Last time

☐ Clipping
Today

☐ Rasterization

☐ In-class Mid-term

- November 5
- Close-book exam
- Notes on 1 page of A4 or Letter size paper
Where We Stand

☐ At this point we know how to:
  ■ Convert points from local to screen coordinates
  ■ Clip polygons and lines to the view volume

☐ Next thing:
  ■ Determine which pixels to fill for any given point, line or polygon
Drawing Points

- When points are mapped into window coordinates, they could land anywhere - not just at a pixel center
- Solution is the simple, obvious one
  - Map to window space
  - Fill the closest pixel
  - Can also specify a radius - fill a square of that size, or fill a circle
- Square is faster
Drawing Points (2)
Task: Decide which pixels to fill (samples to use) to represent a line

We know that all of the line lies inside the visible region (clipping gave us this!)
Line Drawing Algorithms

- Consider lines of the form \( y = mx + c \), where \( m = \frac{\Delta y}{\Delta x} \), 0 < \( m < 1 \), integer coordinates
  - All others follow by symmetry, modify for real numbers

- Variety of slow algorithms (Why slow?):
  - step \( x \), compute new \( y \) at each step by equation, rounding:
  - step \( x \), compute new \( y \) at each step by adding \( m \) to old \( y \), rounding:

\[
\begin{align*}
  x_{i+1} &= x_i + 1, & y_{i+1} &= \text{round}(mx_{i+1} + b) \\
  x_{i+1} &= x_i + 1, & y_{i+1} &= \text{round}(y_i + m)
\end{align*}
\]
Bresenham’s Algorithm Overview

- Aim: For each $x$, plot the pixel whose $y$-value is closest to the line
- Given $(x_i, y_i)$, must choose from either $(x_i+1, y_i+1)$ or $(x_i+1, y_i)$
- Idea: compute a decision variable
  - Value that will determine which pixel to draw
  - Easy to update from one pixel to the next
- Bresenham’s algorithm is the midpoint algorithm for lines
  - Other midpoint algorithms for conic sections (circles, ellipses)
Midpoint Methods

- Consider the midpoint between \((x_i+1, y_i+1)\) and \((x_i+1, y_i)\)
- If it’s above the line, we choose \((x_i+1, y_i)\), otherwise we choose \((x_i+1, y_i+1)\)
Midpoint Decision Variable

- Write the line from \((x_1, y_1)\) to \((x_2, y_2)\) in *implicit form*:
  \[ F(x, y) = ax + by + c = \Delta x \cdot y - \Delta y \cdot x + (\Delta y \cdot x_1 - \Delta x \cdot y_1) \]
  - Assume \(x_1 \leq x_2\)
  - \(\Delta x = x_2 - x_1, \Delta y = y_2 - y_1\)

- The value of \(F(x, y)\) tells us where points are with respect to the line
  - \(F(x, y) = 0\): the point is on the line
  - \(F(x, y) > 0\): The point is above the line
  - \(F(x, y) < 0\): The point is below the line

- The decision variable is the value of \(d_i = 2F(x_i + 1, y_i + 0.5)\)
  - The factor of two makes the math easier
What Can We Decide?

\[ d_i = 2\Delta xy_i - 2\Delta y(x_i + 1) + 2(\Delta y \cdot x_i - \Delta x \cdot y_i) + \Delta x \]

- \(d_i\) positive => next point at \((x_i+1, y_i)\)
- \(d_i\) negative => next point at \((x_i+1, y_i+1)\)
- At each point, we compute \(d_i\) and decide which pixel to draw
- How do we update it? What is \(d_{i+1}\)?
Updating The Decision Variable

- $d_{k+1}$ is the old value, $d_k$, plus an increment:
  \[ d_{k+1} = d_k + (d_{k+1} - d_k) \]

- If we chose $y_{i+1} = y_i + 1$:
  \[ d_{k+1} = d_k - 2\Delta y + 2\Delta x \]

- If we chose $y_{i+1} = y_i$:
  \[ d_{k+1} = d_k - 2\Delta y \]

- What is $d_1$ (assuming integer endpoints)?
  \[ d_1 = \Delta x - 2\Delta y \]

- Notice that we don’t need $c$ any more
Bresenham’s Algorithm

- For integers, slope between 0 and 1:
  - \( x = x_1, \ y = y_1, \ d = dx - 2dy \), draw \((x, y)\)
  - until \( x = x_2 \)
    - \( x = x + 1 \)
    - If \( d < 0 \) then \{ \( y = y + 1 \), draw \((x, y)\), \( d = d - 2\Delta y + 2\Delta x \) \}
    - If \( d > 0 \) then \{ \( y = y \), draw \((x, y)\), \( d = d - 2\Delta y \) \}

- Compute the constants \((2\Delta y - 2\Delta x \text{ and } 2\Delta y)\) once at the start
  - Inner loop does only adds and comparisons

