CSE 373: Computer Graphics

SCAN CONVERSION

Primitive graphic objects

Computer-generated images are produced using primitive graphic objects such as:

- Points
- Straight lines
- Circles

Line Drawing

Many computer-generated pictures are composed of straight-line segments.

A line segment is displayed by **turning on** a set of **adjacent pixels**.

In order to draw a line, it is necessary to determine which pixels lie nearest the line and provide the best approximation to the desired line.

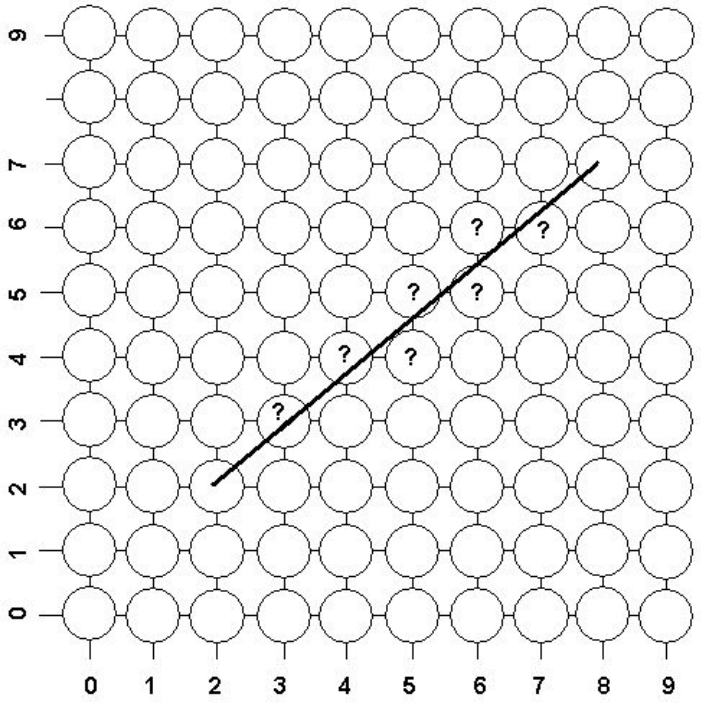
The line drawing routine should be **accurate**, **fast**, and **easy to implement**.

The Problem of Scan Conversion

A line segment is defined by the coordinate positions of the line endpoints.

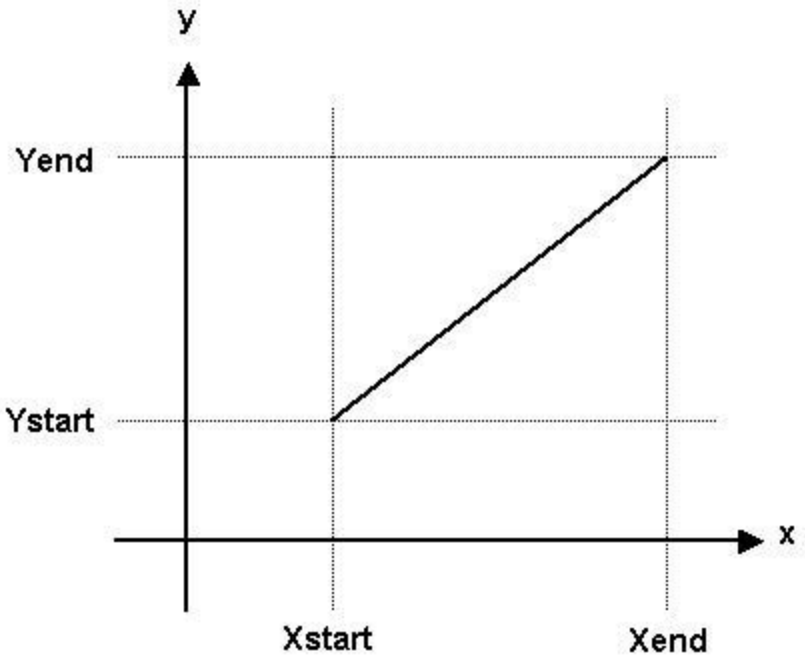
We have a line with the endpoints (2,2) and (8,7) and we want to draw this line on a pixel based display.

How do we choose which pixels to turn on?



Slope-Intercept Line Equation

A line segment in a scene is defined by the coordinate positions of the line endpoints. The starting point is (**Xstart, Ystart**) and the ending point is (**Xend, Yend**).



Slope-Intercept Line Equation

Line Equation is defined as:

$$y = m\,x + b$$

Where **m** is the slope of the line, and defined as the change in **y** values divided by the change in **x** values:

$$m = \frac{Yend - Ystart}{Xend - Xstart}$$

b is the y-intercept. Recall that the y-intercept is the **line's y value** when **x** equals **zero**.

For example, the line defined by equation **y=5x+3** the y-intercept is b=3.

Slope-Intercept Line Equation

The y-intercept can be calculated by the following equation in terms of the coordinate of the starting points.

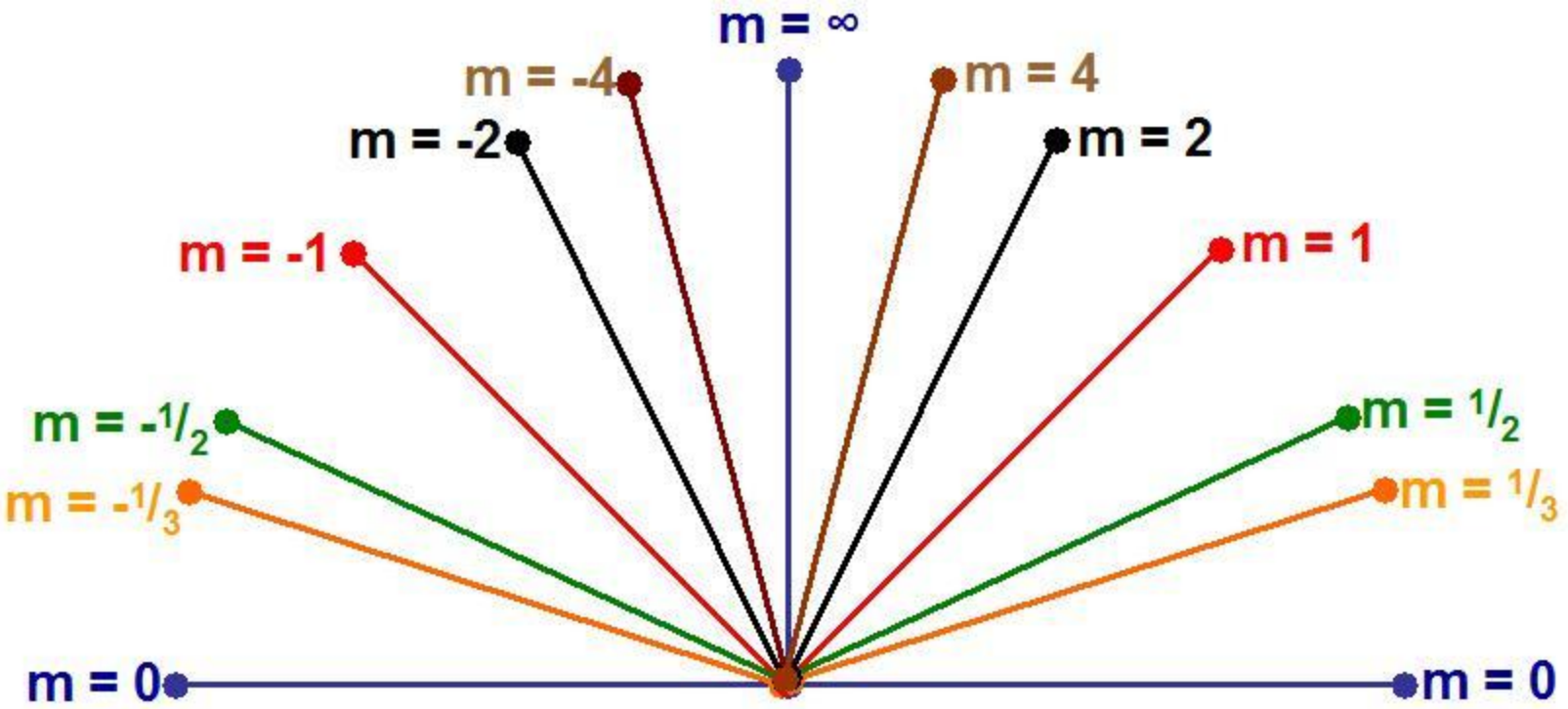
$$b = Ystart - m\ Xstart$$

The y-intercept can also be calculated by the following equation in terms of the coordinate of the ending points.

