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

**Parametric**

- **line**
  \[ x(t) = x_0 + t(x_1 - x_0) \]
  \[ y(t) = y_0 + t(y_1 - y_0) \]
  \[ t \in [0,1] \]
  \[ P(t) = P_0 + t(P_1 - P_0) \]
  \[ P(t) = (1-t)P_0 + t P_1 \]

- **circle**
  \[ x(\theta) = r \cos(\theta) \]
  \[ y(\theta) = r \sin(\theta) \]
  \[ \theta \in [0,2\pi] \]

---

Lines and Curves

**Implicit**

- **line**
  \[ F(x, y) = (x - x_0) dy - (y - y_0) dx \]
  \[ F(x, y) = 0 \] (\(x,y\) is on line)
  \[ F(x, y) > 0 \] (\(x,y\) is below line)
  \[ F(x, y) < 0 \] (\(x,y\) is above line)

- **circle**
  \[ F(x, y) = x^2 + y^2 - r^2 \]
  \[ F(x, y) = 0 \] (\(x,y\) is on circle)
  \[ F(x, y) > 0 \] (\(x,y\) is outside)
  \[ F(x, y) < 0 \] (\(x,y\) 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>faces</th>
<th>vertex list</th>
<th>vertex list # x,y,z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0,2,3,1</td>
<td>0,1,1</td>
</tr>
<tr>
<td>1</td>
<td>1,3,7,5</td>
<td>1,1,1</td>
</tr>
<tr>
<td>2</td>
<td>5,7,6,4</td>
<td>2,0,1</td>
</tr>
<tr>
<td>3</td>
<td>4,6,2,0</td>
<td>3,1,0</td>
</tr>
<tr>
<td>4</td>
<td>4,0,1,5</td>
<td>4,0,1</td>
</tr>
<tr>
<td>5</td>
<td>2,6,7,3</td>
<td>5,1,0</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>6,0,0</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>7,1,0</td>
</tr>
</tbody>
</table>
Triangle Meshes: Winged Edge Data Structure

Mesh Operations
Find boundary edges

Edge collapse

Winged Edge Data Structure

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
- …
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

**Triangle Meshes:**
**Winged Edge Data Structure**

**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 *right Succ; (S_r)`
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 scan conversion process](https://via.placeholder.com/150)

Scan Conversion

**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

![Diagram of edge table construction](https://via.placeholder.com/150)
Scan Conversion

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

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

Scan Converting Triangles

- most graphics APIs will internally break polygons into triangles
- split at the mid-vertex to give two regions with continuous left and right edges

\[
\text{scanTrapezoid}(x_3, x_m, y_3, y_m, \frac{1}{m_{13}}, \frac{1}{m_{12}})
\]

\[
\text{scanTrapezoid}(x_2, x_2, y_2, y_3, \frac{1}{m_{23}}, \frac{1}{m_{12}})
\]
Scan Conversion: the big picture (recap)

- **given vertices in DCS, fill in the pixels**
  - arbitrary polygons (non-simple, non-convex)
    - 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
  - triangles
    - split into two regions

Scan Conversion: the big picture

- **OpenGL**
  - simple convex polygons
    - *break into triangles, trivial*
      - `glBegin(GL_POLYGON) ... glEnd()`
  - concave or non-simple polygons
    - *break into triangles, more effort*
      - `gluNewTess(), 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 $z$

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

extra parameters

for ($y = y_B$ to $y_T$) { // step up scanlines
  $dz = (z_R - z_L) / (x_R - x_L)$
  $z = z_L$
  for ($x = x_L$ to $x_R$) { // step across pixels
    setPixel($x, y, r, g, b, z$);
    $z += dz$
  }
  $x_L = x_L + \Delta x_L$
  $x_R = x_R + \Delta x_R$
  $z_L = z_L + \Delta z_L$
  $z_R = z_R + \Delta z_R$
}

© Michiel van de Panne
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

• \textit{how to compute} $\alpha, \beta, \gamma$ ?
  • use bilinear interpolation or plane equations

\[
\begin{align*}
\alpha &= a \cdot x + b \cdot y + c \\
\beta &= \ldots
\end{align*}
\]

\[
\begin{align*}
\text{interpolate } \alpha, \beta, \gamma \\
\text{just like we did for } z
\end{align*}
\]

• once computed, use to interpolate any # of parameters from their vertex values
  \[
  \begin{align*}
  z &= \alpha \cdot z_1 + \beta \cdot z_2 + \gamma \cdot z_3 \\
  r &= \alpha \cdot r_1 + \beta \cdot r_2 + \gamma \cdot r_3 \\
  g &= \alpha \cdot g_1 + \beta \cdot g_2 + \gamma \cdot g_3 \\
  \end{align*}
  \]

\text{etc.}

Next class...

• clipping