Rasterization

COMP 575/770
Spring 2013
The Rasterization Pipeline

1. **APPLICATION**
2. **COMMAND STREAM**
3. **VERTEX PROCESSING**
4. **TRANSFORMED GEOMETRY**
5. **RASTERIZATION**
6. **FRAGMENTS**
7. **FRAGMENT PROCESSING**
8. **FRAMEBUFFER IMAGE**
9. **DISPLAY**

- 3D transformations; shading
- Conversion of primitives to pixels
- Blending, compositing, shading
- User sees this
Rasterization

1. Project a primitive onto the screen

2. Find which pixels lie inside the projection

3. Interpolate attributes at each pixel
   - These are quantities that help in shading

4. Perform shading
Outline

- Overview
- Line Rasterization
Rasterizing Lines

- Most pixels won’t lie on the line!
- Approximate line using thin rectangle
Rasterizing Lines

- Mark all pixels inside the thin rectangle?

- **Problem**: Sometimes marks adjacent pixels
Rasterizing Lines
Midpoint Algorithm

- Mark only one pixel per column
  - The closest one

- Basically, line width defined parallel to pixel grid
Midpoint Algorithm
Midpoint Algorithm

- Slope–intercept form of line equation:
  \[ y = mx + b \]

- We assume \( m \in (0,1] \)

- There are three other analogous cases:
  - \( m \in (-\infty,-1] \)
  - \( m \in (-1,0] \)
  - \( m \in (1,\infty) \)
**Midpoint Algorithm**

- Evaluate line equation per column
- Endpoints at $x_0 < x_1$

```
for x = ceil(x_0) to floor(x_1)
    y = m*x + b
    mark(x, round(y))
end
```

$y = 1.91 + 0.37x$
Midpoint Algorithm

- Evaluate line equation per column
- Endpoints at $x_0 < x_1$

\[
x = \text{ceil}(x_0)
\]\[
\textbf{while} \ x < \text{floor}(x_1)
\]\[
\quad y = m \times x + b
\]\[
\quad \text{mark}(x, \text{round}(y))
\]\[
\quad x += 1
\]\[
\textbf{end}
\]

\[
y = 1.91 + 0.37 \times x
\]
Optimized Midpoint Algorithm

- Two slow operations:
  - Multiply: \( y = m*x + b \)
  - Round: \( \text{mark}(x, \text{round}(y)) \)

- \( y \) varies predictably:

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

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

\textbf{while} \ x < \text{floor}(x_1)

\[
\text{mark}(x, \text{round}(y))
\]
\[
x += 1
\]
\[
y += m
\]
\textbf{end}
Optimized Midpoint Algorithm

- Only two options when moving to next column
- Which does line pass closer to?
  - \( d = m(x + 1) + b - y \)
  - If \( d > 0.5 \), line is closer to NE
    - Otherwise, closer to E
Optimized Midpoint Algorithm

- Incrementally update $d$

- If we choose E:
  - Increment $d$ by $m$

- If we choose NE:
  - Increment $d$ by $m - 1$

- Approach also called digital differential analyzer (DDA)

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

\[
x = \text{ceil}(x_0) \\
y = \text{round}(m \cdot x + b) \\
d = m \cdot (x + 1) + b - y \\
\textbf{while } x < \text{floor}(x_1) \\
\hspace{1em} \textbf{if } d > 0.5 \\
\hspace{2em} y += 1 \\
\hspace{2em} d -= 1 \\
\hspace{1em} \textbf{end} \\
\hspace{1em} x += 1 \\
\hspace{1em} d += m \\
\hspace{1em} \text{mark}(x, y) \\
\textbf{end}
Outline

- Overview
- Line Rasterization
- Line Attributes
Attribute Interpolation

- **Attributes** are often attached to vertices/endpoints
  - E.g., color of hair drawn using lines
  - Want color to vary smoothly along line segments

- Linear interpolation:

\[ f(x) = (1 - \alpha)y_0 + \alpha y_1 \]
\[ \alpha = \frac{x - x_0}{x_1 - x_0} \]

- \( \alpha \) is the fraction of distance from \((x_0, y_0)\) to \((x_1, y_1)\)
Suppose endpoint attributes are $A_0$ and $A_1$

$$A(x) = A_0 + \alpha(A_1 - A_0)$$

Since $\alpha$ is linear in $x$, can write an incremental expression
Outline

- Overview
- Line Rasterization
- Line Attributes
- Triangle Rasterization
Triangle Rasterization

