.



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



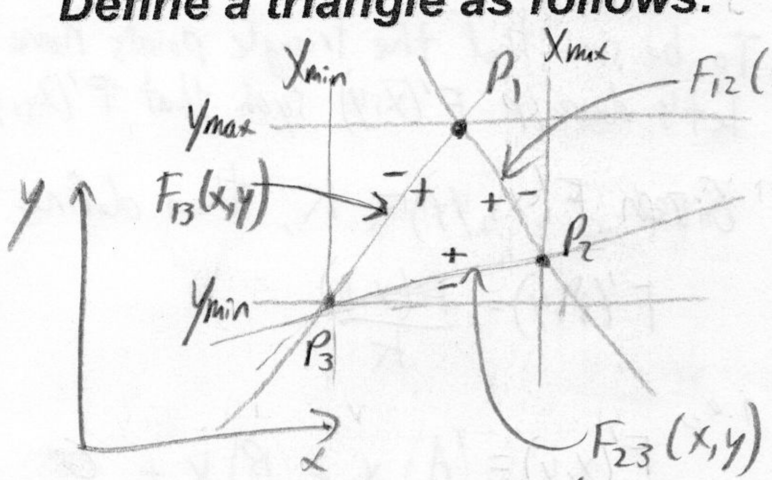
one possible scan conversion



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

Modern Rasterization

Define a triangle as follows:



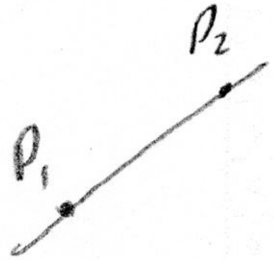
– compute three implicit line equations: F_{12}, F_{13}, F_{23}

– compute X_{min}, X_{max}, y_{min}, y_{max}

– for each pixel, set pixel if $F_{12}(x,y) \geq 0$, $F_{13}(x,y) \geq 0$, $F_{23}(x,y) \geq 0$

Implicit Line Equation

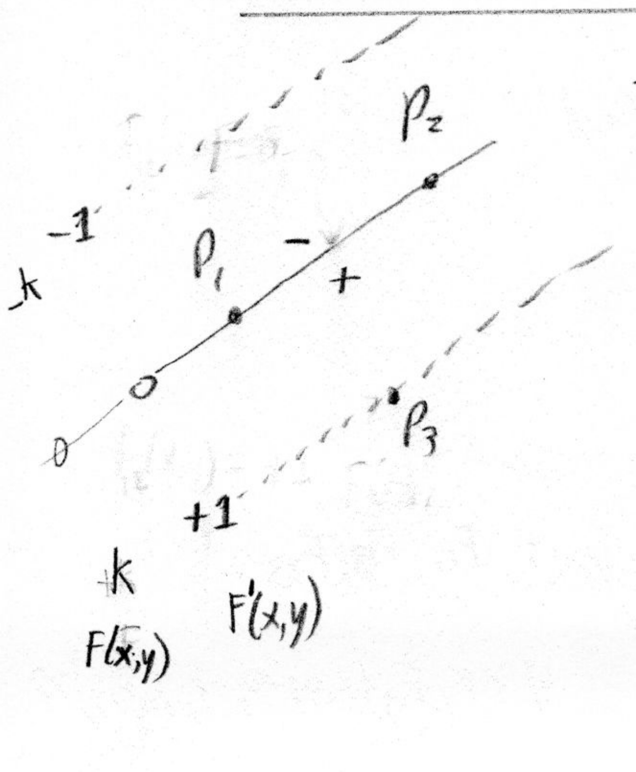
From before:



