Rasterization
or “Know where to draw the line”
Reminder - Pipeline

Polygon at [(2,9), (5,7), (8,9)]

3D Model  Transformations
Raster Display

• The screen is a discrete grid of elements called *pixels*

• Shapes drawn by setting some pixels “on”
Rasterization

• How to draw geometric primitives?
  – Convert from geometric definition to pixels
  – *rasterization* = selecting the pixels

• Will be done frequently
  – must be fast:
    • use integer arithmetic
    • use addition instead of multiplication
Terminology

• **Pixel**: Picture element
  – Smallest accessible element in picture
  – Usually rectangular or circular

• **Aspect Ratio**: Ratio between physical dimensions of pixel (not necessarily 1 !!)

• **Dynamic Range**: Ratio between minimal (not zero!) and maximal light intensity emitted by displayed pixel. Measured in bits.
Terminology

• **Resolution**: number of distinguishable rows and columns on device. Measured in
  – Absolute values (1K x 1K)
  – Relative values (300 dots per inch)

• **Screen Space**: discrete 2D Cartesian coordinate system of screen pixels

• **Object Space**: 3D Cartesian coordinate system of the universe where the objects (to be displayed) are embedded
Today

- Drawing lines

- Filling polygons
Naïve Algorithm for Lines

• Line definition: \( ax + by + c = 0 \)
• Also expressed as: \( y(x) = mx + g \)
  - \( m = \) slope
  - \( g = y(0) \)

For \( x=x_{\text{min}} \) to \( x_{\text{max}} \)

\( y = m \times x + g \)
light pixel \((x, y)\)
Slope Dependency

- Only works with \(-1 \leq m \leq 1\): 

\[ m = 3 \]

Extend by symmetry for \(m > 1\)
Problems

• 2 floating-point operations per pixel

• Improvements:
  
  \[ y = m \times x_{\text{min}} + g \]
  For \( x = x_{\text{min}} \) to \( x_{\text{max}} \)
  
  \[ y += m \]
  
  light pixel \((x, y)\)

• Still 1 floating-point operation per pixel

• Compute in floats, pixels in integers
Bresenham Algorithm: Idea

• At each step, choice between 2 pixels (0 ≤ m ≤ 1)
Bresenham Algorithm

• Need a criterion to pick
• Distance between line and center of pixel:
  – the error associated with this pixel

\[ y(x) = mx + g \]

Error pixel 1

Error pixel 2
Error vs decision

Special cases

\[ err_k = y(x_k) - y_k \]

- **m=0**
  - Go right
  - err=0
- **m=1**
  - Go right & up
  - err=0
- **m=1/2**
  - right
  - right & up
  - err = 1/2  err = -1/2
Bresenham Algorithm

- Choose by sign of \( e = e_1 - e_2 \)

\[
\text{if } e < 0 \\
\text{go right} \\
\text{update } e
\]

\[
\text{Else} \\
\text{go right and up} \\
\text{update } e
\]
Bresenham Algorithm

• Choose by sign of \( e = e_1 - e_2 \)

\[
y = y_{\text{min}} \\
\text{For } x = x_{\text{min}} \text{ to } x_{\text{max}} \\
\quad \text{if } e < 0 \\
\quad \quad // \ y \text{ stays fixed} \\
\quad \quad \text{update } e \\
\quad \text{Else} \\
\quad \quad y++ \\
\quad \quad \text{update } e \\
\quad \text{light pixel } (x, y)\]
Bresenham Algorithm

• Needed $y += m$

$y = y_{\text{min}}$
For $x=x_{\text{min}}$ to $x_{\text{max}}$
  if $e < 0$
    // $y$ stays fixed
    $e += m$
  Else
    $y++$
    $e += m-1$
light pixel $(x,y)$
Bresenham Algorithm

• Initialize e

\[ e = m - \frac{1}{2} \]

\[ y = y_{\text{min}} \]

For \( x=x_{\text{min}} \) to \( x_{\text{max}} \)

if \( e < 0 \)

\[ // \text{ y stays fixed} \]

\[ e += m \]

Else

\[ y++ \]

\[ e += m-1 \]

light pixel \((x,y)\)
Bresenham Algorithm
In integers

\[ e = m - \frac{1}{2} \]

\[ y = y_{\text{min}} \]

For \( x = x_{\text{min}} \) to \( x_{\text{max}} \)
  
  if \( e < 0 \)
    
    // \( y \) stays fixed
    
    \[ e += m \]
  
  Else
    
    \[ y++ \]
    
    \[ e += m - 1 \]
  
light pixel \((x, y)\)

\[ m = \frac{\Delta y}{\Delta x} = \frac{y_{\text{max}} - y_{\text{min}}}{x_{\text{max}} - x_{\text{min}}} \]

\[ d = 2 \ e \ \Delta x \]
Bresenham Algorithm
In integers

$$d = 2\Delta y - \Delta x$$
y = y_{\text{min}}
For x = x_{\text{min}} \text{ to } x_{\text{max}}
\quad \text{if } d < 0
\quad \quad // \ y \text{ stays fixed}
\quad \quad d += 2\Delta y
\quad \text{Else}
\quad \quad y++
\quad \quad d += 2\Delta y - 2\Delta x
\quad \text{light pixel } (x, y)

$$m = \frac{\Delta y}{\Delta x} = \frac{y_{\text{max}} - y_{\text{min}}}{x_{\text{max}} - x_{\text{min}}}$$

$$d = 2 \ e \ \Delta x$$
The Error

\[ e_k = e_1 - e_2 \]
\[ e_1 = y(x_{k+1}) - y_k \]
\[ e_2 = y_{k+1} - y(x_{k+1}) \]

\[ e_k = e_1 - e_2 = y(x_{k+1}) - y_k - (y_{k+1} - y(x_{k+1})) = 2y(x_{k+1}) - y_{k+1} - y_k = 2mx_{k+1} + 2g - y_{k+1} - y_k = 2m(x_{k+1} + 1) + 2g - (y_k + 1) - y_k = 2m(x_{k} + 1) + 2g - 2y_k - 1 \]

\[ y(x) = mx + g \]
\[ x_{k+1} = x_k + 1 \]
\[ y_{k+1} = y_k + 1 \]
The Error
Integers

