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

Lines and Curves

Explicit

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

Parametric

- Line
  \[ x(t) = x_0 + t(x_1 - x_0) \]
  \[ y(t) = y_0 + t(y_1 - y_0) \]
  \[ P(t) = P_0 + t(P_1 - P_0) \]
  \[ P(t) = (1-t)P_0 + tP_1 \]
- Circle
  \[ x(\theta) = r \cos(\theta) \]
  \[ y(\theta) = r \sin(\theta) \]
  \[ \theta \in [0, 2\pi] \]

Implicit

- Line
  \[ F(x, y) = (x - x_0)dy - (y - y_0)dx = 0 \]
  \[ F(x, y) > 0 \] is on line
  \[ F(x, y) < 0 \] is below line
- Circle
  \[ F(x, y) = x^2 + y^2 - r^2 \]
  \[ F(x, y) = 0 \] is on circle
  \[ F(x, y) > 0 \] is outside
  \[ F(x, y) < 0 \] is inside

Polygons

Basic Types

- Simple convex
- Simple concave
- Non-simple (self-intersection)

Polygons: Vertex List

- Ordered list of references to a vertex list

<table>
<thead>
<tr>
<th>Face Index</th>
<th>Vertex List</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0,2,3,1,1</td>
</tr>
<tr>
<td>1</td>
<td>1,3,7,5,1</td>
</tr>
<tr>
<td>2</td>
<td>2,5,7,6,4</td>
</tr>
<tr>
<td>3</td>
<td>3,4,6,2,3</td>
</tr>
<tr>
<td>4</td>
<td>4,0,1,5</td>
</tr>
<tr>
<td>5</td>
<td>5,1,0,1</td>
</tr>
<tr>
<td>6</td>
<td>6,0,0,0</td>
</tr>
<tr>
<td>7</td>
<td>7,1,0,0</td>
</tr>
</tbody>
</table>
Triangle Meshes: Winged Edge Data Structure

Mesh Operations
- Find boundary edges
- Edge collapse

Edge:
- Vertex *start: \(x\)
- Vertex *end: \(y\)
- Face *left: \(F_l\)
- Face *right: \(F_r\)
- Edge *leftPred: \(P_l\)
- Edge *leftSucc: \(S_l\)
- Edge *rightPred: \(P_r\)
- Edge *rightSucc: \(S_r\)

Winged Edge Data Structure

Features
- Data structure for polygonal meshes
- Holds required topological information
- Relatively storage efficient
- Works for arbitrary polygons without holes
  - not just triangles
- Represents only manifold surfaces

Primitives
- Vertices
- Edges
- Faces (polygons)
  - in many cases: only triangles

Topological Relationships:
- Faces adjacent to an edge
- Edges adjacent to a face
- Vertices adjacent to a face
  - ...

Scan Conversion

Basic Algorithm
- Intersect each scanline with all edges
- Sort intersections in \(x\)
- Calculate parity to determine in/out
- Fill the ‘in’ pixels

Scan Conversion
- Exclude horizontal edges
- Vertices lying on scanlines
  - local extrema: leave as is (count twice)
  - otherwise: shorten edge (count once)
**Scan Conversion**

**Exploit Spatial Coherence**
- neighboring scanlines likely to cross same edges

![Diagram of Exploit Spatial Coherence](image)

**Building the Edge Table**
- for each edge
  - skip if horizontal
  - if not local extrema, shorten upper vertex
  - add edge to linked list for scanline corresponding to lower vertex, storing
    - end y: last scanline to consider
    - start x: starting x coord for edge
    - 1/m: use for incrementing x

**Using the Edge Table**
- maintain an Active Edge List (AEL)
  - for each scanline
    - add new edges to AEL from edge table
    - if (AEL != NIL) {
      - sort AEL by x
      - fill pixels between edge pairs
      - delete finished edges
      - update edge x values
    }
  - OpenGL only guarantees that simple convex polygons are drawn correctly

**Scan Converting Z**
- can use linear interpolation
  - but this is not orientation independent for >3 vertices

**Scan Converting a Trapezoid**
- easy because of continuous L and R edges

\[ \text{scanTrapezoid}(x_L, x_R, y_L, y_R, \Delta x_L, \Delta x_R) \]

**Scan Converting Triangles**
- most graphix APIs will internally break polygons into triangles
- split at the mid-vertex to given two regions with continuous left and right edges

\[ \text{scanTrapezoid}(x_2, x'_2, y_2, y'_2, \frac{1}{m_21}, \frac{1}{m_2}) \]
\[ \text{scanTrapezoid}(x_3, x'_3, y_3, y'_3, \frac{1}{m_31}, \frac{1}{m_3}) \]
Scan Conversion: the big picture (recap)

- given vertices in DCS, fill in the pixels
  - arbitrary polygons (non-simple, non-convex)
  - triangles
  - build edge table
  - for each scanline
    - obtain list of intersections, i.e., AEL
    - use parity test to determine in/out and fill in the pixels

Scan Conversion: the big picture

- OpenGL
  - simple convex polygons
    - break into triangles, trivial
    - \texttt{glBegin(GL_POLYGON)} ... \texttt{glEnd()}
  - concave or non-simple polygons
    - break into triangles, more effort
    - \texttt{gluNewTess()}, \texttt{gluTessCallback()}, ...

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 left-hand and right-hand edges, as a function of \( y \)
  - then interpolate quantity as a function of \( x \)
- only triangles guarantee orientation-independent interpolation
  - compute efficiently by using known values at previous scanline, previous pixel

Bilinear Interpolation Across a Trapezoid

- this example: interpolate \( z \)

```latex
\text{for } (y = y_b \text{ to } y_f) \{ \quad \text{\# step up scanlines} \\
\quad \text{dz} = (z_b - z_f) / (x_f - x_b) \\
\quad z = z_i \\
\quad \text{for } (x = x_i \text{ to } x_f) \{ \quad \text{\# step across pixels} \\
\quad \text{setPixel}(x,y,r,g,b,z); \\
\quad z += \text{dz} \\
\quad x_L = x_L + \Delta x_L \\
\quad x_R = x_R + \Delta x_R \\
\quad z_L = z_L + \Delta z_L \\
\quad z_R = z_R + \Delta z_R \\
\quad \} \\
\text{\}}
```

Bilinear Interpolation Across a Trapezoid

- extra parameters
2. Plane Equation

- \( z = f(x, y) \)
  - Implicit plane equation
  - Parametric plane equation
  - Explicit plane equation

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

- **how to compute** \( \alpha, \beta, \gamma \)
  - use bilinear interpolation or plane equations
  \[
  \alpha = a \cdot x + b \cdot y + c \\
  \beta = \ldots
  \]

  - 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.

Next class...

- clipping