CS4620/5620: Lecture 12

Rasterization

Announcements

• Turn in HW 1

• PPA 1 out

• Friday lecture
  – History of graphics
  – PPA 1 in 4621
The graphics pipeline

- The standard approach to object-order graphics
- Many versions exist
  - software, e.g. Pixar’s REYES architecture
    - many options for quality and flexibility
  - hardware, e.g. graphics cards in PCs
    - amazing performance: millions of triangles per frame
- We’ll focus on an abstract version of hardware pipeline
Primitives

• Points
• Line segments
  – and chains of connected line segments
• Triangles
• And that’s all!
  – Curves? Approximate them with chains of line segments
  – Polygons? Break them up into triangles
  – Curved regions? Approximate them with triangles
• Hardware desire: minimal primitives
  – simple, uniform, repetitive: good for parallelism
  – and of course, cyclical; now you can send curves, and the vertex shader will convert to primitives

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
Rasterizing lines

- Define line as a rectangle
- Specify by two endpoints
- Ideal image: black inside, white outside

Point sampling

- Approximate rectangle by drawing all pixels whose centers fall within the line
- Problem: sometimes turns on adjacent pixels
Bresenham lines (midpoint alg.)

- Point sampling unit width rectangle leads to uneven line width
- Goal: draw thinnest line possible
  - Define line width parallel to pixel grid
  - That is, turn on the single nearest pixel in each column
  - Note that 45° lines are now thinner
Algorithms for drawing lines

- line equation:
  \[ y = b + m \times \]
- Simple algorithm:
  evaluate line equation per column

- W.l.o.g. \( x_0 < x_1 \);
  \( 0 \leq m \leq 1 \)

  for \( x = \text{ceil}(x_0) \) to \( \text{floor}(x_1) \)
  \[ y = b + m \times x \]
  output(\( x \), round(\( y \)))

\[ y = 1.91 + 0.37 \times \]
Bresenham lines (midpoint alg.)

- round (y)?
  - cutoff at midpt
  \[
  y = mx + b \\
  d = mx + b - y
  \]

- \(d(x+1, y+0.5)\)
  \[
  = m(x + 1) + b - (y + 0.5)
  \]

- \(d > 0 \)? NE : E

Optimizing line drawing

- Multiplying and rounding: slow
- At each pixel
  - only options are E and NE
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)

\[
d = m(x + 1) + b - y
\]

• Only need to update \( d \) for integer steps in \( x \) and \( y \)
• Now test \( d \) against 0.5
Midpoint line algorithm

\[ x = \text{ceil}(x_0) \]
\[ y = y_0 = \text{round}(m \times x + b) \]

output \((x, y)\)
\[ d = m \times (x + 1) + b - y \]
while \(x < \text{floor}(x_1)\)
  if \(d > 0.5\)
    \(y += 1\)
    \(d -= 1\)
  \(x += 1\)
  \(d += m\)
output\((x, y)\)
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
  – 1D: \( f(x) = (1 - \alpha) y_0 + \alpha y_1 \)
  – 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

\[
x = \text{ceil}(x_0) \\
y = \text{round}(m \cdot x + b) \\
d = m \cdot x + b - y \\
\text{output } (x, y)
\]

while \( x < \text{floor}(x_1) \)
\[
\text{if } d > 0.5 \\
\quad y += 1; \ d -= 1 \\
\text{else} \\
\quad x += 1; \ d += m \\
\text{if } -0.5 < d \leq 0.5 \\
\quad \text{output}(x, y)
\]

Midpoint algorithm in action