Scan Conversion (fixed function)

OCS → **modeling transformation** → WCS → **viewing transformation** → VCS → **projection transformation** → CCS → **/ h** → NDCS → **viewport transformation** → DCS

P_1, P_2, P_3

clipping coordinate system

fixed function

P_1'', P_2'', P_3''

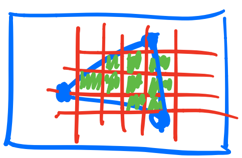
scan conversion

for each pixel

fragment

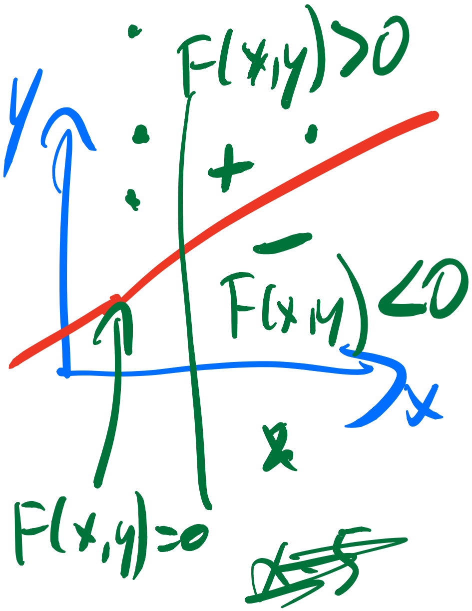
OCS - object coordinate system
WCS - world coordinate system
VCS - viewing coordinate system
CCS - clipping coordinate system
NDCS - normalized device coordinate system
DCS - device coordinate system

Scan Conversion:
The process of identifying all the pixels that lie within a given triangle

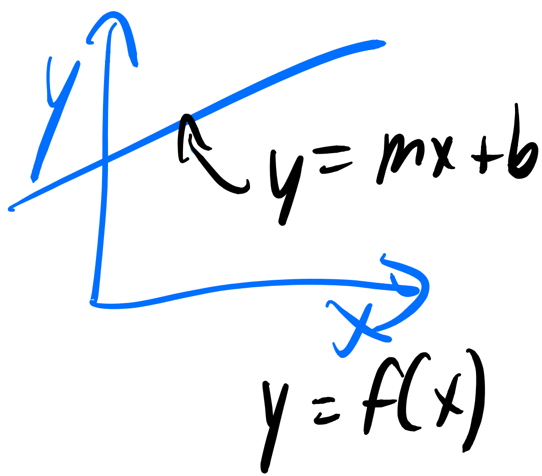


Implicit, Explicit, and Parametric equations for defining geometry

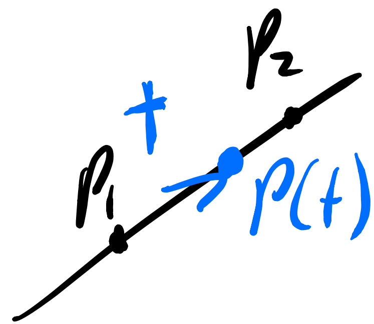
① Implicit



② Explicit



③ Parametric

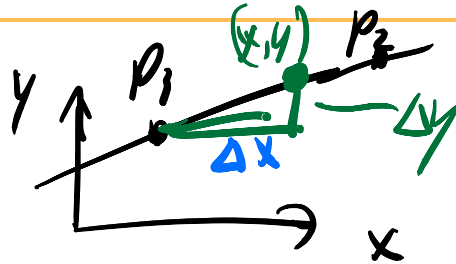


Point P(t) is a function of an underlying parameter, t. Useful to think of t as being time.