• Line from \((x_1, y_1)\) to \((x_2, y_2)\)

\[
m = \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1}
\]

\[
e_k = 2 \frac{\Delta y}{\Delta x} (x_k + 1) + 2g - 2y_k - 1
\]

\[
d_k = e_k \Delta x
\]
Updating the Error

\[ e_k = 2m(x_k + 1) + 2g - 2y_k - 1 \]

\[
d_k = e_k \Delta x = 2\Delta y(x_k + 1) + 2g\Delta x - 2y_k\Delta x - \Delta x = 2x_k\Delta y - 2y_k\Delta x + 2\Delta y - \Delta x + 2g\Delta x\]

constant

\[
d_{k+1} = 2x_{k+1}\Delta y - 2y_{k+1}\Delta x + \text{const}
\]

\[
d_{k+1} - d_k = 2\Delta y(x_{k+1} - x_k) - 2\Delta x(y_{k+1} - y_k)
\]

Always 1

1 or 0
Initializing $d$

\[ y = mx + g \]
\[ y - y_1 = \frac{y_2 - y_1}{x_2 - x_1} (x - x_1) \]
\[ y = \frac{\Delta y}{\Delta x} x + \left( y_1 - \frac{\Delta y}{\Delta x} x_1 \right) \]
\[ g = \left( y_1 - \frac{\Delta y}{\Delta x} x_1 \right) \]

\[ d_1 = e_1 \Delta x = 2x_1 \Delta y - 2y_1 \Delta x + 2\Delta y - \Delta x + 2g\Delta x = \]
\[ = 2x_1 \Delta y - 2y_1 \Delta x + 2\Delta y - \Delta x + 2y_1 \Delta x - 2x_1 \Delta y = \]
\[ = 2\Delta y - \Delta x \]
Midpoint Line Algorithm

Let's try!
Generalizations

• Circles

• Other algebraic curves

• Line intensity

• Line thickness

• Anti-aliasing
Polygon Fill
The Problem

• Problem:
  – Given a closed 2D polygon fill its interior with specified color on graphics display

• Assumptions:
  – Polygon is simple
    • No self intersections
  – Polygon is simply connected
    • No holes

• Solutions:
  – Flood fill
  – Scan conversion
Flood Fill Algorithm

• Let $P$ be a polygon whose boundary is already drawn
• Let $C$ be the color to fill the polygon
• Let $p = (x, y) \in P$ be a point inside $P$
Flood Fill

FloodFill(Polygon \( P \), \( x \), \( y \), \( C \))
if not (OnBoundary\((x, y, P)\) or Colored\((x, y, C)\))
begin
    PlotPixel\((x, y, C)\);
    FloodFill\((P, x + 1, y, C)\);
    FloodFill\((P, x, y + 1, C)\);
    FloodFill\((P, x, y - 1, C)\);
    FloodFill\((P, x - 1, y, C)\);
end ;
Pros and Cons?

- Correctness
- Simplicity
- Efficiency in time and space
- Limitations
  - Very large stack required
Basic Scan Conversion Algorithm

• Let $P$ be a polygon with $n$ vertices $v_0$ to $v_{n-1}$ ($v_n = v_0$)
• Let $C$ be the color
• Each intersection of straight line with boundary moves in/out the polygon
• Detect (and set) pixels inside the polygon boundary
Basic Scan Conversion

\textbf{ScanConvert} (Polygon }P\text{, Color }C\text{)

For }y := 0 \text{ to ScreenYMax do

\[ I \leftarrow \text{Points of intersections of edges of } P \text{ with line } Y = y ; \]

Sort }I\text{ in increasing }X\text{ order and

Fill with color }C\text{ alternating segments ;

end ;
Special Cases
## Comparison

<table>
<thead>
<tr>
<th>Flood Fill</th>
<th>Scan Conversion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Very simple</td>
<td>More complex</td>
</tr>
<tr>
<td>Discrete algorithm in screen space</td>
<td>Discrete algorithm in object and/or screen space</td>
</tr>
<tr>
<td>Requires <code>GetPixelVal</code> system call</td>
<td>Device independent</td>
</tr>
<tr>
<td>Requires a seed point</td>
<td>No seed point required</td>
</tr>
<tr>
<td>Requires very large stack</td>
<td>Requires small stack</td>
</tr>
<tr>
<td>Common in paint packages</td>
<td>Used in image rendering</td>
</tr>
<tr>
<td>Unsuitable for line-based Z-buffer</td>
<td>Suitable for line-based Z-buffer</td>
</tr>
</tbody>
</table>
References

• Interactive Computer Graphics, Angel, 6\textsuperscript{th} edition, chapters 6.9,6.10