$$b = Yend - m\ Xend$$

Slope-Intercept Line Equation

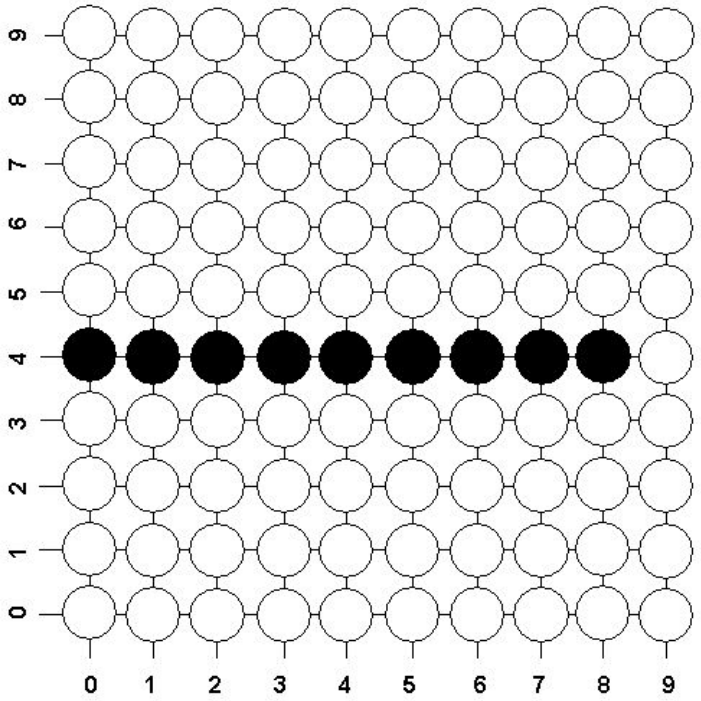
The slope of a line (*m*) is defined by its start and end coordinates. The diagram below shows some examples of lines and their slopes, in all cases **b=0**.



Horizontal Line Drawing Algorithm

The screen coordinates of the points on a horizontal Line are obtained by keeping the value of **y constant** and repeatedly **incrementing** the **x** value by one unit.

horizontal line with starting point (0,4) and ending point (8,4) on a pixel based display



Horizontal Line Drawing Algorithm

The following code can be used to draw a horizontal line from (Xstart, Y) to (Xend, Y),

where **Xstart** ≤ **Xend**

x = **Xstart**

y = **Y**

Next: Set pixel (x, y) with desired color

x = x + 1

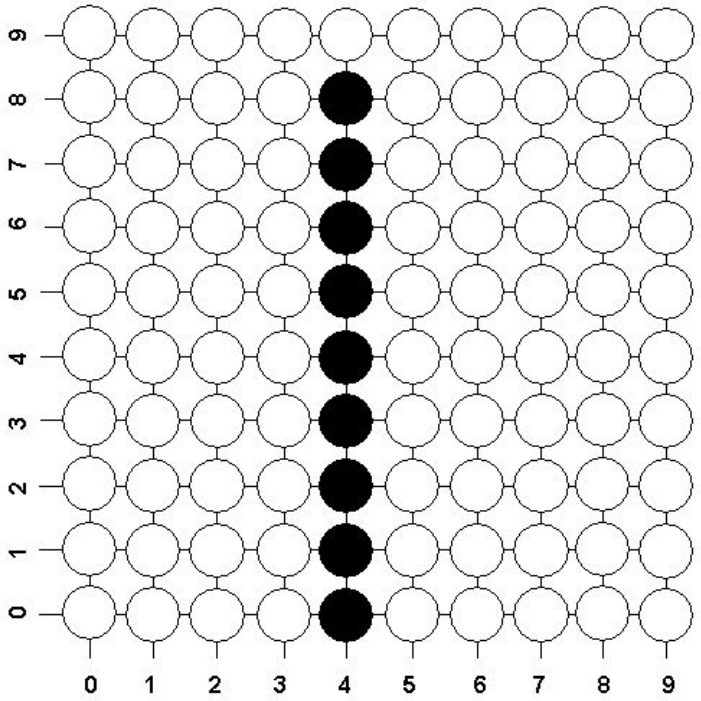
If x **≤** Xend then go to Next

End

Vertical Line Drawing Algorithm

The screen coordinates of the points on a Vertical Line are obtained by keeping the value of **x constant** and repeatedly **incrementing** the **y** value by one unit.

a vertical line with starting point (4,0) and ending point (4,8) on a pixel based display.



Vertical Line Drawing Algorithm

The following code can be used to draw a vertical line from (X, Ystart) to (X, Yend),

where **Ystart** ≤ **Yend**

x = **X**

y = **Ystart**

Next: Set pixel (x, y) with desired color

y = y+ 1

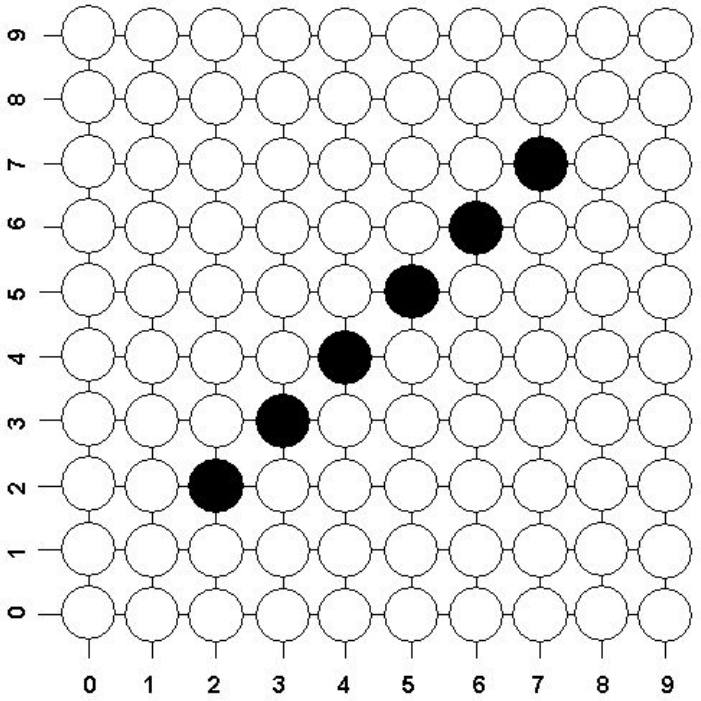
If y **≤** Yend **then** go to **Next**

End

Diagonal Line Drawing Algorithm (**m=1**)

To draw a diagonal line with a slope equals +1 (**m=1**), we need only repeatedly increment by one unit both the **x** and **y** values from the starting to the ending pixels.

a diagonal line (**m=1**) with starting point (2,2) and ending point (7,7) on a pixel based display



Diagonal Line Drawing Algorithm (m=1)

