Triangulating Polygons

• simple convex polygons
  • trivial to break into triangles
  • pick one vertex, draw lines to all others not immediately adjacent
  • OpenGL supports automatically - glBegin(GL_POLYGON), glEnd()
• concave or non-simple polygons
  • more effort to break into triangles
  • simple approach may not work
  • OpenGL can support at extra cost - gluNewTess(), gluTessCallback()

Making It Fast: Reuse Computation

• midpoint: if \( f(x+1, y+.5) < 0 \) then \( y = y+1 \)
• on previous step evaluated \( f(x-1, y-.5) \) or \( f(x+1, y-.5) \)
• \( f(x+1, y) = f(x, y) + (y_0 - y_1) \)
• \( f(x+1, y+.5) = f(x, y) + (y_0 - y_1)(x_0 + 1) \)
  \( + (x_1 - x_0)(2y_0 + 1) \)

Review: Midpoint Algorithm

• we're moving horizontally along x direction (first octant)
• only two choices: draw at current y value, or move up vertically to \( y+1 \)
  • check if midpoint between two possible pixel centers above or below line
• candidates:
  • top pixel: \((x+1, y+1)\)
  • bottom pixel: \((x+1, y)\)
• midpoint: \((x+1, y)\)
• check if midpoint above or below line
  • below: pick top pixel
  • above: pick bottom pixel
• key idea behind Bresenham
  • reuse computation from previous step
  • integer arithmetic by doubling values

Making It Fast: Integer Only

• avoid dealing with non-integer values by doubling both sides

Review: YIQ Color Space

• color model used for color TV
  • Y is luminance (same as CIE)
  • I & Q are color (not same I as HSI!)  
  - using Y backwards compatible for B/W TVs
  - conversion from RGB is linear
  
  \[
  \begin{align*}
  Y &= 0.299R + 0.587G + 0.114B \\
  I &= -0.1687R - 0.3313G + 0.5000B \\
  Q &= 0.5000R - 0.3313G - 0.1687B
  \end{align*}
  \]

Review: Basic Line Drawing

• goals
  • integer coordinates
  • thinnest line with no gaps
  • preserve one octant, other cases symmetric
  • how can we do this more quickly?

Problem

• simple algorithm
  • draw edges of polygon
  • use flood-fill to draw interior
Flood Fill
• draw edges
• run:

Flood Fill Drawbacks
• pixels visited up to 4 times to check if already set
• need per-pixel flag indicating if set already
• must clear for every polygon!

Scanline Algorithms
• scanline: a line of pixels in an image
• set pixels inside polygon boundary along horizontal lines one pixel apart vertically

General Polygon Rasterization
• how do we know whether given pixel on scanline is inside or outside polygon?

General Polygon Rasterization
• idea: use a parity test
  for each scanline
  edgeCnt = 0;
  for each pixel on scanline (1 to x)
    if (oldpixel-mpixel crosses edge)
      edgeCnt ++;
    // draw the pixel if edgeCnt odd
    if (edgeCnt % 2)
      setPixel(pixel);

Making It Fast: Bounding Box
• smaller set of candidate pixels
  loop over xmin, xmax and ymin,ymax instead of all x, all y

Triangle Rasterization Issues
• exactly which pixels should be lit?
• pixels with centers inside triangle edges
• what about pixels exactly on edge?
• draw them: order of triangles matters (it shouldn’t)
• don’t draw them: gaps possible between triangles
• need a consistent (if arbitrary) rule
• example: draw pixels on left or top edge, but not on right or bottom edge
• example: check if triangle on same side of edge as offscreen point

Interpolation During Scan Conversion
• drawing pixels in polygon requires interpolating many values between vertices
  • r,g,b colour components
  • use for shading
  • z values
  • u,v texture coordinates
  • N_i,N_j,N_k surface normals
  • equivalent methods (for triangles)
  • barycentric interpolation
  • barycentric coordinates

Bilinear Interpolation
• interpolate quantity along L and R edges, as a function of y
  • then interpolate quantity as a function of x

Barycentric Coordinates
• non-orthogonal coordinate system based on triangle itself
  • origin: P_1, basis vectors: (P_2-P_1) and (P_3-P_1)
  \[ P = P_1 + \beta(P_2-P_1) + \gamma(P_3-P_1) \]

Using Barycentric Coordinates
• weighted combination of vertices
  • smooth mixing
  • speedup
  • compute once per triangle
  \[ P = \alpha \cdot P_1 + \beta \cdot P_2 + \gamma \cdot P_3 \]
  \[ \alpha + \beta + \gamma = 1 \]
  \[ 0 \leq \alpha, \beta, \gamma \leq 1 \] for points inside triangle

Deriving Barycentric From Bilinear
• from bilinear interpolation of point P on scanline
  \[ P = P_1 + \frac{d_1}{d_1 + d_2} (P_2 - P_1) \]
  \[ d_1 = \frac{1}{d_1 + d_2} \]
  \[ d_2 = \frac{1}{d_1 + d_2} \]
Deriving Barycentric From Bilinear

• similarly

\[ \begin{align*}
P_1 &= P_2 + \frac{h_1}{h_1 + h_2} (P_3 - P_2) \\
&= (1 - \frac{h_2}{h_1 + h_2})P_2 + \frac{h_1}{h_1 + h_2} P_3 = \\
&= \frac{h_1}{h_1 + h_2} P_1 + \frac{h_2}{h_1 + h_2} P_2 \\
gives \end{align*} \]

Deriving Barycentric From Bilinear

• combining

\[ \begin{align*}
P_1 &= \frac{d_1}{d_1 + d_2} P_1 + \frac{d_2}{d_1 + d_2} P_2 \\
P_2 &= \frac{d_2}{d_1 + d_2} P_1 + \frac{d_1}{d_1 + d_2} P_2 \\
\end{align*} \]

• gives

\[ \begin{align*}
\alpha &= \frac{d_1}{d_1 + d_2} \\
\beta &= \frac{d_2}{d_1 + d_2} \\
\gamma &= \frac{d_1}{d_1 + d_2} \\
\end{align*} \]

Deriving Barycentric From Bilinear

• thus \( P = \alpha P_1 + \beta P_2 + \gamma P_3 \) with

\[ \begin{align*}
\alpha &= \frac{d_1}{d_1 + d_2} \\
\beta &= \frac{d_2}{d_1 + d_2} \\
\gamma &= \frac{d_1}{d_1 + d_2} \\
\end{align*} \]

• can verify barycentric properties

\[ \alpha + \beta + \gamma = 1, \quad 0 \leq \alpha, \beta, \gamma \leq 1 \]

Computing Barycentric Coordinates

• 2D triangle area

\[ A = \overrightarrow{PA_1} + \overrightarrow{PA_2} + \overrightarrow{PA_3} \]

• half of parallelogram area

\[ \text{from cross product} \]

\[ A = \alpha \overrightarrow{P_1} + \beta \overrightarrow{P_2} + \gamma \overrightarrow{P_3} \]

• can verify barycentric properties

\[ \alpha + \beta + \gamma = 1, \quad 0 \leq \alpha, \beta, \gamma \leq 1 \]