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

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

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

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

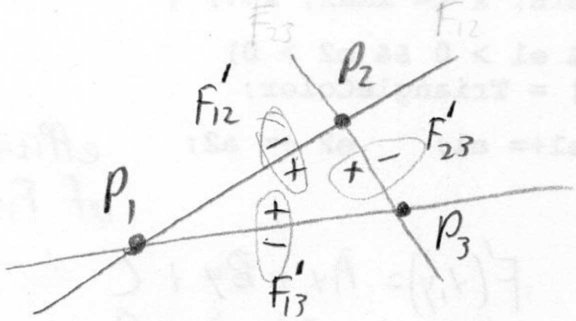
i.e.,

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

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;     = F'12(x,y)
            float e1 = a1*x + b1*y + c1;     = F'23(x,y)
            float e2 = a2*x + b2*y + c2;     = F'13(x,y)
            if (e0 > 0 && e1 > 0 && e2 > 0)        "inside" wrt
                        Image[x][y] = TriangleColor;    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;
      }
}
```

$\left.\begin{array}{l}\texttt{e0}\\\texttt{e1}\\\texttt{e2}\end{array}\right\}$ setup

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

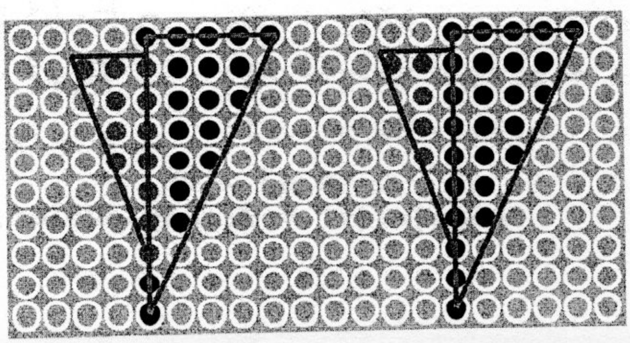
$$F'(x,y) = Ax + By + C$$
$$F'(x+1,y) = A(x+1) + By + C$$
$$\Delta F = A$$

Triangle Rasterization Issues

Exactly which pixels should be lit?

A: Those pixels inside the triangle edges

What about pixels exactly on the edge?



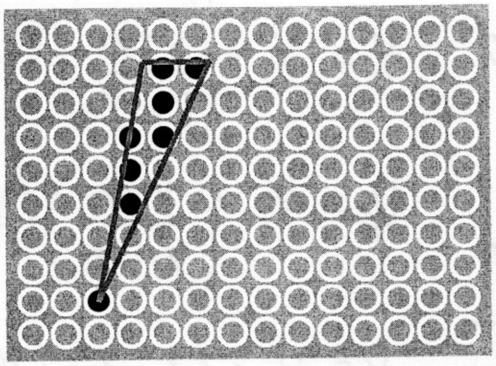
Choices:

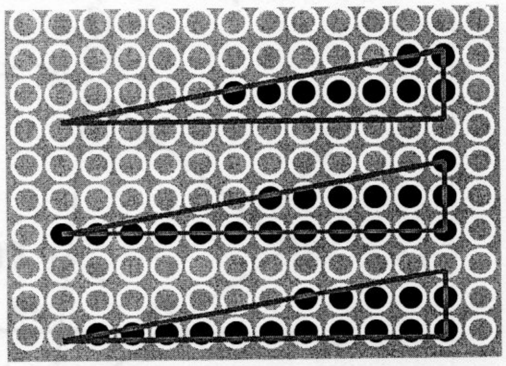
1. Draw them. $\Rightarrow$: result depends on triangle order
2. Don't draw them $\Rightarrow$ gap
3. Use a consistent-but-arbitrary rule

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

Triangle Rasterization Issues

Sliver | ***Moving Slivers***





Partial solution: antialiasing – set a pixel "partly on" based on the fraction of a pixel area that is covered by a triangle.

Interpolation During Scan Conversion

- interpolate between vertices: (demo)
  - *z*
