MID POINT CIRCLE DRAWING ALGORITHM

BY:

Jagjit Bhatia,

Dept. of Computer Science & IT.

Midpoint Circle Algorithm:

A circle is defined as a set of points that are all at a given distance *r* from a center positioned at. (x_c, y_c).

This is represented mathematically by the equation

$$(x - x_c)^2 + (y - y_c)^2 = r^2 \quad \text{---------- (1)}$$

Using equation (1) we can calculate the value of y for each given value of x as

$$y = y_c \pm \sqrt{r^2 - (x_c - x)^2} \quad \text{---------- (2)}$$

Thus one could calculate different pairs by giving step increments to x and calculating the corresponding value of y. But this approach involves considerable computation at each step and also the resulting circle has its pixels sparsely plotted for areas with higher values of the slope of the curve.

Midpoint Circle Algorithm uses an alternative approach, wherein the pixel positions along the circle are determined on the basis of incremental calculations of a decision parameter.

Let

$$f(x,y) = (x - x_c)^2 + (y - y_c)^2 - r^2 \quad \text{---------- (3)}$$

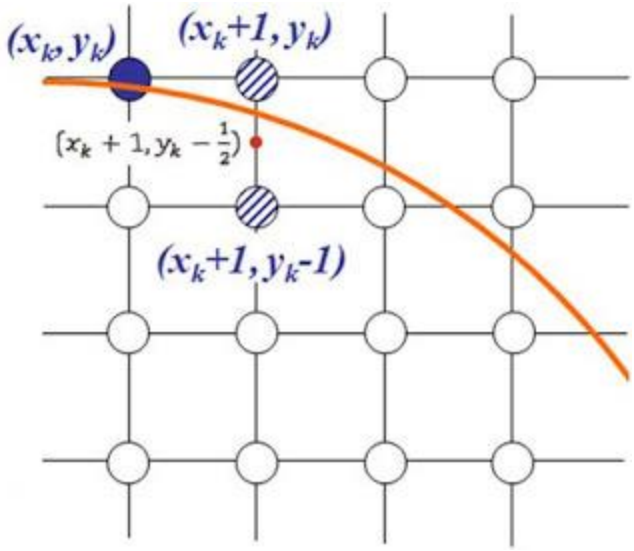
Thus f(x,y)=0 represents the equation of a circle.

Further, we know from coordinate geometry, that for any point , the following holds:

1. $f(x,y) = 0 \quad \Rightarrow$ *The point lies on the circle.*
2. $f(x,y) < 0 \quad \Rightarrow$ *The point lies within the circle.*
3. $f(x,y) > 0 \quad \Rightarrow$ *The point lies outside the circle.*

In Midpoint Circle Algorithm, the decision parameter at the k^{th} step is the circle function evaluated using the coordinates of the midpoint of the two pixel centres which are the next possible pixel position to be plotted.

Let us assume that we are giving unit increments to x in the plotting process and determining the y position using this algorithm. Assuming we have just plotted the k^{th} pixel at (X_k, Y_k), we next need to determine whether the pixel at the position (X_{k+1}, Y_k) or the one at (X_{k+1}, Y_{k-1}) is closer to the circle.



Our decision parameter p_k at the k^{th} step is the circle function evaluated at the midpoint of these two pixels.

The coordinates of the **midpoint** of these two pixels are (X_k+1, Y_k-1/2).

Thus p_k

$$p_k = f\left(x_k + 1, y_k - \frac{1}{2}\right) = (x_k + 1)^2 + (y_k - \frac{1}{2})^2 - r^2 \quad \text{---------- (4)}$$

x

Successive decision parameters are obtained using incremental calculations, thus avoiding a lot of computation at each step. We obtain a recursive expression for the next decision parameter i.e. at the $k+1^{th}$ step, in the following manner.

Using Equ. (4), at the k$+1^{th}$ step, we have:

$$p_k = f\left(x_k + 1, y_k - \frac{1}{2}\right) = (x_k + 1)^2 + (y_k - \frac{1}{2})^2 - r^2$$

$$p_{k+1} = f\left(x_{k+1}+1, y_{k+1}-\frac{1}{2}\right) = (x_{k+1}+1)^2 + (y_{k+1}-\frac{1}{2})^2 - r^2$$

$$\text{Or, } p_{k+1} = (x_k+1+1)^2 + (y_{k+1}-\frac{1}{2})^2 - r^2$$

$$\text{Or, } p_{k+1} = (x_k+2)^2 + (y_{k+1}-\frac{1}{2})^2 - r^2 \text{ ----------(5)}$$

