Scan Conversion (fixed function)

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

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

3. Parametric
   \[ y = f(t) \]

Point \( P(t) \) is a function of an underlying parameter. Useful to think of \( t \) as being time.
Lines and Curves

Explicit

- line
  \[ y = mx + b \]
  \[ y = y_1 + \Delta y = y_1 + m \Delta x \]
  \[ y = y_1 + \frac{(y_2-y_1)(x-x_1)}{(x_2-x_1)} \]
  \[ y = \pm \sqrt{r^2 - x^2} \]

- circle

- plane
  \[ z = Ax + By + C \]
  \[ z = \pm \sqrt{r^2 - x^2 - y^2} \]

- sphere
Lines and Curves

**Parametric**

- **line**
  \[ \mathbf{P}(t) = \mathbf{P}_1 + t(\mathbf{P}_2 - \mathbf{P}_1) \]
  \[ = (1-t)\mathbf{P}_1 + t\mathbf{P}_2 \]

- **circle**
  \[ x(t) = r \cos(t) \quad \text{and} \quad y(t) = r \sin(t) \quad \text{with} \quad t \in [0, 2\pi] \]

- **plane**
  \[ \mathbf{P}(s,t) = \mathbf{P}_0 + s(\mathbf{P}_1 - \mathbf{P}_0) + t(\mathbf{P}_2 - \mathbf{P}_0) \]
  \[ \begin{bmatrix} x(s,t) \\ y(s,t) \\ z(s,t) \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} + s \begin{bmatrix} x_1 - x_0 \\ y_1 - y_0 \\ z_1 - z_0 \end{bmatrix} + t \begin{bmatrix} x_2 - x_0 \\ y_2 - y_0 \\ z_2 - z_0 \end{bmatrix} \]
Lines and Curves

**Implicit**

2D

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

3D

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

Line

\[ y = y_1 + \Delta y = y_1 + m \Delta x \]

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

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

Circle

\[ r^2 = x^2 + y^2 \]

\[ 0 = x^2 + y^2 - r^2 \]

\[ F(x, y) = 0 \text{ on circle} \]

\[ F(x, y) < 0 \text{ inside} \]

\[ F(x, y) > 0 \text{ outside} \]

\[ 0 = x(y_2 - y_1) + y(x_1 - x_2) + y_1 x_2 - y_2 x_1 + y_1 x_1 \]

\[ O = Ax + By + C \]
Polygons

Interactive graphics uses polygons

Simple: edges do not self intersect
Convex: interior angles \( \theta_i \leq 180^\circ \)

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

Set all pixels/fragments whose center point is "inside"
Modern Rasterization

Define a triangle as follows:

- $F_{13}(x, y)$
- $F_{12}(x, y)$
- $F_{23}(x, y)$

Points $P_1$, $P_2$, $P_3$ with coordinates $(x_{min}, y_{min})$, $(x_{max}, y_{max})$. DCS (DirectConnectedSystem) axis.
Scaled Implicit Line Equation

From before:  \( O = x(y_2 - y_1) + y(x_1 - x_2) + y_1x_2 - y_2x_1 \)

\[ F(x_1, y_1) = A x + B y + C \]

Now develop \( F(x_1, y_1) \) such that \( F'(x_3, y_3) = 1 \)

Define \( K = F(x_3, y_3) \)

Then \( F'(x_1, y_1) = \frac{F(x_1, y_1)}{K} \)

\[ F'(x_1, y_1) = \left( \frac{A}{K} \right) x + \left( \frac{B}{K} \right) y + \frac{C}{K} \]
Edge Equations: Code

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

What about pixels exactly on the edge?

Choice:
1. Draw them both, x result depends on order
2. Don’t draw them, x gop
3. Use a consistent but arbitrary rule, e.g., draw pixels on left or top boundaries

Partial Solutions: anti-aliasing: set a pixel “partly on” based on the fraction of coverage.
Interpolation During Scan Conversion

• interpolate between vertices: (demo)
  – $z$
  – $r, g, b$ \textit{colour components}
  – $u, v$ \textit{texture coordinates}
  – $N_x, N_y, N_z$ \textit{surface normals}

• three equivalent ways of viewing this (for triangles)
  1. \textit{bilinear interpolation}
  2. \textit{plane equation}
  3. \textit{barycentric coordinates}

\textit{Known (given) at the vertices}
1. Bilinear Interpolation

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

\[
\begin{align*}
V_L &= V_2 + \frac{(y - y_2)}{(y_3 - y_2)} (V_3 - V_2) \\
V_R &= V_2 + \frac{(y - y_2)}{(y_1 - y_2)} (V_1 - V_2) \\
V &= V_L + \frac{(x - x_L)}{(x_R - x_L)} (V_R - V_L)
\end{align*}
\]
2. Plane Equation

\[ v = Ax + By + C \]

\[
\begin{align*}
V_1 &= Ax_1 + By_1 + C \\
V_2 &= Ax_2 + By_2 + C \\
V_3 &= Ax_3 + By_3 + C
\end{align*}
\]

\{ \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

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

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

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

"convex combination of points"
Barycentric Coordinates

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

\[ v = \alpha \cdot v_1 + \beta \cdot v_2 + \gamma \cdot v_3 \]

- Computing Barycentric coordinates:

\[ \alpha, \beta, \gamma \in [0, 1] \]
\[ \alpha + \beta + \gamma = 1 \]

\[ r = \alpha r_1 + \beta r_2 + \gamma r_3 \]
\[ g = \alpha g_1 + \beta g_2 + \gamma g_3 \]
\[ b = \alpha b_1 + \beta b_2 + \gamma b_3 \]

E.g., for color interpolation.
Interpolation:
Screen vs World Space

\[ P_0(x,y,z) \]

\[ P_0'(x',y') \]

\[ P_1(x,y,z) \]

\[ P_1'(x',y') \]

This means we want non-linear interpolation in screen space.
Perspective-correct interpolation

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

\[ v = \frac{Barycentric\left(\frac{v_1}{h_1}, \frac{v_2}{h_2}, \frac{v_3}{h_3}\right)}{Barycentric\left(\frac{1}{h_1}, \frac{1}{h_2}, \frac{1}{h_3}\right)} \]