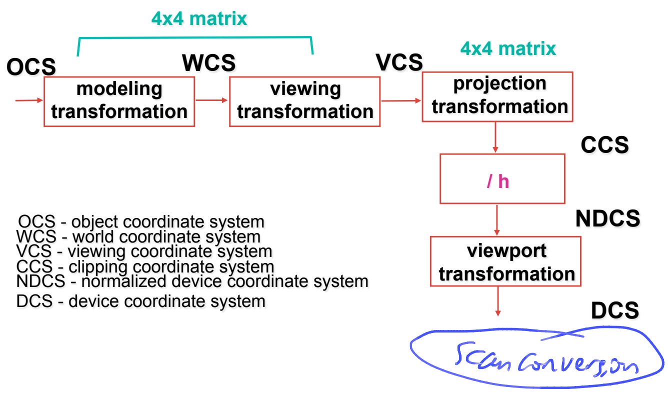
Scan Conversion

Projective Rendering Pipeline

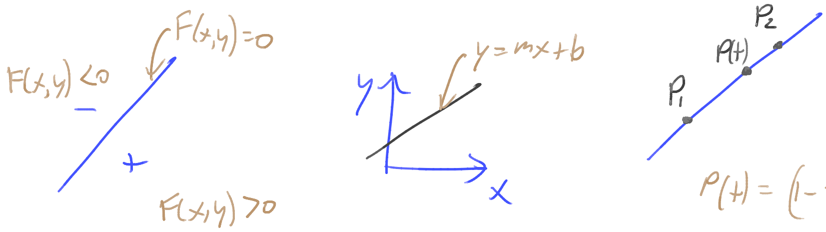


OCS - object coordinate system
WCS - world coordinate system
VCS - viewing coordinate system
CCS - clipping coordinate system
NDCS - normalized device coordinate system
DCS - device coordinate system

Implicit, Explicit, and Parametric equations for defining geometry

University of British Columbia

Implicit | Explicit | Parametric



$$P(t) = (1-t)P_1 + tP_2$$

Lines and Curves

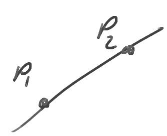
University of British Columbia

Explicit

line

$$y = mx + b$$

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



circle

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

plane

$$z = f(x,y)$$
$$= Ax + By + Cz$$

sphere

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

Lines and Curves

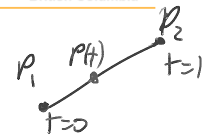
University of British Columbia

Parametric

line

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

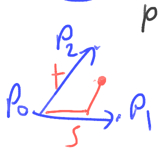
"basis functions"



circle

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

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

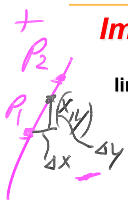


Lines and Curves

University of British Columbia

Implicit

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



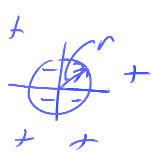
line

$$y = mx + b$$
$$0 = mx + b - y = F(x,y)$$
$$y = y_1 + m \cdot \Delta x$$
$$0 = (y_1 - y) + \left(\frac{y_2 - y_1}{x_2 - x_1}\right)\overbrace{(x - x_1)}^{\Delta x}$$
$$0 = (x_2 - x_1)(y_1 - y) + (y_2 - y_1)(x - x_1) = F(x,y)$$

$Ax + By + C$

0: pt is on line
>0 : above line
<0 : below line

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

University of British Columbia

Interactive graphics uses Polygons

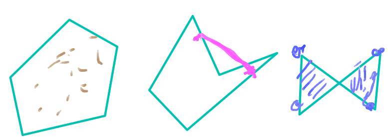
- Can represent any surface *with arbitrary accuracy*
  - *Splines, mathematical functions, ...*
- simple, regular rendering algorithms
  - *embed well in hardware*

Polygons

University of British Columbia

Basic Types

Simple: edges do not self intersect



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

Set $C \subseteq \mathbb{R}^d$ is convex if for any two points $p, q \in C$ and any $\alpha \in [0,1]$, $\alpha p + (1-\alpha) q \in C$

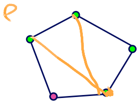
2D projection of convex 3D shapes are also convex.

From Polygons to Triangles

University of British Columbia

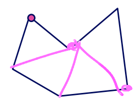
- why? triangles are always planar, always convex
- simple convex polygons
  - *trivial to break into triangles*

triangle fan



Polygon with hole

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



polygon triangulation

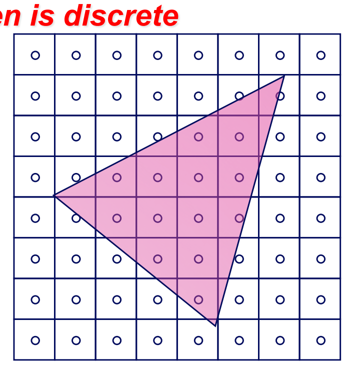
simple polygon: $O(n)$ time → complex algorithms

