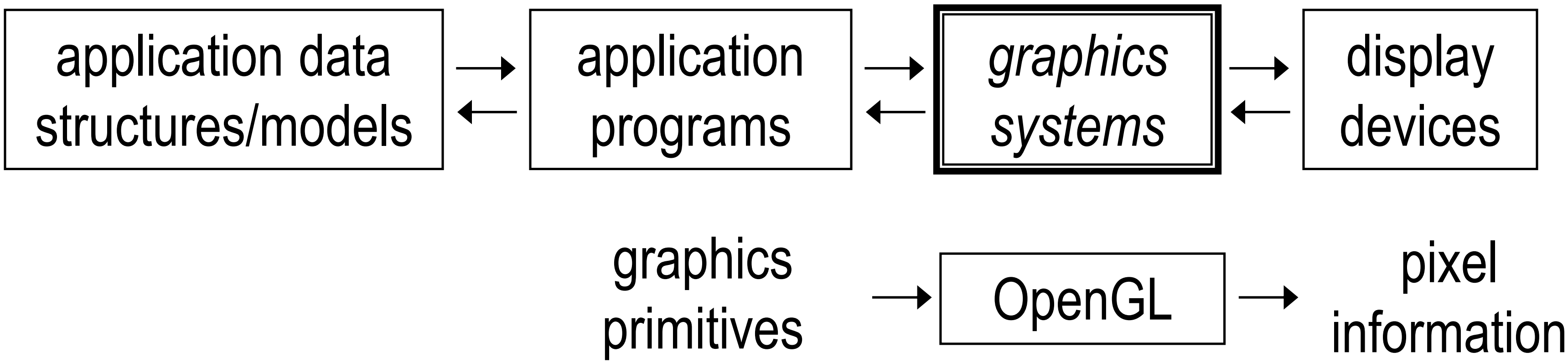
Line Drawing
and Generalization

Outline

- overview
- line drawing
- circle drawing
- curve drawing

1. Overview



application data structures/models
application programs
graphics systems
display devices
graphics primitives
OpenGL
pixel information

Geometric Primitives

Building blocks for a 3D object

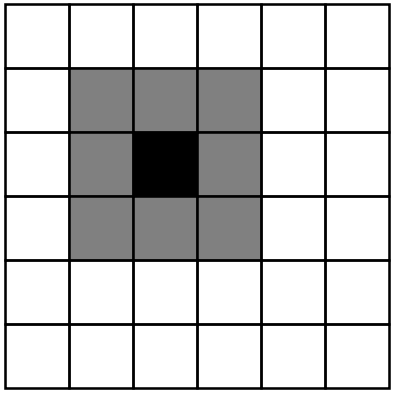
Application programs must describe their service requests to a graphics system using geometric primitives !!!

↑

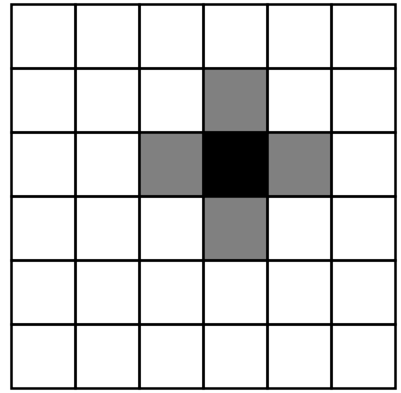
points, lines, polygons

Why not providing data structures directly to the graphics system?

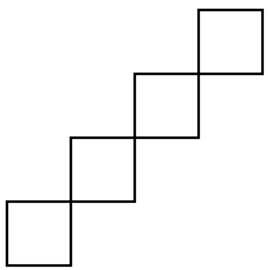
Continuity

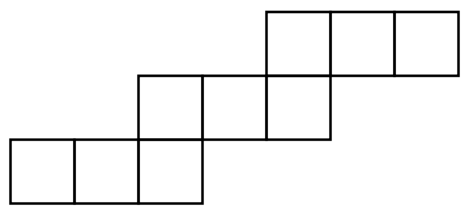


8-connectivity
(king – continuity)

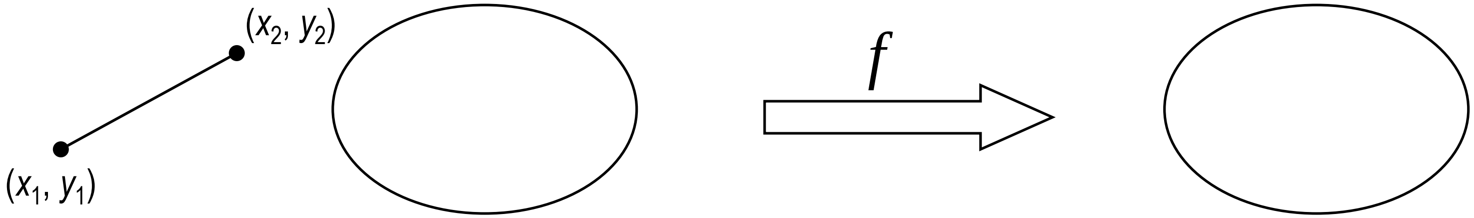


4-connectivity
(rook – continuity)





2. Line Drawing



$L_1 = \{(x,y) \mid x_1 \le x \le x_2, y_1 \le y \le y_2, ax+by+c=0\}$

$|L_1| = \infty$

$L_2 = \{(x_i, y_i) \mid i = 1,2,\ldots,n\}$

$|L_2| < \infty$

$\boxed{f : L_1 \Rightarrow L_2}$

$\therefore$ quality degradation!!

Line drawing is at the heart of many graphics programs.

$\therefore$ Smooth, even, and <u>continuous</u> as much as possible !!!

Simple and fast !!!

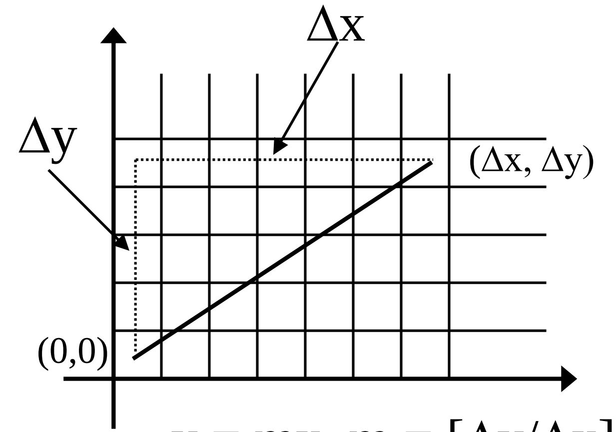
Bresenham's Line Drawing Algorithm
via
Pragram Transformation

- additions / subtractions only
- integer arithmetic
- not programmers' point of view but system developers' point of view

```
var yt : real; Δx, Δy, xi, yi : integer;
for xi := 0 to Δx do begin
            yt := [Δy/Δx]*xi;      *
            yi := trunc(yt+[1/2]);
            display(xi,yi);
end;
```

Eliminate multiplication !!!

```
var yt : real; Δx, Δy, xi, yi : integer;
yt := 0;
for xi := 0 to Δx do begin                  **
            yi := trunc(yt + [1/2]);
            display(xi,yi);
            yt := yt+[Δy/Δx]
end;
```



$y = mx$, $m = [\Delta y/\Delta x]$

$\Delta x \geq \Delta y \quad \therefore m \leq 1$

Δx, Δy: positive integers

```
var ys : real; Δx, Δy, xi, yi : integer;

ys := 1/2;

for xi := 0 to dx do begin

        yi := trunc(ys);   ***

        display(xi,yi);

        ys := ys+[Δy/Δx]

end;
```

$$y_s = \underline{y_{si}} + \underline{y_{sf}}$$

integer part   fractional part

```
var ysf : real; Δx, Δy, xi, ysi : integer;

ysi := 0;

ysf := 1/2;

for xi := 0 to Δx do begin

        display(xi,ysi);

        if ysf+[Δy/Δx] < 1 then begin      ****

                ysf := ysf + [Δy/Δx];

        end else begin

                ysi := ysi + 1;

                ysf := ysf + [Δy/Δx-1];

        end;

end;
```

$$y_{sf} = \frac{1}{2} + \frac{\Delta y}{\Delta x} - 1 = \frac{\Delta y}{\Delta x} - \frac{1}{2}$$

$$y_{sf} + \boxed{\frac{\Delta y}{\Delta x} - 1} < 0$$

$$\downarrow$$

$$y_{sf} < 0$$

$$2\Delta x \cdot y_{sf} \stackrel{\triangle}{=} r$$

var Δx, Δy, xi, ysi, r : integer;

ysi := 0;