The following code can be used to draw a diagonal line from (Xstart, Ystart) to (Xend, Yend), where **Xstart ≤ Xend** and **Ystart ≤ Yend**

x = **Xstart**

y = **Ystart**

Next: Set pixel (x, y) with desired color

y = y+ 1

x = x + 1

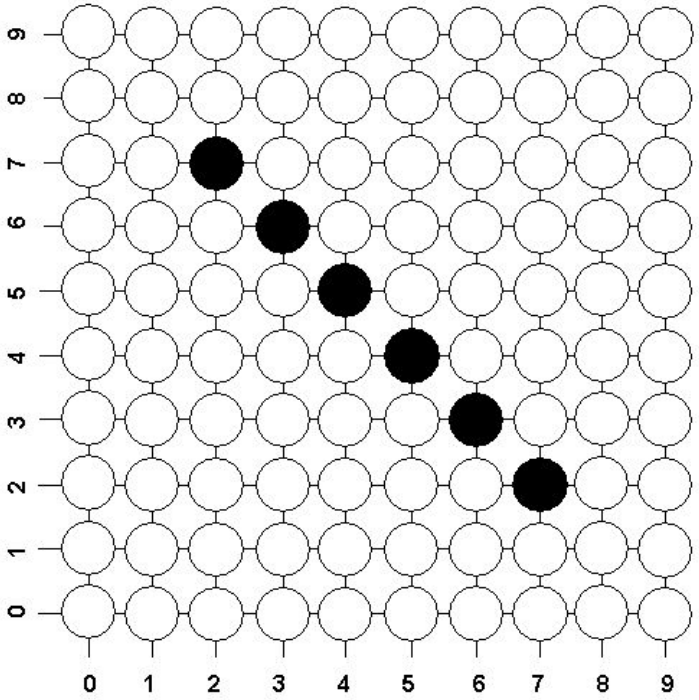
If x ≤ Xend **then** go to **Next**

End

Diagonal Line Drawing Algorithm (m=-1)

To draw a diagonal line with a slope equals –1 (**m=-1**), we need only repeatedly increment by one unit the **x** and decrementing by one unit the **y** values from the starting to the ending pixels.

a diagonal line (**m=-1**) with starting point (2,7) and ending point (7,2) on a pixel based display.



Diagonal Line Drawing Algorithm (m=-1)

The following code can be used to draw a diagonal line from (Xstart, Ystart) to (Xend, Yend), where **Xstart ≤ Xend** and **Ystart ≥ Yend**

x = **Xstart**

y = **Ystart**

Next: Set pixel (x, y) with desired color

y = y − 1

x = x + 1

If x **≤** Xend **then** go to **Next**

End

Arbitrary Lines Drawing Algorithm

Drawing lines with arbitrary slope creates several problems.

- ✔ **Direct Line Drawing Algorithm**
- ✔ **Simple Digital Differential Analyzer (simple DDA) Line Drawing Algorithm**
- ✔ **Bresenham ’s Line Drawing Algorithm**

Direct Line Drawing Algorithm

Perhaps the most natural method of generating a straight line is to use its equation. First we calculate the **slope (m)** and the **y-intercept (b)** using these equations:

$$\mathbf{m} = \frac{\mathbf{Yend - Ystart}}{\mathbf{Xend - Xstart}}$$

$$\mathbf{b = Ystart - m\ Xstart} \quad \text{or} \quad \mathbf{b = Yend - m\ Xend}$$

We then draw the line by incrementing the **x** value one unit from (Xstart, Ystart) to (Xend, Yend) and at each step solve for the corresponding **y** value. For non-integer **y** value, we must first determine the nearest integer coordinate.

Direct Line Drawing Algorithm

The following code can be used to draw a line from (Xstart, Ystart) to (Xend, Yend), where **Xstart ≤ Xend**

x = **Xstart**
y = **Ystart**
m= (Yend-Ystart)/(Xend-Xstart)
b= Ystart – m Xstart

Next: Set pixel (x, Round(y)) with desired color
x= x + 1
y = mx + b
If x **≤** Xend **then** go to **Next**
End

Direct Line Drawing Algorithm

Note that: the **Round function** is used to obtain an integer coordinate value.

There are two fundamental **problems** with this method:

The first problem is the computational time required to draw the line. On many computer systems the operations of multiplication and division can take from 50 to 200 times longer to perform than an addition.

Direct Line Drawing Algorithm

- ✔ a division operation used to calculate the slop (m)
- ✔ a multiplication operation used to calculate b
- ✔ There are (Xend-Xstart+1) multiplication operations used to calculate the y value

The **second problem** concerns lines with a slope whose absolute value is greater than 1 (**m > 1**). These lines will be near the vertical. Using the direct method, the displayed lines will have gaps between plotted points.

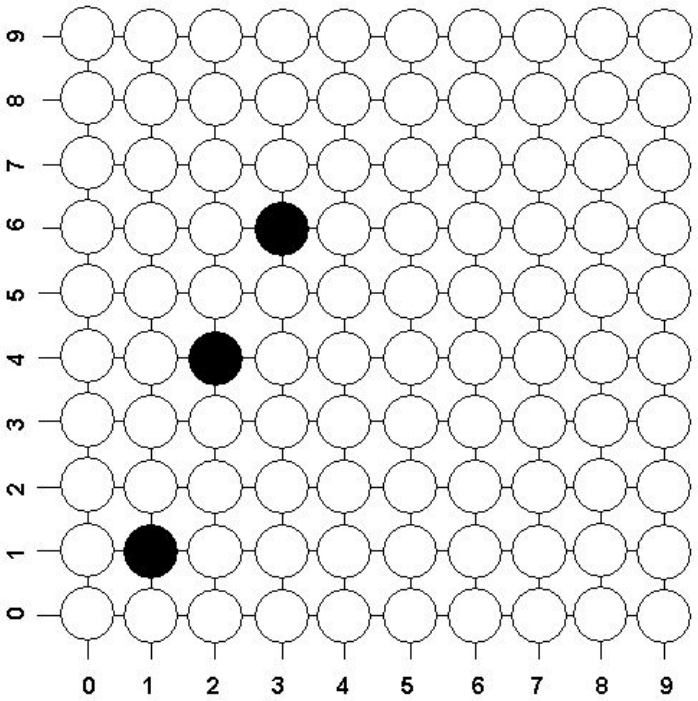
Direct Line Drawing Algorithm

The following example illustrates this problem. We will use the **direct method** to draw a line with starting point (1,1) and ending point (3,6) on a pixel based display.

$$\mathbf{m} = \frac{\mathbf{Yend - Ystart}}{\mathbf{Xend - Xstart}} = \frac{\mathbf{6 - 1}}{\mathbf{3 - 1}} = \frac{\mathbf{5}}{\mathbf{2}}$$

Direct Line Drawing Algorithm

x	y	Round(y)
1	1	1
2	(5/2)*2-(3/2) = 7/2=3.5	4
3	(5/2)*3-(3/2) = 12/2 =6	6



Direct Line Drawing Algorithm

There are several methods of correcting both problems.

- ✔ Performing incremental calculations at each step based on the previous step can eliminate repeated multiplication.
- ✔ The second problem can be resolved by incrementing **y** by one unit and solving the straight line equation for **x** when absolute value of the slope is greater than 1 (**abs(m) > 1**).

Simple Digital Differential Analyzer (simple DDA) Line Drawing Algorithm

To illustrate the idea of DDA algorithm, we still want to draw the line segment with endpoints (Xstart, Ystart) and (Xend, Yend) having slope :

$$\mathbf{m = \frac{Yend - Ystart}{Xend - Xstart}}$$

Any two consecutive points (x1, y1), (x2, y2) lying on this line satisfies the equation:

$$\mathbf{\frac{y2 - y1}{x2 - x1} = m} \qquad \textbf{Equation 1}$$

Simple DDA Line Drawing Algorithm

The algorithm is divided into **two cases** that depend on the absolute value of the slope of the line. **Note that:**

- We should test the line endpoints to ensure that the line is neither horizontal (**Xstart = Xend**) nor vertical (**Ystart =Yend**). If the line is horizontal use the horizontal line drawing algorithm and use the vertical line drawing algorithm when it is vertical.
- The starting and ending points of the line are plotted separately since these values are known from the given data.
- In the two cases below the computed incremental values for y2 and x2 may not be integers. The Round function must be used to obtain an integer coordinate value.

