2D Line

- Implicit representation:

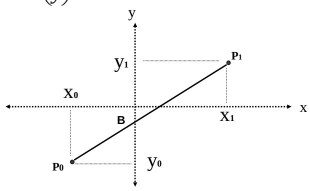
$$\alpha x + \beta y + \gamma = 0$$

- Explicit representation:

$$y = mx + B \qquad m = \frac{y_1 - y_0}{x_1 - x_0}$$

- Parametric representation:

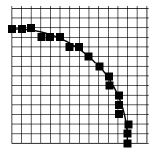
$$P = \begin{pmatrix} x \\ y \end{pmatrix} \qquad P = P_0 + (P_1 - P_0)t \quad t \in [0..1]$$



Scan Conversion

Lines and Circles

(Chapter 3 in Foley & Van Dam)



Scan Conversion - Lines

y = mx + B

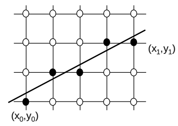
Basic naïve algorithm

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

Scan Conversion - Lines

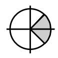
y = mx + B



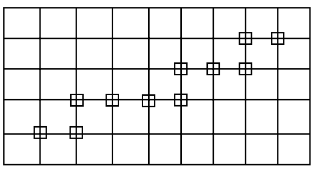
slope = m = $\frac{y_1 - y_0}{x_1 - x_0}$

offset= B = y_1-mx_1

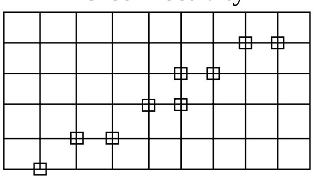
Assume $|m| \leq 1$
Assume $x_0 \leq x_1$



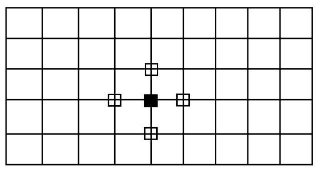
4-connectivity



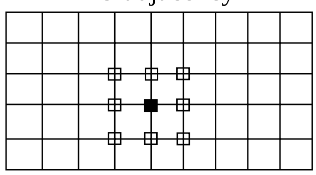
8-connectivity



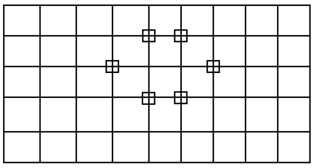
4-adjacency



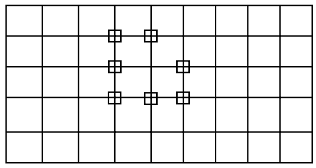
8-adjacency



An 8-conncected closed curve with a hole



A 4-connected open arc with a hole



Incremental Algorithm:

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

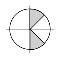
if $\Delta x = 1$ then $\boxed{y_{i+1} = y_i + m}$

Algorithm

```
y=y0
For x = x0  to  x1
     PlotPixel(x,round(y))
     y =  y  +  m
end;
```

Symmetric Cases:

$|m| \geq 1$



```
x = x0
For  y = y0 to  y1
     PlotPixel(round(x),y)
     x =  x  +  1/m
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))

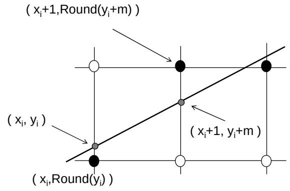
Basic Line Drawing:

For each iteration: 1 addition, 1 Round.
Drawback:
- Accumulated error
- float arithmetic
- Round operations

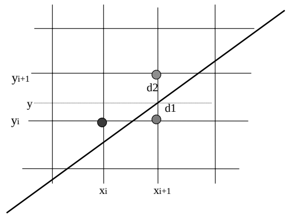
Pseudo Code for Basic Line Drawing:

Assume $x_1 > x_0$ and line slope absolute value is ≤ 1