$\boxed{y_{sf} = \frac{\Delta y}{\Delta x} - \frac{1}{2}}$ $\longleftarrow$ $2\Delta x \cdot y_{sf} = 2\Delta y - \Delta x$

for xi := 0 to Δx do begin

display(xi,ysi);

if $\boxed{y_{sf} < 0}$ **then begin** $\longleftarrow$ $2\Delta x \cdot y_{sf} < 0$

$\boxed{y_{sf} = y_{sf} + \frac{\Delta y}{\Delta x}}$ $\longleftarrow$ $2\Delta x \cdot y_{sf} = 2\Delta x \cdot y_{sf} + 2\Delta y$

end else begin

ysi := ysi + 1;

$\boxed{y_{sf} = y_{sf} + \left[\frac{\Delta y}{\Delta x} - 1\right]}$ $\longleftarrow$ $2\Delta x \cdot y_{sf} = 2\Delta x \cdot y_{sf} + [2\Delta y - 2\Delta x]$

end;

end;

$$2\Delta x \cdot y_{sf} \stackrel{\triangle}{=} r$$

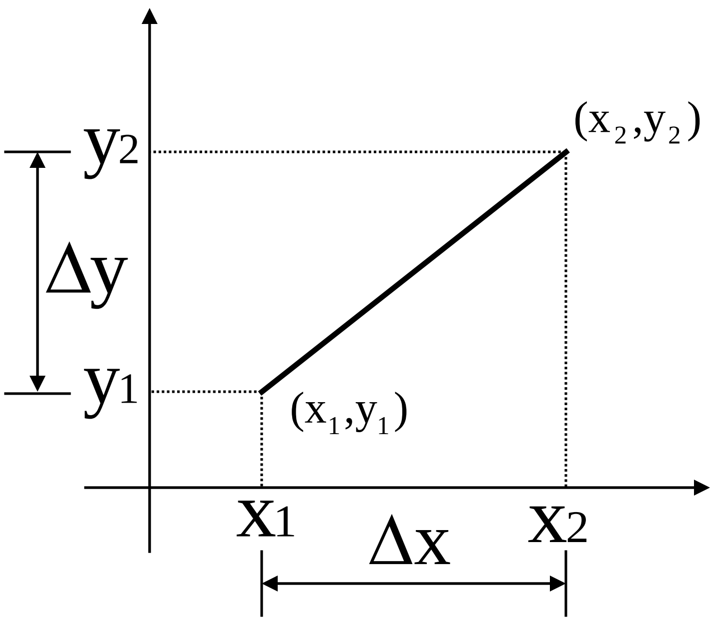
```
var Δx, Δy, xi, ysi, r : integer;
ysi := 0;
r := 2*Δy - Δx;
for xi := 0 to Δx do begin
        display(xi,ysi);
        if r < 0 then begin
                r := r + [2*Δy];
        end else begin
                ysi := ysi + 1;
                r := r + [2*Δy -2*Δx ];
        end;
end;
```

Bresenham's Algorithm !!!

No multiplication/ division.

No floating point operations.

Line-Drawing Algorithms



$$y = mx + b$$

$$b = y_1 - mx_1$$

$$m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{\Delta y}{\Delta x}$$

$$\boxed{\therefore \Delta y = m \Delta x}$$

Assumptions

$x_1 < x_2 \quad (\Delta x > 0)$

$m > 0$

DDA(Digital Differential Analyzer) Algorithm

basic idea

$$\boxed{\begin{aligned}\Delta y &= m\Delta x \\ \Delta x &= \frac{1}{m}\Delta y\end{aligned}} \qquad m > 0$$

Take unit steps with one coordinate and

calculate values for the other coordinate

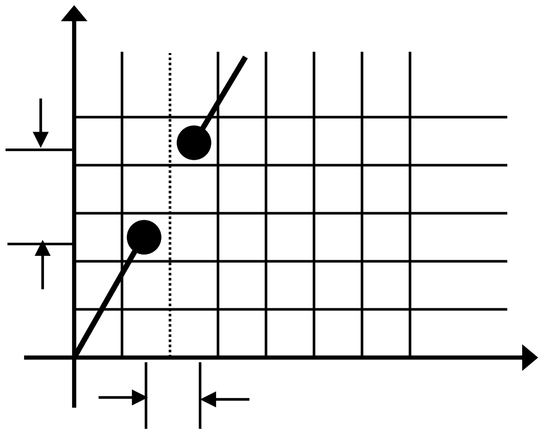
i.e.

$$\boxed{\begin{aligned}x_{i+1} &:= x_i + 1 \\ y_{i+1} &:= y_i + m\end{aligned}} \qquad 0 < m \le 1$$

or

$$\boxed{\begin{aligned}y_{i+1} &:= y_i + 1 \\ x_{i+1} &:= x_i + \frac{1}{m}\end{aligned}} \qquad 1 < m$$

Why?



discontinuity !!

Assumption : $0 \le m < 1$, $x1 < x2$

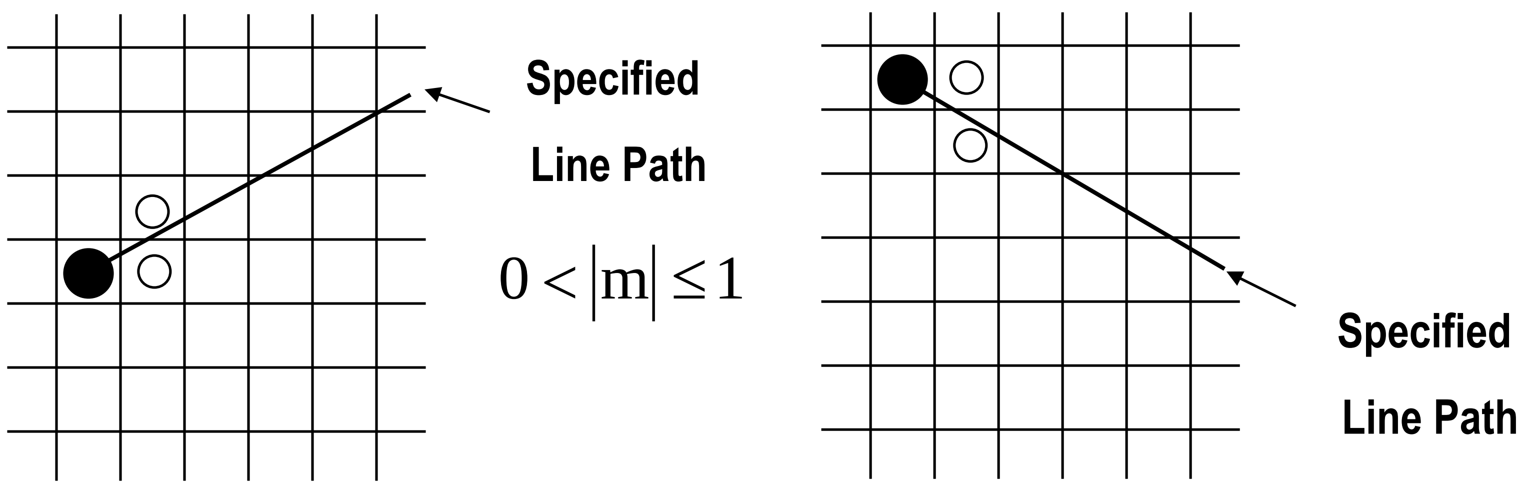
```
procedure dda (x1, y1, x2, y2 : integer);
  var
  Δx, Δy, k : integer;
  x, y : real
  begin
         Δx := x2 - x1;
         Δy := y2 - y1;
         x := x1; y := y1;
         display(x,y);
         for k := 1 to Δx do begin
                x := x + 1;
                y := y + [Δy/Δx];
                 display(round(x),round(y));
         end { for k }
   end; { dda }
```