(5)-(4) gives

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

$$\text{Or, } p_{k+1} = p_k + (2x_k+3).1 + (y_{k+1}+y_k-1)(y_{k+1}-y_k) \text{ -------(6)}$$

Now if $P_k <= 0$, then the midpoint of the two possible pixels lies within the circle, thus north pixel is nearer to the theoretical circle. Hence, $Y_{k+1} = Y_k$. Substituting this value of in Equ. (6), we have

$$p_{k+1} = p_k + (2x_k+3) + (y_k+y_k-1)(y_k-y_k)$$

$$\text{Or, } p_{k+1} = p_k + (2x_k+3)$$

If $p_k > 0$ then the midpoint of the two possible pixels lies outside the circle, thus south pixel is nearer to the theoretical circle. Hence, $Y_{k+1} = Y_k - 1$. Substituting this value of in Equ. (6), we have

$$p_{k+1} = p_k + (2x_k+3) + (y_k-1+y_k-1)(y_k-y_k-1)$$

$$\text{Or, } p_{k+1} = p_k + 2(x_k-y_k) + 5$$

For the boundary condition, we have x=0, y=r. Substituting these values in (4), we have

$$p_0 = (0+1)^2 + (r-\frac{1}{2})^2 - r^2 = 1 + r^2 + \frac{1}{4} - r - r^2 = \frac{5}{4} - r$$

For integer values of pixel coordinates, we can approximate $P_0 = 1-r$,

Thus we have:

If $p_k \leq 0$: $y_{k+1} = y_k$ and $p_{k+1} = p_k + (2x_k + 3)$

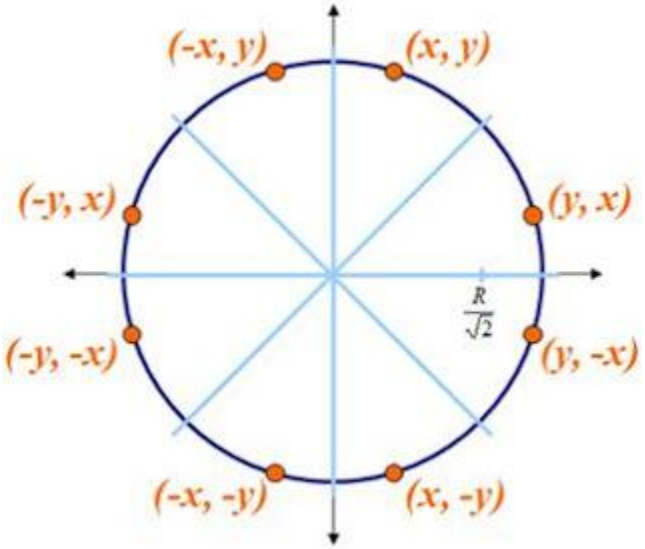
If $p_k > 0$: $y_{k+1} = y_k - 1$ and $p_{k+1} = p_k + 2(x_k - y_k) + 5$

Also, $p_0 = 1 - r$

Drawing the circle:

We can reduce our calculation drastically (8^{th} fraction) by making use of the fact that a circle has 8 way symmetry. Thus after calculating a pixel position (x,y) to be plotted, we get 7 other points on the circle corresponding to it. These are:

(x,y); (x,-y); (-x,y); (-x,-y); (y,x); (y,-x); (-y,x); (-y,-x)



Bresenham's Circle Algorithm

Introduction

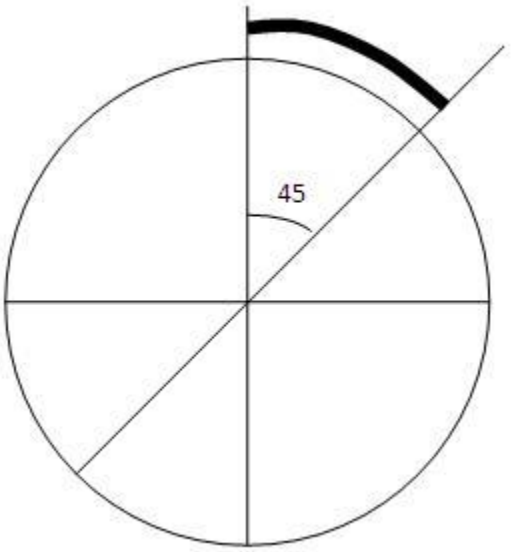
Jack E. Bresenham invented this algorithm in 1962. The objective was to optimize the graphic algorithms for basic objects, in a time when computers were not as powerful as they are today.

