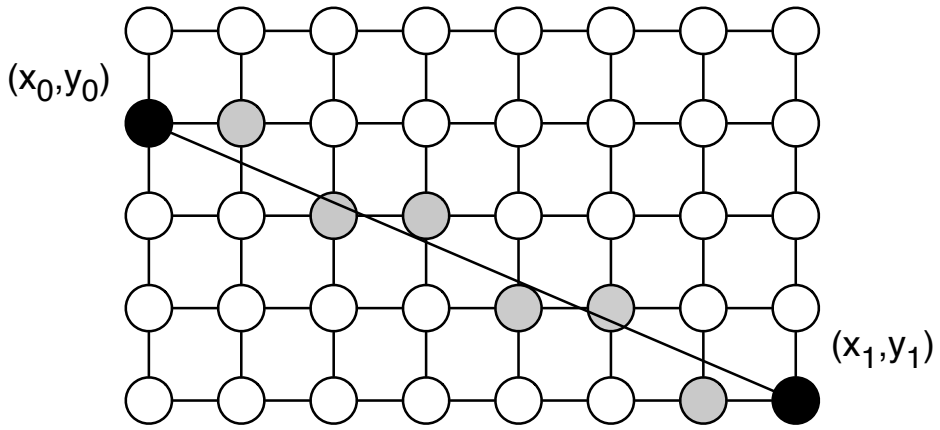
Line Rasterization



- Line begins and ends at pixel coordinates (x_0, y_0) and (x_1, y_1).
- Assume slope $|m| \leq 1$ and $x_0 \leq x_1$
- Which pixels do we choose?

Digital Differential Analyzer (DDA)

- Slope (store as fixed or floating point)

$$m = \frac{y_1 - y_0}{x_1 - x_0}$$

- Increment x by 1 on each step ($|m| < 1, x_0 \leq x_1$)

$$x_{i+1} = x_i + 1$$

- Update y via line equation

$$\begin{aligned} y_{i+1} &= m x_{i+1} + b \\ &= m(x_i + 1) + b \\ &= y_i + m \end{aligned}$$

DDA for line

$|m| \leq 1,\ x_0 \leq x_1$

```
m = (y1 - y0)/(x1 - x0);        /* m : fixed or float */
y = y0;                         /* y : fixed or float */
for (x = x0; x <= x1; x++) {    /* x : integer */
  setPixel(x, round(y));
  y += m;
}
```

- `setPixel(x,y)` "sets" the pixel at the logical address `(x,y)` in the *framebuffer*.
- `y` is rounded to the nearest scan line.

DDA for line

$|m| > 1,\ y_0 \leq y_1$

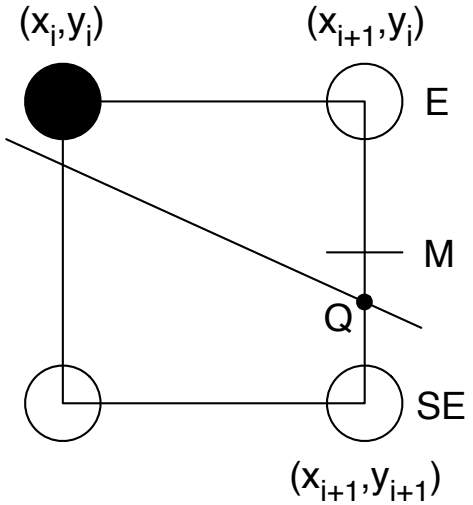
```
m = (x1 - x0)/(y1 - y0);        /* m : inverse slope */
x = x0;
for (y = y0; y <= y1; y++) {
  setPixel(round(x), y);
  x += m;
}
```

- Roles of x and y reversed.

Midpoint Line Algorithm

- Classic Bresenham (1965) line algorithm uses *only* integer arithmetic.
- Computes (x_{i+1}, y_{i+1}) incrementally by using the calculation already done at (x_i, y_i).
- Algorithm generalizes to circles and "oriented" ellipses, but not to other more general conic sections.
- Assume $0 < m < 1$ for now.