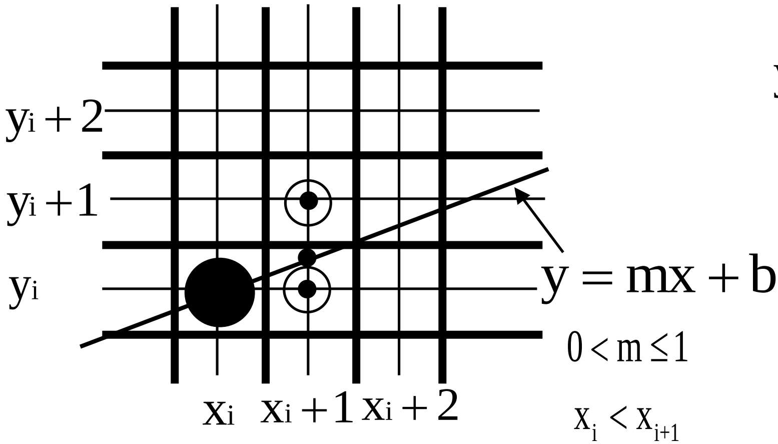
no *'s

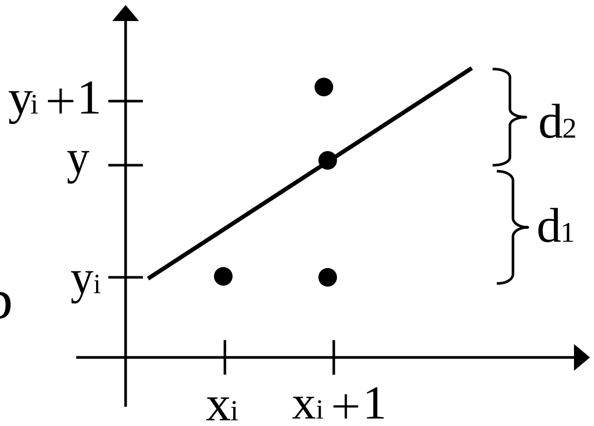
expensive !!

Bresenham's Line Algorithm

- basic idea
  - Find the closest integer coordinates to the actual line path using only integer arithmetic
  - Candidates for the next pixel position







$$y_i = mx_i + b$$

$$\therefore\ d_1 = y - y_i = m\underline{(x_i + 1) + b - y_i}$$

$$d_2 = y_i + 1 - y = y_i + 1 - \underline{m(x_i + 1) - b}$$

$$d = d_1 - d_2 = 2m(x_i + 1) - 2y_i + 2b - 1 \quad (\text{where } m = \Delta y/\Delta x)$$

$$p_i = \Delta x(d_1 - d_2) = 2\Delta y(x_i + 1) - 2\Delta x y_i + \Delta x(2b - 1)$$

$$= 2\Delta y x_i - 2\Delta x y_i + \underline{(2\Delta y + \Delta x(2b - 1))} \qquad \triangleq c$$

$$= 2\Delta y x_i - 2\Delta x y_i + c$$

$$p_i = \underline{2\Delta y x_i - 2\Delta x y_i + c}$$

if $p_i < 0$ then $(x_i + 1, y_i)$

if $p_i \ge 0$ then $(x_i + 1, y_i + 1)$

$$p_i = 2\Delta y x_i - 2\Delta x y_i + c$$

$$p_{i+1} = 2\Delta y x_{i+1} - 2\Delta x y_{i+1} + c$$

$$\therefore p_{i+1} - p_i = 2\Delta y(\underbrace{x_{i+1} - x_i}_{=1}) - 2\Delta x(y_{i+1} - y_i)$$

$$\therefore p_{i+1} = p_i + 2\Delta y - 2\Delta x(\underline{y_{i+1}} - y_i)$$

$$y_{i+1} = \begin{cases} y_i & \text{if } p_i < 0 \\ y_i + 1 & \text{otherwise} \end{cases}$$

$$\boxed{\therefore p_{i+1} = \begin{cases} p_i + 2\Delta y & \text{if } p_i < 0 \\ p_i + 2\Delta y - 2\Delta x & \text{otherwise} \end{cases}}$$

Now,

$$p_1 = 2\Delta y x_1 - 2\Delta x y_1 + \underline{c}$$

$$c = 2\Delta y + \Delta x(2\underline{b} - 1)$$

$$b = y_1 - (\Delta y/\Delta x)x_1$$

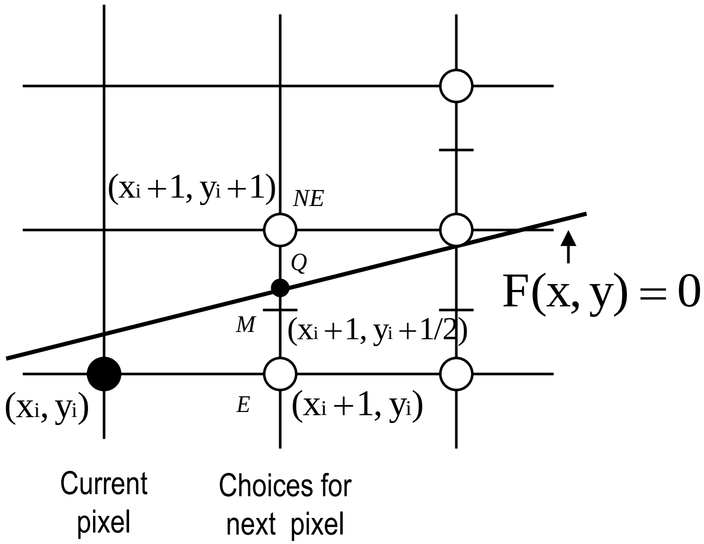
$$= 2\Delta y - \Delta x$$

$$\boxed{\therefore p_1 = 2\Delta y - \Delta x}$$

```
procedure bres_line (x1, y1, x2, y2 : integer);
   var
   Δx, Δy, x,y,p,incrE,incrNE : integer;
   begin
         Δx := x2 – x1;
         Δy := y2 – y1;
         p := 2*Δy - Δx;
         incrE := 2*Δy;
         incrNE := 2*(Δy - Δx);
         x := x1; y := y1;
         display(x,y);
```

```
         while x < x2 do begin
               if p<0 then begin
                     p := p + incrE;
                     x := x + 1;
               end; { then begin }
               else begin
                     p := p + incrNE;
                     y := y + 1;
                     x := x + 1;
               end; { else begin }
               display (x, y);
         end { while  x < x2 }
end; { bres_line}
```

Midpoint Line Algorithm



$$y = \frac{\Delta y}{\Delta x} x + B \Rightarrow \Delta y \cdot x - \Delta x \cdot y + \Delta x \cdot B = 0$$

$$F(x, y) \equiv \Delta y \cdot x - \Delta x \cdot y + \Delta x \cdot B$$

Letting $a = \Delta y,\ b = -\Delta x, \text{ and } c = \Delta x \cdot B,$

$$F(x, y) = ax + by + c$$

$$F(x, y) = 0 \quad \text{if } (x, y) \text{ is on the line}$$

$$Q \text{ is closer to} \begin{cases} E & \text{if } F(x_i + 1, y_i + \frac{1}{2}) < 0 \\ NE & \text{if } F(x_i + 1, y_i + \frac{1}{2}) \geq 0 \end{cases}$$

Van Aken, “An Efficient Ellipse - Drawing Algorithm” IEEE CG & A, 4(9), 24-35, 1984.

$$F(x_i+1, y_i+\frac{1}{2}) = a(x_i+1)+b(y_i+\frac{1}{2})+c$$

$$\text{Let } P_i \equiv 2F(x_i+1, y_i+\frac{1}{2}) = 2a(x_i+1)+b(2y_i+1)+c$$

Since $a = \Delta y$ and $b = -\Delta x$,

$$P_i = 2\Delta y(x_i+1) - \Delta x(2y_i+1)+2c$$

$$P_{i+1} = 2\Delta y(x_{i+1}+1) - \Delta x(2y_{i+1}+1)+2c$$

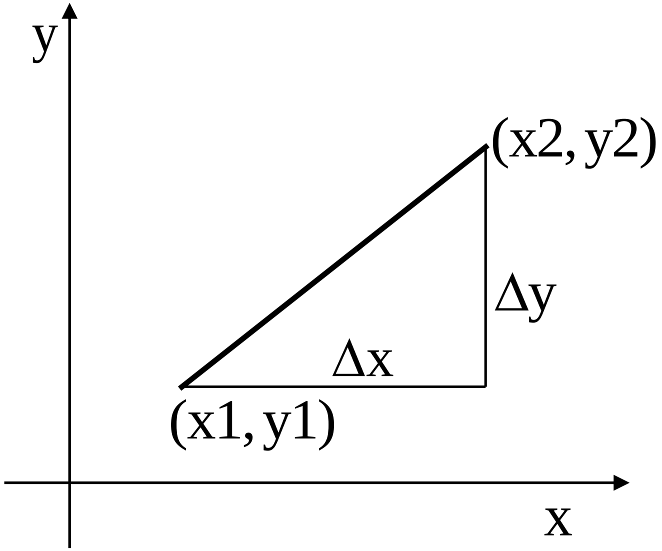
$$\therefore P_{i+1} = P_i + 2\Delta y(x_{i+1}-x_i) - 2\Delta x(y_{i+1}-y_i)$$