Simple DDA Line Drawing Algorithm

Case 1: For **abs(m) < 1** and **Xstart < Xend**,

we generate the line by incrementing the previous **x** value one unit until **Xend** is reached and then solve for **y**. if **Xstart > Xend,** swap the two endpoints. Thus for these consecutive points:

$$x2 = x1 + 1 \quad \text{or} \quad x2 - x1 = 1$$

Substituting this difference into equation 1 yields:

$$(y2 - y1)/1 = m \quad \text{or} \quad y2 = y1 + m \qquad \textbf{Equation 2}$$

Simple DDA Line Drawing Algorithm

Equation 2 enables us to calculate successive values of y from the previous value by replacing the repeated multiplication with floating point addition.

This method of obtaining the current value by adding a constant to the previous value is an example of **incremental calculation**. Using knowledge of one point to compute the next is a great time saving technique.

The following code can be used to draw a line from (Xstart, Ystart) to (Xend, Yend) using simple DDA algorithm **(case 1)**:

Simple DDA Line Drawing Algorithm

m = (Yend-Ystart) / (Xend-Xstart)
If (abs(m)<1 and Xstart>Xend) **then**
    Swap endpoints Xstart ↔ Xend and Ystart ↔ Yend
end if
Set pixel (Xstart, Ystart) with desired color
If abs(m) < 1 **then**
    y = **Ystart**
    x = **Xstart + 1**
Next: y = y + m
    Set pixel (x, Round(y)) with desired color
    x = x + 1
    If x ≤ Xend-1 **then** go to **Next**
endif
Set pixel (Xend, Yend) with desired color

Simple DDA Line Drawing Algorithm

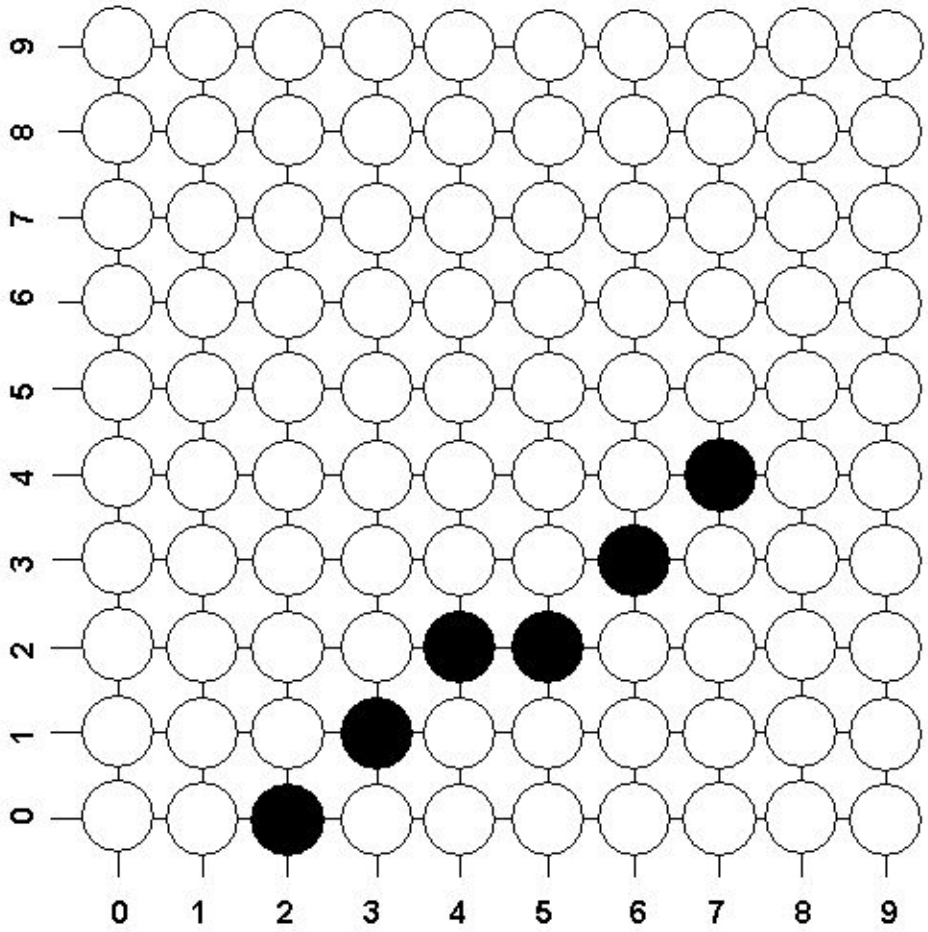
We will use the **simple DDA algorithm** to draw a line with starting point (2,0) and ending point (7,4) on a pixel based display. Firstly, we compute the **slope m**:

m =(Yend–Ystart)/(Xend–Xstart)=(4–0)/(7–2)=4/5 = 0.8

y = **Ystart** = 0 x = **Xstart** + 1 = 2 + 1 =3

x	y	Round(y)
2	0	
3	y = y + m = 0 + 0.8=0.8	1
4	y = y + m = 0.8 + 0.8=1.6	2
5	y = y + m = 1.6 + 0.8=2.4	2
6	y = y + m = 2.4 + 0.8=3.2	3
7	y = y + m = 3.2 + 0.8=4.0	4

Simple DDA Line Drawing Algorithm



Simple DDA Line Drawing Algorithm

Case 2: For **abs(m) > 1** and **Ystart < Yend**,

we generate the line by reversing the above procedure. Namely, we increment the **y** value one unit until **Yend** is reached and then solve for **x.** if **Ystart > Yend,** swap the two endpoints. For these consecutive points:

y2 = y1 + 1 or y2 – y1 = 1

Substituting this difference into equation 1 yields:

1/(x2 – x1) = m or x2 = x1 + 1/m Equation 3

Simple DDA Line Drawing Algorithm

Equation 3 is the desired incremental line equation.

The following code can be used to draw a line from (Xstart, Ystart) to (Xend, Yend) using simple DDA algorithm **(case 2):**

Simple DDA Line Drawing Algorithm

m = (Yend-Ystart) / (Xend-Xstart)
If (abs(m)>1 and Ystart>Yend) **then**
    Swap endpoints Xstart ↔ Xend and Ystart ↔ Yend
end if
Set pixel (Xstart, Ystart) with desired color
If abs(m) > 1 **then**
    m = 1/m , y = **Ystart + 1**
    x = **Xstart**
Next: x= x + m
    Set pixel (Round(x), y) with desired color
    y = y + 1
    If y ≤ Yend-1 **then** go to **Next**
endif
Set pixel (Xend, Yend) with desired color