  - *r,g,b* *colour components*
  - *u,v* *texture coordinates*
  - N_x, N_y, N_z *surface normals*

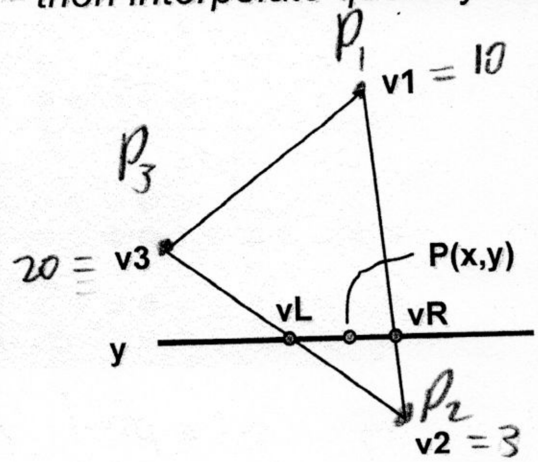
value V known at the vertices.

- three equivalent ways of viewing this (for triangles)
  1. *bilinear interpolation*
  2. *plane equation*
  3. *barycentric coordinates*

1. Bilinear Interpolation

fraction of the way towards P_3 from P_2

- interpolate quantity along LH and RH edges, as a function of y
  - *then interpolate quantity as a function of x*



$$V_L = V_2 + \left(\frac{y-y_2}{y_3-y_2}\right)(V_3 - V_2)$$

$$V_R = V_2 + \left(\frac{y-y_2}{y_1-y_2}\right)(V_1 - V_2)$$

$$V = V_L + \left(\frac{x-x_L}{x_R-x_L}\right)(V_R - V_L)$$

2. Plane Equation

- v = Ax + By + C

$$\left.\begin{aligned} A x_1 + B y_1 + C &= V_1 \\ A x_2 + B y_2 + C &= V_2 \\ A x_3 + B y_3 + C &= V_3 \end{aligned}\right\} \text{Solve for } A, B, C$$

At any given pixel x,y. compute v using

$$v = Ax + By + C$$

3. Barycentric Coordinates

•weighted combination of vertices

$$\begin{cases} P = \alpha \cdot P_1 + \beta \cdot P_2 + \gamma \cdot P_3 \\ \alpha + \beta + \gamma = 1 \\ 0 \le \alpha, \beta, \gamma \le 1 \end{cases}$$

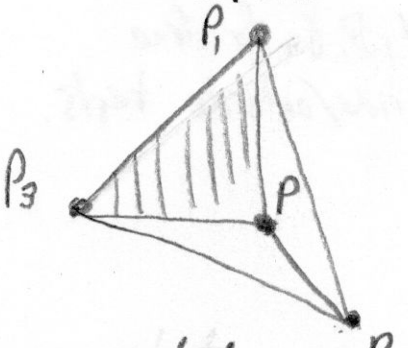
α, β, γ are weights

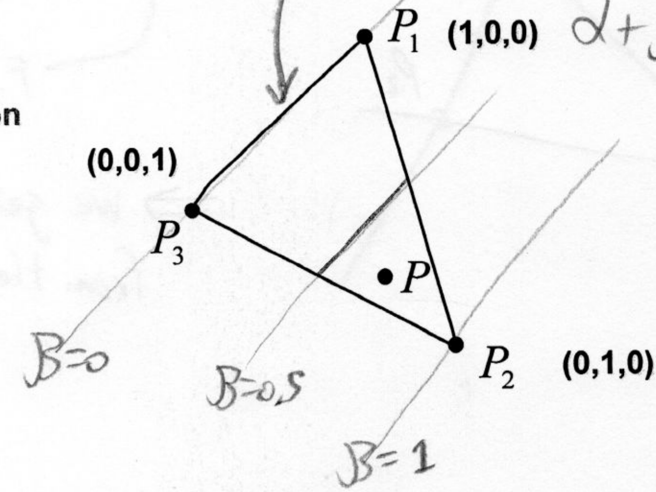
$\alpha, \beta, \gamma \in [0,1]$

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

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

"convex combination of points"





equivalent:

$$\beta = \frac{area(\Delta P P_1 P_3)}{area(\Delta P_1 P_2 P_3)}$$

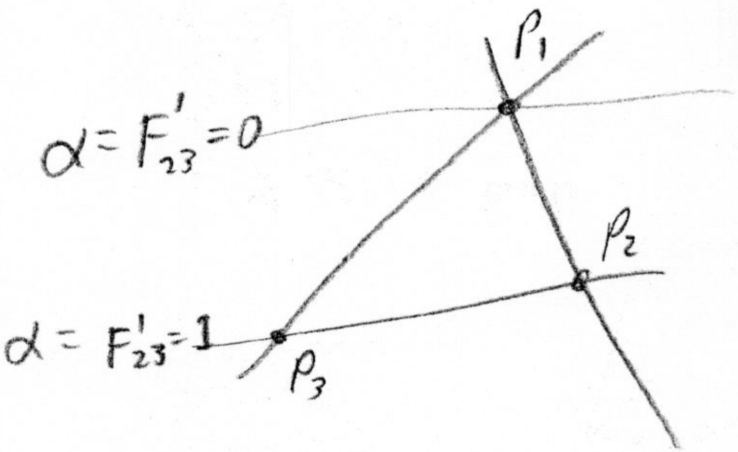
Barycentric Coordinates

- once computed, use to interpolate any # of parameters from their vertex values