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

$$y_{i+1} = \begin{cases} y_i & \text{if } P_i < 0 \\ y_i+1 & \text{otherwise} \end{cases}$$

$$\therefore P_{i+1} = \begin{cases} \underline{P_i + 2\Delta y} & \text{if } P_i < 0 \\ \underline{P_i + 2\Delta y - 2\Delta x} & \text{otherwise} \end{cases}$$

$$P_1 = 2F(x_1+1, y_1+\frac{1}{2}) = 2(\Delta y x_1 + \Delta x y_1 + c) + 2\Delta y - \Delta x = 2\Delta y - \Delta x$$



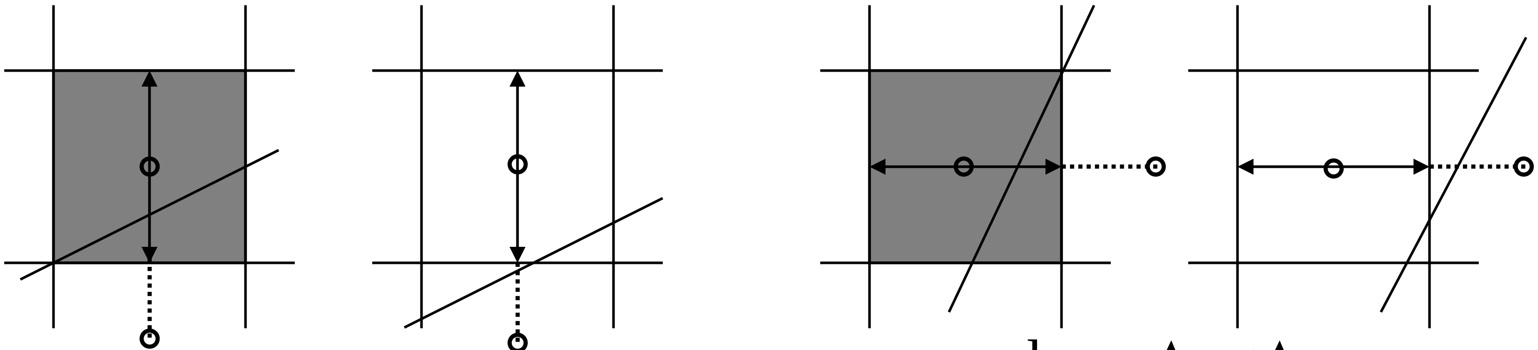
$$\therefore P_{i+1} = \begin{cases} P_i + 2\Delta y & \text{if } P_i < 0 \\ P_i + 2\Delta y - 2\Delta x & \text{otherwise} \end{cases}$$

$$P_1 = 2\Delta y - \Delta x$$

$$(x_{i+1}, y_{i+1}) = \begin{cases} (x_i + 1, y_i) & \text{if } P_i < 0 \\ (x_i + 1, y_i + 1) & \text{otherwise} \end{cases}$$

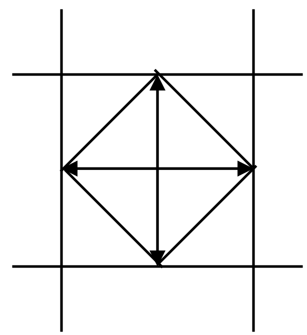
```
procedure mid_point (x1, y1, x2, y2,value : integer);
  var
  Δx, Δy,incrE,incrNE,p,x,y : integer;
  begin
        Δx := x2 – x1;
        Δy := y2 – y1;
        p := 2*Δy - Δx;
        incrE := 2*Δy;
        incrNE := 2*(Δy-Δx);
        x := x1;  y := y1;
        display(x,y);
  while x ≤ x2 do begin
        if p < 0 then begin
              p := p + incrE;
              x := x + 1;
        end; { then begin }
        else begin
              p := p + incrNE;
              y := y + 1;
              x := x + 1;
        end; { else begin }
        display(x,y);
    end; { while x ≤ x2 }
end; { mid_point }
```