Lines and Curves

Explicit

line



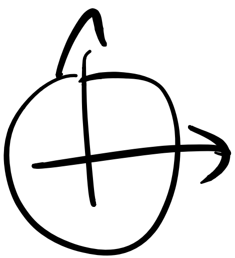
$$y = mx + b$$

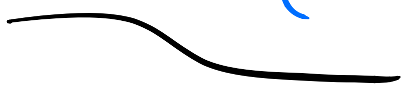
$$y = y_1 + \Delta y$$

$$= y_1 + m\Delta x$$

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

circle





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

plane

$$z = Ax + By + C$$

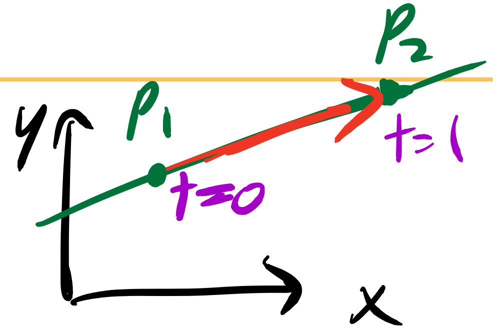
sphere

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

Lines and Curves

Parametric

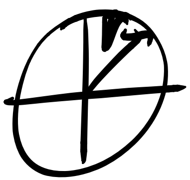
line



$$p(t) = P_1 + t(P_2 - P_1)$$
$$= (1-t)P_1 + tP_2$$

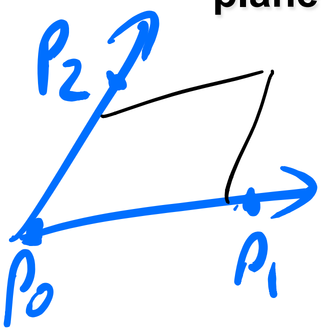
"basis functions"

circle



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

plane



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

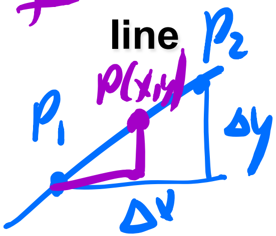
$$\begin{bmatrix} x(s,t) \\ y(s,t) \\ z(s,t) \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} + s\begin{bmatrix} x_1 - x_0 \\ y_1 - y_0 \\ z_1 - z_0 \end{bmatrix} + t\begin{bmatrix} x_2 - x_0 \\ y_2 - y_0 \\ z_2 - z_0 \end{bmatrix}$$

Lines and Curves

2D $\qquad$ 3D

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

+ line



$$y = y_1 + \Delta y$$
$$= y_1 + m\Delta x$$

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

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

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

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

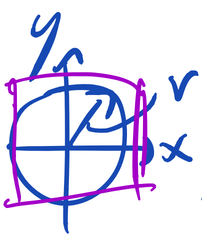
$$0 = x\underline{(y_2-y_1)} + y\underline{(x_1-x_2)}$$
$$\underline{+y_1x_2 - y_2x_1}$$

$$0 = Ax + By + C$$

−

$$y = mx + b$$
$$0 = -y + mx + b$$

circle



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

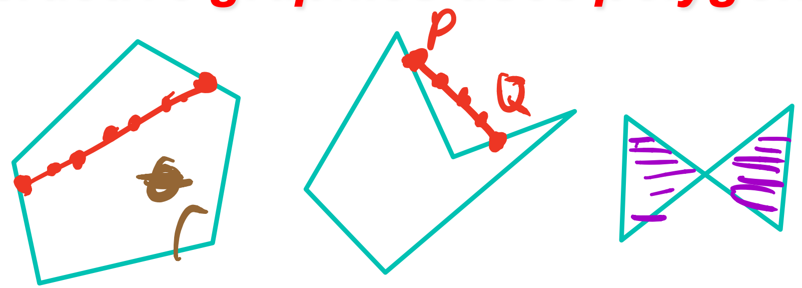
$F(x,y) = 0$ on circle

$F(x,y) < 0$ inside

$F(x,y) > 0$ outside

Polygons

Interactive graphics uses polygons



simple
convex

simple
concave

non-simple
(self-intersection)

Simple : edges do not self intersect

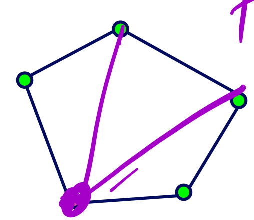
Convex : interior angles $\theta_i \leq 180°$