$$z = \alpha \cdot z_1 + \beta \cdot z_2 + \gamma \cdot z_3$$
$$r = \alpha \cdot r_1 + \beta \cdot r_2 + \gamma \cdot r_3$$
$$g = \alpha \cdot g_1 + \beta \cdot g_2 + \gamma \cdot g_3$$

etc.

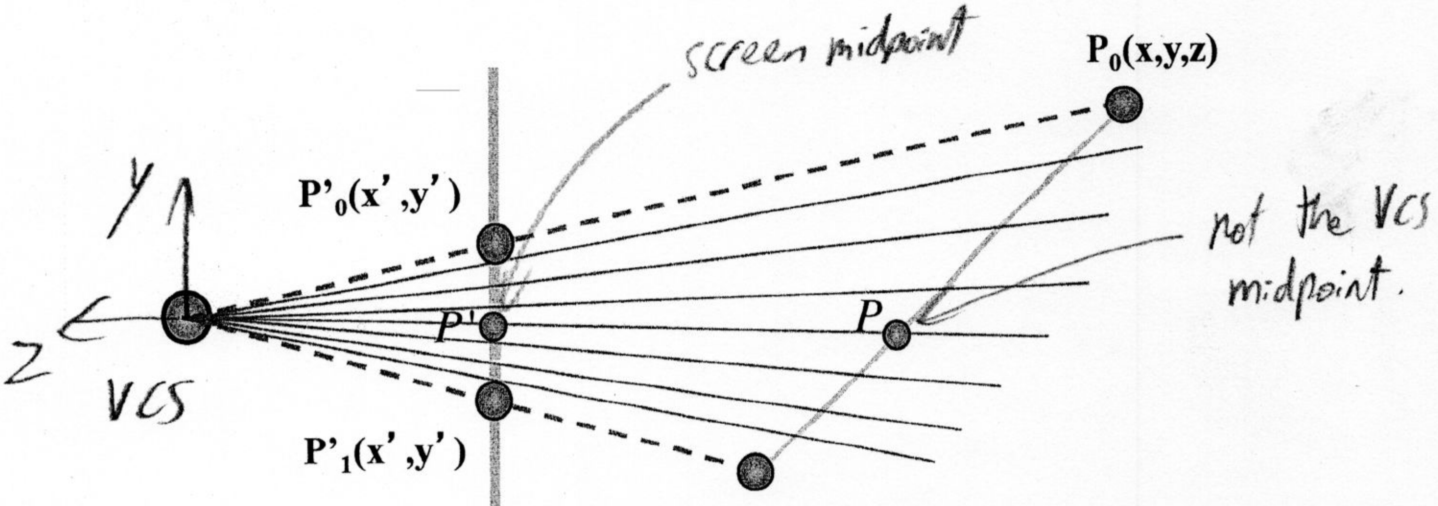
Computing Barycentric Coords



$$V = \underbrace{\alpha}_{F'_{23}} V_1 + \underbrace{\beta}_{F'_{13}} V_2 + \underbrace{\gamma}_{F'_{12}} V_3$$

$\Rightarrow$ we get α, β, γ for free from the inside/outside tests.

Interpolation: Screen vs World Space



as described so far

Perspective-correct interpolation

DCS space

Linear interpolation of v in screen space

as described before

compute α,β,γ such that $P' = \alpha \cdot P_1' + \beta \cdot P_2' + \gamma \cdot P_3'$

$v = Barycentric(v_1, v_2, v_3)$ $\quad v = \alpha \cdot v_1 + \beta \cdot v_2 + \gamma \cdot v_3$

VCS space

Linear interpolation of v in ~~world~~ VCS space

compute α,β,γ such that $P = \alpha \cdot P_1 + \beta \cdot P_2 + \gamma \cdot P_3$

$v = Barycentric(v_1, v_2, v_3)$ $\quad v = \alpha \cdot v_1 + \beta \cdot v_2 + \gamma \cdot v_3$

$$v = \frac{Barycentric(\frac{v_1}{h_1}, \frac{v_2}{h_2}, \frac{v_3}{h_3})}{Barycentric(\frac{1}{h_1}, \frac{1}{h_2}, \frac{1}{h_3})} \qquad v = \frac{\alpha \cdot v_1 / h_1 + \beta \cdot v_2 / h_2 + \gamma \cdot v_3 / h_3}{\alpha / h_1 + \beta / h_2 + \gamma / h_3}$$

interpolate $\frac{1}{h}$

"perspective correct interpolation"

- ⇒ use screen-space barycentric interpolation of $1/h$
- i.e., $\frac{1}{h} = \alpha\left(\frac{1}{h_1}\right) + \beta\left(\frac{1}{h_2}\right) + \gamma\left(\frac{1}{h_3}\right)$.
- then interpolate using

$$V = \frac{\alpha\left(V_1/h_1\right) + \beta\left(V_2/h_2\right) + \gamma\left(V_3/h_3\right)}{1/h}$$