Chapter 5

Scan Conversion – Drawing Polygons on Raster Display

Rasterizing Polygons/Triangles

- Basic surface representation in rendering
- Why?
  - Lowest common denominator
  - Can approximate any surface with arbitrary accuracy
    - All polygons can be broken up into triangles
- Guaranteed to be:
  - Planar
  - Triangles - Convex
- Simple to render
  - Can implement in hardware

Triangulation

- Convex polygons easily triangulated
- Concave polygons present a challenge

OpenGL Triangulation

- Simple convex polygons
  - break into triangles, trivial
  - glBegin(GL_POLYGON) ... glEnd()
- Concave or non-simple polygons
  - break into triangles, more effort
  - glNewTess(), gluTessCallback(), ...

Problem

- Input: closed 2D polygon
- Problem: Fill its interior with specified color on graphics display
- Assumptions –
  - simple - no self intersections
  - simply connected
- Solutions
  - Flood fill
  - Scan conversion
  - Implicit test

Flood Fill Algorithm

- $P$ polygon with $n$ vertices $v_0$ to $v_n$ ($v_0 = v_n$)
- $C$ color
- $P = (x,y) \in P$ point inside $P$
Flood Fill

- Draw edges
- Run:

```c
FloodFill (Polygon P, int x, int y, Color C)
if not (OnBoundary (x, y, P) or Colored (x, y, C))
begin
  PlotPixel (x, y, C);
  FloodFill (P, x + 1, y, C);
  FloodFill (P, x, y + 1, C);
  FloodFill (P, x, y - 1, C);
  FloodFill (P, x - 1, y, C);
end;
```
- Drawbacks?

Scanline Algorithm

- P polygon with n vertices v₀ to vₙ₋₁ (v₀=vₙ)
- C color
- Each intersection of straight line with boundary moves it from/into polygon
- Detect (& set) pixels inside polygon boundary (simple closed curve) with set of horizontal lines (pixel apart)

Scanline

```c
ScanConvert (Polygon P, Color C)
For y := 0 to ScreenMaxY do
  I := Points of intersections of edges of P with line Y = y;
  Sort I in increasing X order and
  Fill with color C alternating segments;
end;
```
- Limit to bounding box to speed up
- Other enhancements....

Bounding Box

Edge Walking

- Scanline is more efficient for specific polygons – trapezoids (triangles)
- Past graphics hardware
- Exploit continuous L and R edges on trapezoid
- Use Bresenham

```c
scanTrapezoid((x₁, y₁), (x₂, y₂), (x₃, y₃), (x₄, y₄), Δx₁, Δx₂)
```
Edge Walking

```c
for (y=yB; y<=yT; y++) {
    for (x=xL; x<=xR; x++)
        setPixel(x,y);
    xL += DxL;
    xR += DxR;
}
```

Edge Walking Triangles

- Split triangles into two regions with continuous left and right edges

```
scanTrapezoid(x0,x1,y0,y1, y0-y1, 1/mu1)
scanTrapezoid(x2,x3,y2,y3, y2-y3, 1/mu3)
```

Edge Walking Triangles

- Issues
  - Many small triangles
    - setup cost is non-trivial
  - Clipping triangles produces non-triangles

Modern Rasterization

Define a triangle from implicit edge equations:

Computation of Edge Equations

- Computing $A,B,C$ from $(x_1,y_1), (x_2,y_2)$

\[
\begin{align*}
Ax_1 + By_1 + C &= 0 \\
Ax_2 + By_2 + C &= 0
\end{align*}
\]

- Two equations, three unknowns
- Express $A, B$ in terms of $C$

Computing Edge Equations

```
\begin{bmatrix} x_0 & y_0 & A \\ x_1 & y_1 & B \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ \end{bmatrix}
```

```
\begin{bmatrix} A \\ B \end{bmatrix} = \begin{bmatrix} y_1 - y_0 \\ x_1 y_1 - x_1 y_0 \end{bmatrix}
```

- choose $C = x_0 y_2 - x_2 y_0$ for convenience
- Then $A = y_0 - y_1$ and $B = x_0 - x_1$
Edge Equations

- Given \( P_0, P_1, P_2 \), what are our three edges?
- Half-spaces defined by the edge equations must share the same sign on the interior of the triangle
- Consistency (Ex: \( [P_0 \ P_1], [P_1 \ P_2], [P_2 \ P_0] \))
- How do we make sure that sign is positive?
  - Test & flip if needed (\( A= -A, B= -B, C= -C \))

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
```c
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?**
  - Pixels inside triangle edges
- **What about pixels exactly on the edge?**
  - Draw - BUT order of triangles matters (it shouldn’t)
  - Don’t draw - BUT gaps possible between triangles
- Need consistent (if arbitrary) rule
  - Example: draw pixels on left or top edge, but not on right or bottom edge
### Triangle Rasterization Issues
- Moving Slivers

### Interpolation
- Interpolate between vertices:
  - \( Z \)
  - \( r,g,b \) - colour components
  - \( u,v \) - texture coordinates
  - \( N_x, N_y, N_z \) - surface normals
- Equivalent
  - Bilinear interpolation
  - Barycentric coordinates

### Barycentric Coordinates
- Weighted combination of vertices
- For point \( P \) on scanline:
  \[
P = a_1 P_1 + a_2 P_2 + a_3 P_3
  \]
  \[
a_1 + a_2 + a_3 = 1
  \]
  \[
0 \leq a_1, a_2, a_3 \leq 1
  \]

\[
\begin{align*}
P &= (1,0,0) \\
a_1 &= 0 \\
a_2 &= 0.5 \\
a_3 &= 1
\end{align*}
\]

- Alternative formula
  \[
P_x = P_1 + \frac{d_1}{d_1 + d_2} (P_2 - P_1)
  \]
  \[
= \frac{d_2}{d_1 + d_2} P_2 + \frac{d_1}{d_1 + d_2} P_3
  \]

### Barycentric Coordinates
- Area
  \[
A = \frac{1}{2} \left| P_1 P_2 \times P_3 \right|
  \]
- Barycentric coordinates
  \[
a_1 = A_{P_1 P_2} / A, a_2 = A_{P_2 P_3} / A,
  \]
  \[
a_3 = A_{P_3 P_1} / A,
  \]
  \[
P = a_1 P_1 + a_2 P_2 + a_3 P_3
  \]
Computing Barycentric Coords

- similarly:

\[
P_R = P_2 + \frac{b_1}{b_1 + b_2}(P_1 - P_2) = (1 - \frac{b_1}{b_1 + b_2})P_2 + \frac{b_1}{b_1 + b_2}P_1 = \frac{b_2}{b_1 + b_2}P_2 + \frac{b_1}{b_1 + b_2}P_1
\]

Computing Barycentric Coords

- combining

\[
P = \frac{c_1}{c_1 + c_2}P_1 + \frac{c_2}{c_1 + c_2}P_2
\]

- gives

\[
P = \frac{d_1}{d_1 + d_2}P_1 + \frac{d_2}{d_1 + d_2}P_2
\]

Computing Barycentric Coords

- can verify barycentric properties

\[
a_1 + a_2 + a_3 = 1
\]

\[
0 \leq a_1, a_2, a_3 \leq 1
\]

Bilinear Interpolation

- Interpolate quantity along \( L \) and \( R \) edges, as a function of \( y \)

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