CS 335
Graphics and Multimedia

2D Drawings: Lines
Digital Concepts of Drawing in Raster Arrays

PIXEL is a single array element at (x,y)

- No smaller drawing unit exists
Pixel Value = 1 bit, 8 bit, 24 bit, ….

= address into hardware look up table to determine display color, intensity.

**Pixel based Geometry**

Points, lines, circles, conics, curves, splines, polygons, shaded polygons, text fonts & icons.

all basically reduce to scan conversion problem:

FIND Digital algorithms for continuous geometric concepts
Drawing Lines

From \((x_1,y_1)\) to \((x_2,y_2)\)

**Equation:**

\[ y = mx + b \]

\[ m = \frac{(y_2-y_1)}{(x_2-x_1)} \]

\[ b = y_1 - m \times x_1 \]

compute -- \(m, b\)
for \(x=x_1\) to \(x_2\), step 1

\[ y = m \times x + b \]

pixel( \(x\), round(\(y\)), color)

loop
DDA Algorithm

(Digital Differential Analyzer)
[DDA avoids the multiple by slope \( m \)]

Equation:

\[
\begin{align*}
x_{k+1} &= x_k + 1 \\
y_{k+1} &= y_k + m
\end{align*}
\]

if slope \( |m| > 1 \)

\[
\begin{align*}
y_{k+1} &= y_k + 1 \\
x_{k+1} &= x_k + \frac{1}{m}
\end{align*}
\]

(see page 88 – Hearn)
DDA Algorithm

(Digital Differential Analyzer)
[avoid the float multiple by slope m]

slope \( m, y \) – floats

Problems:
1. Necessary to perform float addition
2. Necessary to have float representation
3. Necessary to perform -- \texttt{round}()

Float representations/operations are more expensive than integer operations. We would like to avoid them if possible.
Mid-Point Line Algorithm (Bresenham)

Uses only integer math to draw a line

Lets assume slope-m is positive and less than 1

we know that: $x_{k+1} = x_k + 1$

at $x_{k+1}$ we need to make a decision for next pixel

draw either at:

choice 1 -- $(x_k + 1, y_k)$
choice 2 -- $(x_k + 1, y_k + 1)$

How do we decide between choice 1 or 2?
Mid-Point Line Algorithm (Bresenham)

Consider the distance with the actual line \((y=mx+b)\) at the two choices:

\[
d_1 = y - y_k = m(x_k+1) + b - y_k
\]

\[
d_2 = (y_k+1) - y = y_k + 1 - m(x_k+1) - b
\]

Difference between these two separations is:

\[
d = d_1 - d_2
\]

- if \(d > 0\), move to \((y_k + 1)\)
- else stay at \(y_k\)
Mid-Point Line Algorithm (Bresenham)

Still have a “float” operation in calculation of “d”

Let's create a new decision operator by multiplying by $\Delta x$ (recall $m = \Delta y / \Delta x$)

$$p_k = \Delta x (d_1 - d_2) \quad \sim \text{note } p_k\text{ sign is the same as } (d_1 - d_2)$$

$$= 2 \Delta y x_k - 2 \Delta x y_k + 2 \Delta y + \Delta x (2b - 1) \quad [\text{Eq 1}]$$

$$= 2 \Delta y x_k - 2 \Delta x y_k + C$$

We can use $p_k$ as a decision operator – instead of $(d_1 - d_2)$
Mid-Point Line Algorithm (Bresenham)

\[ p_{k+1} = 2 \Delta y \ x_{k+1} - 2 \Delta x \ y_{k+1} + C \]

What is the change from \( p_k \) to \( p_{k+1} \)?

