Scan Conversion

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

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

2. Explicit
   \[ y = mx + b \]

3. Parametric
   \[ p(t) \]
   \[ t = 0 \]
   \[ t = 1 \]
   \[ P_1 \]
   \[ P_2 \]
   \[ p(t) \]
   \[ t = 1.5 \]

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

Lines and Curves

**Explicit**

- line
  \[ y = mx + b \]
  \[ y = y_1 + \Delta y \]
  \[ = y_1 + m \Delta x \]
  \[ = y_1 + \left( \frac{y_2 - y_1}{x_2 - x_1} \right) (x - x_1) \]

- circle

- plane
  \[ Z = Ax + By + C \]

- sphere
  \[ Z = \pm \sqrt{r^2 - x^2 - y^2} \]
Lines and Curves

**Parametric**

- **Line**
  \[
  p(t) = p_1 + t(p_2 - p_1) = (1-t)p_1 + tp_2
  \]

- **Circle**
  \[
  x(t) = r \cos(t), \quad y(t) = r \sin(t), \quad t \in [0, 2\pi]
  \]

- **Plane**
  \[
  p(s, t) = p_0 + s(p_1 - p_0) + t(p_2 - p_0)
  \]

Three points uniquely define a 3D plane.

**Implicit**

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

- **Circle**
  \[
  O = (x, y) = (y_2 - y_1, x_2 - x_1)
  \]

- **Equation**
  \[
  F(x, y) = 0, \quad F(x, y, z) = 0
  \]

- **Circle Equation**
  \[
  O = x^2 + y^2 - r^2
  \]

- **Signs**
  \[
  O = x^2 + y^2 - r^2
  \]
  \[
  F(x, y) = 0 : \text{on circle}
  \]
  \[
  < 0 : \text{inside}
  \]
  \[
  > 0 : \text{outside}
  \]
Polygons

*Interactive graphics uses polygons*

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

**Simple:** edges do not self-intersect

**Convex:** interior angles, \( \theta_i \leq 180^\circ \)

More generally: set \( S \subseteq \mathbb{R}^d \) is convex if for any two points \( p, q \in S \) and any \( \alpha \in [0, 1] \),

\[ \alpha p + (1 - \alpha) q \in S. \]

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

- Simple polygon: \( \mathcal{O}(n) \)
- Polygon with holes: \( \mathcal{O}(n \log n) \)
What is Scan Conversion?
(a.k.a. Rasterization)

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

Modern Rasterization

Define a triangle as follows:

- Compute three implicit line equations: $F_{12}(x,y)$, $F_{13}(x,y)$, $F_{23}(x,y)$
- Compute $x_{min}, x_{max}, y_{min}, y_{max}$
- For each pixel, set pixel if $F_{12}(x,y) > 0$, $F_{13}(x,y) > 0$, $F_{23}(x,y) > 0$
**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

```c
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_{13}(x, y)
        float e2 = a2*x + b2*y + c2;  = F_{23}(x, y)
        if (e0 > 0 && e1 > 0 && e2 > 0)
            Image[x][y] = TriangleColor;
    }
}
```
Edge Equations: Code

```c
// 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;
    }
}
```

\[
F'(x,y) = A x + B y + C \\
F'(x+1,y) = A(x+1) + B y + C \\
\Delta F = A
\]

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
Triangle Rasterization Issues

**Sliver**

**Moving Slivers**

Partial solution: anti-aliasing — 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
- Three equivalent ways of viewing this (for triangles)
  1. *bilinear interpolation*
  2. *plane equation*
  3. *barycentric coordinates*
1. Bilinear Interpolation

- Interpolate quantity along LH and RH edges, as a function of $y$
  
  \[ V_L = V_2 + \frac{y - y_2}{y_3 - y_2}(V_3 - V_2) \]
  
  \[ V_R = V_2 + \frac{y - y_2}{y_1 - y_2}(V_1 - V_2) \]
  
  \[ V = V_L + \frac{x - x_L}{x_R - x_L}(V_R - V_L) \]

2. Plane Equation

- $v = Ax + By + C$

\[
\begin{align*}
A x_1 + B y_1 + C &= v_1 \\
A x_2 + B y_2 + C &= v_2 \\
A x_3 + B y_3 + C &= v_3
\end{align*}
\]

At any given pixel $(x, y)$, compute $v$ using

\[ v = A x + B y + C \]
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"

\[
B = \frac{\text{area}(\Delta P_1 P_2 P_3)}{\text{area}(\Delta P_1 P_2 P)}
\]

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

\[ V = \alpha V_1 + \beta V_2 + \gamma V_3 \]

\[ \Rightarrow \text{we get } \alpha, \beta, \gamma \text{ for free from the inside/outside tests.} \]

Interpolation:
Screen vs World Space
**Perspective-correct interpolation**

**Linear interpolation of \( \mathbf{v} \) in screen space**

- compute \( \alpha, \beta, \gamma \) such that
- \( \mathbf{v} = \text{Barycentric}(v_1, v_2, v_3) \)
- \( \mathbf{P}' = \alpha \cdot \mathbf{P}_1' + \beta \cdot \mathbf{P}_2' + \gamma \cdot \mathbf{P}_3' \)
- \( \mathbf{v} = \alpha \cdot \mathbf{v}_1 + \beta \cdot \mathbf{v}_2 + \gamma \cdot \mathbf{v}_3 \)

**Linear interpolation of \( \mathbf{v} \) in world space**

- compute \( \alpha, \beta, \gamma \) such that
- \( \mathbf{v} = \text{Barycentric}(v_1, v_2, v_3) \)
- \( \mathbf{P} = \alpha \cdot \mathbf{P}_1 + \beta \cdot \mathbf{P}_2 + \gamma \cdot \mathbf{P}_3 \)
- \( \mathbf{v} = \alpha \cdot \mathbf{v}_1 + \beta \cdot \mathbf{v}_2 + \gamma \cdot \mathbf{v}_3 \)

\[
\text{Barycentric}(\frac{v_1}{h_1}, \frac{v_2}{h_2}, \frac{v_3}{h_3}) = \frac{1/\alpha \cdot \mathbf{v}_1/h_1 + 1/\beta \cdot \mathbf{v}_2/h_2 + 1/\gamma \cdot \mathbf{v}_3/h_3}{1/\alpha + 1/\beta + 1/\gamma}
\]

"perspective correct interpolation"

- Use screen-space barycentric interpolation of \( \frac{1}{h} \)

- i.e., \( \frac{1}{h} = \alpha \left( \frac{\mathbf{v}_1}{h_1} \right) + \beta \left( \frac{\mathbf{v}_2}{h_2} \right) + \gamma \left( \frac{\mathbf{v}_3}{h_3} \right) \)

- Then interpolate using

\[
\mathbf{V} = \frac{\alpha \left( \frac{\mathbf{V}_1}{h_1} \right) + \beta \left( \frac{\mathbf{V}_2}{h_2} \right) + \gamma \left( \frac{\mathbf{V}_3}{h_3} \right)}{1/h}
\]