Simple DDA Line Drawing Algorithm

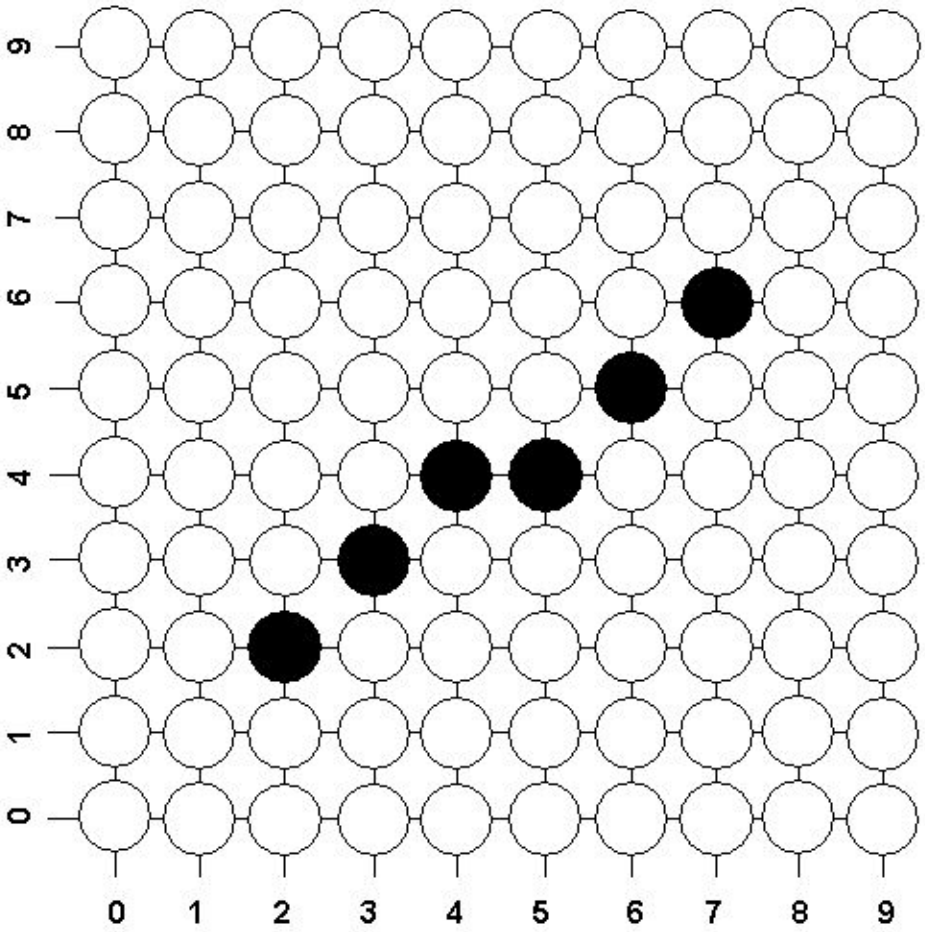
We will use the **simple DDA algorithm** to draw a line with starting point (2,2) and ending point (6,7) on a pixel based display. Firstly, we compute the **slope m**:

m =(Yend–Ystart)/(Xend–Xstart)=(7–2)/(6–2)=5/4

m=1/m = 0.8, y = **Ystart + 1** = 2 + 1 = 3 , x= **Xstart** =2

y	x	Round(x)
2	2	
3	x = x + m = 2 + 0.8=2.8	3
4	x = x + m = 2.8 + 0.8=3.6	4
5	x = x + m = 3.6 + 0.8=4.4	4
6	x = x + m = 4.4 + 0.8=5.2	5
7	x = x + m = 5.2 + 0.8=6.0	6

Simple DDA Line Drawing Algorithm



Bresenham ’s Line Drawing Algorithm

The simple DDA has the disadvantages of using two operations that are expensive in computational time: floating point addition and the round function.

Several good line drawing algorithms avoid these problems by using only integer arithmetic such as Bresenham’s line drawing algorithm.

The Bresenham’s line drawing algorithm is another incremental scan conversion algorithm.

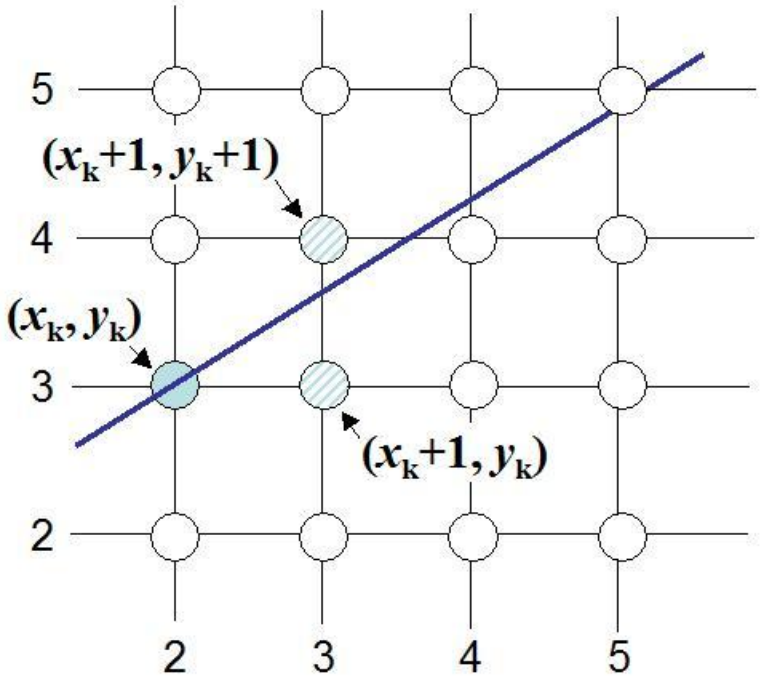
The big advantage of this algorithm is that it uses only integer calculations.

Bresenham 's Line Drawing Algorithm

The main Idea of the Bresenham's line drawing algorithm: Move across the x-axis in unit intervals and at each step choose between two different y coordinates.

For example

from position (2,3) we have to choose between (3,3) and (3,4), we would like the point that is closer to the original line.



Bresenham 's Line Drawing Algorithm

Deriving The Bresenham Line Algorithm

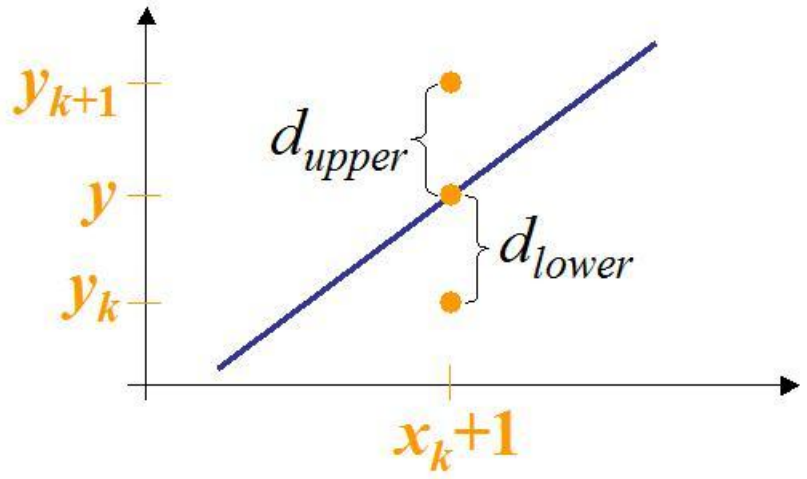
To illustrate Bresenham's approach, we first consider the scan conversion process for lines with positive slope less than 1 **(m<1)**.

Pixel positions along a line path are determined by sampling at unit **x** intervals. Starting from the left endpoint (x_0, y_0) of a given line, we step to each successive column (**x** position) and demonstrate the k^{th} step in this process.

Assuming we have determined that the pixel at (x_k, y_k) is to be displayed, we next need to decide which pixel to plot in column x_k+1. Our choices are the pixels at positions (x_k+1, y_k) and (x_k+1, y_k+1).

