Circle drawing algorithms

Given the circle center point (x_c, y_c) and radius *R,* we want to draw an approximation of the circle. Different methods are based on one of circle equations given next.

[1] Circle equations:

Cartesian equation:

This equation expresses the fact that every point (x, y) on the circle is at a constant distance *R* from the center point (x_c, y_c), thus:

$$(x - x_c)^2 + (y - y_c)^2 = R^2$$

Parametric equation (polar form)

If the angle between the vector from the center point (x_c, y_c) to the point (x, y) and the positive x axis direction is θ then the coordinates of the point (x, y) are:

$$x = x_c + R\ cos(\theta)$$

$$x = y_c + R\ sin(\theta)$$

All circle points can be obtained by changing θ from 0 to 2π. Note here we use the radian system of measuring angles.

[2] Circle symmetry

Consider the case when the center point (x_c, y_c) is at the origin (i.e. $x_c = 0\ and\ y_c = 0$). If some point (a, b) lies on the circle, it will satisfy the circle equation, i.e.:

$$a^2 + b^2 = R^2$$

It is easy to see that changing the signs and exchanging the variables a and b in the left hand side of the above equation gives the same result. So, the following eight points are all on the circle if one of them is on the circle:

$$\{(a, b), (-a, b), (-a, -b), (a, -b), (b, a), (-b, a), (-b, -a), (b, -a)\}$$

This fact can be exploited in circle drawing algorithms by computing the points of only one-eighth of the circle and inferring all the other points by symmetry.

A circle with general center point (x_c, y_c) other than the origin are simply computed by shifting the points of the circle centered about the origin by (x_c, y_c), so if we have some point (a, b) on the circle centered about the origin, the corresponding eight similar points on the general circle are:

$$\{(x_c+a, y_c+b), (x_c-a, y_c+b), (x_c-a, y_c-b), (x_c+a, y_c-b), (x_c+b, y_c+a), (x_c-b, y_c+a), (x_c-b, y_c-a), (x_c+b, y_c-a)\}$$

The following utility function expresses this fact; it will be used in circle drawing algorithms.

```
void Draw8Points(HDC hdc,int xc,int yc, int a, int b,COLORREF color)
{
	SetPixel(hdc, xc+a, yc+b, color);
	SetPixel(hdc, xc-a, yc+b, color);
	SetPixel(hdc, xc-a, yc-b, color);
	SetPixel(hdc, xc+a, yc-b, color);
	SetPixel(hdc, xc+b, yc+a, color);
	SetPixel(hdc, xc-b, yc+a, color);
	SetPixel(hdc, xc-b, yc-a, color);
	SetPixel(hdc, xc+b, yc-a, color);
}
```

[3] Circle drawing methods

Note that in all of the following algorithms, the computations of circle points (x, y) are for points of circles centered about the origin and the utility ‘Draw8Points’ described above is used to exploit the 8-symmetry and shifting properties for general circles.

3-1. Direct Cartesian method

Circle equation ($x^2 + y^2 = R^2$) can be solved for y as:

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

Circle points are obtained by changing x and computing y. To know which octant (one-eighth of the circle) to compute in this case, we must be sure that the slope of the circle at each point of the octant is less than one in magnitude (absolute value). By taking the derivative of both sides of the circle equation with respect to x, we obtain:

$$2x + 2y\frac{dy}{dx} = 0$$

So:

$$slope = \frac{dy}{dx} = -\frac{x}{y}$$

For the absolute value of the slope to be less than or equal to one, we should have:

$$|x| \le |y|$$

One of the octants satisfying this condition is the one satisfying the condition: $0 \le x \le y$, so the main loop of the algorithm starts at x=0 and ends when x=y.

```
void CircleDirect(HDC hdc,int xc,int yc, int R,COLORREF color)
```

```
{
	int x=0,y=R;
	int R2=R*R;
	Draw8Points(hdc,xc,yc,x,y,color);
	while(x<y)
	{
		x++;
		y=round(sqrt((double)(R2-x*x)));
		Draw8Points(hdc,xc,yc,x,y,color);
	}
}
```

3-2. Polar direct method

Using polar equations ($x = R\cos(\theta), y = R\sin(\theta)$), one can change θ from 0 to $\frac{\pi}{4}$ for example and compute both x and y in each iteration to compute the first octant. This octant also satisfy the condition: $0 \le y \le R$. The problem is how to select the step $\Delta\theta$ by which θ is incremented? If $\Delta\theta$ is big, we'll obtain few (disconnected) points. If it is very small, we'll obtain so many points that will be approximated (by rounding) in such a way that the same pixel is drawn many times. To tackle this problem, we may notice that the length of a circle arc making an angle $\Delta\theta$ is $R\Delta\theta$ provided that $\Delta\theta$ is expressed in radians. So, for an arc length of 1, we must have:

$$R\Delta\theta = 1$$

So

$$\Delta\theta = \frac{1}{R}$$

The implementation of the algorithm is as shown below:

```
void CirclePolar(HDC hdc,int xc,int yc, int R,COLORREF color)
{
	int x=R,y=0;
	double theta=0,dtheta=1.0/R;
	Draw8Points(hdc,xc,yc,x,y,color);
	while(x>y)
	{
		theta+=dtheta;
		x=round(R*cos(theta));
		y=round(R*sin(theta));
		Draw8Points(hdc,xc,yc,x,y,color);
	}
}
```