This algorithm is called incremental, because the position of the next pixel is calculated on the basis of the last plotted one, instead of just calculating the pixels from a global formula. Such logic is faster for computers to work on and allows plotting circles without trigonometry. The algorithm use only integers, and that's where the strength is: floating point calculations slow down the processors.

All in all, incremental algorithms are 30% faster than the classical ones, based on floating points and trigonometry.

Concept

Circles have the property of being highly symmetrical, which is handy when it comes to drawing them on a display screen.



We know that there are 360 degrees in a circle. First we see that a circle is symmetrical about the x axis, so only the first 180 degrees need to be calculated. Next, we see that it's also symmetrical about the y axis, so now we only need to calculate the first 90 degrees. Finally, we see that the circle is also symmetrical about the 45 degree diagonal axis, so we only need to calculate the first 45 degrees.

Bresenham's circle algorithm calculates the locations of the pixels in the first 45 degrees. It assumes that the circle is centered on the origin shifting the original center coordinates (centerx,centery). So for every pixel (x,y) it calculates, we draw a pixel in each of the 8 octants of the circle :

putpixel(centerx + x, center y + y)

putpixel(centerx + x, center y - y)

putpixel(centerx - x, center y + y)

putpixel(centerx - x, center y - y)

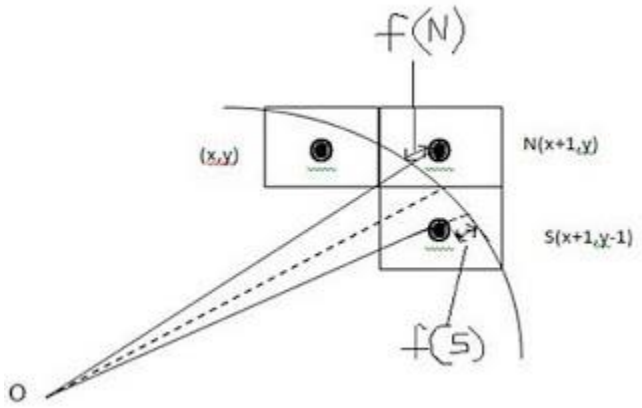
putpixel(centerx + y, center y + x)

putpixel(centerx + y, center y - x)

putpixel(centerx - y, center y + x)

putpixel(centerx - y, center y - x)

Now, consider a very small continuous arc of the circle interpolated below, passing by the discrete pixels as shown.



As can be easily intercepted, the continuous arc of the circle can not be plotted on a raster display device, but has to be approximated by choosing the pixels to be highlighted. At any point (x,y), we have two choices – to choose the pixel on east of it, i.e. N(x+1,y) or the south-east pixel S(x+1,y-1). To choose the pixel, we determine the errors involved with both N & S which are f(N) and f(S) respectively and whichever gives the lesser error, we choose that pixel.

Let $d_i = f(N) + f(S)$, where d can be called as "decision parameter", so that

if $d_i <= 0$,

then, N(x+1,y) is to be chosen as next pixel i.e. $x_{i+1} = x_i+1$ and $y_{i+1} = y_i$,

and if $d_i>0$,

then, S(x+1,y-1) is to be chosen as next pixel i.e. $x_{i+1} = x_i+1$ and $y_{i+1} = y_i-1$.

Derivation

We know that for a circle,

$x^2 + y^2 = r^2$, where r represents the radius of the circle, an input to the algorithm.

Errors can be represented as

$$f(N) = (x_i + 1)^2 + y_i^2 - r^2, \qquad -(1)$$

$$f(S) = (x_i + 1)^2 + (y_i - 1)^2 - r^2 \qquad -(2)$$

As $d_i = f(N) + f(S)$,

$$\_d_i = 2(x_i+1)^2 + y_i^2 + (y_i-1)^2 – 2r^2 \qquad -(3)$$

Calculating next decision parameter,

$$\_d_{i+1} = 2(x_i+2)^2 + y_{i+1}^2 + (y_{i+1}-1)^2 – 2r^2 \qquad -(4)$$

from (4)- (3), we get,

$$d_{i+1} – d_i = 2((x_i+2)^2-(x_i+1)^2) + (y_{i+1}^2 – y_i^2) + ((y_{i+1}-1)^2 + (y_i-1)^2)$$

$$\_d_{i+1} = d_i + 2((x_i+2+x_i+1)(x_i+2-x_i-1)) + ((y_{i+1}+y_i)(y_{i+1}-y_i)) + ((y_{i+1}-1+y_i-1)(y_{i+1}-1-y_i+1))$$

$$\_d_{i+1} = d_i + 2(2x_i+3) + ((y_{i+1}+y_i)(y_{i+1}-y_i)) + ((y_{i+1}-1+y_i-1)(y_{i+1}-1-y_i+1))$$