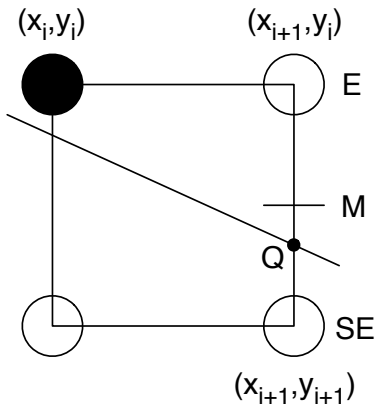
Choosing the next pixel based on the midpoint



- (x_i, y_i) was chosen in the previous iteration.
- Choose either E or SE based Q's relation to M
- Use implicit function for line : $F(x, y) = ax + by + c$

$F(M) > 0$	"above" line
$F(M) < 0$	"below" line
$F(M) = 0$	"on" line

Decision Variable d



$$d = F(M) = F(x_i + 1, y_i + 1/2)$$

- Midpoint criterion
  - If $d > 0$ choose SE pixel
  - If $d < 0$ choose E pixel
  - If $d = 0$ choose either pixel (be consistent)
- We update d after each iteration

Update d using either Δ_E or Δ_{SE}

if E pixel chosen, $d_{new} = d_{old} + \Delta_E$

$$\begin{aligned} d_{new} &= F(x_i + 2, y_i + 1/2) \\ &= a(x_i + 2) + b(y_i + 1/2) + c \\ d_{old} &= F(x_i + 1, y_i + 1/2) \\ &= a(x_i + 1) + b(y_i + 1/2) + c \\ \Delta_E &= d_{new} - d_{old} = a \end{aligned}$$

if SE pixel chosen, $d_{new} = d_{old} + \Delta_{SE}$

$$\begin{aligned} d_{new} &= F(x_i + 2, y_i + 3/2) \\ &= a(x_i + 2) + b(y_i + 3/2) + c \\ d_{old} &= a(x_i + 1) + b(y_i + 1/2) + c \\ \Delta_{SE} &= d_{new} - d_{old} = a + b \end{aligned}$$

Initializing d

Initial value: $d = F(M)$ for the first midpoint M

$$\begin{aligned} d &= F(x_0 + 1, y_0 + 1/2) \\ &= a(x_0 + 1) + b(y_0 + 1/2) + c \\ &= \underbrace{F(x_0, y_0)}_{0} + a + b/2 \end{aligned}$$

We eliminate the fraction $b/2$ by using $2F$ instead of F

$$\begin{aligned} d_0 &= 2F(x_0 + 1, y_0 + 1/2) \\ &= 2a(x_0 + 1) + 2b(y_0 + 1/2) + 2c \\ &= \underbrace{2F(x_0, y_0)}_{0} + 2a + b \\ \Delta_E &= 2a \\ \Delta_{SE} &= 2(a + b) \end{aligned}$$

Bresenham Algorithm

$0 < m < 1,\ x0 < x1$

```
b = x0 - x1;  a = y1 - y0;
d = 2*a + b;
dE = 2*a;
dSE = 2*(a + b);
setPixel(x0, y0);
for (x = x0, y = y0; x < x1; x++) {
  if (d <= 0)  /* choose E pixel */
    d += dE;
  else {
    d += dSE;  /* choose SE pixel */
    y++;
  }
  setPixel(x,y);
}
```

```
void midpoint_line(int x0, int y0, int x1, int y1) {
  int x,y;              /* current pixel coordinate */
  int a,b;              /* major,minor line dimensions */
  int d;                /* decision variable */
  int xdiag, ydiag;     /* minor axis (diaganol) increment */
  int xdom, ydom;       /* major axis (dominant) increment */
  int dd_dom, dd_diag; /* delta d for dominant and diaganol increment */
  int count;            /* number of pixels to set */

  a = x1 - x0;  b = y1 - y0;
  xdiag = (a < 0) ? -1 : +1;   ydiag = (b < 0) ? -1 : +1;
  a = abs(a); b = abs(b);
  if (a < b) {swap(a,b); xdom = xdiag; ydom = 0;}
  else       {           xdom = 0;     ydom = ydiag;}

  dd_dom = b << 1;
  d = dd_dom - a;
  dd_diag = (b - a) << 1;

  x = x0;  y = y0;
  for (count = a+1; count; count--) {
    setPixel(x,y);
    if (d <= 0) {d += dd_dom;  x += xdom;   y += ydom;}
    else        {d += dd_diag; x += xdiag;  y += ydiag;}
  }
}
```