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

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

In practice we use triangles

- why? triangles are always: simple, convex, planar

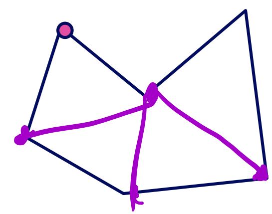


- simple convex polygons
  - *trivial to break into triangles*
- concave or non-simple polygons
  - *more effort to break into triangles*

simple polygon $O(n)$

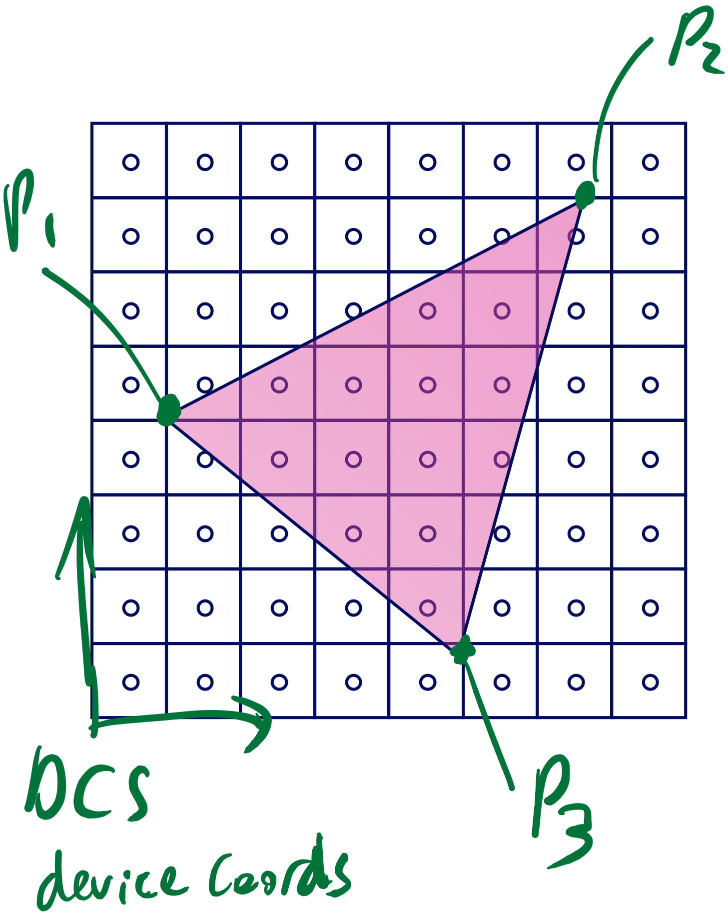
Concave polygon, possibly with holes $O(n \log n)$

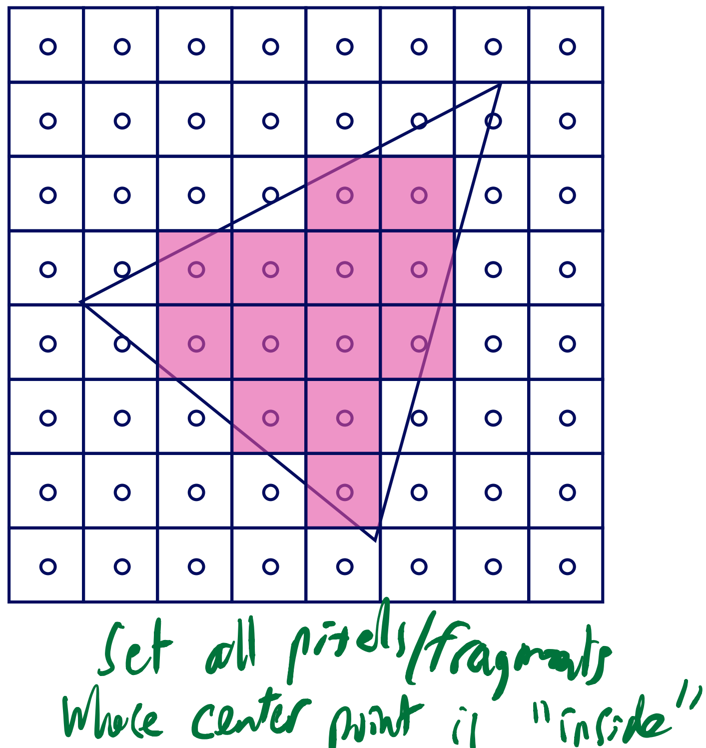
n vertices



Complex algorithms.