Geometric Interpretation

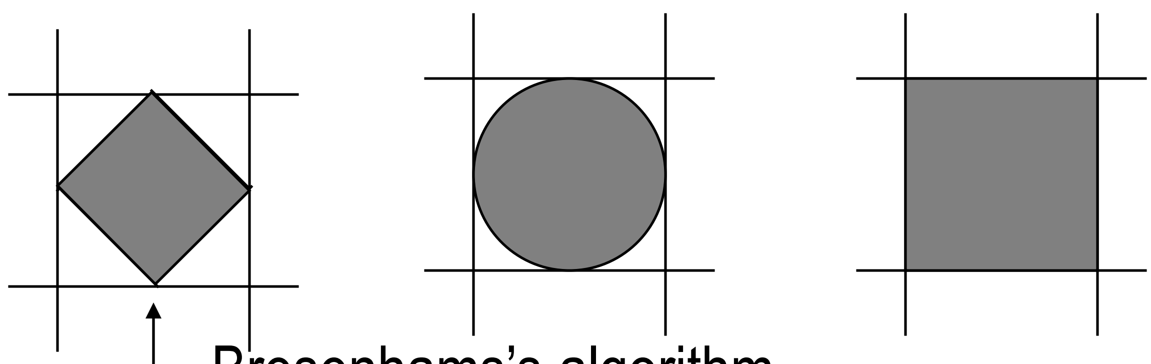


slope: $\Delta x > \Delta y$

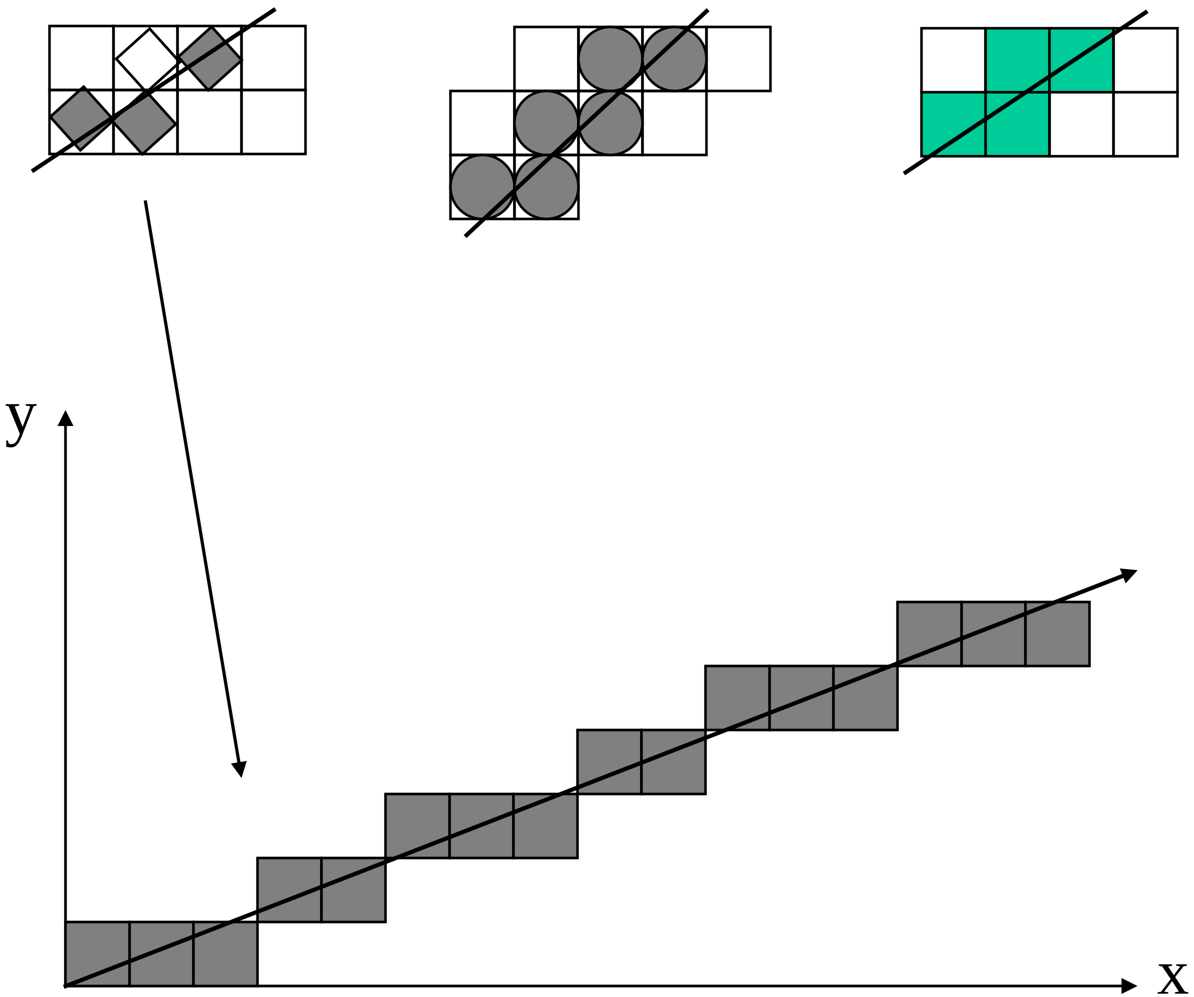
slope: $\Delta x < \Delta y$



any slope



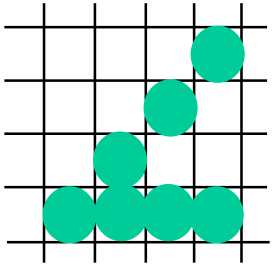
Bresenhams's algorithm


y
x

Aliasing Effects

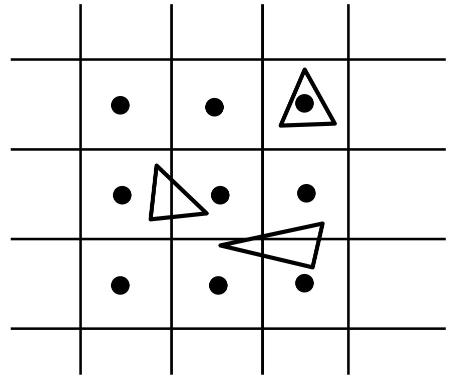
- line drawing

staircases (or jaggies)



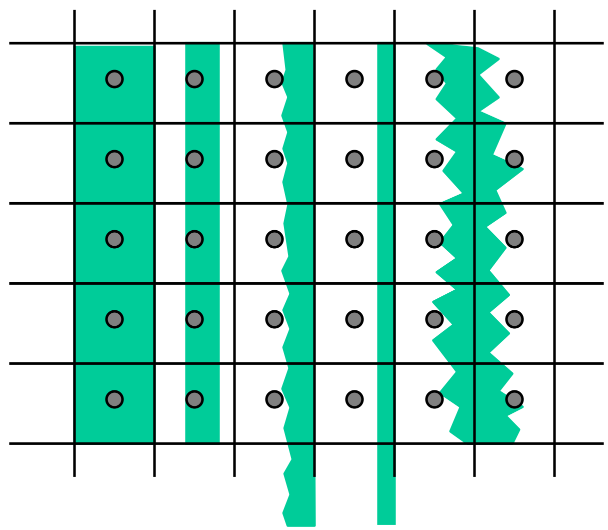
intensity variation

animation



popping-up

texturing



Anti-aliasing Lines

- ❑ Removing the staircase appearance of a line

  - ➢ Why staircases?

    raster effect !!!

    ∴ need some compensation in line-drawing algorithms for this raster effect?

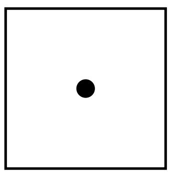
How to anti-alias?

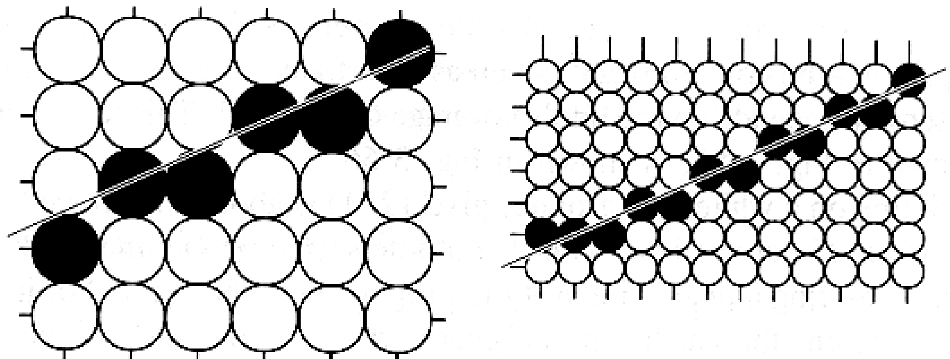
well, …

increasing resolution.

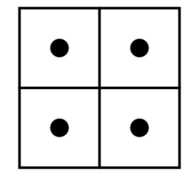
However, ...

Increasing resolution





(a) (b)



memory cost
memory bandwidth
scan conversion time
display device cost

Anti-aliasing Techniques

- super sampling

  (postfiltering)

- area sampling

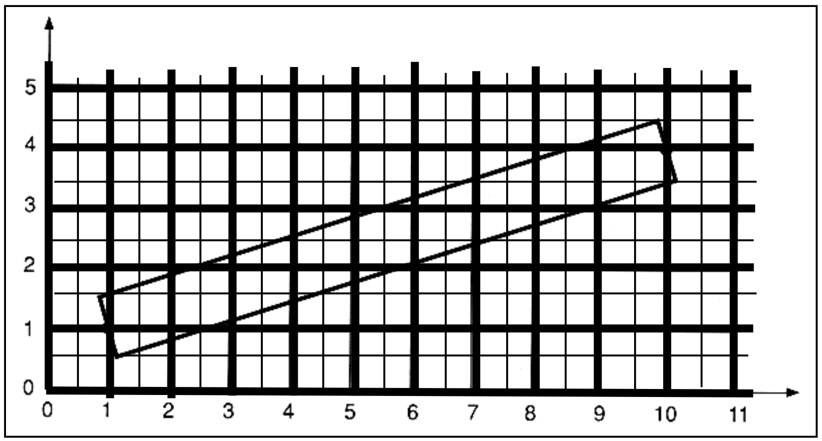
  (prefiltering)

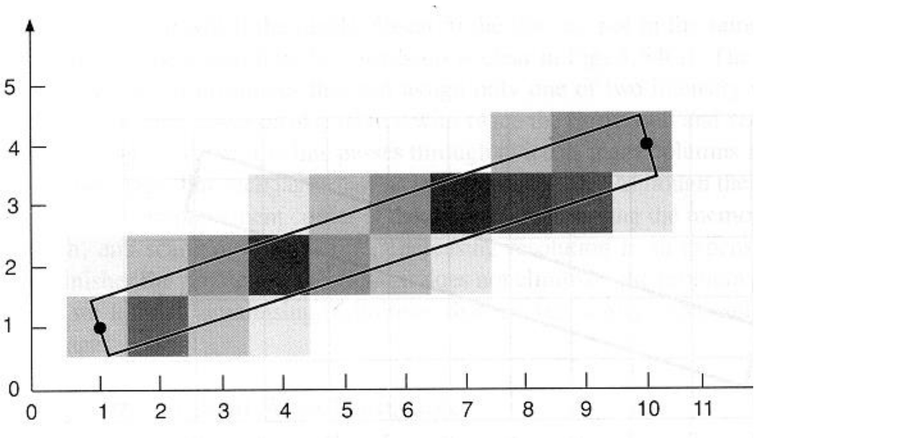
- stochastic sampling

  Cook, ACM Trans. CG, 5(1),

  307-316, 1986.