Now, if $(d_i<=0)$,

$x_{i+1}=x_i+1$ and $y_{i+1}=y_i$

so that $d_{i+1} = d_i + 2(2x_i + 3) + ((y_{i+1}+y_i)(y_i-y_i)) + ((y_i-1+y_i-1)(y_i-1-y_i+1))$

$$\_$$
$d_{i+1} = d_i + 2(2x_i + 3) +$
$((y_{i+1}+y_i)(0)) + ((y_i-1+y_i-1)(0))$

$$\_$$
$d_{i+1} = d_i + 4x_i + 6$

else

$$d_{i+1} = d_i + 2(2x_i+3) + ((y_i-1+y_i)(y_i-1-y_i)) + ((y_i-2+y_i-1)(y_i-2-y_i+1))$$

$$\_ d_{i+1} = d_i + 4x_i+6 + ((2y_i-1)(-1)) + ((2y_i-3)(-1))$$

$$\_ d_{i+1} = d_i + 4x_i+6 - 2y_i - 2y_i + 1 + 3$$

$$\_ d_{i+1} = d_i + 4(x_i - y_i) + 10$$

To know d_{i+1}, we have to know d_i first. The initial value of d_i can be obtained by replacing $x=0$ and $y=r$ in (3). Thus, we get,

$$d_o = 2 + r^2 + (r - 1)^2 - 2r^2$$

$$\_d_o = 2 + r^2 + r^2 + 1 - 2r - 2r^2$$

$$\_ d_o = 3 - 2r$$

The following codes are implement of Circle Algorithm in C++

```
# include <iostream.h>

# include <conio.h>

# include <graphics.h>

# include <math.h>


void MidPointCircleAlgo(int Radius,int xC,int yC); // Prototype


void main()

{

    int gDriver=DETECT, gMode;

    initgraph(&gDriver,&gMode,"c:\\tc\\bgi");
```

```
    int Radius, xC, yC;
    cout<< endl << "Enter Center point coordinates...";
    cout<<endl<<" Xc : ";
    cin>>xC;
    cout<<endl<<" Yc : ";
    cin>>yC;
    cout<<endl<<"Radius : ";
    cin>>Radius;

cleardevice();
    MidPointCircleAlgo(Radius,xC,yC);
    getch();
}

void MidPointCircleAlgo(int Radius,int xC,int yC)
{
    int P;
    int x,y;

void Draw(int x,int y,int xC,int yC); // Function to plots the points

P = 3 -2* Radius;
    x = 0;
    y = Radius;
```

```
    Draw(x,y,xC,yC);



while (x<=y)
    {
        x++;
        if (P<0)
        {
            P += 4 * x + 6;
        }
        else
        {
            P += 4 * (x - y) + 10;
            y--;
        }
        Draw(x,y,xC,yC);
    }

}
```

```
void Draw(int x,int y,int xC,int yC)

{

xC=320+xC;

yC=240-yC;

    putpixel(xC + x,yC + y,1);

    putpixel(xC + x,yC - y,1);

    putpixel(xC - x,yC + y,1);

    putpixel(xC - x,yC - y,1);

    putpixel(xC + y,yC + x,1);

    putpixel(xC - y,yC + x,1);

    putpixel(xC + y,yC - x,1);

    putpixel(xC - y,yC - x,1);

}
```

Using the highlighted formulas, we can easily plot a circle on a raster graphics display device. An implementation in C language is provided below.

Implementation

int x,y,d,r;

y=r;

putpixel(x,y,1);

d=(3-2*r);

while(x<=y)

```
{
    if(d<=0)
        d += (4*x+6);
    else
    {
        d = d+4*(x-y)+10;
        y--;
    }
    x++;
    putpixel(x,y,1);
    putpixel(-x,y,1);
    putpixel(x,-y,1);
    putpixel(-x,-y,1);
    putpixel(y,x,1);
    putpixel(-y,x,1);
    putpixel(y,-x,1);
    putpixel(-y,-x,1);
}
```