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

\[ p_{k+1} = p_k + 2 \Delta y \ (1) - 2 \Delta x \ (y_{k+1} - y_k) \]

either 1 or 0

so –

if \( p_k < 0 \)  // \( y_{k+1} = y_k \)
\[ p_{k+1} = p_k + 2 \Delta y \]

else  // \( y_{k+1} = y_k + 1 \)
\[ p_{k+1} = p_k + 2 \Delta y - 2 \Delta x \]

Use [Eq. 1] with \( x_1 \) and \( y_1 \) to find:

\[ p_0 = 2 \Delta y - \Delta x \]
Mid-Point Line Algorithm (Bresenham)

Compute constants: $\Delta x$, $\Delta y$, $2\Delta y$, $2\Delta y - 2\Delta x$

Calculate: $p_0 = 2\Delta y - \Delta x$

```
plot(x1,y1)
for x=x1 to x2  // let y=y1
    if (p < 0)
        p = p + 2\Delta y
    else
        y++
        p = p + 2\Delta y - 2\Delta x
    end
plot(x,y)
loop
```

Note -- main loop:
Only integer math.
No float representation, or operations needed.
Constants: 2dy, 2dx are integers.
Mid-Point Line Algorithm (Bresenham)

**Example:**

(20,10) to (30,18)

\( \Delta x = 10, \quad \Delta y = 8 \)

(slope 0.8)

\[
p0 = 2\Delta y - \Delta x = 6
\]
\[
2\Delta y = 16 [E]
\]
\[
2\Delta y - 2\Delta x = -4 [NE]
\]

---

<table>
<thead>
<tr>
<th>( k )</th>
<th>( P_k )</th>
<th>((x_{k+1},y_{k+1}))</th>
<th>( k )</th>
<th>( P_k )</th>
<th>((x_{k+1},y_{k+1}))</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>6</td>
<td>(21,11)</td>
<td>5</td>
<td>6</td>
<td>(26,15)</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>(22,12)</td>
<td>6</td>
<td>2</td>
<td>(27,16)</td>
</tr>
<tr>
<td>2</td>
<td>-2</td>
<td>(23,12)</td>
<td>7</td>
<td>-2</td>
<td>(28,16)</td>
</tr>
<tr>
<td>3</td>
<td>14</td>
<td>(24,13)</td>
<td>8</td>
<td>14</td>
<td>(29,17)</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>(25,14)</td>
<td>9</td>
<td>10</td>
<td>(30,18)</td>
</tr>
</tbody>
</table>

---

*full algorithm -- page 90-91 Hearn
adjusts for slope \( m > 1 \)
re-orders \( x_1, x_2, y_1, y_2 \) as necessary*
Addressing the Framebuffer

Logical representation

Physical representation in memory – 1D Array

$X_{\text{max}}$
1D memory as a 2D image

imaData[ (y * width + x) * depth ]

- row
- size of row
- offsets into the proper scanline
- offset in the row
- offsets to the proper pixel

\[ \text{imaData[ (y * width + x) * depth ]} \]
Images with “depth” > 1

- Individual color bands are “packed” into one array
- Called the “packing format”
- Typical arrangements
  - RGB, RGBA, ABGR, BGR
1D memory as a 2D Framebuffer

- Indexing 1D memory
- Coordinates \((x, y)\) [or \((i, j)\)]
  (assume array is unsigned char)

  - 1 byte pixel (grayscale)
    \[
    \text{frameBuffer}[ y \times \text{Xmax} + x ] = \text{grayscale or index color}
    \]

  - 4 byte pixel (RGBA)
    \[
    \text{frameBuffer}[ (y\times\text{Xmax}+x)\times4 + 0 ] = R \\
    \text{frameBuffer}[ (y\times\text{Xmax}+x)\times4 + 1 ] = G \\
    \text{frameBuffer}[ (y\times\text{Xmax}+x)\times4 + 2 ] = B \\
    \text{frameBuffer}[ (y\times\text{Xmax}+x)\times4 + 3 ] = A
    \]
Summary of Concepts

- Implicit representation of a line
- Incremental algorithm for more efficient computation
- Integer algorithm by introducing harmless scale factor
- Logic must be added to handle all line orientations
- Frame-buffer representation