Area Sampling (Prefiltering)

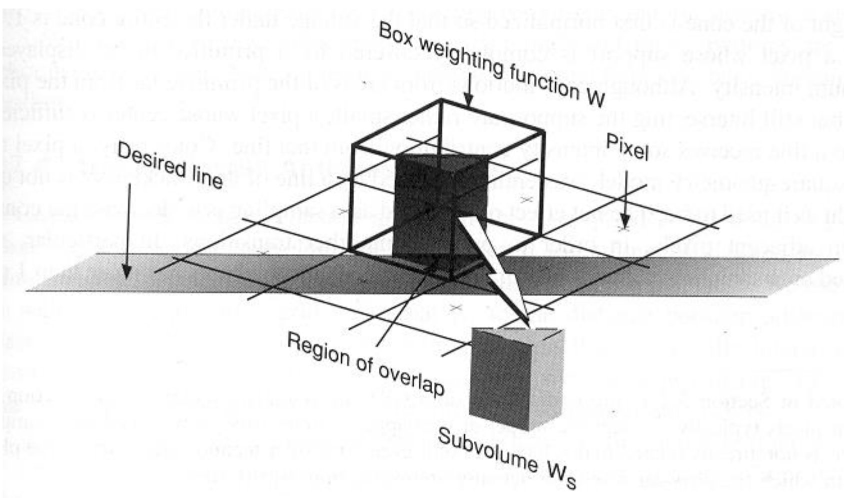


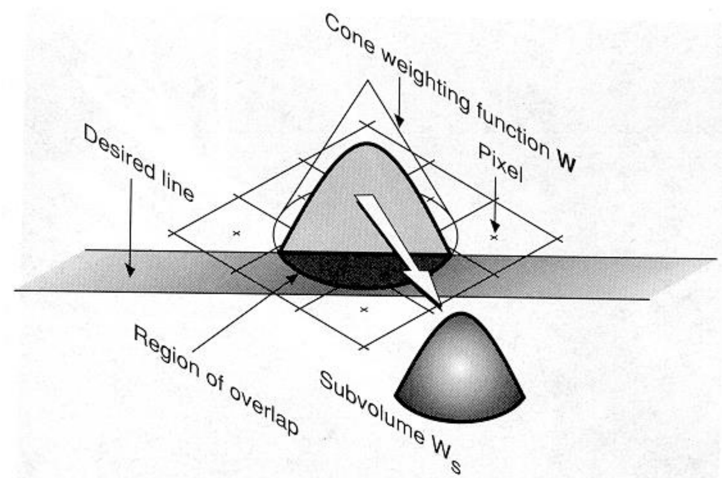


Filters

unweighted area sampling

weighted area sampling

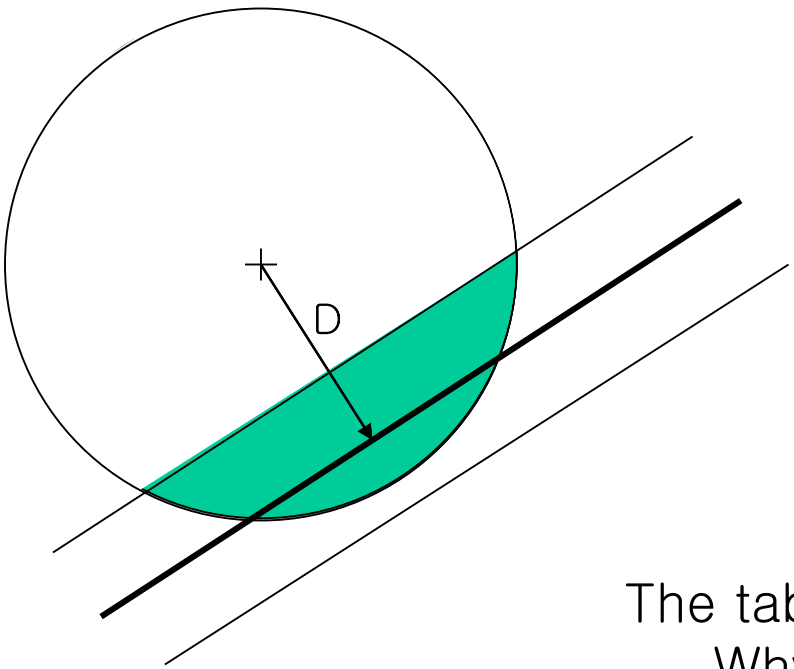




box filter

cone filter

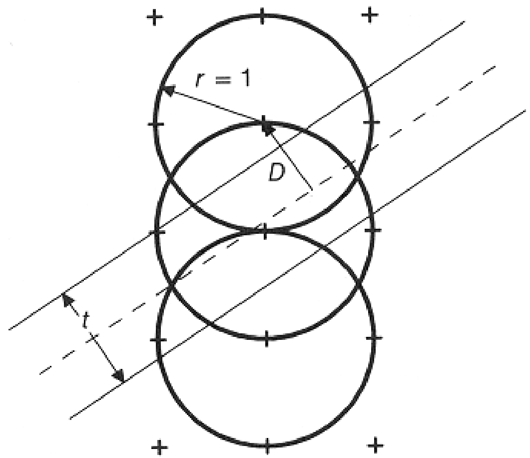
Table look-up for *f*(*D*, *t*)



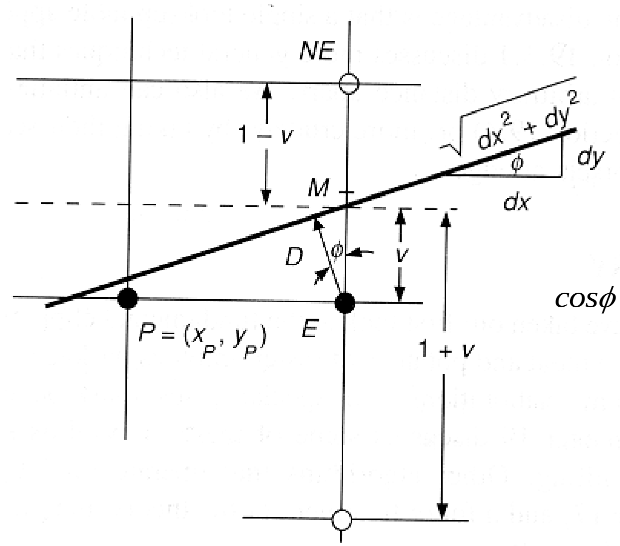
The table is invariant of the slope of line!
Why?

The search key for the table is D!
Why?

Intensity Functions $f(D, t)$
(assumption : t=1)



$v > 0$



$$cos\phi = \frac{dx}{\sqrt{dx^2 + dy^2}}$$

$$D = v\, cos\phi = \frac{vdx}{\sqrt{dx^2 + dy^2}}$$

How to compute D

(assumption : t=1)

$$D = v\cos\phi = \frac{vdx}{\sqrt{dx^2 + dy^2}},$$

$$\text{where} \quad v = y - y_{i+1} = \begin{cases} y - y_i & \text{if} \quad P_i < 0 \\ y - (y_i + 1) & \text{otherwise.} \end{cases}$$

$$P_i = 2F(M) = 2F(x_i + 1, y_i + \frac{1}{2}),$$

where $F(x,y) = ax + by + c$

How to compute *D*, incrementally

Let $dx \equiv \Delta x$ and $dy \equiv \Delta y$.

$$y=\frac{\Delta y}{\Delta x}x+B \Rightarrow \Delta y x-\Delta x y+\Delta x\cdot B=0$$

$$ax+by+c=0$$

$$F(x,y)=ax+by+c$$

$P_i < 0$:

$$v = y - y_i = \frac{a(x_i+1)+c}{-b} - y_i$$

$$\therefore -bv = a(x_i+1)+by_i+c$$

$$vdx = F(x_i+1,y_i)$$

$$\begin{aligned}
2v\Delta x &= 2F(x_i+1,y_i) \\
&= 2a(x_i+1)+2by_i+2c \\
&= 2a(x_i+1)+2b(y_i+\frac{1}{2})+2c-b \\
&= 2F(x_i+1,y_i+\frac{1}{2})+\Delta x \\
&= P_i+\Delta x
\end{aligned}$$

$$\therefore \quad v\Delta x = \frac{1}{2}(P_i+\Delta x)$$

(x_i+1,y_i)

$$\therefore D = \frac{v\Delta x}{\sqrt{\Delta x^2+\Delta y^2}} = \frac{2v\Delta x}{2\sqrt{\Delta x^2+\Delta y^2}} = \frac{(P_i+\Delta x)}{2\sqrt{\Delta x^2+\Delta y^2}},$$