```
Line(x0,y0,x1,y1)
begin
     float dx, dy, x, y, slope;
     dx := x1-x0;
     dy := y1-y0;
     slope := dy/dx;
     y := y0;
     for x:=x0 to x1 do
     begin
          PlotPixel( x,Round(y) );
          y := y+slope;
     end;
end;
```



Bresenham's Line Algorithm



$$y = m(x_i + 1) + b$$

$$d_1 = y - y_i = m(x_i + 1) + b - y_i$$

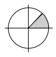
$$d_2 = (y_i + 1) - y = y_i + 1 - m(x_i + 1) - b$$

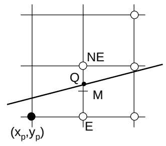
$\boxed{d1 - d2 > 0\ ?}$

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: sign(M-Q) determines NE or E

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

The vertical distance is equivalent to the Euclidean distance

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

$d1 - d2 = 2m(x_i + 1) - 2y_i + 2b - 1$

$d1 - d2 = 2(dy/dx)(x_i + 1) - 2y_i + 2b - 1$

$dx(d1-d2) = 2dy*x_i + 2dy - 2dx*y_i + 2dx*b - dx$

$f_i = dx(d1-d2)$

$f_{i+1} - f_i = 2dy(x_{i+1} - x_i) - 2dx(y_{i+1} - y_i)$

If y is incremented then $f_{i+1} = f_i + 2dy-2dx$

else $f_{i+1} = f_i + 2dy$

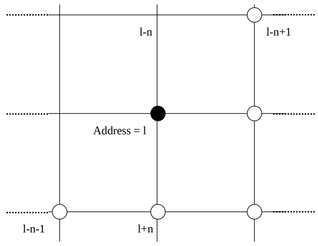
Bresenham's Line Algorithm

```
Const1 = 2dy;
Const2 = 2dy - 2dx;
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;
}
```

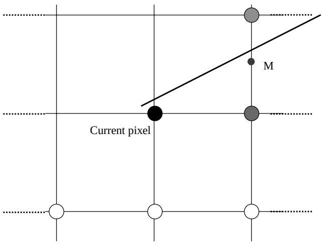
Offsets

- The image is a linear memory…



Mid-point

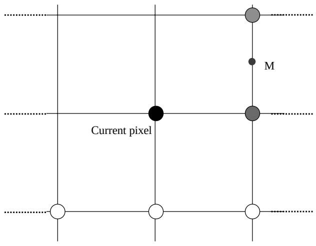
The line passes above M so it is a d move to



Mid-point

In 8-connected choose either a h or d move

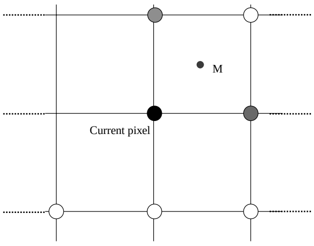
The midpoint M is located at (x +1,y +1/2)



Mid-point

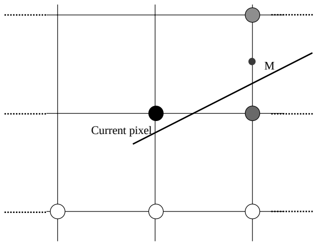
In 4-connected choose either a h or v move

The midpoint M is located at (x + 1/2,y +1/2)



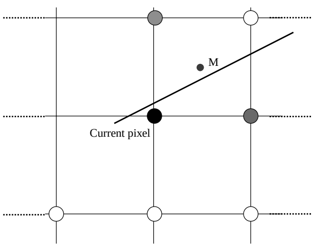
Mid-point

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



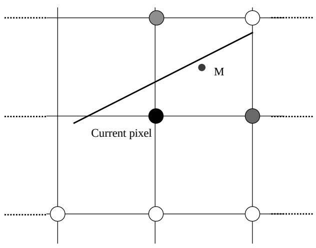
Mid-point

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

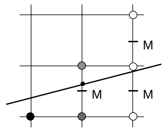


Mid-point

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



How to update f - the value at M



If ● was chosen

Mi = (x,y)
Mi+1 = (x+1,y), thus
f_i = ax + by + c, and

f_{i+1} = a(x+1) + by + c, or

$f_{i+1} = f_i$ + a.

Since a is constant we denote it with Δ_h, and we have:

f += Δ_h

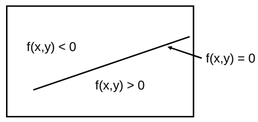
Midpoint Line Drawing (cont.)

$$\boxed{y = \frac{dy}{dx}x + B}$$

Implicit form of a line:

$$\boxed{f(x,y) = ax + by + c = 0}$$

$$f(x,y) = dy\,x - dx\,y + B\,dx = 0$$



Decision Variable :

$f = f(M) = f(x_p+1, y_p+1/2) = a(x_p+1) + b(y_p+1/2) + c$

The sign of f defines the move

Incremental Algorithm:

Initialization:

First point = (x_0,y_0), first MidPoint = $(x_0+1,y_0+1/2)$

$f_{start} = f(x_0+1,y_0+1/2) = a(x_0+1) + b(y_0+1/2) + c$
$= ax_0 + by_0 + c + a + b/2$
$= f(x_0,y_0) + a + b/2 = a + b/2$

$$\boxed{d_{start} = dy - dx/2}$$

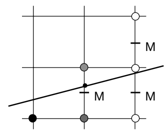
Enhancement:

To eliminate fractions, define:

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

$$\boxed{d_{start} = 2dy - dx}$$

How to update f - the value at M



If ● was chosen

Mi = (x,y)
Mi+1 = (x+1,y+1), thus
f_i = ax + by + c, and

f_{i+1} = a(x+1) + b(y+1) + c, or

$f_{i+1} = f_i$ + a + b.

Since a and b are constants we denote their sum with Δ_d, and we have:

f += Δ_d

Midpoint Line Drawing - Summary

- The sign of $f(x_0+1,y_0+1/2)$ indicates whether to move *East* or *North-East*.
- At the beginning $d=f(x_0+1,y_0+1/2)=2dy-dx$.
- 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.

Mid-point Line Algorithm

