2D Line

- Implicit representation:
  \[ \alpha x + \beta y + \gamma = 0 \]
- Explicit representation:
  \[ y = mx + B \quad m = \frac{y_1 - y_0}{x_1 - x_0} \]
- Parametric representation:
  \[
P = P_0 + (P_1 - P_0) t \quad t \in [0,1]
\]
Scan Conversion - Lines

\[ y = mx + B \]

**Basic Algorithm**

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

\[ y = mx + B \]

PlotPixel(\( x \), round(\( y \)))

end;

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

Incremental Algorithm:

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

if \( \Delta x = 1 \) then \[ y_{i+1} = y_i + m \]

Algorithm

\[ y = y_0 \]

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

PlotPixel(\( x \), round(\( y \)))

\[ y = y + m \]

end;
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 + slope;
        end;
end;
```
Symmetric Cases:

\[ |m| \geq 1 \]

\[ x = x_0 \]

For \( y = y_0 \) to \( y_1 \)

\[ x = x + \frac{1}{m} \]

PlotPixel(round\((x), y)\)

end;

Special Cases:

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

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

Symmetric Cases:

if \( x_0 > x_1 \) for \( |m| \leq 1 \) or \( y_0 > y_1 \) for \( |m| \geq 1 \)

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

Basic Line Drawing:

For each iteration: 1 addition, 1 Round.

Drawback:

- Accumulated error
- float arithmetic
- Round operations

Midpoint (~Bresenham) Line Drawing

Assumptions:

- \( x_0 < x_1 \), \( y_0 < y_1 \)
- \( 0 < \) 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: \( \text{sign}(M-Q) \) determines \( NE \) or \( E \)

\( M = (x_p+1, y_p+1/2) \)
Bresenham’s Line Algorithm

\[ d_1 = y - y_i = m(x_i + 1) + b - y_i \]
\[ d_2 = (y_i + 1) - y = y_i + 1 - m(x_i + 1) - b \]
\[ d_1 - d_2 > 0 ? \]

Bresenham’s Line Algorithm

\[ d_1 - d_2 = 2m(x_i + 1) - 2y_i + 2b - 1 \]
\[ d_1 - d_2 = 2(dy/dx)(x_i + 1) - 2y_i + 2b - 1 \]
\[ dx(d_1-d_2) = 2dy*x_i + 2dy - 2dx*y_i + 2dx*b - dx \]
\[ f_i = dx(d_1-d_2) \]
\[ f_{i+1} - f_i = 2dy(x_{i+1} - x_i) - 2dx(y_{i+1} - y_i) \]
If \( y \) is incremented then \( f_{i+1} = f_i + 2dy - 2dx \)
else \( f_{i+1} = f_i + 2dy \)
Bresenham’s Line Algorithm

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

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

Bresenham’s Line Algorithm

Const1 = 2dy;
Const2 = 2dy - 2dx;
p = A + n*y + x;
offset_h = sign(dx);
offset_d = sign(dx) + n*sign(dy)
f = 2dy - dx;
*p = color;
d8 = dx;

while (d8--){
  if (f < 0){
    f += Const1;
    p += offset_h;
  } else {
    f += Const2;
    p += offset_d;
  }
  *p = color;
}
Midpoint Line Drawing (cont.)

\[ y = \frac{dy}{dx} x + B \]

Implicit form of a line:
\[ f(x,y) = ax + by + c = 0 \]
\[ f(x,y) = dy \cdot dx y + B \cdot dx = 0 \]

Decision Variable:
\[ d = f(M) = f(x_p + 1, y_p + 1/2) = ax_p + b(y_p + 1/2) + c \]

- choose NE if \( d > 0 \)
- choose E if \( d \leq 0 \)

Incremental Algorithm:

What happens at \( x_p + 2 \)?

If E was chosen at \( x_p + 1 \)
\[ M = (x_p + 2, y_p + 1/2) \]
\[ d_{new} = f(x_p + 2, y_p + 1/2) = a(x_p + 2) + b(y_p + 1/2) + c \]
\[ d_{old} = f(x_p + 1, y_p + 1/2) = a(x_p + 1) + b(y_p + 1/2) + c \]
\[ d_{new} = d_{old} + a = d_{old} + dy \]
\[ d_{new} = d_{old} + \Delta_e \]

If NE was chosen at \( x_p + 1 \)
\[ M = (x_p + 2, y_p + 3/2) \]
\[ d_{new} = f(x_p + 2, y_p + 3/2) = a(x_p + 2) + b(y_p + 3/2) + c \]
\[ d_{old} = f(x_p + 1, y_p + 1/2) = a(x_p + 1) + b(y_p + 1/2) + c \]
\[ d_{new} = d_{old} + a + b = d_{old} + dy - dx \]
\[ d_{new} = d_{old} + \Delta_{NE} \]
Incremental Algorithm:

Initialization:
First point = (x₀, y₀), first MidPoint = (x₀ + 1, y₀ + 1/2)
\[ d_{\text{start}} = f(x₀ + 1, y₀ + 1/2) = a(x₀ + 1) + b(y₀ + 1/2) + c = ax₀ + by₀ + c + a + b/2 = f(x₀, y₀) + a + b/2 = a + b/2 \]
\[ d_{\text{start}} = dy - dx/2 \]

Enhancement:
To eliminate fractions, define:
f(x, y) = 2(ax + by + c) = 0
\[ d_{\text{start}} = 2dy - dx \]
• The sign of \( f(x₀ + 1, y₀ + 1/2) \) indicates whether to move **East** or **North-East**.
• At the beginning \( d = f(x₀ + 1, y₀ + 1/2) = 2dy - dx \).
• The increment in \( d \) (after this step) is:
  – If we moved **East**: \( \Delta E = 2dy \)
  – If we moved **North-East**: \( \Delta NE = 2dy - 2dx \)

Midpoint Line Drawing - Summary
• The sign of \( f(x₀ + 1, y₀ + 1/2) \) indicates whether to move **East** or **North-East**.
• At the beginning \( d = f(x₀ + 1, y₀ + 1/2) = 2dy - dx \).
• The increment in \( d \) (after this step) is:
  – If we moved **East**: \( \Delta E = 2dy \)
  – If we moved **North-East**: \( \Delta NE = 2dy - 2dx \)

• Comments:
  – Integer arithmetic (dx and dy are integers).
  – One addition for each iteration.
  – No accumulated errors.
  – By symmetry, we deal with \( 0 > \text{slope} > -1 \)
Pseudo Code for Midpoint Line Drawing:

Assume \( x_1 > x_0 \) and \( 0 < \text{slope} \leq 1 \)

\[
\text{Line}(x_0, y_0, x_1, y_1) \begin{align*}
\text{begin} & \quad \text{int} \ dx, \ dy, \ x, \ y, \ d, \ \Delta_x, \ \Delta_y; \\
\ & \quad x := x_0; \quad y := y_0; \\
\ & \quad dx := x_1 - x_0; \quad dy := y_1 - y_0; \\
\ & \quad d := 2 \cdot dy - dx; \\
\ & \quad \Delta_e := 2 \cdot dy; \quad \Delta_ne := 2 \cdot (dy - dx); \\
\ & \quad \text{PlotPixel}(x, y); \\
\text{while} (x < x_1) \text{ do} & \quad \begin{align*}
\ & \quad \text{if} \ (d < 0) \ \text{then} \\
\ & \quad \quad \begin{align*}
\ & \quad \quad d := d + \Delta_e; \\
\ & \quad \quad x := x + 1; \\
\ & \text{end} \\
\ & \quad \text{else} \ \text{begin} \\
\ & \quad \quad d := d + \Delta_ne; \\
\ & \quad \quad x := x + 1; \\
\ & \quad \quad y := y + 1; \\
\ & \quad \text{end} \\
\ & \quad \text{PlotPixel}(x, y); \\
\ & \end{align*}
\end{align*}
\text{end;}
\end{align*}
\]

Drawing Circles

- Implicit representation (centered at the origin with radius \( R \)):
  \[
  x^2 + y^2 - R^2 = 0
  \]
- Explicit representation:
  \[
  y = \pm \sqrt{R^2 - x^2}
  \]
- Parametric representation:
  \[
  \begin{pmatrix}
  x \\
  y
  \end{pmatrix} =
  \begin{pmatrix}
  R \cos(t) \\
  R \sin(t)
  \end{pmatrix}, \quad t \in [0, 2\pi]
  \]
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 -
\((y,x)\) \((y,-x)\) \((x,-y)\) \((-x,-y)\) \((-y,-x)\) \((-y,x)\) \((-x,y)\)
are on the circle as well.
Therefore we need to compute only one octant (45°) segment.
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 East or South-East.
- Again, \(d(x, y)\) will be a threshold criteria at the midpoint.

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

Threshold Criteria

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

• At the beginning \(d_{\text{start}} = d(x_0 + 1, y_0 - 1/2) = d(1, R - 1/2) = 5/4 - R\)

• If \(d < 0\) we move **East**:
  \[
  \Delta_E = d(x_0 + 2, y_0 - 1/2) - d(x_0 + 1, y_0 - 1/2) = 2x_0 + 3
  \]

• If \(d > 0\) we move **South-East**:
  \[
  \Delta_{SE} = d(x_0 + 2, y_0 - 3/2) - d(x_0 + 1, y_0 - 1/2) = 2(x_0 - y_0) + 5
  \]

• \(\Delta_E\) and \(\Delta_{SE}\) are not constant anymore.

• Since \(d\) is incremented by integer values, we can use \(d_{\text{start}} = 1 - R\), yielding an integer algorithm. This has no affect on the threshold criteria.

---

Midpoint Circle Algorithm

**Circle Octant2 (R)**

```plaintext
begin
  int x, y, d;
  x := 0;
  y := R;
  d := 1 - R;
  PlotPixel(x, y);
  while (y > x) do
      if (d < 0) then /* East */
          begin
              d := d + 2x + 3;
              x := x + 1;
          end;
      else begin
              d := d + 2(x - y) + 5;
              x := x + 1;
              y := y - 1;
              PlotPixel(x, y); (*)
          end;
end;
```

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