(x_i+1,y_i+1):

$$2(1-v)\Delta x = \underline{2\Delta x - 2v\Delta x}$$

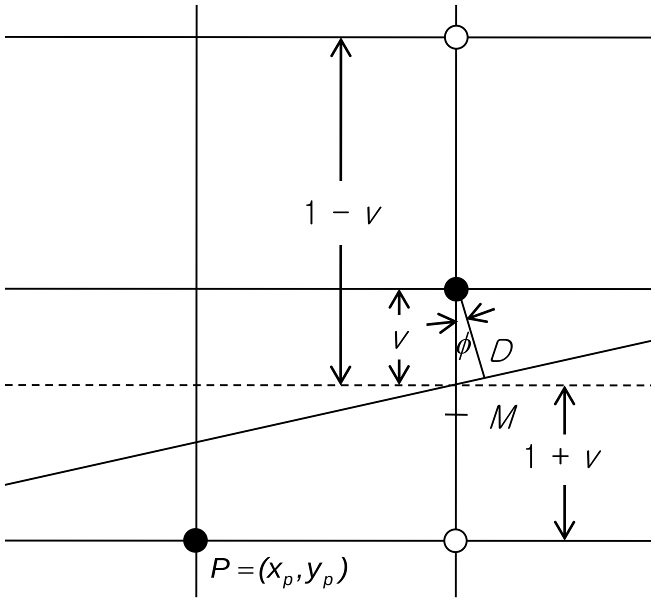
$$\therefore D = \frac{2\Delta x\text{-}2v\Delta x}{2\sqrt{\Delta x^2+\Delta y^2}} = \frac{2\Delta x}{2\sqrt{\Delta x^2+\Delta y^2}} - \frac{2v\Delta x}{2\sqrt{\Delta x^2+\Delta y^2}}$$

(x_i+1,y_i-1):

$$2(1+v)\Delta x = \underline{2\Delta x + 2v\Delta x}$$

$$\therefore D = \frac{2\Delta x}{2\sqrt{\Delta x^2+\Delta y^2}} + \frac{2v\Delta x}{2\sqrt{\Delta x^2+\Delta y^2}}$$

Similarly,



$$P_i \geq 0: \ (x_i + 1, y_i + 1)$$

$$2v\Delta x = P_i - \Delta x$$

$$D = \frac{2v\Delta x}{2\sqrt{\Delta x^2 + \Delta y^2}}$$

$$(x_i + 1, y_i + 2):$$

$$2\Delta x - 2v\Delta x$$

$$D = \frac{2\Delta x}{2\sqrt{\Delta x^2 + \Delta y^2}} - \frac{2v\Delta x}{2\sqrt{\Delta x^2 + \Delta y^2}}$$

$$(x_i + 1, y_i):$$

$$2\Delta x + 2v\Delta x$$

$$D = \frac{2\Delta x}{2\sqrt{\Delta x^2 + \Delta y^2}} + \frac{2v\Delta x}{2\sqrt{\Delta x^2 + \Delta y^2}}$$

```
          Δx := x2 - x1;
          Δy := y2 - y1;
          p := 2 *Δy - Δx;                                   {Initial value p1 as before}
          incrE := 2 * Δy;                                   {Increment used for move to E}
          incrNE := 2 * (Δy - Δx);                           {Increment used for move to NE}
2vΔx  •   two_v_Δx := 0;                                     {Numerator; v = 0 for start pixel}
D     •   invDenom := 1 / (2 * Sqrt(Δx * Δx + Δy * Δy));     {Precomputed inverse denominator}
      •   two_Δx_invDenom := 2 * Δx * invDenom;              {Precomputed constant}
          x := x1;
          y := y1;
      •   IntensifyPixel (x, y, 0);                          {Start pixel}
      •   IntensifyPixel(x, y + 1, two_Δx_invDenom);         {Neighbor}
      •   IntensifyPixel(x, y - 1, two_Δx_invDenom);         {Neighbor}
          while x < x2 do
            begin
              if p < 0 then
                begin                                        {Choose E}
      •           two_v_Δx := p + Δx;
                  p := p + incrE;
                  x := x + 1
                end
              else
                begin                                        {Choose NE}
      •           two_v_Δx := p - Δx;
                  p := p + incrNE;
                  x := x + 1;
                  y := y + 1;
                end;
              {Now set chosen pixel and its neighbors}
      •       IntensifyPixel (x, y, two_v_Δx * invDenom);
      •       IntensifyPixel (x, y + 1, two_Δx_invDenom - two_v_Δx * invDenom);
      •       IntensifyPixel (x, y - 1, two_Δx_invDenom + two_v_Δx * invDenom)
            end {while}
```

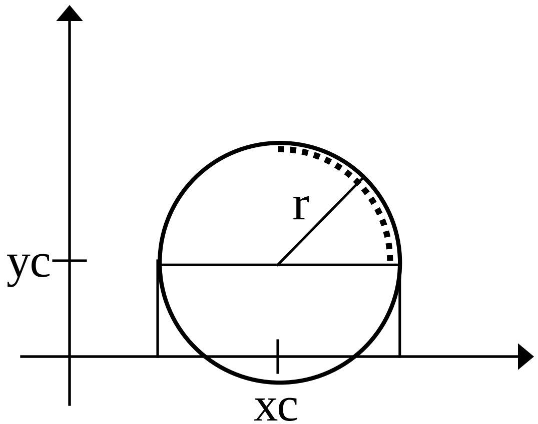
$$\frac{1}{2\sqrt{\Delta x^2 + \Delta y^2}}$$