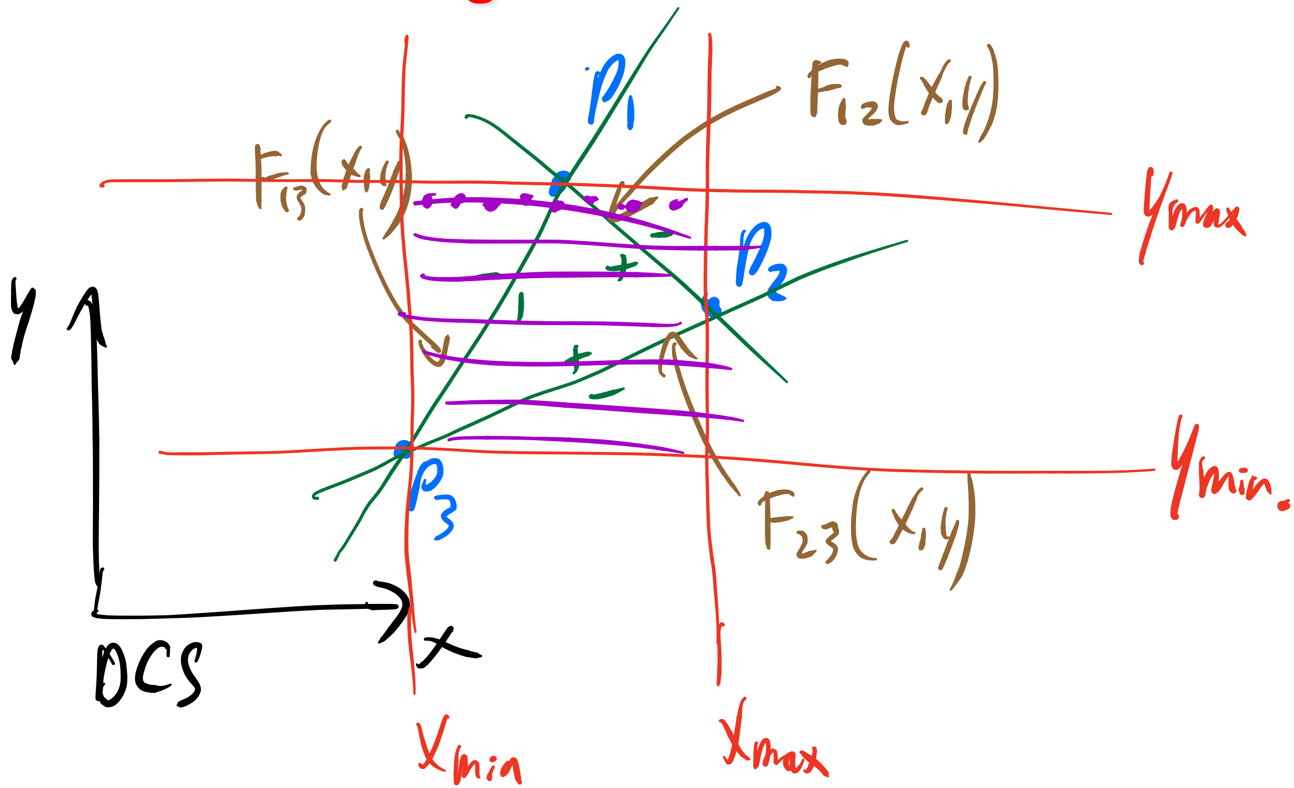
What is Scan Conversion?
(a.k.a. Rasterization)





