A Simple Circle Drawing Algorithm

The equation for a circle is:

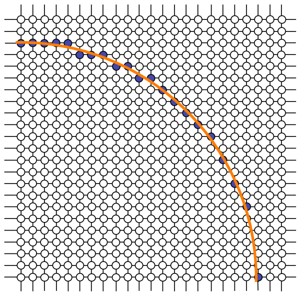
$$x^2 + y^2 = r^2$$

where r is the radius of the circle

So, we can write a simple circle drawing algorithm by solving the equation for y at unit x intervals using:

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

A Simple Circle Drawing Algorithm (cont…)



$$y_0 = \sqrt{20^2 - 0^2} \approx 20$$

$$y_1 = \sqrt{20^2 - 1^2} \approx 20$$

$$y_2 = \sqrt{20^2 - 2^2} \approx 20$$

$$\vdots$$

$$y_{19} = \sqrt{20^2 - 19^2} \approx 6$$

$$y_{20} = \sqrt{20^2 - 20^2} \approx 0$$

A Simple Circle Drawing Algorithm (cont…)

However, unsurprisingly this is not a brilliant solution!

Firstly, the resulting circle has large gaps where the slope approaches the vertical

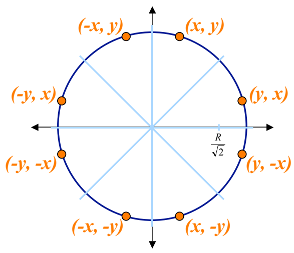
Secondly, the calculations are not very efficient

- The square (multiply) operations
- The square root operation – try really hard to avoid these!

We need a more efficient, more accurate solution

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

Similarly to the case with lines, there is an incremental algorithm for drawing circles – the *mid-point circle algorithm*

In the mid-point circle algorithm we use eight-way symmetry so only ever calculate the points for the top right eighth of a circle, and then use symmetry to get the rest of the points

The mid-point circle algorithm was developed by Jack Bresenham, who we heard about earlier. Bresenham's patent for the algorithm can be viewed here.

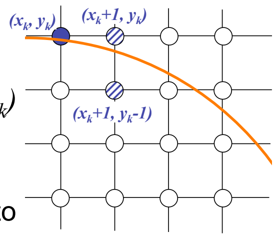
Mid-Point Circle Algorithm (cont…)

Assume that we have just plotted point (x_k, y_k)

The next point is a choice between (x_k+1, y_k) and (x_k+1, y_k-1)

We would like to choose the point that is nearest to the actual circle

So how do we make this choice?



Mid-Point Circle Algorithm (cont…)

Let's re-jig the equation of the circle slightly to give us:

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

The equation evaluates as follows:

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

By evaluating this function at the midpoint between the candidate pixels we can make our decision

Mid-Point Circle Algorithm (cont…)

Assuming we have just plotted the pixel at (x_k, y_k) so we need to choose between (x_k+1, y_k) and (x_k+1, y_k-1)

Our decision variable can be defined as:

$$\begin{aligned} p_k &= f_{circ}(x_k + 1, y_k - \tfrac{1}{2}) \\ &= (x_k + 1)^2 + (y_k - \tfrac{1}{2})^2 - r^2 \end{aligned}$$

If $p_k < 0$ the midpoint is inside the circle and and the pixel at y_k is closer to the circle

Otherwise the midpoint is outside and y_k-1 is closer

Mid-Point Circle Algorithm (cont…)

To ensure things are as efficient as possible we can do all of our calculations incrementally

First consider:

$$\begin{aligned} p_{k+1} &= f_{circ}\left(x_{k+1} + 1, y_{k+1} - \tfrac{1}{2}\right) \\ &= [(x_k + 1) + 1]^2 + \left(y_{k+1} - \tfrac{1}{2}\right) - r^2 \end{aligned}$$

or:

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

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

Mid-Point Circle Algorithm (cont…)

The first decision variable is given as:

$$\begin{aligned} p_0 &= f_{circ}(1, r - \tfrac{1}{2}) \\ &= 1 + (r - \tfrac{1}{2})^2 - r^2 \\ &= \tfrac{5}{4} - r \end{aligned}$$

Then if $p_k < 0$ then the next decision variable is given as:

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

If $p_k > 0$ then the decision variable is:

$$p_{k+1} = p_k + 2x_{k+1} + 1 - 2y_k + 1$$

The Mid-Point Circle Algorithm

MID-POINT CIRCLE ALGORITHM

- Input radius r and circle centre (x_c, y_c), then set the coordinates for the first point on the circumference of a circle centred on the origin as:

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

- Calculate the initial value of the decision parameter as:

$$p_0 = \tfrac{5}{4} - r$$

- Starting with $k = 0$ at each position x_k, perform the following test. If $p_k < 0$, the next point along the circle centred on $(0, 0)$ is (x_k+1, y_k) and:

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

The Mid-Point Circle Algorithm (cont…)

Otherwise the next point along the circle is (x_k+1, y_k-1) and:

$$p_{k+1} = p_k + 2x_{k+1} + 1 - 2y_{k+1}$$

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

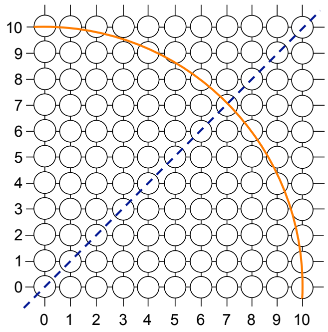
$$x = x + x_c \qquad y = y + y_c$$

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

Mid-Point Circle Algorithm Example

To see the mid-point circle algorithm in action lets use it to draw a circle centred at (0,0) with radius 10

Mid-Point Circle Algorithm Example (cont…)

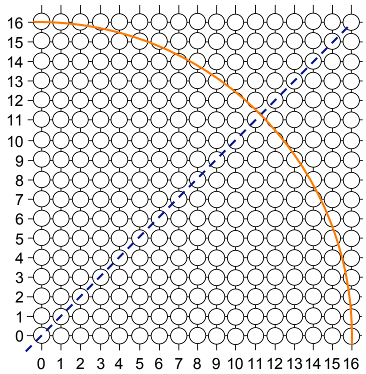


k	p_k	(x_{k+1},y_{k+1})	$2x_{k+1}$	$2y_{k+1}$
0				
1				
2				
3				
4				
5				
6				

Mid-Point Circle Algorithm Exercise

Use the mid-point circle algorithm to draw the circle centred at (0,0) with radius 15

Mid-Point Circle Algorithm Example (cont…)



k	p_k	(x_{k+1},y_{k+1})	$2x_{k+1}$	$2y_{k+1}$
0				
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				

Mid-Point Circle Algorithm Summary

The key insights in the mid-point circle algorithm are:

- Eight-way symmetry can hugely reduce the work in drawing a circle
- Moving in unit steps along the x axis at each point along the circle's edge we need to choose between two possible y coordinates