Announcements

- Reminder important dates - still to come
  - Assignment 1 due: Oct 14
  - Assignment 2 due: Nov 4
  - Assignment 3 due:
    - Theory: Nov 25
    - Programming: Nov 28
  - Quiz 1: Oct 20
  - Quiz 2: Nov 10

A1Q3: “Given a line segment $S=(P_0, P_1)$ in 2D and a point $P$, write an algorithm to find if the point is on the line segment.”

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
- Convexity - formal definition:
  Object $S$ is convex iff for any two points $P, Q \in S$, $nP + (1 - n)Q \in S$, $n \in [0, 1]$.

OpenGL Triangulation

- Simple convex polygons
  - break into triangles, trivial
  - `glBegin(GL_POLYGON) ... glEnd()`

- Concave or non-simple polygons
  - break into triangles, more effort
  - `gluNewTess(), gluTessCallback(), ...`

Polygon Rasterization

- Assumptions - well behaved
  - simple - no self intersections
  - simply connected
  - (no holes)
- Solutions
  - Flood fill
  - Scan line
  - Implicit test
**Formulation**
- **Input**
  - polygon $P$ with rasterized edges
- **Problem:** Fill its interior with specified color on graphics display

**Flood Fill Algorithm**
- **Input**
  - polygon $P$ with rasterized edges
  - $P = (x,y) \in P$ point inside $P$

**Flood Fill**

**Flood Fill - Drawbacks**
- How do we find a point inside?
- Pixels visited up to 4 times to check if already set
- Need per-pixel flag indicating if set already
  - clear for every polygon!

**Scanline Algorithm**
- **Observation:** 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)**
Scan Conversion - Polygons

Scanline

ScanConvert (Polygon P, Color C)
For y := 0 to ScreenYMax 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)

  \[
  \text{scanTrapezoid}(x_L, x_R, y_B, y_T, x'_L, x'_R)
  \]

  scanTrapezoid\( (x_L, x_R, y_B, y_T, x'_L, x'_R)\) \( (x_L, x_R, y_B, y_T, x'_L, x'_R)\)

  \[
  y_B \leq y = y_T \leq y_B \\
  x_L \leq x \leq x_R
  \]

  \[
  y_B \leq y = y_T \leq y_B \\
  x_L \leq x \leq x_R
  \]

  \[
  y_B \leq y = y_T \leq y_B \\
  x_L \leq x \leq x_R
  \]

Edge Walking

- Exploit continuous L and R edges

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

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

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

  \[
  \text{scanTrapezoid}(x_L, x_R, y_B, y_T, \Delta x_L, \Delta x_R)
  \]
**Edge Walking Triangles**

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

\[
\text{scanTrapezoid}(x_0, y_0, x_1, y_1, \frac{1}{m_1}, \frac{1}{m_2})
\]

\[
\text{scanTrapezoid}(x_2, y_2, x_3, y_3, \frac{1}{w_1}, \frac{1}{w_2})
\]

**Issues**

- Many small triangles
- Setup cost is non-trivial
- Clipping triangles produces non-triangles

**Modern Rasterization**

- Define a triangle from implicit edge equations:

\[
\begin{align*}
&Ax_1 + By_1 + C = 0 \\
&Ax_2 + By_2 + C = 0 \\
&Ax_3 + By_3 + C = 0
\end{align*}
\]

- Two equations, three unknowns
- Express \( A, B \) in terms of \( C \)

**Computing Edge Equations**

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

\[
\begin{align*}
Ax_1 + By_1 + C &= 0 \\
Ax_2 + By_2 + C &= 0 \\
Ax_3 + By_3 + C &= 0
\end{align*}
\]

- Two equations, three unknowns
- Express \( A, B \) in terms of \( C \)

**Computing Edge Equations**

- Choose \( C = x_2 y_1 - x_1 y_2 \) for convenience
- Then \( A = y_2 - y_1 \) and \( B = x_1 - x_2 \)
- Our original implicit formula
- Note - in literature you can find same equation multiplied by -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 (\( A = -A, B = -B, C = -C \))

---

Copyright 2005, Alla Sheffer, UBC
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:
    - evaluate edge equations
    - draw pixel if all three are positive

```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
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

- Sliver

- Moving Slivers
Triangle Rasterization Issues

- Shared Edge Ordering

Interpolation - access triangle interior

- 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

- Area
  \[ A = \frac{1}{2} \left| P_1P_2 \times P_3P_2 \right| \]
- Barycentric coordinates
  \[ a_i = A_{P_2P_3} / A, \quad a_2 = A_{P_3P_1} / A, \]
  \[ P = a_1P_1 + a_2P_2 + a_3P_3 \]

Barycentric Coords: Alternative formula

- For point \( P \) on scanline:
  \[ P_x = P_2 + \frac{d_1}{d_1 + d_2} (P_3 - P_2) \]
  \[ = (1 - \frac{d_1}{d_1 + d_2})P_2 + \frac{d_1}{d_1 + d_2}P_3 = \]
  \[ = \frac{d_1}{d_1 + d_2}P_2 + \frac{d_2}{d_1 + d_2}P_3 \]

Computing Barycentric Coords

- similarly:
  \[ P_x = P_2 + \frac{b_1}{b_1 + b_2} (P_3 - P_2) \]
  \[ = (1 - \frac{b_1}{b_1 + b_2})P_2 + \frac{b_1}{b_1 + b_2}P_3 = \]
  \[ = \frac{b_2}{b_1 + b_2}P_2 + \frac{b_1}{b_1 + b_2}P_3 \]
Computing Barycentric Coords

- combining
  \[ P = \frac{c_1}{c_1 + c_2} P_1 + \frac{c_2}{c_1 + c_2} P_2 \]
- gives
  \[ P = \frac{c_1 d_3 + d_1 P_1 + d_2}{d_1 + d_2} P_2 + \frac{c_2}{c_1 + c_2} \left( \frac{b_1}{b_1 + b_2} P_1 + \frac{b_2}{b_1 + b_2} P_2 \right) \]

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 \( x \)