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

modeling transformation
viewing transformation
projection transformation
viewport transformation

vertex shader
fragment shader
Implicit, Explicit, and Parametric equations for defining geometry

1. Implicit
   \[ F(x,y) > 0 \]
   \[ F(x,y) \leq 0 \]

2. Explicit
   \[ y = mx + b \]
   \[ y = f(x) \]

3. Parametric
   \[ p(t) = [x(t), y(t)] \] for \( t = 0 \), \( t = 1 \), \( t = 0.5 \)
Lines and Curves

Explicit

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

- circle
  \[ y = \pm \sqrt{r^2 - x^2} \]

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

Parametric

**line**

\[ \mathbf{p}(t) = \mathbf{p}_1 + t(\mathbf{p}_2 - \mathbf{p}_1) = (1-t)\mathbf{p}_1 + t\mathbf{p}_2 \quad t \in [0,1] \]

**circle**

\[ \begin{align*}
x(t) &= \mathbf{r} \cos(t) \\
y(t) &= \mathbf{r} \sin(t)
\end{align*} \quad t \in [0,2\pi] \]

**plane**

\[ \mathbf{p}(s,t) = \mathbf{p}_0 + s(\mathbf{p}_1 - \mathbf{p}_0) + t(\mathbf{p}_2 - \mathbf{p}_0) \]

\[(s,t) = (1,1) = \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} + s \begin{bmatrix} x_1-x_0 \\ y_1-y_0 \\ z_1-z_0 \end{bmatrix} + t \begin{bmatrix} x_2-x_0 \\ y_2-y_0 \\ z_2-z_0 \end{bmatrix} \]
Lines and Curves

**Implicit**

- **line**
  \[ F(x, y) = 0 \]
  \[ y = y_1 + \frac{(y_2 - y_1)(x - x_1)}{x_2 - x_1} \]
  \[ 0 = (y_1 - y)(x_2 - x_1) + (y_2 - y_1)(x - x_1) \]
  \[ 0 = Ax + By + C \]

- **circle**
  \[ r^2 = x^2 + y^2 \]
  \[ 0 = x^2 + y^2 - r^2 = F(x, y) \]

- **signed distance**
  \[ d = Ax + By + C \]
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 iff for any $P, Q \in C$ and any $\alpha$,

$\Rightarrow$ triangles are always simple & convex $\alpha P + (1-\alpha)Q \in C$
In practice we use triangles

- They are always planar, always convex

- why?

- simple convex polygons
  - trivial to break into triangles

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

- $O(n)$ for a simple polygon
- $O(n \log n)$ for a concave polygon possibly with holes
What is Scan Conversion? (a.k.a. Rasterization)

better solution for edges
(weighted combination of
triangle color and background)

set all pixels/fragments
whose center point lies "inside"

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

Define a triangle as follows:

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

AABB: axis-aligned bounding box

\[ y_{\text{max}} = y_1 \]

\[ y_{\text{min}} = y_2 \]

\[ x_{\text{max}} = x_2 \]

\[ x_{\text{min}} = x_3 \]
Scaled Implicit Line Equation

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$$

Define $K = F(x_3, y_3)$

Then $F(x, y) = F(x_3, y_3)$

Now define $F(x, y)$ such that $F(x, y) = 1$

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

$$2x - 2xy + y^2 = x_2 y_2$$
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;
    }
}
```

CALL FRAGMENT SHADER
// 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?

- **sliver**
- **moving slivers**
Interpolation During Scan Conversion

- interpolate values from vertices to interior pixels:
  - \( z \) depth values
  - \( r,g,b \) colour components
  - \( u,v \) texture coordinates
  - \( N_x, N_y, N_z \) surface normals

- equivalent ways of viewing this (for triangles)
  - plane equation
  - barycentric coordinates
  - bilinear interpolation
Plane Equation

• $v = Ax + By + C$

scalar quantity that we know at the vertices and wish to interpolate

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

solve for $A$, $B$, $C$
Barycentric Coordinates

- weighted combination of vertices

\[ P = \alpha \cdot P_1 + \beta \cdot P_2 + \gamma \cdot P_3 \]
\[ \alpha + \beta + \gamma = 1 \]
\[ 0 \leq \alpha, \beta, \gamma \leq 1 \]

To interpolate a scalar quantity, \( v \), whose values are known at the vertices:

\[ v = \alpha \cdot v_1 + \beta \cdot v_2 + \gamma \cdot v_3 \]
1. Bilinear Interpolation

- interpolate quantity along LH and RH edges, as a function of $y$
  - then interpolate quantity as a function of $x$
Interpreting Barycentric Coordinates

- (1) coordinate system using edges as basis vectors
- (2) fractional distances
- (3) fractional areas
- (4) fractional weights

\[ p = \alpha P_1 + \beta P_2 + \gamma P_3 \]

\[ p = P_3 + \alpha \frac{\mathbf{r}_1}{|\mathbf{r}_1|} + \beta \frac{\mathbf{r}_2}{|\mathbf{r}_2|} \]

where \( \mathbf{r}_1 = P_1 - P_3 \)

\( \mathbf{r}_2 = P_2 - P_3 \)

\[ p = P_3 + \alpha (P_1 - P_3) + \beta (P_2 - P_3) \]

\[ p = \alpha P_1 + \beta P_2 + (1 - \alpha - \beta) P_3 \]

\( \alpha \) is the fractional distance away from the line \( P_2P_3 \) towards \( P_1 \).
Imagine splitting 1 kg of clay into lumps $m_1, m_2, m_3$ placed at the vertices. Then center-of-mass is at point $P$

$$(m_1, m_2, m_3) = (\alpha, \beta, \gamma)$$
Interpolation:
Screen vs World Space

As a result, we get unwanted artifacts in the interpolation.
Perspective-correct interpolation

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

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

\[ v = \frac{1}{Barycentric(\frac{1}{h_1}, \frac{1}{h_2}, \frac{1}{h_3})} \]

\[ Barycentric(a_1, a_2, a_3) = \alpha a_1 + \beta a_2 + \gamma a_3 \]

\[ h_1, h_2, h_3 \text{ are the CCS } h \text{ values, i.e., before } \frac{1}{h} \]