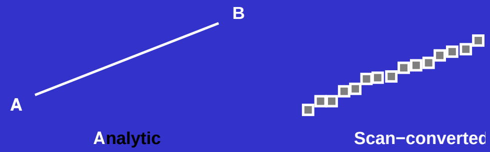
Raster Algorithms

"The process of converting geometric primitives into their discrete approximations"

Scan Conversion:

⇒ Approximate geometric primitives (analytically defined) by a set of pixels, stored in frame-buffer or memory.



Clipping:

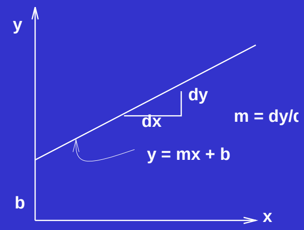
⇒ Only sections of primitives determined to be within a clipping region is drawn (scan-converted).

Speed:

⇒ Algorithms must be very efficient. Why?

Drawing Lines

Given the line end points (x_0, y_0) and (x_1, y_1)



Line Equation

$$y = mx + b$$

m = Slope, b = y intercept

To determine

the sequence of points between (x_0, y_0) and (x_1, y_1) on the raster grid (end points are in screen coordinates).

Brute Force Algorithm

for $i = x_0$ **to** $i = x_1$

$\{$

$y_i = mx_i + b$

PLOT (x_i, ROUND (y_i))

$\}$

Inefficient: Involves multiply and rounding.

DDA Algorithm

Features

- Exploits the fact that the line equation is a linear function that needs to be evaluated over a regular lattice.
- **Constant** increments in both dimensions to obtain successive points along the line eliminates multiplies in the inner loop.

$$
\begin{aligned}
y_{i+1} &= mx_{i+1} + b \\
&= m(x_i + \Delta x) + b \\
&= (mx_i + b) + m\Delta x \\
&= y_i + m(1)
\end{aligned}
$$

DDA Algorithm

Algorithm:

$m = (y_1 - y_0)/(x_1 - x_0)$
for x_i = x_0 **to** $x_i = x_1$
$\{$
    plot ($x_i, Round(y_i)$)
    $y_i = y_i + m$
    $x_i = x_i + 1$
$\}$

Inefficient, involves division to compute slope, and rounding.

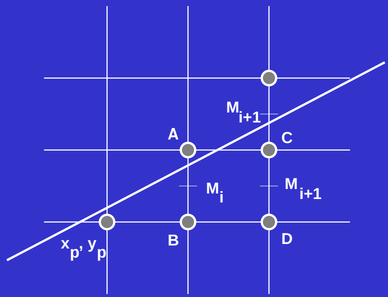
MidPoint Line Algorithm

For lines and circles, same as **Bresenham's** algorithm.

Features:

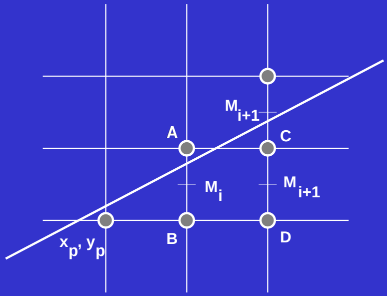
- Uses only integer operations.
- Uses incremental calculations to determine successive pixels.

Idea



- Determine location of mid point, M_i with respect to the line.
- Mid points M_i are computed using incremental calculations.

MidPoint Line Algorithm(contd)

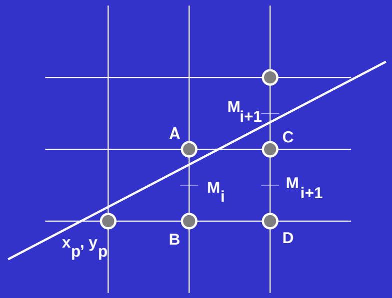


$$
\begin{aligned}
y &= (dy/dx).x + B \\
dx.y &= dy.x + B.dx, \text{ or}
\end{aligned}
$$

$$
\begin{aligned}
F(x, y) &= dy.x - dx.y + B.dx = 0 \\
&= a.x + b.y + c = 0, \quad \text{(a = dy, b = -dx, c = B .dx)}
\end{aligned}
$$

$$
\begin{aligned}
d = F(x, y) &= 0, & x, y \textbf{ on } \text{the line} \\
&> 0, & x, y \textbf{ below } \text{the line} \\
&< 0, & x, y \textbf{ above } \text{the line}
\end{aligned}
$$

