2D Line

- Implicit representation:
  \[ax + by + c = 0\]

- Explicit representation:
  \[y = mx + b\]
  \(m = \frac{y_1 - y_0}{x_1 - x_0}\)

- Parametric representation:
  \[P(t) = \begin{pmatrix} x \\ y \end{pmatrix} = P_0 + t(P_1 - P_0), \ t \in [0, 1]\]

Scan Conversion - Lines

\[y = mx + B\]

*Basic naïve algorithm*

For \(x = x_0\) to \(x_1\)

\[y = mx + B\]

PlotPixel \((x, \text{round}(y))\)

end;

For each iteration: 1 float multiplication, 1 addition, 1 Round

Scan Conversion - Lines

\[y = mx + B\]

slope = \(m = \frac{y_1 - y_0}{x_1 - x_0}\)

offset = \(B = y_1 - mx_1\)

Assume \(|m| < 1\)

Assume \(y_0 < y_1\)

4-connectivity

8-connectivity

4-adjacency

8-adjacency
An 8-connected closed curve with a hole

A 4-connected open arc with a hole

Incremental Algorithm:

\[
y_{i+1} = mx_{i+1} + B = mx_i + m \Delta x + B = y_i + m \Delta x
\]

if \( m = 1 \) then \( y_{i+1} = y_i + m \)

Algorithm:

\[y_{i+1} = mx_i + B\]

For \( x = x_0 \) to \( x_1 \)

PlotPixel(round(x),y)

end;

\[x = x_0\]

Special Cases:

\( m = \pm 1 \) (diagonals)

\( m = 0 \) (horizontal, vertical)

Symmetric Cases:

if \( x_0 > x_1 \) for \( |m| \neq 1 \)

\[y_{i+1} = y_i + m\]

swap((x_0,y_0),(x_1,y_1))

Basic Line Drawing:

For each iteration:

1 addition, 1 Round.

Drawback:

- Accumulated error
- Round arithmetic
- Round operations

Pseudo Code for Basic Line Drawing:

Assume \( x_1 > x_0 \) and line slope absolute value is \( \leq 1 \)

Line(x_0,y_0,x_1,y_1)
begin
float dx, dy, x, y, slope;
dx = x_1 - x_0;
dy = y_1 - y_0;
slope = dy/dx;
y = y_0;
for x = x_0 to x_1 do
begin
PlotPixel(x,round(y));
y = y + m;
end;
end;
Midpoint (Bresenham) Line Drawing

Assumptions:
- $x_0 < x_1$, $y_0 < y_1$
- $0 < \text{slope} < 1$

Given $(x_p, y_p)$:
- next pixel is $E = (x_p + 1, y_p)$ or $NE = (x_p + 1, y_p + 1)$

Bresenham's sign($M - Q$) determines NE or E

$M = (x_p + 1, y_p + 1/2)$

The vertical distance is equivalent to the Euclidean distance $d_1 = y - y_i = m(x_i + 1) + b - y_i$
$d_2 = (y_i + 1) - y = y_i + 1 - m(x_i + 1) - b$

$d_1 - d_2 > 0$?

Bresenham's Line Algorithm

Const1 = 2dy;
Const2 = 2dy - 2dx;
set_pixel(x1, y1);
$x = x1; y = y1;

while (x < x2) {
if (f < 0) {f += Const1;}
else {
if ($y$ is incremented) f += Const2; y++;
}
set_pixel(x, y);

Offsets

The image is a linear memory...
In 8-connected choose either a h or d move
The midpoint M is located at (x + 1, y + 1/2)

The line passes above M so it is a d move to

In 4-connected choose either a h or v move
The midpoint M is located at (x + 1/2, y + 1/2)

The line passes above M so it is a d move to

The line passes below M so it is a h move to

The line passes below M so it is a h move to

The line passes above M so it is a v move to
Midpoint Line Drawing (cont.)

\[ y = \frac{dy}{dx} \]

Implicit form of a line:

\[ f(x, y) = ax + by + c = 0 \]

Decision Variable:

\[ f(x, y) = 0 \]

\[ f(x, y) > 0 \]

\[ f(x, y) < 0 \]

If \( f(M) \) was chosen

First point = \((x_0, y_0)\), first MidPoint = \((x_0 + 1, y_0 + \frac{1}{2})\)

\[ f_{\text{start}} = f(x_0 + 1, y_0 + \frac{1}{2}) = ax_0 + by_0 + c + a + \frac{b}{2} \]

\[ d_{\text{start}} = dy - dx \]

Enhancement:

To eliminate fractions, define:

\[ f(x, y) = 2(ax + by + c) = 0 \]

\[ d_{\text{start}} = 2dy - 2dx \]

Mid-point Line Algorithm

\[ \Delta y = 2dy; \]

\[ \Delta x = 2dx; \]

\[ f = 2dy - dx; \]

set_pixel(x1, y1);

\( x = x1; y = y1; \)

while \((x < x2)\) {
    if \((f < 0)\) {
        \( f = f + \Delta y; \)
    } else {
        \( f = f + \Delta x; \)
    }
    set_pixel(x, y);
}

Incremental Algorithm:

Initialization:

First point = \((x_0, y_0)\), first MidPoint = \((x_0 + 1, y_0 + \frac{1}{2})\)

\[ f_{\text{start}} = f(x_0 + 1, y_0 + \frac{1}{2}) = ax_0 + by_0 + c + a + \frac{b}{2} \]

Enhancement:

To eliminate fractions, define:

\[ f(x, y) = 2(ax + by + c) = 0 \]

\[ d_{\text{start}} = 2dy - 2dx \]

Midpoint Line Drawing - Summary

- The sign of \( f(M) \) indicates whether to move East or North-East.
- At the beginning \( d = 2(y_1 - y_0) = 2(2dy - dx) \).
- The increment in \( d \) (after this step) is:
  - If we moved East: \( \Delta y = 2dy \)
  - If we moved North-East: \( \Delta x = 2dx \)

Comments:

- Integer arithmetic (\( dx \) and \( dy \) are integers).
- One addition for each iteration.
- No accumulated errors.
**Drawing Circles**

- Implicit representation (centered at the origin with radius R):
  \[ x^2 + y^2 - R^2 = 0 \]
- Explicit representation:
  \[ y = \sqrt{R^2 - x^2} \]
- Parametric representation:
  \[
  \begin{bmatrix}
  x \\
  y
  \end{bmatrix} = \begin{bmatrix}
  R \cos \theta \\
  R \sin \theta
  \end{bmatrix}
  \quad t \in [0, \pi]
  \]

**Scan Conversion - Circles**

**Basic Algorithm**

For \( x = -R \) to \( R \)

\[ y = \sqrt{R^2 - x^2} \]

PlotPixel(\( x, \text{round}(y) \))

PlotPixel(\( x, -\text{round}(y) \))

**Comments:**
- Square-root operations are expensive.
- Float arithmetic.
- Large gap for \( x \) values close to \( R \).

**Exploiting Eight-Way Symmetry**

For a circle centered at the origin:
- If \((x, y)\) is on the circle then \(-y, x\), \(y, -x\), \(-y, -x\), \(-x, -y\), \(-x, y\), \(y, x\) are on the circle as well.

Therefore we need to compute only one octant (45°) segment.

**Threshold Criteria**

\[
d(x,y) = f(x,y) = x^2 + y^2 - R^2 = 0
\]

- \( f(x,y) > 0 \)
- \( f(x,y) < 0 \)
- \( f(x,y) = 0 \)

**Circle Midpoint (for one octant)**

(The circle is located at \((0,0)\) with radius \(R\))

- We start from \((x_0, y_0) = (0, -R)\).
- One can move either \( h \) or \( d \).
- Again, \( f(x, y) \) will be a decision variable at the midpoint.
Mid-point circle (for one octant) Algorithm

One may mirror (*), creating the other seven octants.

Initialization: \( A_0 = B_0 \) (home exercise)
\[
f = (1/2)^2 + (-R - 1/2)^2 - R^2
\]
\[
\text{set_pixel}(x = x_1, y = y_1);
\]

while (in the octant){
  if \( f < 0 \) {
    \( f += A_0; \quad A_0 += 2; \)
    \( x++; \)
  } else {
    \( f += B_0; \quad A_0 += 2; \)
    \( y++; \)
  }
\}
\text{set_pixel}(x,y);

How to update \( f \) - the value at \( M \)

If \( a \) was chosen, as in lines we update \( M \)

\( M_{i+1} = (x+1, y), \) thus
\[
f_{i+1} = (x+1)^2 + y^2 - R^2
\]
and
\[
f_{i+1} = f_i + 2x + 1.
\]

Now, \( A_0 \) is NOT a constant; it is a linear term, so we update it as well:
\[
A_{i+1} = 2(x+1) + 1, \text{ which is } A_{i+1} = A_i + 2.
\]

Similarly if \( b \) was chosen

Mid-point

The arc passes above \( M \) so it is a \( v \) move to  •

Mid-point circle (for one octant) Algorithm

One may mirror (*), creating the other seven octants.