- For floating point, initialization is harder, \( \Delta x \) and \( \Delta y \)
  will be floating point, **but still no rounding required**
Example: (2,2) to (7,6)

$\Delta x = 5, \Delta y = 4$

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>-3</td>
</tr>
</tbody>
</table>
Example: (2,2) to (7,6)

\[ \Delta x = 5, \quad \Delta y = 4 \]

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>-3</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>-1</td>
</tr>
</tbody>
</table>
Example: (2,2) to (7,6)

\[ \Delta x = 5, \ \Delta y = 4 \]

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>-3</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>-1</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>
Example: (2,2) to (7,6)

\[ \Delta x = 5, \quad \Delta y = 4 \]

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>-3</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>-1</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>-7</td>
</tr>
</tbody>
</table>
Example: (2,2) to (7,6)

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>-3</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>-1</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>-7</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>-5</td>
</tr>
</tbody>
</table>
Example: (2,2) to (7,6)

Δx=5, Δy=4

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>-3</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>-1</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>-7</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>-5</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>-3</td>
</tr>
</tbody>
</table>
Filling Triangles
Filling Triangles
Algorithm

☐ Decide which pixels to fill (samples to use) to represent a triangle?
☐ Calculate the color for each pixel?
Barycentric coordinates

\[ P = \alpha P_0 + \beta P_1 + \lambda P_2 \]

\[ \alpha + \beta + \lambda = 1 \]

\[ 0 \leq \alpha, \beta, \lambda \leq 1 \]
Barycentric coordinates

\[ \alpha = \frac{f_{12}(x, y)}{f_{12}(x_0, y_0)} \]
\[ \beta = \frac{f_{20}(x, y)}{f_{20}(x_1, y_1)} \]
\[ \lambda = \frac{f_{01}(x, y)}{f_{01}(x_2, y_2)} \]

\[ f_{12}(x, y) = (y_1 - y_2)x + (x_2 - x_1)y + x_1y_2 - x_2y_1 \]
\[ f_{20}(x, y) = (y_2 - y_0)x + (x_0 - x_2)y + x_2y_0 - x_0y_2 \]
\[ f_{01}(x, y) = (y_0 - y_1)x + (x_1 - x_0)y + x_0y_1 - x_1y_0 \]
Rasterizing Triangle

- \( y_{\min} = \min(y_0, y_1, y_2) \), \( y_{\max} = \max(y_0, y_1, y_2) \)
- \( x_{\min} = \min(x_0, x_1, x_2) \), \( x_{\max} = \max(x_0, x_1, x_2) \)
- For \( y = y_{\min} \) to \( y_{\max} \)
  - For \( x = x_{\min} \) to \( x_{\max} \)
    - Calculate \( \alpha, \beta, \) and \( \lambda \)
    - If \( 0 \leq \alpha, \beta, \) and \( \lambda \leq 1 \)
      \[ c = \alpha c_0 + \beta c_1 + \lambda c_2 \]
      - Draw \((x, y)\) with color \( c\)
Anti-Aliasing

- Recall: We can’t sample and then accurately reconstruct an image that is not band-limited
  - Infinite Nyquist frequency
  - Attempting to sample sharp edges gives “jaggies”, or stair-step lines
- Solution: Band-limit by filtering (pre-filtering)
  - What sort of filter will give a band-limited result?
- In practice, difficult to do for graphics rendering
Alpha-based Anti-Aliasing

- Set the $\alpha$ of a pixel to simulate a thick line
  - The pixel gets the line color, but with $\alpha \leq 1$

- This supports the correct drawing of primitives one on top of the other
  - Draw back to front, and composite each primitive over the existing image
  - Only some hidden surface removal algorithms support it
Calculating $\alpha$

- Consider a line as having thickness (all good drawing programs do this)
- Consider pixels as little squares
- Set $\alpha$ according to the proportion of the square covered by the line
- The sub-pixel coverage interpretation of $\alpha$

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1/8</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1/4</td>
<td>.914</td>
<td>1/8</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1/4</td>
<td>.914</td>
<td>1/4</td>
<td>0</td>
</tr>
<tr>
<td>1/8</td>
<td>.914</td>
<td>1/4</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1/8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Weighted Sampling

- Instead of using the proportion of the area covered by the line, use convolution to do the sampling
  - Equivalent to filtering the line then point sampling the result

- Place the “filter” at each pixel, and integrate product of pixel and line

- Common filters are cones (like Bartlett) or Gaussians
Post-Filtering (Supersampling)

- Sample at a higher resolution than required for display, and filter image down
  - Easy to implement in hardware
  - Typical is 2x2 sampling per pixel, with simple averaging to get final
    - What kind of filter?

- More advanced methods generate different samples (eg. not on regular grid) and filter properly
  - Issues of which samples to take, and how to filter them
Next Time

- Hidden Surface Removal

-