Rasterizing triangles

- The most common case in most applications
  - with good antialiasing can be the only case
  - some systems render a line as two skinny triangles
- Triangle represented by three vertices
- Simple way to think of algorithm follows the pixel-walk interpretation of line rasterization
  - walk from pixel to pixel over (at least) the polygon’s area
  - evaluate linear functions as you go
  - use those functions to decide which pixels are inside

Rasterizing triangles

- Input:
  - three 2D points (the triangle’s vertices in pixel space)
    - \((x_0, y_0); (x_1, y_1); (x_2, y_2)\)
  - parameter values at each vertex
    - \(q_{00}, \ldots, q_{0n}; q_{10}, \ldots, q_{1n}; q_{20}, \ldots, q_{2n}\)
- Output: a list of fragments, each with
  - the integer pixel coordinates \((x, y)\)
  - interpolated parameter values \(q_0, \ldots, q_n\)

Incremental linear evaluation

- A linear (affine, really) function on the plane is:
  \[ q(x, y) = c_x x + c_y y + c_k \]
- Linear functions are efficient to evaluate on a grid:
  \[
  q(x + 1, y) = c_x (x + 1) + c_y y + c_k = q(x, y) + c_x \\
  q(x, y + 1) = c_x x + c_y (y + 1) + c_k = q(x, y) + c_y
  \]

Incremental linear evaluation

```
lEval(xl, xh, yl, yh, cx, cy, ck) {
    // setup
    qRow = cx*xl + cy*yl + ck;
    // traversal
    for y = yl to yh {
        qPix = qRow;
        for x = xl to xh {
            output(x, y, qPix);
            qPix += cx;
        }
        qRow += cy;
    }
}
```
Defining parameter functions

• To interpolate parameters across a triangle we need to find the \( c_x \), \( c_y \), and \( c_k \) that define the (unique) linear function that matches the given values at all 3 vertices:
  
  \[
  c_x x + c_y y + c_k = q_0 \quad ( \text{each states that the function agrees with the given value at one vertex} )
  \]

  leading to a 3x3 matrix equation for the coefficients:

  \[
  \begin{bmatrix}
  x_0 & y_0 & 1 \\
  x_1 & y_1 & 1 \\
  x_2 & y_2 & 1
  \end{bmatrix}
  \begin{bmatrix}
  c_x \\
  c_y \\
  c_k
  \end{bmatrix}
  =
  \begin{bmatrix}
  q_0 \\
  q_1 \\
  q_2
  \end{bmatrix}
  \]

  (singular if triangle is degenerate)

More efficient version: shift origin to \((x_0, y_0)\):

\[
q(x, y) = c_x (x - x_0) + c_y (y - y_0) + q_0
\]

\[
q(x_1, y_1) = c_x (x_1 - x_0) + c_y (y_1 - y_0) + q_0 = q_1
\]

\[
q(x_2, y_2) = c_x (x_2 - x_0) + c_y (y_2 - y_0) + q_0 = q_2
\]

• Now this is a 2x2 linear system (since \( q_0 \) falls out):

\[
\begin{bmatrix}
  (x_1 - x_0) & (y_1 - y_0) \\
  (x_2 - x_0) & (y_2 - y_0)
\end{bmatrix}
\begin{bmatrix}
  c_x \\
  c_y
\end{bmatrix}
= \begin{bmatrix}
  q_1 - q_0 \\
  q_2 - q_0
\end{bmatrix}
\]

• Solve using Cramer’s rule (see Shirley):

\[
c_x = (\Delta q_1 \Delta y_2 - \Delta q_2 \Delta y_1) / (\Delta x_1 \Delta y_2 - \Delta x_2 \Delta y_1)
\]

\[
c_y = (\Delta q_2 \Delta x_1 - \Delta q_1 \Delta x_2) / (\Delta x_1 \Delta y_2 - \Delta x_2 \Delta y_1)
\]

Interpolating several parameters

\[
\text{linInterp}(x_l, x_h, y_l, y_h, n, x_0, y_0, q_0[0], x_1, y_1, q_1[0], x_2, y_2, q_2[0])
\]

// setup
for k = 0 to n-1
// compute \( c_x[k] \), \( c_y[k] \), \( q_{Row}[k] \)
// from \( q_0[k] \), \( q_1[k] \), \( q_2[k] \)

// traversal
for y = y_l to y_h
for k = 1 to n, qPix[k] = qRow[k]
for x = x_l to x_h
output(x, y, qPix)
fork = 1 to n, qPix[k] += c_x[k]
fork = 1 to n, qRow[k] += c_y[k]

Clipping to the triangle

• Interpolate three barycentric coordinates across the plane

  - each barycentric coord is 1 at one vert and 0 at the other two

  • Output fragments only when all three are > 0.

Rasterizing triangles

• Summary

  1. evaluation of linear functions on pixel grid
  2. functions defined by parameter values at vertices
  3. using extra parameters to determine fragment set
Barycentric coordinates

- Basis: a coordinate system for triangles
  \[ \alpha = 1 - \beta - \gamma \]
  \[ p = a + \beta(b - a) + \gamma(c - a) \]

- in this view, the triangle interior test is just
  \[ \beta > 0; \ \gamma > 0; \ \beta + \gamma < 1 \]

Edge equations

- In plane, triangle is the intersection of 3 half spaces
  \[ (x - a) \cdot (b - a)^\perp > 0 \]
  \[ (x - b) \cdot (c - b)^\perp > 0 \]
  \[ (x - c) \cdot (a - c)^\perp > 0 \]

Walking edge equations

- We need to update values of the three edge equations with single-pixel steps in x and y
- Edge equation already in form of dot product
- Components of vector are the increments

Pixel-walk (Pineda) rasterization

- Conservatively visit a superset of the pixels you want
- Interpolate linear functions
- Use those functions to determine when to emit a fragment
Rasterizing triangles

- Exercise caution with rounding and arbitrary decisions
  - need to visit these pixels once
  - but it's important not to visit them twice!