- The most common primitive in most applications
  - Can represent any object using many triangles
  - A triangle always projects to a triangle

- Triangle represented by 3 vertices
  - \( a = (x_a, y_a) \), \( b = (x_b, y_b) \), and \( c = (x_c, y_c) \)

- Need to figure out which pixels are inside the triangle
Bounding Rectangle

- Smallest rectangular portion of screen which contains triangle
- No pixels outside it could possibly be in the triangle

\[
\begin{align*}
x_{\text{min}} &= \text{floor}(\min(x_a, x_b, x_c)) \\
x_{\text{max}} &= \text{ceil}(\max(x_a, x_b, x_c)) \\
y_{\text{min}} &= \text{floor}(\min(y_a, y_b, y_c)) \\
y_{\text{max}} &= \text{ceil}(\max(y_a, y_b, y_c))
\end{align*}
\]
Triangle Rasterization

\[ x_{\text{min}} = \text{floor}(\min(x_a, x_b, x_c)) \]
\[ x_{\text{max}} = \text{ceil}(\max(x_a, x_b, x_c)) \]
\[ y_{\text{min}} = \text{floor}(\min(y_a, y_b, y_c)) \]
\[ y_{\text{max}} = \text{ceil}(\max(y_a, y_b, y_c)) \]

\begin{align*}
\text{for } y &= y_{\text{min}} \text{ to } y_{\text{max}} \\
\text{for } x &= x_{\text{min}} \text{ to } x_{\text{max}} \\
\text{if } (x, y) \text{ is in triangle} \\
& \quad \text{mark}(x, y) \\
\end{align*}
Barycentric Coordinates

- A triangle is a **convex** shape

- Any point in the triangle is a **convex combination** of the triangle’s vertices:

  \[ \mathbf{p} = \alpha \mathbf{a} + \beta \mathbf{b} + \gamma \mathbf{c} \]

- Subject to:

  \[ \alpha + \beta + \gamma = 1 \]
Barycentric Coordinates

- $\alpha, \beta, \gamma$ are called the barycentric coordinates of $\mathbf{p}$
- $\mathbf{p}$ lies inside the triangle if $\alpha \geq 0$, $\beta \geq 0$, and $\gamma \geq 0$
- Alternatively: $\beta \geq 0$, $\gamma \geq 0$, and $\beta + \gamma \leq 1$
- Main goal: determine $\beta, \gamma$
Barycentric Coordinates

- Rewriting:
  \[ \mathbf{p} = \mathbf{a} + \beta (\mathbf{b} - \mathbf{a}) + \gamma (\mathbf{c} - \mathbf{a}) \]

- We’ve seen this sort of thing before!

- \((\beta, \gamma)\) are coordinates of \(\mathbf{p}\) in a different frame:
  - \(\mathbf{a}\) is the origin
  - \((\mathbf{b} - \mathbf{a})\) and \((\mathbf{c} - \mathbf{a})\) are the axes
Barycentric Coordinates
Barycentric Coordinates

- Construct the “frame–to–canonical” matrix:

\[
\begin{bmatrix}
  b - a & c - a & a \\
  0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
  \beta \\
  \gamma \\
  1
\end{bmatrix} = F
\begin{bmatrix}
  \beta \\
  \gamma \\
  1
\end{bmatrix} =
\begin{bmatrix}
  x \\
  y \\
  1
\end{bmatrix}
\]

- Invert to get the “canonical–to–frame” matrix:

\[
\begin{bmatrix}
  \beta \\
  \gamma \\
  1
\end{bmatrix} = F^{-1}
\begin{bmatrix}
  x \\
  y \\
  1
\end{bmatrix} =
\begin{bmatrix}
  b - a & c - a & a \\
  0 & 0 & 1
\end{bmatrix}^{-1}
\begin{bmatrix}
  x \\
  y \\
  1
\end{bmatrix}
\]
Barycentric Coordinates

- The final answer is:

\[
\beta = \frac{(y_a - y_c)x + (x_c - x_a)y + x_ay_c - x_cy_a}{(y_a - y_c)x_b + (x_c - x_a)y_b + x_ay_c - x_cy_a}
\]

\[
\gamma = \frac{(y_a - y_b)x + (x_b - x_a)y + x_ay_b - x_by_a}{(y_a - y_b)x_c + (x_b - x_a)y_c + x_ay_b - x_by_a}
\]

\[
\alpha = 1 - \beta - \gamma
\]
Triangle Rasterization

\[ x_{\text{min}} = \text{floor}(\min(x_a, x_b, x_c)) \]
\[ x_{\text{max}} = \text{ceil}(\max(x_a, x_b, x_c)) \]
\[ y_{\text{min}} = \text{floor}(\min(y_a, y_b, y_c)) \]
\[ y_{\text{max}} = \text{ceil}(\max(y_a, y_b, y_c)) \]