Bresenham 's Line Drawing Algorithm

At sampling position x_k+1, the vertical separations from the mathematical line path are labelled as d_{upper} and d_{lower} as shown in the following figure:



The **y** coordinate on the mathematical line at pixel column $\mathbf{x_k+1}$ is calculated as: $\mathbf{y = m(x_k+1) + b}$

Bresenham 's Line Drawing Algorithm

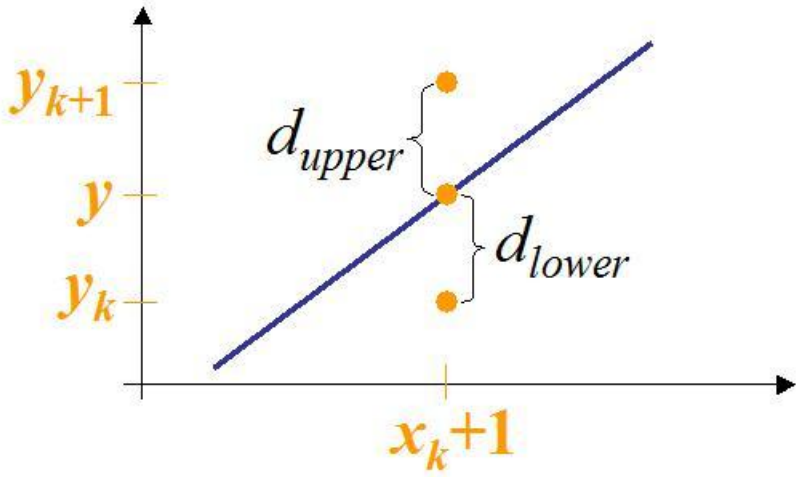
So, d_{upper} and d_{lower} are given as follows:

$$d_{lower} = y - y_k = m(x_k + 1) + b - y_k$$

$$d_{upper} = y_k + 1 - y = y_k + 1 - m(x_k + 1) - b$$

The difference between these two separations is

$$d_{lower} - d_{upper} = 2m(x_k + 1) - 2y_k + 2b - 1$$



Bresenham 's Line Drawing Algorithm

$$d_{lower} - d_{upper} = 2m(x_k + 1) - 2y_k + 2b - 1$$

Let's substitute m with $\Delta y/\Delta x$ where Δx and Δy are the differences between the end-points:

$$d_{lower} - d_{upper} = 2(\Delta y/\Delta x)(x_k + 1) - 2y_k + 2b - 1$$

$$= (1/\Delta x)[\, 2\Delta y\, x_k + 2\Delta y - 2\Delta x\, y_k + 2\Delta x\, b - \Delta x]$$

$$= (1/\Delta x)[\, 2\Delta y\, x_k - 2\Delta x\, y_k + 2\Delta x\, b - \Delta x + 2\Delta y]$$

$$= (1/\Delta x)[\, 2\Delta y\, x_k - 2\Delta x\, y_k + c]$$

Where $c = 2\Delta x\, b - \Delta x + 2\Delta y$

$$\Delta x\,(d_{lower} - d_{upper}) = 2\Delta y\, x_k - 2\Delta x\, y_k + c$$

Bresenham 's Line Drawing Algorithm

$$\Delta x\,(d_{lower} - d_{upper}) = 2\Delta y\, x_k - 2\Delta x\, y_k + c$$

This equation is used as a simple decision about which pixel is closer to the mathematical line.

So, a decision parameter p_k for the k^{th} step along a line is given by:

$$p_k = \Delta x\,(d_{lower} - d_{upper}) = 2\Delta y\, x_k - 2\Delta x\, y_k + c$$

The sign of the decision parameter p_k is the same the sign of $d_{lower} - d_{upper}$,

if the pixel at y_k is closer to the line path than the pixel at y_k+1 (that is $d_{lower} < d_{upper}$), then decision parameter p_k is **negative**. In this case we plot the lower pixel, otherwise we plot the upper pixel.

Bresenham 's Line Drawing Algorithm

Coordinate changes along the line occur in unit steps in either the x or y directions. Therefore, we can obtain the values of successive decision parameters using incremental integer calculations.

At step k+1 the decision parameter is given as:

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

Subtracting p_k from this we get:

$$p_{k+1} - p_k = 2\Delta y(x_{k+1} - x_k) - 2\Delta x(y_{k+1} - y_k)$$

But, x_{k+1} is the same as x_k+1 so that

$$p_{k+1} = p_k + 2\Delta y - 2\Delta x(y_{k+1} - y_k)$$

Where $y_{k+1} - y_k$ is either 0 or 1, depending on the sign of parameter p_k.

Bresenham 's Line Drawing Algorithm

The recursive calculation of decision parameters is performed at each integer **x** position, starting at the left coordinate endpoint of the line.

The first decision parameter $\mathbf{p_0}$ is evaluated at the starting pixel position $\mathbf{(x_0, y_0)}$ and with **m** evaluated as $\mathbf{(\Delta y/\Delta x)}$:

$$\mathbf{P_0 = 2\Delta y - \Delta x}$$

The following steps summarize Bresenham's Line Drawing Algorithm for a line with positive slope less than 1 $\mathbf{(|m|<1)}$

Bresenham 's Line Drawing Algorithm

1. Input the two line end-points, storing the left end-point in (x_0, y_0)
2. Plot the point (x_0, y_0)
3. Calculate the constants Δx, Δy, $2\Delta y$, and $(2\Delta y - 2\Delta x)$ and get the first value for the decision parameter as:

$$P_0 = 2\Delta y - \Delta x$$

4. At each x_k along the line, starting at $k=0$, perform the following test:

   If $p_k < 0$, the next point to plot is (x_k+1, y_k) and

$$p_{k+1} = p_k + 2\Delta y$$

   Otherwise, the next point to plot is (x_k+1, y_k+1) and

$$p_{k+1} = p_k + 2\Delta y - 2\Delta x$$

5. Repeat step 4 (Δx) times

Bresenham 's Line Drawing Algorithm

Bresenham's Line Drawing Example

To illustrate the algorithm, we digitize the line with endpoints**(20,10)** and **(30, 18)**. This line has a slope of 0.8, with

$$\Delta x = 10,\ \Delta y = 8$$

The initial decision parameter has the value $P_0 = 2\Delta y - \Delta x = 6$

And the increments for calculation successive decision parameters are:

$$2\Delta y = 16,$$

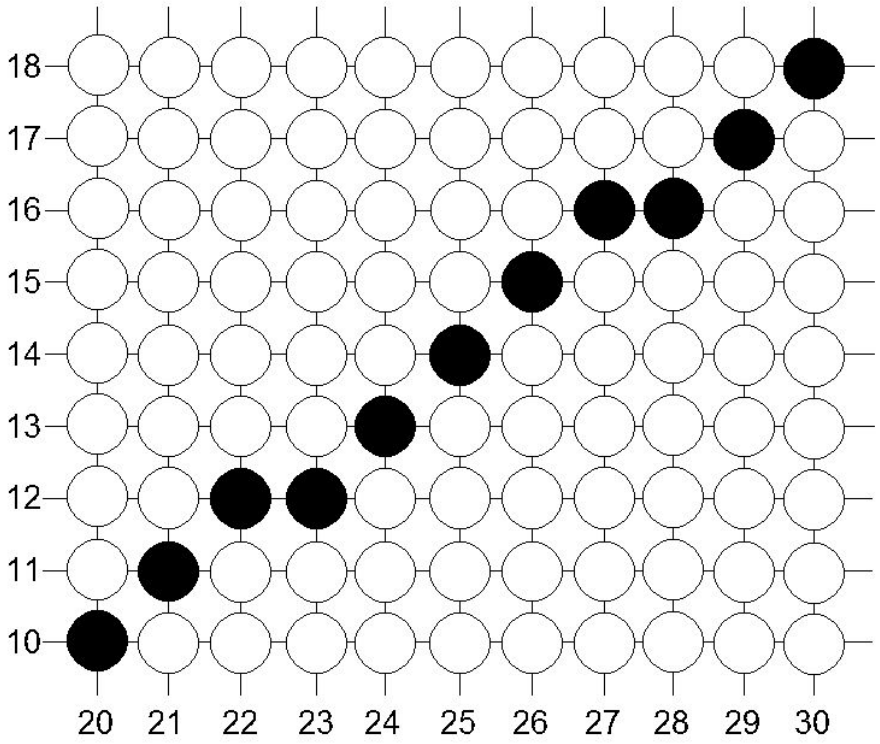
$$2\Delta y - 2\Delta x = -4$$

We plot the initial point **(20, 10)** , and determine successive pixel positions along the line path from the decision parameters as:

Bresenham 's Line Drawing Algorithm

Bresenham's Line Drawing Example

k	p_k	(x_k+1, y_k+1)
0	6	(21, 11)
1	2	(22, 12)
2	-2	(23, 12)
3	14	(24, 13)
4	10	(25, 14)
5	6	(26, 15)
6	2	(27, 16)
7	-2	(28, 16)
8	14	(29, 17)
9	10	(30, 18)



Circle Drawing

Since the circle is a frequently used component in pictures and graphs, a procedure for generating either full circles or circular arc is included in most graphics packages.

