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

4x4 matrix

OCS → WCS

transformations

WCS → VCS

OCS

VCS

WCS

 CCS

 projection

transformation

NDCS

DCS

Lines and Curves

Explicit

- line: \[ \begin{align*}
  y &= mx + b \\
  y &= \frac{(y_2 - y_1)(x - x_1) + y_1}{x_2 - x_1}
\end{align*} \]

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

Implicit

- line: \[ F(x,y) = (x-x_0)(y-y_0) dx + (y-y_0) dy = 0 \]

- plane: \[ z = Fx + Fy + D \]

- sphere: \[ z = \pm \sqrt{r^2 - x^2 - y^2} \]

Lines and Curves

Parametric

- line: \[ \begin{align*}
  x(t) &= x_0 + t(x_1 - x_0) \\
  y(t) &= y_0 + t(y_1 - y_0) \\
  t &\in [0,1] \\
  P(t) &= P_0 + t(P_1 - P_0) \\
  P(t) &= (1-t)P_0 + tP_1
\end{align*} \]

- circle: \[ \begin{align*}
  x(\theta) &= r \cos(\theta) \\
  y(\theta) &= r \sin(\theta) \\
  &\theta \in [0,2\pi]
\end{align*} \]

- plane: \[ P(1,t) = P_0 + t(P_1 - P_0) + t(P_2 - P_0) \]

Polygons

- 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!
From Polygons to Triangles

- why? triangles are planar and convex
- simple convex polygons
  - break into triangles, trivial
  - `glBegin(GL_POLYGON) ... glEnd()`
- concave or non-simple polygons
  - break into triangles, more effort
  - `gluNewTess(), gluTessCallback(), ...`

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

- screen is discrete

Scan Conversion

A General Algorithm

- intersect each scanline with all edges
- sort intersections in x
- calculate parity to determine in/out
- fill the ‘in’ pixels

Edge Walking

past graphics hardware

- works for arbitrary polygons
- efficiency improvement:
  - exploit row-to-row coherence using “edge table”

\[
\text{scanTrapezoid}(x_L, x_R, y_T, y_B, \Delta x_L, \Delta x_R)\]

\[
\frac{y_T - y_B}{\Delta x_R} = \frac{1}{\Delta x_R} \implies m_R = \frac{1}{\Delta x_R}
\]
Edge Walking Triangles

Issues
- many applications have small triangles
  - setup cost is non-trivial
- clipping triangles produces non-triangles

Using Edge Equations

- clip to window by checking the bounds to the window
Edge Equations

- So...we can find edge equation from two verts.
- Given $P_0$, $P_1$, $P_2$, what are our three edges?
  
  How do we make sure the half-spaces defined by the edge equations all share the same sign on the interior of the triangle?
- A: Be consistent (Ex: $[P_0\ P_1\ P_2\ P_0]$)
  
  How do we make sure that sign is positive?
- A: Test, and flip if needed ($A=A, B=B, C=C$)

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

```
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;
    }
}
```

Edge Equations: Code

```
// 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?

- Draw them: order of triangles matters (it shouldn’t)
- Don’t draw them: gaps possible between triangles

We need a consistent (if arbitrary) rule

- Example: draw pixels on left or top edge, but not on right or bottom edge
Triangle Rasterization Issues

Sliver

Moving Slivers

Triangle Rasterization Issues

Shared Edge Ordering

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 methods (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 \( y \)
  - then interpolate quantity as a function of \( x \)

2. Plane Equation

• \( v = Ax + By + C \)

\[
\begin{align*}
    \text{Plane eq:} & \quad Ax + By + Cz + D = 0 \\
    & \Rightarrow A(x-x_1) + B(y-y_1) + C(z-z_1) = 0 \\
    & \Rightarrow v = A(x - x_1) + B(y - y_1) + C(z - z_1) \\
    \text{Computing } & \quad \begin{pmatrix} A & B & C \end{pmatrix} \begin{pmatrix} x_1 & y_1 & z_1 \end{pmatrix} \\
    & \Rightarrow \begin{pmatrix} x_2 & y_2 & z_2 \end{pmatrix} = (P-P_1) \times (P-P_2) \\
    & \Rightarrow N \cdot \rho = 0 \\
    & \Rightarrow \rho = -N \cdot \rho \\
    & \text{choose any } \rho;
\end{align*}
\]
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"

- 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 Coordinates

1. Compute equation for \( \overline{RB} \)
   \[ F(x, y) = Ax + By + C \]

2. Compute \( F(a) = \ell \) such that
   \[ F(a) = 1 \]
   \[ \ell = \sqrt{F(a)} \]
   \[ \kappa = \frac{1}{\ell} \]
   \[ a = F(x, y) = Ax + By + C \]
   where \( \ell = \kappa \cdot C \)
   \[ C = \kappa \cdot C \]

Note that we could use \( a, \overline{RB} \) for the input test for scan conversion.