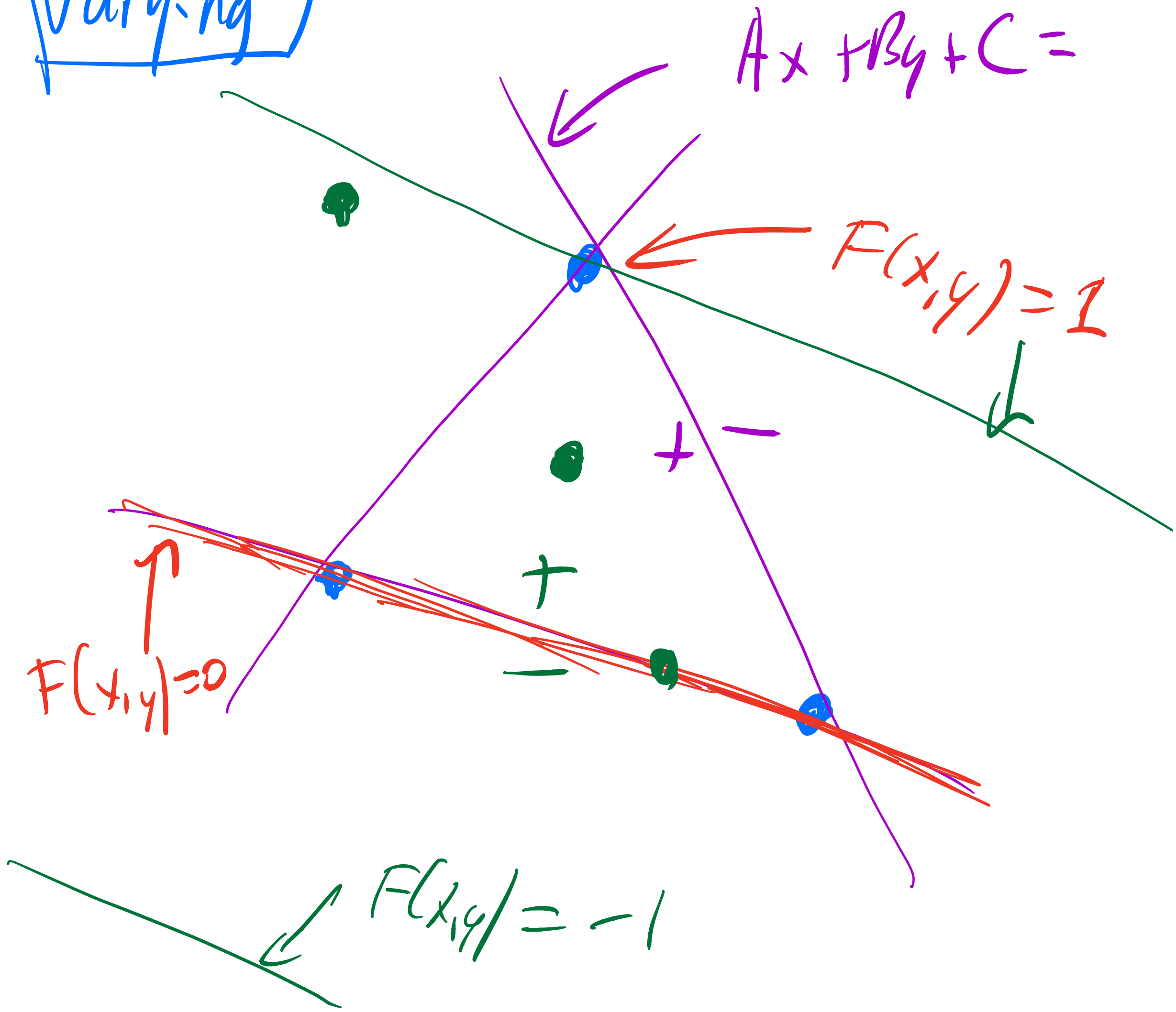
Modern Rasterization

Define a triangle as follows:

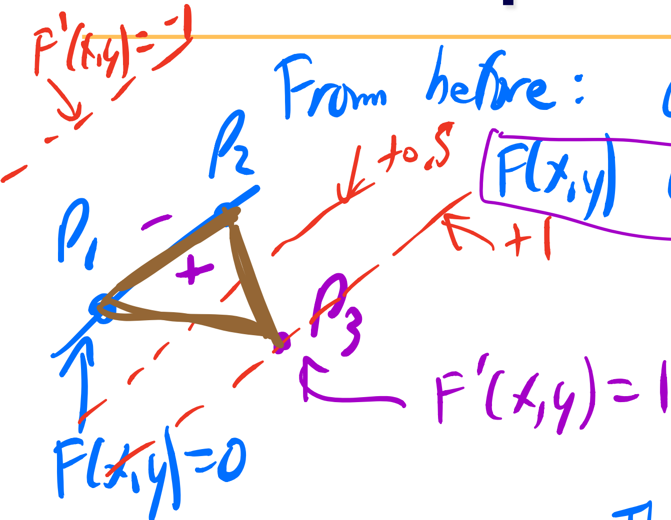


p_3

$\boxed{\text{Varying}}$



Scaled Implicit Line Equation



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

$\boxed{F(x,y) \quad 0 = Ax + By + C}$

Now develop $F'(x,y)$ such that

$F'(x_3,y_3) = +1$

Define $K = F(x_3,y_3)$

Then $F'(x,y) = \dfrac{F(x,y)}{K}$

i.e., $\boxed{F'(x,y) = \left(\dfrac{A}{K}\right)x + \left(\dfrac{B}{K}\right)y + \dfrac{C}{K}}$