Direct Algorithm

The equation for a circle is:

$$x^2 + y^2 = r^2$$

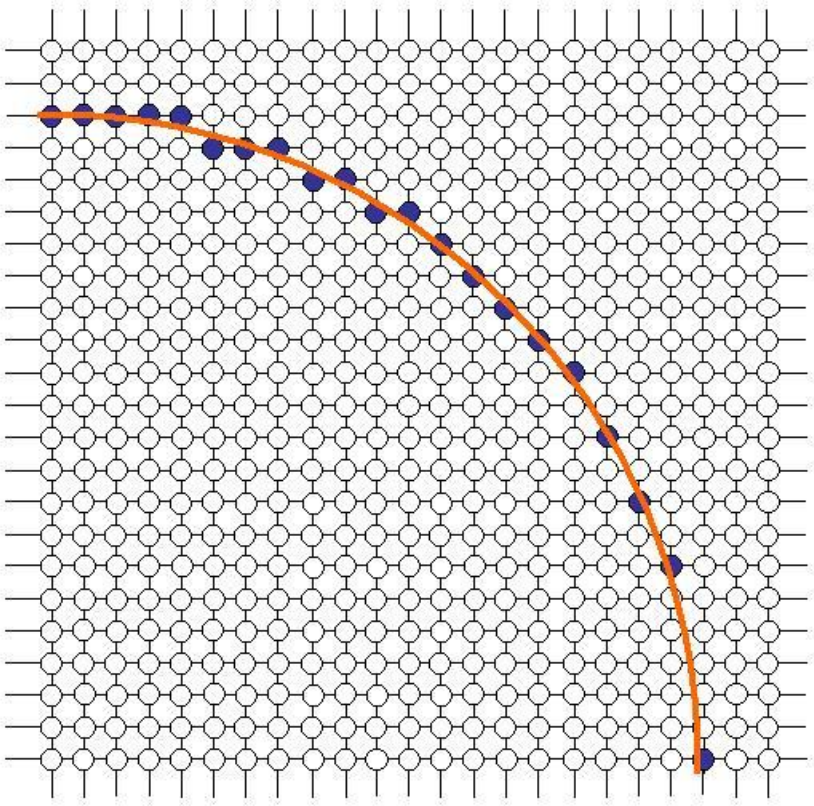
Where **r** is the **radius** of the circle. So, we can write a direct circle drawing algorithm by solving the equation for **y** at unit **x** intervals using:

$$y = \pm(r^2 - x^2)^{1/2}$$

Direct Algorithm

To draw a circle with radius=20

$y(0) = \pm(20^2 - 0^2)^{1/2} \cong 20$
$y(1) = \pm(20^2 - 1^2)^{1/2} \cong 20$
$y(2) = \pm(20^2 - 2^2)^{1/2} \cong 20$
.....
$y(19) = \pm(20^2 - 19^2)^{1/2} \cong 6$
$y(20) = \pm(20^2 - 20^2)^{1/2} = 0$

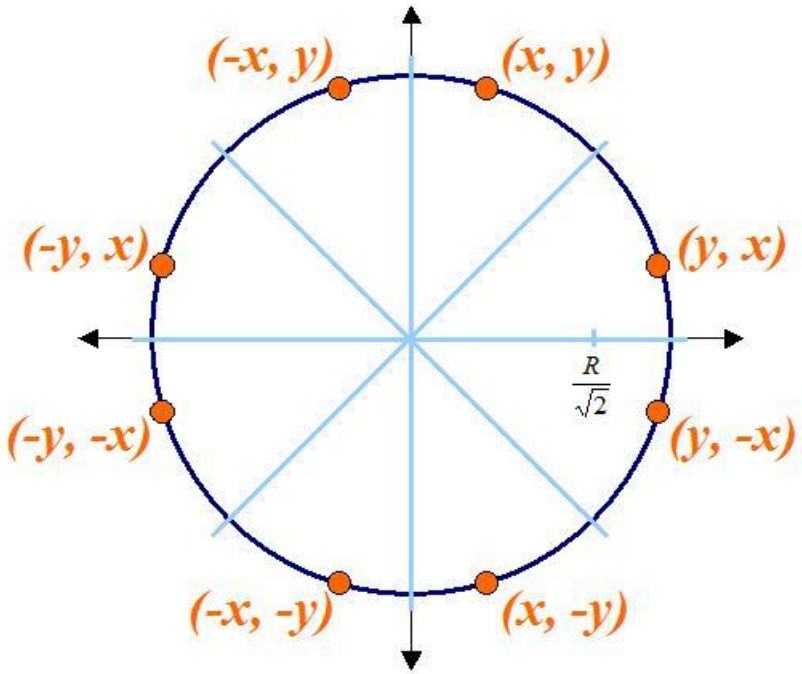


the resulting circle has large gaps where the slope approaches the vertical. And the calculations are not very efficient:

- ✔ The square (multiply) operations
- ✔ The square root operation

Eight-Way Symmetry

The first thing we can notice to make our circle drawing algorithm more efficient is that circles centred at (0, 0) have eight-way symmetry



Mid-Point Circle Algorithm

As in the raster line algorithm, we sample at unit intervals and determine the closest pixel position to the specified circle path at each step.

- For a given radius **r** and screen center position ($\mathbf{x_c}$, $\mathbf{y_c}$), we can first set up our algorithm to calculate pixel positions around a circle path centered at the coordinate (0, 0).
- Then each calculated position (x, y) is moved to its proper screen position by adding $\mathbf{x_c}$ to **x** and $\mathbf{y_c}$ to **y**.

Mid-Point Circle Algorithm

- Along the circle section from x = 0 to x = y in the first quadrant, the slope of the curve varies from 0 to –1. Therefore, we can take unit steps in the positive x direction over this octant and use a decision parameter to determine which of the two possible y positions is closer to the circle path at each step positions in the other seven octants are then obtained by symmetry.
- To apply the midpoint algorithm, we define a circle function:

$$f_{circle}(x, y) = x^2 + y^2 - r^2$$

Mid-Point Circle Algorithm

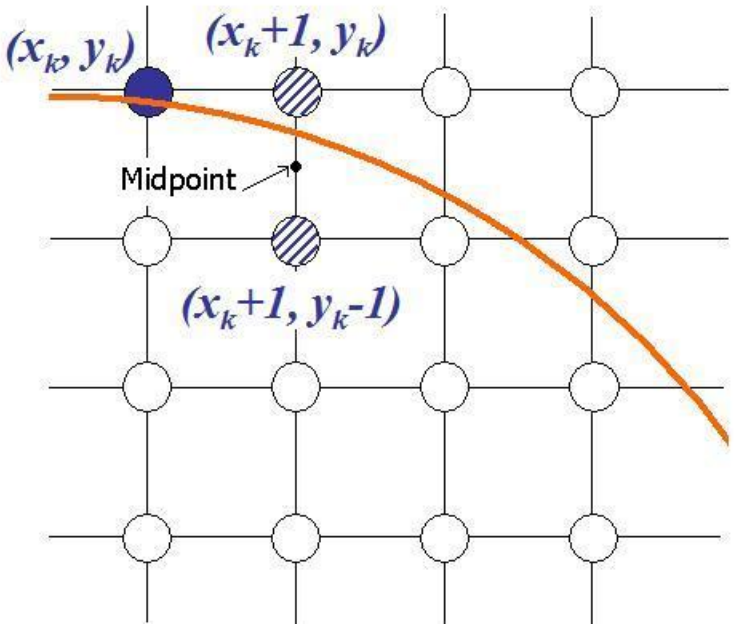
- The relative position of any point (x, y) can be determined by checking the sign of the on the boundary of the circle function:

$$f_{circle}(x, y) \begin{cases} < 0 & \text{If (x, y) is inside the circle boundary} \\ = 0 & \text{If (x, y) is on the circle boundary} \\ > 0 & \text{If (x, y) is outside the circle boundary} \end{cases}$$

The circle function tests are performed for the midpoints positions between pixels near the circle path at each sampling step. Thus, circle function is decision parameter in the midpoint algorithm, and we can set up incremental calculations for this function as we did in the line algorithm.