\textbf{for} y = y_{\text{min}} \textbf{to} y_{\text{max}} \\
\hspace{1em} \textbf{for} x = x_{\text{min}} \textbf{to} x_{\text{max}} \\
\hspace{2em} \text{compute } \beta, \gamma \text{ given } x, y \\
\hspace{2em} \textbf{if } \beta > 0 \textbf{ and } \gamma > 0 \textbf{ and } \beta + \gamma < 1 \\
\hspace{3em} \text{mark}(x, y) \\
\hspace{1em} \textbf{end} \\
\textbf{end} \\
\textbf{end}
- $\beta, \gamma$ are linear functions of $x, y$:

$$\beta = \frac{(y_a - y_c)x + (x_c - x_a)y + x_ay_c - x_cy_a}{(y_a - y_c)x_b + (x_c - x_a)y_b + x_ay_c - x_cy_a}$$

$$\gamma = \frac{(y_a - y_b)x + (x_b - x_a)y + x_ay_b - x_by_a}{(y_a - y_b)x_c + (x_b - x_a)y_c + x_ay_b - x_by_a}$$

$$\alpha = 1 - \beta - \gamma$$

- Rewriting:

$$\beta(x, y) = \beta_0 + \beta_xx + \beta_yy$$

$$\gamma(x, y) = \gamma_0 + \gamma_xx + \gamma_yy$$
Optimized Triangle Rasterization

- This gives the following recurrences:

\[ \beta(x + 1, y) - \beta(x, y) = \beta_x \]
\[ \beta(x, y + 1) - \beta(x, y) = \beta_y \]
\[ \gamma(x + 1, y) - \gamma(x, y) = \gamma_x \]
\[ \gamma(x, y + 1) - \gamma(x, y) = \gamma_y \]

- So now we can write an incremental algorithm!
compute $x_{\text{min}}, x_{\text{max}}, y_{\text{min}}, y_{\text{max}}$
compute $\beta, \gamma$ given $x_{\text{min}}, y_{\text{min}}$
n = $(x_{\text{max}} - x_{\text{min}}) + 1$

for $y = y_{\text{min}}$ to $y_{\text{max}}$
  for $x = x_{\text{min}}$ to $x_{\text{max}}$
    if $\beta > 0$ and $\gamma > 0$ and $\beta + \gamma < 1$
      mark($x, y$)
    end
    $\beta + = \beta_x$
    $\gamma + = \gamma_x$
  end
  $\beta + = \beta_y - n*\beta_x$
  $\gamma + = \gamma_y - n*\gamma_x$
end
Outline

- Overview
- Line Rasterization
- Line Attributes
- Triangle Rasterization
- Triangle Attributes
Attribute Interpolation

- Given attributes $A_a, A_b, A_c$ at vertices $a, b, c$

- Attribute at $p = (x, y)$:

$$A(x, y) = A_a + \beta (A_b - A_a) + \gamma (A_c - A_a)$$

- $\beta, \gamma$ are just the barycentric coordinates
Attribute Interpolation

- Easy to incorporate into incremental algorithm
- Also called Gouraud interpolation
- Just one way of doing interpolation
Shared Edges

- Some pixels may lie exactly on an edge shared by two triangles
- What color to assign them?
- More than one way to do this
Shared Edges

- Pixel \((x, y)\) is on an edge if:
  \[ \beta = 0 \text{ or } \gamma = 0 \text{ or } \beta + \gamma = 1 \]

- Ignore pixels on an edge when on top row or last column in a row

- OpenGL/Direct3D convention

Outline

- Overview
- Line Rasterization
- Line Attributes
- Triangle Rasterization
- Triangle Attributes
- Clipping
Clipping

- Rasterizer assumes triangle’s pixels are on-screen
  - Bad things happen if triangle crosses near plane

- After applying perspective matrix, need to clip against canonical view volume
  - Clip triangle against planes \( \{x, y, z\} = \pm 1 \)
Clipping

- 4 cases, based on which/how many vertices are inside the clipping plane:
  - All inside: retain triangle as-is
  - All outside: don’t draw triangle
  - One inside, two outside: one clipped triangle
  - Two inside, one outside: two clipped triangles