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

3D Model

Transformations

Polygon at [(2,9), (5,7), (8,9)]
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 \( x_{\text{max}} \)

\[ y = mx + g \]

light pixel \((x, y)\)
Slope Dependency

• Only works with $-1 \leq m \leq 1$:

$m = 3$

$m = 1/3$

Extend by symmetry for $m > 1$
Problems

- 2 floating-point operations per pixel
- Improvements:
  \[ y = m \cdot x_{\text{min}} + g \]
  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 \(0 \leq m \leq 1\)
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 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 \( e = e_1 - e_2 \)

if \( e < 0 \)
  go right
  update \( e \)

Else
  go right and up
  update \( e \)
Bresenham Algorithm

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

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

For \( x = x_{\text{min}} \) to \( x_{\text{max}} \)
  
  if \( e < 0 \)
  
  // \( y \) stays fixed
  
  update \( e \)
  
Else
  
  \( y++ \)
  
  update \( e \)
  
light pixel \((x, y)\)
Update $e$

- $m < 1/2$, first iterations

$$(x_k, y_k)$$

$$e_1 = m, \quad e_2 = 1 - m$$

$$e = 2m - 1$$

$$(x_k + 1, y_k)$$

$$e_1 = 2m, \quad e_2 = 1 - 2m$$

$$e = 4m - 1$$

$$e += 2m$$
Update $e$

- $m > 1/2$, first iterations

$$(x_k, y_k)$$

$$e_1 = m, \quad e_2 = 1 - m$$

$$e = 2m - 1$$

$$(x_k + 1, y_k + 1)$$

$$e_1 = 2m - 1, \quad e_2 = 2 - 2m$$

$$e = 4m - 3$$

$$e += 2m - 2$$
Update $e$

$y = y_{\text{min}}$

For $x=x_{\text{min}}$ to $x_{\text{max}}$

if $e < 0$

    // $y$ stays fixed
    $e += 2m$

Else

    $y++$
    
    $e += 2m-2$

light pixel $(x,y)$
Initialize $e$

- Initialize $e$
  
  $e = 2^m - 1$
  
  $y = y_{\text{min}}$

  For $x=x_{\text{min}}$ to $x_{\text{max}}$
    
    if $e < 0$
      
      // $y$ stays fixed
      
      $e += 2^m$
    
    Else
      
      $y++$
      
      $e += 2^{m-2}$

    light pixel $(x,y)$
**Bresenham Algorithm**

In integers

\[
e = 2m - 1 \quad d = 2\Delta y - \Delta x
\]

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

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

if \(e < 0\)

// \(y\) stays fixed

\[
e \mathrel{+}= 2m \quad d \mathrel{+}= 2\Delta y
\]

Else

\[y++\]

\[
e \mathrel{+}= 2m - 2 \quad d \mathrel{+}= 2\Delta y - 2\Delta x
\]

light pixel \((x, y)\)

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

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

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

\[ m = \frac{\Delta y}{\Delta x} = \frac{y_{\text{max}} - y_{\text{min}}}{x_{\text{max}} - x_{\text{min}}} \]
\[ d = e \Delta x \]
Bresenham Algorithm
In integers

\textbf{Line}(x_1, y_1, x_2, y_2)

\begin{align*}
x &= x_1 \\
y &= y_1 \\
\Delta x &= x_2 - x_1 \\
\Delta y &= y_2 - y_1 \\
d &= 2\Delta y - \Delta x \\
\Delta e &= 2\Delta y \\
\Delta ne &= 2\Delta y - 2\Delta x
\end{align*}

\textbf{PlotPixel}(x, y)

\begin{align*}
\text{while } (x < x_2) \text{ do} \\
&\text{For } x = x_1 \text{ to } x_2 \\
&\text{if } d < 0 \quad \text{then} \quad d += \Delta e \\
&\text{else} \quad y++ \\
&\text{d } += \Delta ne \\
&\text{PlotPixel} \ (x, y)
\end{align*}
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

\[
\text{FloodFill}(\text{Polygon } P, \text{ int } x, \text{ int } y, \text{ Color } C) \\
\text{if not } (\text{OnBoundary}(x, y, P) \text{ or Colored}(x, y, C)) \\
\text{begin} \\
\quad \text{PlotPixel}(x, y, C); \\
\quad \text{FloodFill}(P, x + 1, y, C); \\
\quad \text{FloodFill}(P, x, y + 1, C); \\
\quad \text{FloodFill}(P, x, y - 1, C); \\
\quad \text{FloodFill}(P, x - 1, y, C); \\
\text{end;} 
\]
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