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

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

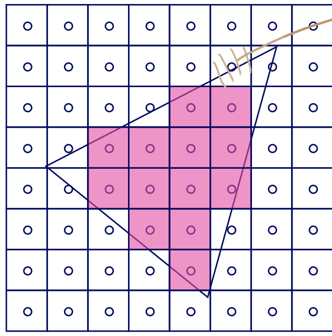
University of British Columbia

screen is discrete



University of British Columbia

one possible scan conversion



later:

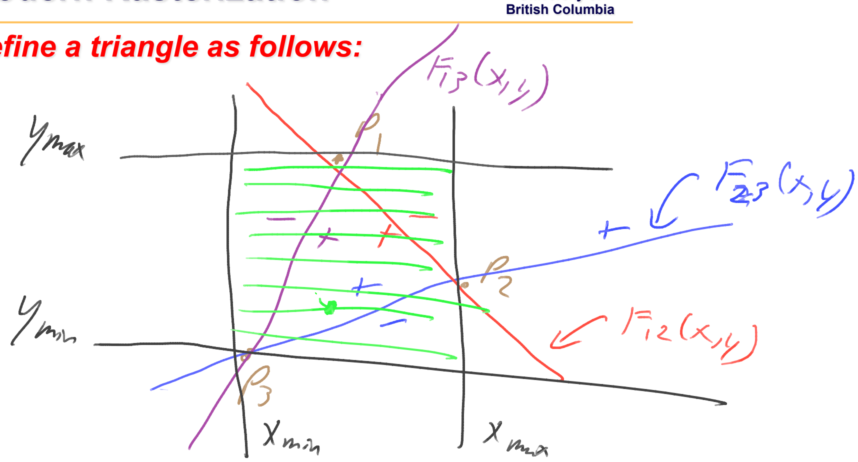
"anti aliasing"

→ also render partly covered pixels

Modern Rasterization

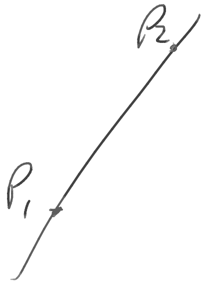
University of British Columbia

Define a triangle as follows:



Computing Edge Equations

University of British Columbia



from before:

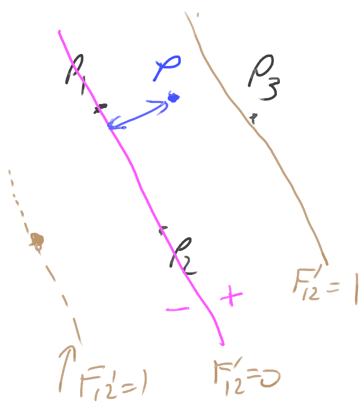
$$F(x,y) = (y_1 - y)(x_2 - x_1) + (y_2 - y_1)(x - x_1)$$

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

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

Computing Edge Equations

University of British Columbia



Now we want $F_{12}(x_3, y_3) > 0$

Let's constrain $F'_{12}(x_3, y_3) = 1$

Let $\underline{k} = F_{12}(x_3, y_3)$

Then define $F'_{12}(x,y) = \dfrac{F_{12}(x,y)}{k}$ $\qquad \dfrac{k}{k}$

$$F'_{12}(x,y) = \underbrace{\frac{A}{k}}_{A'} x + \underbrace{\frac{B}{k}}_{B'} y + \underbrace{\frac{C}{k}}_{C'}$$

Edge Equations: Code

University of British Columbia

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

University of British Columbia

```
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;
        }
}
```

$F_{12}(x,y)$

$F_{23}(x,y)$

$F_{13}(x,y)$

evaluate edge eq'ns

If "inside" wrt all three edges ..