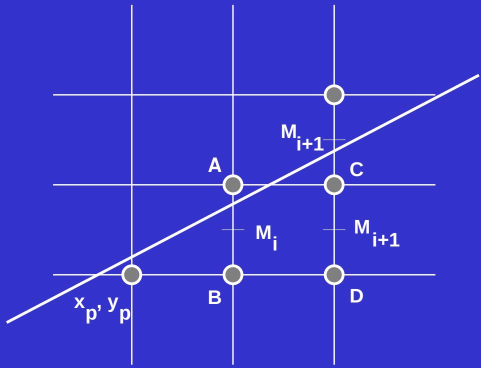
MidPoint Line Algorithm(contd)



Strategy:

- The sign of $d = F(x, y)$, **the decision variable**, will determine whether A or B is chosen as the next pixel on the line.
- *“Insert M_i into $F(x, y)$ and check the sign of F.”*

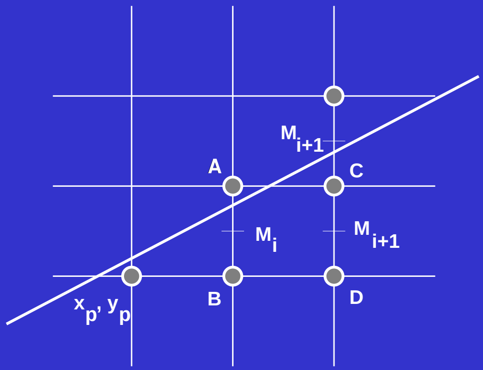
Efficient calculation of d



Case 1: A is chosen

$$
\begin{aligned}
M_i &= F(x_p + 1, y_p + 1/2) \\
M_{i+1} &= F(x_p + 2, y_p + 3/2) \\
M_{i+1} - M_i &= F(x_p + 2, y_p + 3/2) - F(x_p + 1, y_p + 1/2) \\
&= \{a(x_p + 2) + b(y_p + 3/2) + c\} - \\
&\quad \{a(x_p + 1) + b(y_p + 1/2) + c\} \\
&= (a + b) \\
M_{i+1} &= M_i + (a + b) = M_i + dy - dx
\end{aligned}
$$

Efficient calculation of d(contd)



Case 2: B is chosen

$$
\begin{aligned}
M_{i+1} &= F(x_p + 2, y_p + 1/2) \\
M_{i+1} - M_i &= F(x_p + 2, y_p + 1/2) - F(x_p + 1, y_p + 1/2) \\
&= a \\
M_{i+1} &= M_i + (a) \\
&= M_i + dy
\end{aligned}
$$

Initialization

$$
\begin{aligned}
d &= F(x_0 + 1, y_0 + 1/2) \\
&= a(x_0 + 1) + b(y_0 + 1/2) + c \\
&= (ax_0 + by_0 + c) + a + b/2 \\
&= a + b/2
\end{aligned}
$$

or

$d = 2a + b$ (only sign of d is important)

Hence

$$
\begin{aligned}
d_1 = 2(a + b) &= 2(dy - dx) \\
d_2 = 2a &= 2(dy)
\end{aligned}
$$

Final Algorithm (Quadrant 1 only)

```
d = 2 dy - dx
d1 = 2 dy - 2 dx
d2 = 2 dy

for x = x0 to x1 by 1
{
        if (d ≤ 0
                d = d + d1
        else
        {
                d = d + d2
                y = y + 1
        }
        plot (x, y)
}
```