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

**Implicit**

\[ F(x, y) = 0 \]

\[ F(x, y) < 0 \]

\[ F(x, y) > 0 \]

**Explicit**

\[ y = mx + b \]

**Parametric**

\[ P(t) = (t)P_1 + tP_2 \]

---

**Lines and Curves**

**Explicit**

**Line**

\[ y = mx + b \]

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

**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**

**Parametric**
- Line: \( p(t) = p_1 + t(p_2 - p_1) \)
- Circle: \( x(t) = r \cos(t) \), \( y(t) = r \sin(t) \) for \( t \in [0, 2\pi] \)
- Plane: \( p(t) = p_0 + s(p_1 - p_0) + t(p_2 - p_0) \)

**Implicit**
- Line: \( F(x, y) = 0 \)
- Circle: \( F(x, y, z) = 0 \)

**Notes:**
- A point is on a line if \( F(x, y, z) < 0 \).
- A point is below a line if \( A \cdot x + B \cdot y + C > 0 \).
Interactive graphics uses Polygons

- Can represent any surface with arbitrary accuracy
  - Splines, mathematical functions, ...
- Simple, regular rendering algorithms
  - Embed well in hardware

Even hippos are made of polygons!

Basic Types

- Simple: edges do not self intersect
- Convex
- Concave
- Non-simple (self-intersection)

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

2D projection of convex 3D shapes are also convex.
From Polygons to 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

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

*screen is discrete*
one possible scan conversion

Modern Rasterization

Define a triangle as follows:
From before:

\[
F(x, y) = (y_2 - y_1)(x_2 - x_1) + (y_2 - y_1)(x - x_1)
\]

\[
= x_1(y_2 - y_1) + y_1(x_1 - x_1) + y_2(x_2 - x_1) - x_1y_2 + x_1y_1
\]

\[
F(x, y) = Ax + By + C
\]

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

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

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

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

\[
F'_{12}(x, y) = \left( \frac{A}{k} \right)x + \left( \frac{B}{k} \right)y + \left( \frac{C}{k} \right)
\]
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
// 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;
    }
}

Triangle Rasterization Issues

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 \(\Delta s\)

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

**Sliver**

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

**Moving Slivers**

- Three equivalent ways of viewing this (for triangles)
  1. Bilinear interpolation
  2. Plane equation
  3. Barycentric coordinates

Interpolation During Scan Conversion

\( \Rightarrow \) One solution: "Antialiasing" - Set a pixel "partly on" based on the fraction of pixel covered by the triangle.
1. Bilinear Interpolation

- Interpolate quantity along LH and RH edges, as a function of y.
  - Then interpolate quantity as a function of x.

\[ V_L = \frac{y - y_L}{y_2 - y_L} (V_2 - V_L) \]
\[ V_R = \frac{x - x_L}{x_2 - x_L} (V_2 - V_L) \]
\[ V = V_L + \frac{y - y_L}{y_2 - y_L} (V_2 - V_L) \]

2. Plane Equation

- \( v = Ax + By + C \)

\[ \begin{align*}
  A x_1 + B y_1 + C &= V_1 \\
  A x_2 + B y_2 + C &= V_2 \\
  A x_3 + B y_3 + C &= V_3
\end{align*} \]

Solve for \( A, B, C \) at any given pixel \( x, y \):

\[ V = Ax + By + C \]
3. Barycentric Coordinates

• weighted combination of vertices

\[ P = \alpha \cdot P_1 + \beta \cdot P_2 + \gamma \cdot P_3 \]

\[ \alpha + \beta + \gamma = 1 \]

\[ 0 \leq \alpha, \beta, \gamma \leq 1 \]

“convex combination of points”

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

\[ \alpha = \frac{F'(x, y)}{2s} \]

Interpolation: Screen vs World Space

Not the midpoint in VCS

midpoint in Screen Space
In world space

\[ P = \text{Barycentric}(P) \]
\[ v = \text{Barycentric}(v) \]

\[ P = \alpha \cdot P_1 + \beta \cdot P_2 + \gamma \cdot P_3 \]
\[ v = \alpha \cdot v_1 + \beta \cdot v_2 + \gamma \cdot v_3 \]

In screen space

\[ P' = \text{Barycentric}(P') \]
\[ v = \text{Barycentric}(v) \]

\[ P' = \alpha \cdot P_1' + \beta \cdot P_2' + \gamma \cdot P_3' \]
\[ v = \alpha \cdot v_1 + \beta \cdot v_2 + \gamma \cdot v_3 \]

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

(derivation is a bit messy) (costly compared to +\star)

\( \frac{\alpha}{h_1} + \frac{\beta}{h_2} + \frac{\gamma}{h_3} \)