Mid-Point Circle Algorithm

Assuming we have just plotted point ($\mathbf{x_k}$, $\mathbf{y_k}$), we next need to determine whether the pixel at position ($\mathbf{x_k+1}$, $\mathbf{y_k}$) or the one at position ($\mathbf{x_k+1}$, $\mathbf{y_k-1}$) is closer to the circle path. Our decision parameter is The circle function evaluated at the midpoint between these two pixels:



Mid-Point Circle Algorithm

$$p_k = f_{circle}(x_k + 1, y_k - 1/2)$$
$$= (x_k + 1)^2 + (y_k - 1/2)^2 - r^2$$

- If $p_k < 0$, this midpoint is inside the circle and the pixel at y_k is closer to the circle boundary.
- Otherwise the midpoint is outside or on the circle boundary, and we select the pixel at $y_k - 1$.

✔ Successive decision parameters are obtained using incremental calculations. We obtain a recursive expression for the next decision parameter by evaluating the circle function at sampling position $x_{k+1} + 1 = x_k + 2$

$$p_{k+1} = f_{circle}(x_{k+1} + 1, y_{k+1} - 1/2)$$
$$= ((x_k + 1) + 1)^2 + (y_k - 1/2)^2 - r^2$$

Mid-Point Circle Algorithm

$$p_{k+1} = p_k + 2(x_k+1) + (y_{k+1}^2 - y_k^2) - (y_{k+1} - y_k) + 1$$

Where y_{k+1} is either y_k or y_{k-1} depending on the sign of p_k

Increments for obtaining p_{k+1} are either $2x_{k+1} + 1$ (if p_k is negative) or $2x_{k+1} + 1 - 2y_{k+1}$.

Evaluation of the terms $2x_{k+1}$ and $2y_{k+1}$ can also be done incrementally as:

$$2x_{k+1} = 2x_{k+1} + 2$$

$$2y_{k+1} = 2y_{k+1} - 2$$

At the starting position (0, r), these two terms have the values **0** and **2r**, respectively. Each successive value is obtained by **adding 2** to the previous value of **2x** and **subtracting 2** from the previous value of **2y**.

Mid-Point Circle Algorithm

The initial value of the decision parameter is obtained by evaluating the circle function at the start position

$$(x_0, y_0) = (0, r)$$

$$P_0 = f_{circle}(1, r - 1/2) = 1 + (r - 1/2)^2 - r^2$$

Or

$$P_0 = 5/4 - r$$

If the radius **r** is specified as an integer, we can simply round P_0 to

$$P_0 = 1 - r \quad \text{(for integer r)}$$

Since all increments are integers.

Mid-Point Circle Algorithm

1. Input radius *r* and circle centre **(xc, yc),** and obtain the first point on the circumference of a circle centred on the origin as:

$$(x_0, y_0) = (0, r)$$

2. Calculate the initial value of the decision parameter as:

$$P_0 = 5/4 - r$$

3. At each position x_k Starting with **k=0**, perform the following test.

If pk < 0, the next point along the circle centred on (0, 0) is (x_{k+1}, y_k) and $p_{k+1} = p_k + 2x_{k+1} + 1$

Otherwise the next point along the circle is (x_{k+1}, y_{k-1}) and $p_{k+1} = p_k + 2x_{k+1} + 1 - 2y_{k+1}$

where $2x_{k+1} = 2x_{k+1} + 2$ and $2y_{k+1} = 2y_{k+1} - 2$

Mid-Point Circle Algorithm

4. Determine symmetry points in the other seven octants
5. Move each calculated pixel position (x, y) onto the circular path centred at **(xc, yc)** and plot the coordinate values:

$$X = x + \mathbf{xc} \qquad y = y + \mathbf{yc}$$

6. Repeat steps 3 to 5 until **x >= y**

Mid-Point Circle Algorithm

Mid-Point Circle Algorithm Example

Given a circle **radius=10**, we demonstrate the midpoint circle algorithm by determining positions along the circle octant in the first quadrant from x = 0 to x = y. the initial value of the decision parameter is $\mathbf{P_0 = 1 - r = -9}$

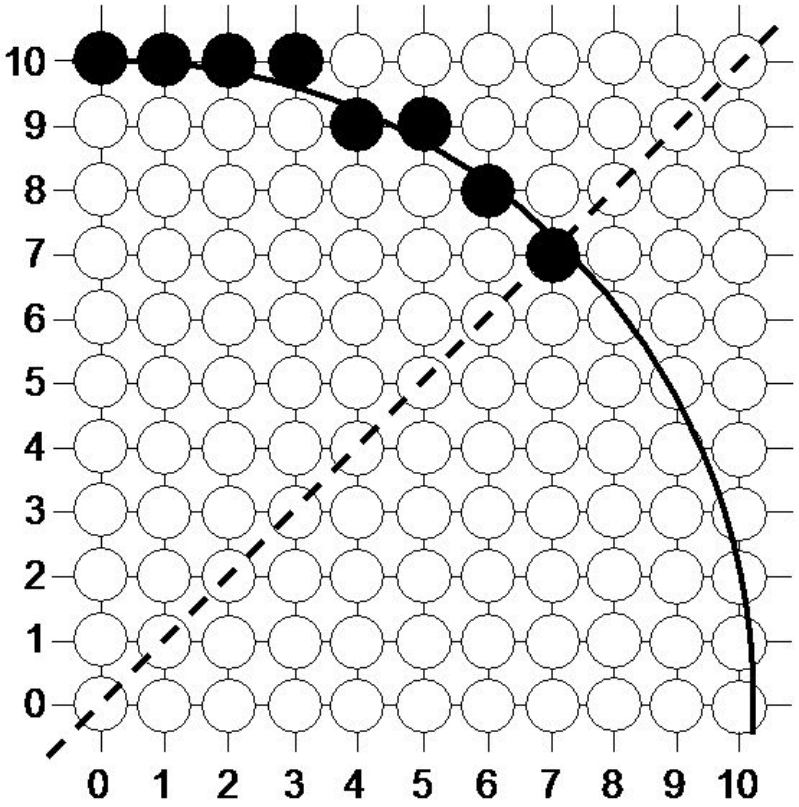
For the circle centred on the origin, the initial point is $\mathbf{(x_0, y_0) = (0, 10)}$, and the initial increment term for calculating the decision parameters are

$$\mathbf{2x_0 = 0} \text{ and } \mathbf{2y_0 = 20}$$

Successive decision parameters values and positions along the circle path are calculated using midpoint algorithm as:

Mid-Point Circle Algorithm

k	P_k	x_{k+1}, y_{k-1}	$2x_{k+1}$	$2y_{k+1}$
0	-9	(1, 10)	2	20
1	-6	(2, 10)	4	20
2	-1	(3, 10)	6	20
3	6	(4, 9)	8	18
4	-3	(5, 9)	10	18
5	8	(6, 8)	12	16
6	5	(7, 7)	14	14



A plot of the generated pixel positions in the first quadrant is shown in the figure