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;
    }
}
```

loop over bounding box

$F_0(x,y)$

$F_1(x,y)$

$F_2(x,y)$

"inside"

Edge Equations: Code

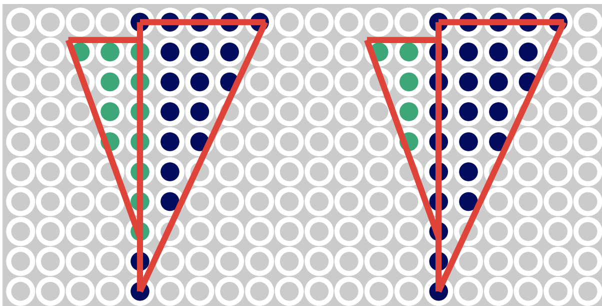
```
  // more efficient inner loop
for (int y = yMin; y <= yMax; y++) {
    float e0 = a0*xMin + b0*y + c0;
    float e1 = a1*xMin + b1*y + c1;
    float e2 = a2*xMin + b2*y + c2;
    for (int x = xMin; x <= xMax; x++) {
      if (e0 > 0 && e1 > 0 && e2 > 0)
        Image[x][y] = TriangleColor;
    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

α β γ

more efficient

Triangle Rasterization Issues

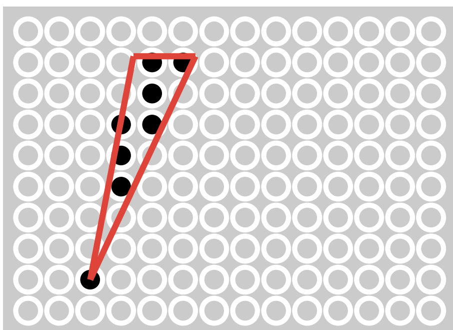
What about pixels exactly on the edge?



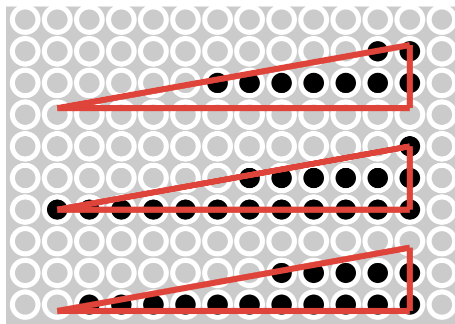
Choices:

1. Draw them both
   × result depends on order
2. Don't draw them
   × gap
3. Use a consistent but arbitrary rule
   eg: draw pixels on left or top boundaries

sliver



moving slivers



Partial Solutions: antialiasing: set a pixel "partly on" based on the fraction of coverage

Interpolation During Scan Conversion

value

- interpolate between vertices: (demo)
  - z
  - *r,g,b* *colour components*
  - *u,v* *texture coordinates*
  - N_x, N_y, N_z *surface normals*
- three equivalent ways of viewing this (for triangles)
  1. *bilinear interpolation*
  2. *plane equation*
  3. *barycentric coordinates*

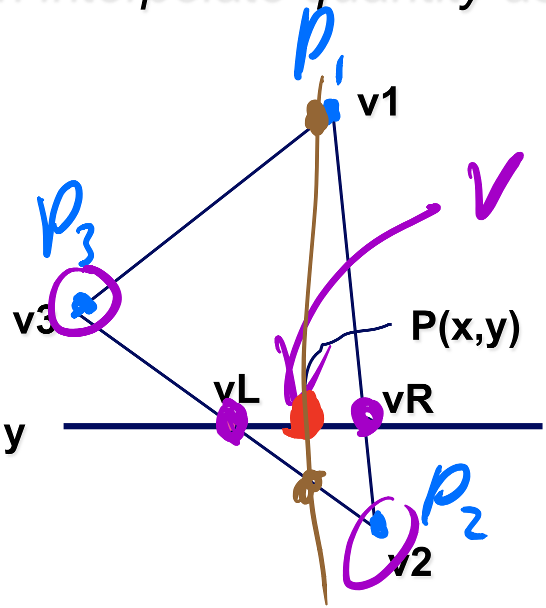
known (given) at the vertices

graphics "lingo" ← LERPs

linear interpolation

1. Bilinear Interpolation

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



fraction of the way towards P_3 from P_2

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

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

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

2. Plane Equation

- $v = Ax + By + C$

$$\left.\begin{aligned} V_1 &= Ax_1 + By_1 + C \\ V_2 &= Ax_2 + By_2 + C \\ V_3 &= Ax_3 + By_3 + C \end{aligned}\right\} \text{Solve for } A, B, C$$