```
Δh = 2dy;
Δd = 2dy - 2dx;
f = 2dy - dx;
set_pixel(x1,y1);
x = x1; y = y1;

while (x++ < x2){
   if (f < 0)
         f += Δh;
   else {
         f += Δd;
         y++;
        }
set_pixel(x,y);
}
```

Scan Conversion - Circles

<u>Basic Algorithm</u>

```
For x = -R  to  R
    y =  sqrt(R²-x²)
    PlotPixel(x,round(y))
    PlotPixel(x,-round(y))
 end;
```

Comments:

- square-root operations are expensive.
- Float arithmetic.
- Large gap for x values close to R.

Drawing Circles

- Implicit representation (centered at the origin with radius R):

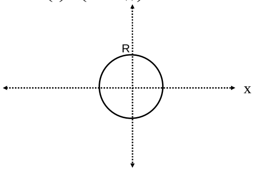
$$x^2 + y^2 - R^2 = 0$$

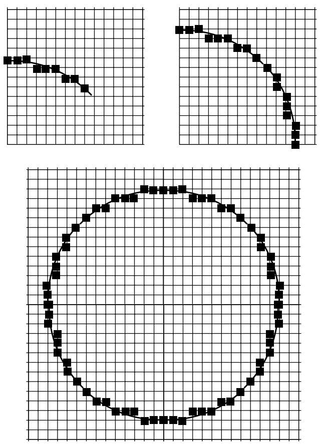
- Explicit representation:

$$y = \pm\sqrt{R^2 - x^2}$$

- Parametric representation:

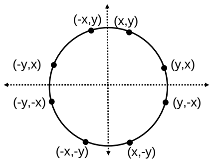
$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} R\cos(t) \\ R\sin(t) \end{pmatrix} \qquad t \in [0..2\pi]$$





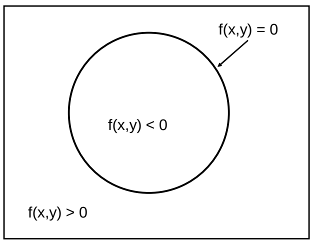
Exploiting Eight-Way Symmetry

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 (45º) segment.



Threshold Criteria

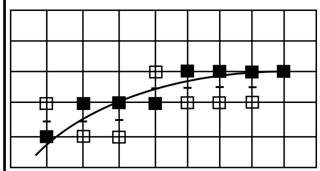
d(x,y)=f(x,y) = $x^2 + y^2 - R^2 = 0$



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.



How to update f - the value at M

If was chosen, as in lines we update M

Mi+1 = (x+1,y), thus

$f_{i+1} = (x+1)^2 + y^2 - R^2$

and

$f_{i+1} = f_i + 2x + 1.$

Now, Δ_h is NOT a constant ,
but a linear term, so we update
it as well:

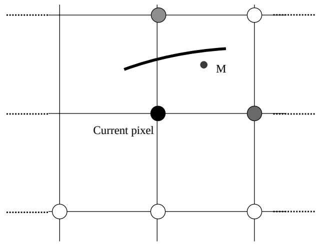
$\Delta_{h+1} = 2(x+1) + 1$, which is

$\Delta_{h+1} = \Delta_h + 2.$

Similarly if 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 octans.

Initialize Δ_h and Δ_d (home exercise)

f = $(1/2)^2 + (-R - 1/2)^2 - R^2$

```
set_pixel(x = x1,y = y1);
while (in the octant){
   if (f < 0) {
         f += Δh;  Δh += 2;
         X++;
   else {
         f += Δv;  Δh += 2;
        Y++;
         }
set_pixel(x,y);
}
```