**Announcements**

- HW 2 extension till next Monday
- Grading slots: sign up sheet
  - Need to sign up, else will not be graded
- HW 1 (make regrade requests online)
- Course split
  - 45% 2 prelims
  - 30% 3 programming assignments
  - 25% 4 HWs

---

**Rasterization**

- First job: enumerate the pixels covered by a primitive
  - simple, aliased definition: pixels whose centers fall inside
- Second job: interpolate values across the primitive
  - e.g. colors computed at vertices
  - e.g. normals at vertices
  - will see applications later on

---

**Optimizing line drawing**

- Only need to update \( d \) for integer steps in \( x \) and \( y \)
- Do that with addition
- Known as “DDA” (digital differential analyzer)

---

**Midpoint line algorithm**

\[
\begin{align*}
x &= \text{ceil}(x_0) \\
y &= y_0 + \text{round}(m \cdot x + b) \\
d &= m \cdot (x + 1) + b - y \\
\text{while } x < \text{floor}(x_1) & \text{ do }
\begin{align*}
\text{if } d > 0.5 & \text{ then } \\
y &= y + 1 \\
d &= d - m \\
x &= x + 1 \\
d &= d + m \\
\text{output}(x, y)
\end{align*}
\end{align*}
\]

---

**Linear interpolation**

- We often attach attributes to vertices
  - e.g. computed diffuse color of a hair being drawn using lines
  - want color to vary smoothly along a chain of line segments
- Basic definition of interpolation
  - \( \text{ID}: f(x) = (1 - \alpha) \cdot y_0 + \alpha \cdot y_1 \)
  - where \( \alpha = (x - x_0) / (x_1 - x_0) \)
- In the 2D case of a line segment, alpha is just the fraction of the distance from \((x_0, y_0)\) to \((x_1, y_1)\)
Linear interpolation

- Pixels are not exactly on the line
- Define 2D function by projection on line
  - this is linear in 2D
  - therefore can use DDA to interpolate

Alternate interpretation

- We are updating $d$ and $\alpha$ as we step from pixel to pixel
  - $d$ tells us how far from the line we are
  - $\alpha$ tells us how far along the line we are
- So $d$ and $\alpha$ are coordinates in a coordinate system oriented to the line

Alternate interpretation

- View loop as visiting all pixels the line passes through
  - Interpolate $d$ and $\alpha$ for each pixel
  - Only output frag. if pixel is in band
- This makes linear interpolation the primary operation

Pixel-walk line rasterization

$x = \text{ceil}(x_0)$
$y = \text{round}(m \cdot x + b)$
output $x, y$
$d = m \cdot x + b - y$
while $x < \text{floor}(x_1)$
  - if $d > 0.5$
    $y += 1; d -= 1;$
  - else
    $x += 1; d -= m;$
  - if $-0.5 < d \leq 0.5$
    output $x, y$

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
      - $q_{00}, \cdots, q_{0n}; q_{10}, \cdots, q_{1n}; q_{20}, \cdots, q_{2n}$
- Output: a list of fragments, each with
  - the integer pixel coordinates $(x, y)$
  - interpolated parameter values $q_0, \ldots, q_n$
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

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

\[
\text{linEval}(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;
}
\]

c_x = .005; c_y = .005; c_k = 0
(image size 100x100)

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