3-3. Iterative polar algorithm

The above algorithm computes the trigonometric functions sin and cos which take a very long time because they are computed using infinite series expansions. [$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots, \cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots$]

To speed up the algorithm the following iterative approach is used:

Let (x, y) be a point on the circle centered about the origin. Let (R, θ) be its corresponding polar coordinates. Mathematically:

$$x = R\cos\theta$$

$$y = R\sin\theta$$

Next point (x', y') to draw on the circle is obtained by incrementing θ by $\Delta\theta$, thus it has polar coordinates (R, $\theta + \Delta\theta$). Mathematically:

$$x' = R\cos(\theta + \Delta\theta)$$

$$y' = R\sin(\theta + \Delta\theta)$$

Expanding x' and y' using some trigonometric identities:

$$\begin{aligned} x' &= R(\cos\theta\cos\Delta\theta - \sin\theta\sin\Delta\theta) = (R\cos\theta)\cos\Delta\theta - (R\sin\theta)\sin\Delta\theta \\ &= x\cos\Delta\theta - y\sin\Delta\theta \end{aligned}$$

$$\begin{aligned} y' &= R\,(\cos\theta\sin\Delta\theta + \sin\theta\cos\Delta\theta) = (R\cos\theta)\sin\Delta\theta + (R\sin\theta)\cos\Delta\theta \\ &= x\sin\Delta\theta + y\cos\Delta\theta \end{aligned}$$

The idea is to start at some point (R, 0) and use the above formulae to compute next points iteratively. The following functions implements the idea.

```
void CircleIterativePolar(HDC hdc,int xc,int yc, int R,COLORREF color)
{
	double x=R,y=0;
	double dtheta=1.0/R;
	double cdtheta=cos(dtheta),sdtheta=sin(dtheta);
	Draw8Points(hdc,xc,yc,R,0,color);
	while(x>y)
	{
		double x1=x*cdtheta-y*sdtheta;
		y=x*sdtheta+y*cdtheta;
		x=x1;
		Draw8Points(hdc,xc,yc,round(x),round(y),color);
	}
}
```

3-4. Bresenham (midpoint) algorithm

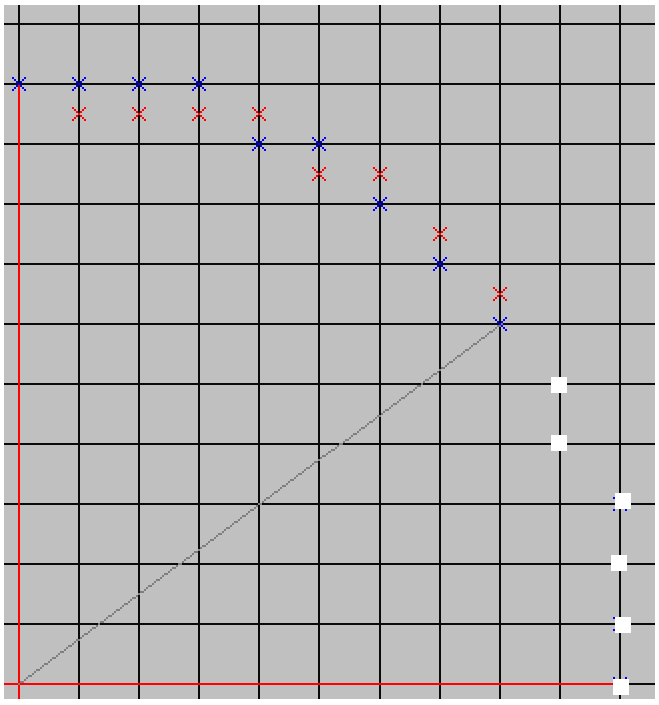
The following function can be used to test if some point is on the circle, inside or outside the circle (centered at the origin)

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

Using this function to test a point (x, y) is as follows:

$$f(x,y) = \begin{cases} 0 & (x,y) \textit{is on the circle} \\ < 0 & (x,y) \textit{is inside the circle} \\ > 0 & (x,y) \textit{is outside the circle} \end{cases}$$

The idea is to start at the point (0, R). This point is considered the current point (x, y). The circle testing function is then used to test whether the next midpoint (x+1, y-1/2) is inside or outside the circle. If it is inside, next point to draw is (x+1, y), otherwise next point is (x+1, y-1). In the shown Figure, drawn points are represented as blue stars while the midpoints are represented as red stars.