At any given pixel (x, y), compute V using

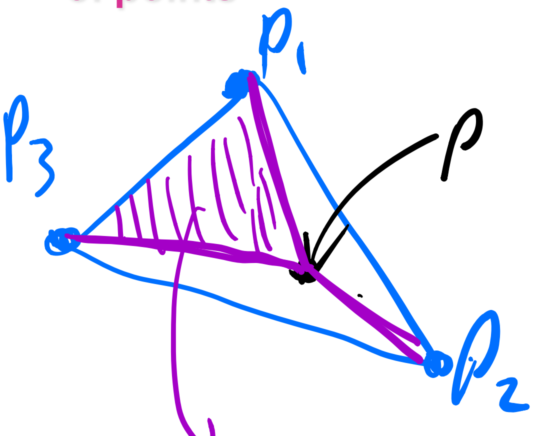
$$V = Ax + By + C$$

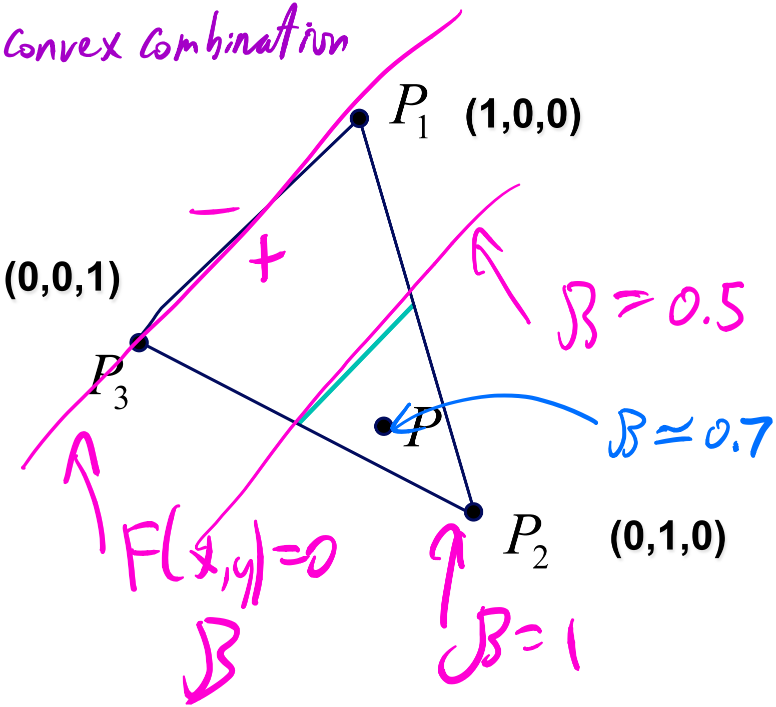
3. Barycentric Coordinates : α, β, γ

- ***weighted combination of vertices***

$$\begin{cases} P = \alpha \cdot P_1 + \beta \cdot P_2 + \gamma \cdot P_3 \\ \left.\begin{array}{l} \alpha + \beta + \gamma = 1 \\ 0 \le \alpha, \beta, \gamma \le 1 \end{array}\right\} \text{convex combination} \end{cases}$$

"convex combination of points"