6 x
6 + per pixel

Edge Equations: Code

University of
British Columbia

```
  // 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 edge eqns

⇒ 3 additions

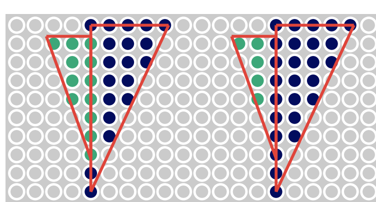
Triangle Rasterization Issues

University of
British Columbia

Exactly which pixels should be lit?

A: Those pixels inside the triangle edges

What about pixels exactly on the edge?



① Draw them
→ problem: result is dependent on order of △s

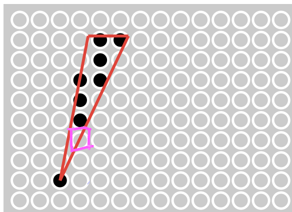
② Don't draw them
→ problem: gap

⇒ ③ use a consistent but arbitrary rule
e.g. draw pixels on a left or top boundary, but not on right or bottom.

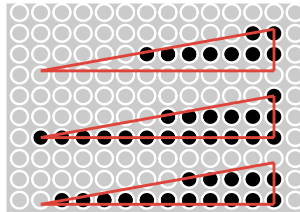
Triangle Rasterization Issues

University of British Columbia

Sliver



Moving Slivers



⇒ one solution "antialiasing"

– set a pixel "partly on" based the fraction of pixel covered by the triangle.

Interpolation During Scan Conversion

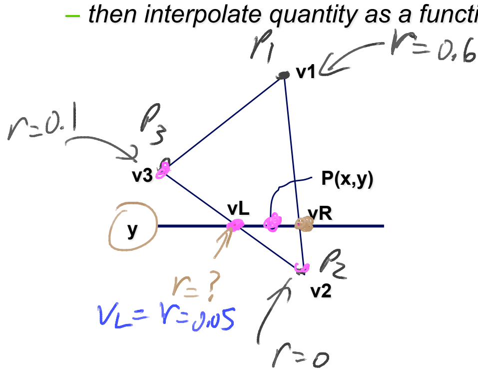
University of British Columbia

- 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*

1. Bilinear Interpolation

University of
British Columbia

- 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 V_3
$f \in [0,1]$

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

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

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

2. Plane Equation

University of
British Columbia

- v = Ax + By + C

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

At any given pixel x, y

$$V = Ax + By + C$$

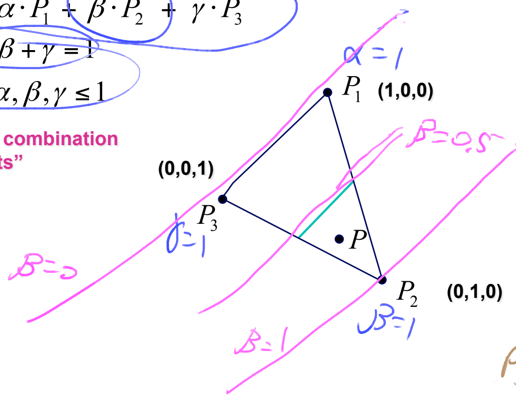
3. Barycentric Coordinates

University of
British Columbia

- *weighted combination of vertices*

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

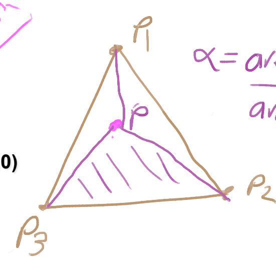
"convex combination of points"



α, β, γ are weights

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

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



$\alpha = \dfrac{\text{area}(\Delta P P_2 P_3)}{\text{area}(\Delta P_1 P_2 P_3)}$

Barycentric Coordinates

University of
British Columbia

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

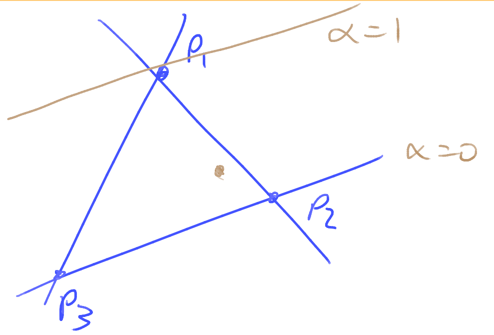
$$z = \alpha \cdot z_1 + \beta \cdot z_2 + \gamma \cdot z_3$$
$$r = \alpha \cdot r_1 + \beta \cdot r_2 + \gamma \cdot r_3$$
$$g = \alpha \cdot g_1 + \beta \cdot g_2 + \gamma \cdot g_3$$

etc.

$V = \alpha V_1 + \beta V_2 + \gamma V_3$

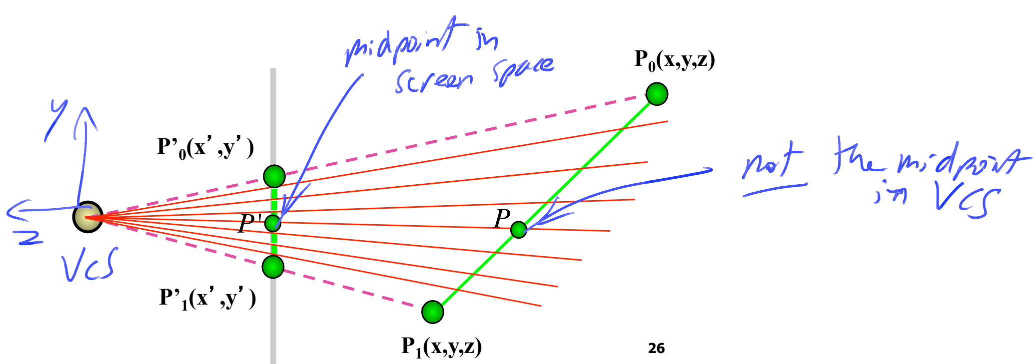
$F'_{23}(x,y)$ $F'_{13}(x,y)$ $F'_{12}(x,y)$

Computing Barycentric Coords



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

Interpolation: Screen vs World Space



what we would like, i.e, linear in WCS or VCS

University of British Columbia

In world space

$$P = Barycentric(P) \qquad P = \alpha \cdot P_1 + \beta \cdot P_2 + \gamma \cdot P_3$$

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

In screen space

$$P' = Barycentric(P') \qquad P' = \alpha \cdot P_1' + \beta \cdot P_2' + \gamma \cdot P_3'$$

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

easy to do, but is wrong, i.e, results are not perspective correct

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

(derivation is a bit messy)

per pixel division (costly compared to +, *)