An iterative approach is used to compute the testing function at the middle point as follows:

Define d(x, y) as the mid-point test function to be tested after drawing the point (x, y). d(x, y) is thus given by:

$$d(x,y) = f\left(x+1, y-\frac{1}{2}\right) = (x+1)^2 + (y-\frac{1}{2})^2 - R^2 = x^2 + 2x + 1 + y^2 - y + \frac{1}{4} - R^2$$
$$= (x^2 + y^2 - R^2) + 2x - y + \frac{5}{4}$$

So,

$$d(x,y) = f(x,y) + 2x - y + \frac{5}{4}$$

Initial value of d(x, y) is:

$$d_{(initial)} = d(0,R) = f(0,R) + 2(0) - R + \frac{5}{4} = \frac{5}{4} - R$$

Change in d(x, y):

If d(x, y)<0 then (x, y) is inside the circle and next point to draw is (x+1, y) so the change in d is given by:

$$\Delta d = d(x+1,y) - d(x,y)$$

So:

$$\begin{aligned}\Delta d &= f(x+1,y) + 2(x+1) - y + \frac{5}{4} - \left(f(x,y) + 2x - y + \frac{5}{4}\right)\\ &= f(x+1,y) - f(x,y) + 2 = (x+1)^2 + y^2 - R^2 - (x^2 + y^2 - R^2) + 2\\ &= 2x + 3\end{aligned}$$

If d(x, y)>0 then (x, y) is outside the circle and next point to draw is (x+1, y-1) and the change in d is thus:

$$\Delta d = d(x+1,y-1) - d(x,y)$$

So:

$$\begin{aligned}\Delta d &= f(x+1,y-1) + 2(x+1) - (y-1) + \frac{5}{4} - \left(f(x,y) + 2x - y + \frac{5}{4}\right)\\ &= f(x+1,y-1) - f(x,y) + 3\\ &= (x+1)^2 + (y-1)^2 - R^2 - (x^2 + y^2 - R^2) + 3 = 2(x-y) + 5\end{aligned}$$

The initial value of d contains the constant value $\frac{5}{4}$ which is a real number. Fortunately $\frac{5}{4} = 1 + \frac{1}{4}$ and removing $\frac{1}{4}$ from this constant will not affect the algorithm since the sign of d will not be affected by this removal. So we will use d=1-R as the initial value of d and the algorithm will be purely integer. Bresenham algorithm for circle drawing is given below:

```
void CircleBresenham(HDC hdc,int xc,int yc, int R,COLORREF color)
{
	int x=0,y=R;
	int d=1-R;
	Draw8Points(hdc,xc,yc,x,y,color);
	while(x<y)
	{
		if(d<0)
			d+=2*x+2;
		else
		{
			d+=2*(x-y)+5;
			y--;
		}
		x++;
		Draw8Points(hdc,xc,yc,x,y,color);
	}
}
```

Further enhancement of the above algorithm can be obtained if the computation of the changes (2x+3) and (2(x-y)+5) are done iteratively as follows:

Define the functions $c_1(x,y)$ and $c_2(x,y)$ as:

$$c_1(x,y) = 2x + 3$$

$$c_2(x,y) = 2(x - y) + 5$$

Initial values of the two functions are thus given by:

$$c_1(0,R) = 2(0) + 3 = 3$$

$$c_2(0,R) = 2(0 - R) + 5 = 5 - 2R$$

Changes in the two functions with changing decision d are as follows

If d(x,y) < 0 then

$$\Delta c_1(x,y) = c_1(x+1,y) - c_1(x,y) = 2(x+1) + 3 - (2x+3) = 2$$

$$\Delta c_2(x,y) = c_2(x+1,y) - c_2(x,y) = 2(x+1-y) + 5 - (2(x-y)+5) = 2$$

Else

$$\Delta c_1(x,y) = c_1(x+1,y-1) - c_1(x,y) = 2(x+1) + 3 - (2x+3) = 2$$

$$\Delta c_2(x,y) = c_2(x+1,y-1) - c_2(x,y) = 2(x+1-y+1) + 5 - (2(x-y)+5) = 4$$

End if

The implementation of the following is given below:

```
void CircleFasterBresenham(HDC hdc,int xc,int yc, int R,COLORREF color)
{
	int x=0,y=R;
	int d=1-R;
	int c1=3, c2=5-2*R;
	Draw8Points(hdc,xc,yc,x,y,color);
	while(x<y)
	{
		if(d<0)
		{
			d+=c1;
			c2+=2;
		}
		else
		{
			d+=c2;
			c2+=4;
			y--;
		}
		c1+=2;
		x++;
		Draw8Points(hdc,xc,yc,x,y,color);
	}
}
```