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

Barycentric Coordinates

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

$$v = \alpha \cdot v_1 + \beta \cdot v_2 + \gamma \cdot v_3$$

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

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

- computing Barycentric coordinates

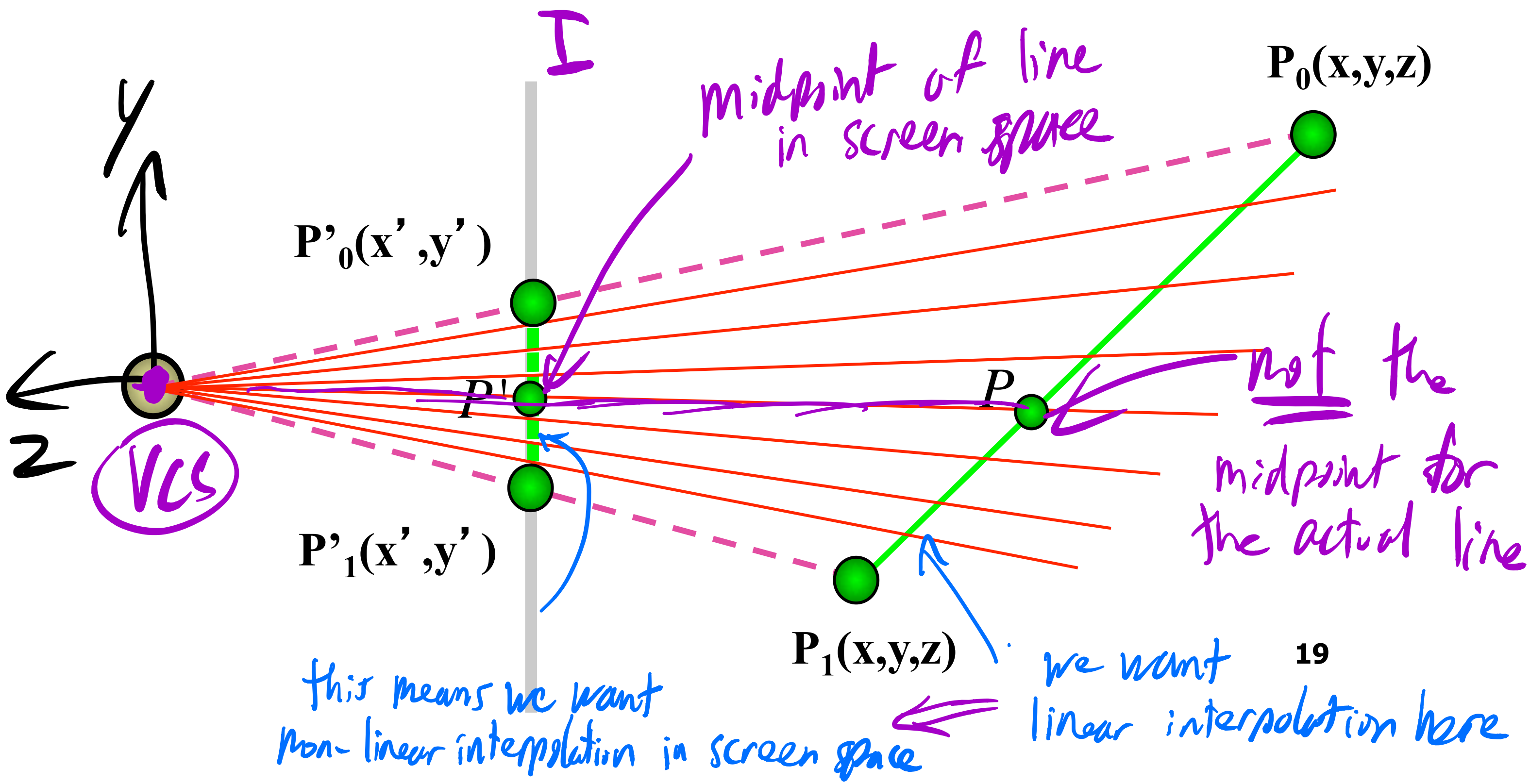
e.g., for colour interpolation

$$r = \alpha r_1 + \beta r_2 + \gamma r_3$$

$$g = \alpha g_1 + \beta g_2 + \gamma g_3$$

$$b = \alpha b_1 + \beta b_2 + \gamma b_3$$

Interpolation:
Screen vs World Space



Perspective-correct interpolation

Solution to the problem on the previous slide.

$$v = \frac{\alpha \cdot v_1 / h_1 + \beta \cdot v_2 / h_2 + \gamma \cdot v_3 / h_3}{\alpha / h_1 + \beta / h_2 + \gamma / h_3}$$

This really does the following

$$v = \frac{Barycentric(\frac{v_1}{h_1}, \frac{v_2}{h_2}, \frac{v_3}{h_3})}{Barycentric(\frac{1}{h_1}, \frac{1}{h_2}, \frac{1}{h_3})}$$