initially,

$v = 0$

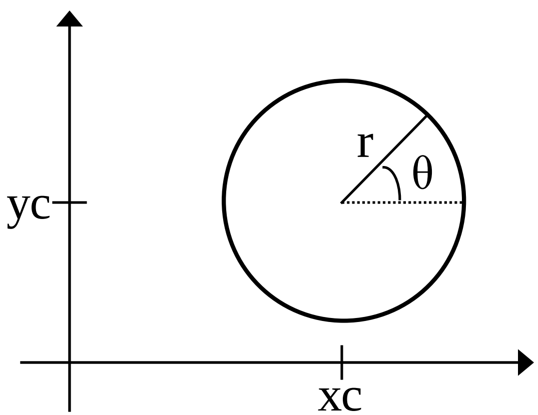
$\therefore 2v\Delta x = 0$

3. Circle Drawing



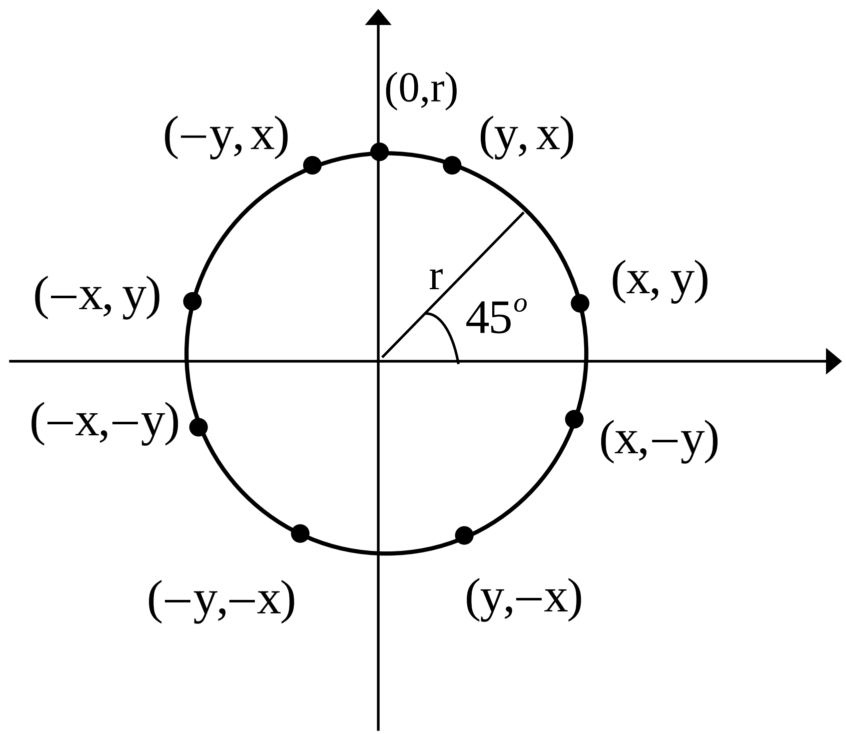
$$(x - xc)^2 + (y - yc)^2 = r^2$$

$$y = yc \pm \sqrt{r^2 - (x - xc)^2}$$



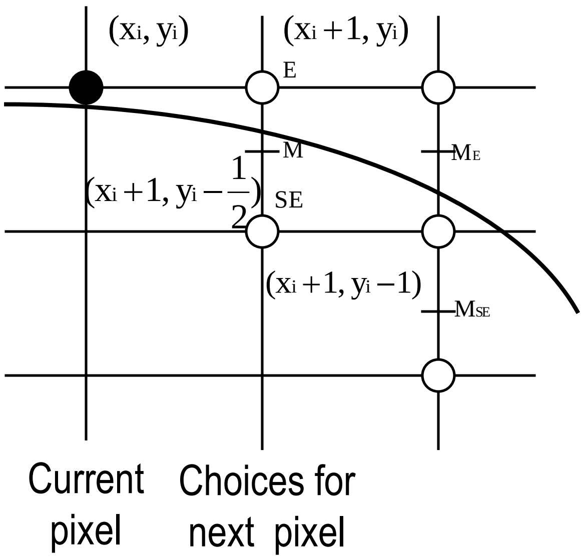
$$x = xc + r\cos\theta$$

$$y = yc + r\sin\theta$$



using symmetry

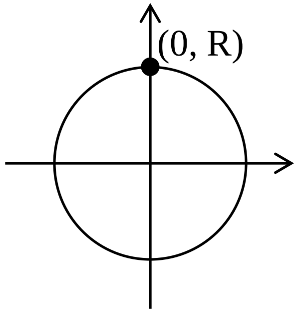
Midpoint Circle Algorithm



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

$$\text{Let } F(x, y) = x^2 + y^2 - R^2$$

$$(x_{i+1}, y_{i+1}) = \begin{cases} (x_i + 1, y_i) & \text{if } F(M) < 0 \\ (x_i + 1, y_i - 1) & \text{otherwise} \end{cases}$$



$$\text{Let } P_i = F(x_i + 1, y_i - \frac{1}{2}) = (x_i + 1)^2 + (y_i - \frac{1}{2})^2 - R^2$$

$$P_{i+1} = F(x_{i+1} + 1, y_{i+1} - \frac{1}{2}) = (x_i + 2)^2 + (y_{i+1} - \frac{1}{2})^2 - R^2, \text{ where}$$

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

$$y_{i+1} = \begin{cases} y_i & \text{if } P_i < 0 \\ y_i - 1 & \text{otherwise} \end{cases}$$

$$\therefore P_{i+1} = \begin{cases} P_i + 2x_i + 3 & \text{if } P_i < 0 \\ P_i + 2(x_i - y_i) + 5 & \text{otherwise} \end{cases}$$

$$P_1 = F(x_1 + 1, y_1 - \frac{1}{2}) = (0+1)^2 + (R - \frac{1}{2})^2 - R^2 = \frac{5}{4} - R \quad \text{since } (x_1, y_1) = (0, R)$$

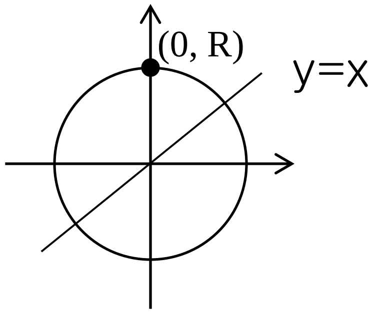
In summary,

$$(x_{i+1}, y_{i+1}) = \begin{cases} (x_i + 1, y_i) & \text{if } P_i < 0 \\ (x_i + 1, y_i - 1) & \text{otherwise} \end{cases}$$

$$\therefore P_{i+1} = \begin{cases} P_i + (2x_i + 3) & \text{if } P_i < 0 \\ P_i + (2x_i - 2y_i + 5) & \text{otherwise} \end{cases}$$

$$P_1 = \frac{5}{4} - R$$

```
procedure MidpointCircle (radius,value : integer);
  var
      x,y : integer; P: real;
  begin
      x := 0;                 { initialization }
      y := radius;
      P := 5/4 - radius;                          *
      CircePoints(x,y,value);
      while y > x do begin
          if P < 0 then           { select E }    **
              P : = P + 2*x + 3;                  ***
              x := x + 1;
          end
          else begin              { select SE }
              P := P + 2*(x - y) + 5;             ****
              x := x + 1;
              y := y - 1;
          end
          CirclePoints(x,y,value)
      end { while }
  end; { MidpointCircle }
```



$$d = P - 1/4$$

$$P = d + 1/4$$

* $\quad d = 1 - \text{radius}$

** $\quad d < -1/4 \Rightarrow d < 0$

why?

*** $\quad d := d + 2*x + 3$

**** $\quad d := d + 2(x-y) + 5$

```
procedure MidpointCircle (radius,value : integer);
{ Assumes center of circle is at origin. Integer arithmetic only }
  var
        x,y,d : integer;
  begin
        x := 0;                 { initialization }
        y := radius;
        d := 1 - radius;
        CirclePoints(x,y,value);
        while y > x do begin
              if d < 0 then             { select E }
                    d := d + 2*x + 3;
                    x := x + 1;
              end
              else begin                { select SE }
                    d := d+2*(x - y) + 5;
                    x := x + 1;
                    y := y - 1;
              end
              CirclePoints(x,y,value)
          end { while }
    end; { MidpointCircle }
```

$$P_{i+1} = P_i + \Delta P_i$$

$$\Delta P_i = P_{i+1} - P_i = \begin{cases} 2x_i + 3 & \text{if } P_i < 0 \qquad (\Delta P_i^E) \\ 2(x_i - y_i) + 5 & \text{otherwise} \qquad (\Delta P_i^{SE}) \end{cases}$$

$$(x_i, y_i) \rightarrow (x_i + 1, y_i)$$

$$\Delta P_{i+1}^E = 2(x_i + 1) + 3 = \Delta P_i^E + 2$$

$$\Delta P_{i+1}^{SE} = 2(x_i + 1 - y_i) + 5 = \Delta P_i^{SE} + 2$$

$$(x_i, y_i) \rightarrow (x_i + 1, y_i - 1)$$

$$\Delta P_{i+1}^E = 2(x_i + 1) + 3 = \Delta P_i^E + 2$$

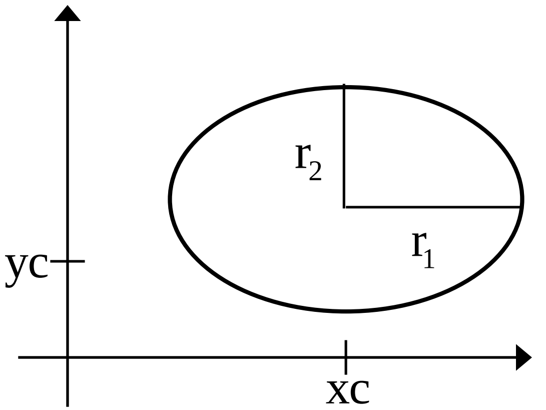
$$\Delta P_{i+1}^{SE} = 2(x_i + 1 - y_i + 1) + 5 = \Delta P_i^{SE} + 4$$

At (x_1, y_1)

$$\Delta P_1 = \begin{cases} 3 & \text{if } P_1 < 0 \qquad (\Delta P_1^E) \\ -2R + 5 & \text{otherwise} \qquad (\Delta P_1^{SE}) \end{cases}$$

```
procedure MidpointCircle (radius,value : integer);
/* This procedure uses second-order partial differences to compute
increments in the decision variable. Assumes center of circle is origin. */
  var
       x,y,d,deltaE,deltaSE : integer;
  begin
       x := 0;                { initialization }
       y := radius;
       d := 1 - radius;
       deltaE := 3;
       deltaSE := -2*radius + 5;
       CirclePoints(x,y,value);
       while y > x do begin
            if d < 0 then          { select E }
                 d := d + deltaE;
                 deltaE := deltaE + 2;
                 deltaSE := deltaSE + 2;
                 x := x + 1;
            end
            else begin             { select SE }
                 d := d + deltaSE;
                 deltaE := deltaE + 2;
                 deltaSE := deltaSE + 4;
                 x := x + 1;
                 y := y - 1
            end
            CirclePoints(x,y,value)
       end { while }
  end; { MidpointCircle }
```

Drawing Ellipse



$$\frac{x^2}{r_1^2} + \frac{y^2}{r_2^2} = 1$$

$$y^2 = r_2^2 \left(1 - \frac{x^2}{r_1^2}\right)$$

4. Curve Drawing

- $y = f(x)$

- parametric equation

$$\boxed{\begin{aligned} y &= g(t) \\ x &= h(t) \end{aligned}}$$

- discrete data set
  - curve fitting
  - piecewise linear