1 Bresenham's Circle Drawing Algorithm

Classic equation for a circle looks like this:

$$x^2 + y^2 = r^2 \tag{1}$$

The idea for Bresenham's algorithm comes from defining the above equation like this:

$$x^2 + y^2 - r^2 = 0 \tag{2}$$

That way we can think of it in terms of three cases:

$$x^2 + y^2 - r^2 = 0 \tag{3}$$
$$x^2 + y^2 - r^2 < 0 \tag{4}$$
$$x^2 + y^2 - r^2 > 0 \tag{5}$$

When our equation is '≤ 0' then we can be sure our point $(x; y)$ is somewhere on a circle or inside it, if our equation is '> 0' however then we need to correct our point, because it's outside of our defined radius 'r'. This gives us a nice 'error function':

$$x^2 + y^2 - r^2 = F(x, y) \tag{6}$$

The idea of a 'midpoint' comes from the fact that we are working with discrete coordinates, meaning we can move 1 pixel left/right or up/down, it's not really* possible for us to move 0.125 of a pixel to the side.

In other words, if we start at some point $(r; 0)$ and always move up (increasing y), then we just need to change our x only in certain cases, adjusting it by 1 pixel (either x stays the same or decreases by 1):

Figure 1: Circle 'pixel staircase'

This interesting fact gives us two points that we need to consider **(x, y + 1)** and **(x - 1, y + 1)**, if we average them we get: **(x - 0.5, y + 1)**. This is a very crucial observation that will help us adjust our 'error function' (Eq. 6).

We also need to consider two cases because, while our y always changes, our 'next' x can either stay the same or decrease by 1:

$$x_{n+1} = x_n \vee x_n - 1 \tag{7}$$
$$y_{n+1} = y_n + 1 \tag{8}$$

This gives us the following 'error functions'.

$$F(x_n, y_n) = (x_n - 0.5)^2 + (y_n + 1)^2 - r^2 \tag{9}$$
$$F(x_{n+1}, y_{n+1}) = (x_{n+1} - 0.5)^2 + (y_{n+1} + 1)^2 - r^2 \tag{10}$$

Let's modify the equation for $F(x_{n+1}, y_{n+1})$:

$$\begin{aligned}
F(x_{n+1}, y_{n+1}) &= (x_{n+1} - 0.5)^2 + [(y_n + 1) + 1]^2 - r^2 \\
F(x_{n+1}, y_{n+1}) &= (x_{n+1} - 0.5)^2 + (y_n + 1)^2 + 2(y_n + 1) + 1 - r^2 \\
F(x_{n+1}, y_{n+1}) &= (x_{n+1} - 0.5)^2 + (y_n + 1)^2 + 2(y_n + 1) + 1 - r^2 \\
&\quad + (x_n - 0.5)^2 - (x_n - 0.5)^2 \\
F(x_{n+1}, y_{n+1}) &= F(x_n, y_n) + (x_{n+1} - 0.5)^2 - (x_n - 0.5)^2 + 2(y_n + 1) + 1 \\
F(x_{n+1}, y_{n+1}) &= F(x_n, y_n) + x_{n+1}^2 - x_{n+1} + 0.25 - x_n^2 + x_n - 0.25 + 2(y_n + 1) + 1 \\
F(x_{n+1}, y_{n+1}) &= F(x_n, y_n) + (x_{n+1}^2 - x_n^2) - (x_{n+1} - x_n) + 2(y_n + 1) + 1
\end{aligned}$$

Terribly sorry for the above Wall of Math^{TM}, but I feel it's neccessary to show where it's all coming from, instead of providing 'black-boxes' and telling people to just blindly trust them. The final 'error function' is therefore:

$$F(x_{n+1}, y_{n+1}) = F(x_n, y_n) + (x_{n+1}^2 - x_n^2) - (x_{n+1} - x_n) + 2(y_n + 1) + 1 \tag{11}$$

For $x_{n+1} = x_n$ we get:

$$F(x_{n+1}, y_{n+1}) = F(x_n, y_n) + 2(y_n + 1) + 1 \tag{12}$$

For $x_{n+1} = x_n - 1$ we get:

$$F(x_{n+1}, y_{n+1}) = F(x_n, y_n) + 2(y_n + 1) - 2(x_n - 1) + 1 \tag{13}$$

That's all nice, but our 'error function' depends on the previous point, meaning that we can only calculate it in terms of $F(x_{n+1}, y_{n+1})$. That's not a problem, since we know our starting point $(r; 0)$, which gives us the following:

$$F(x_n, y_n) = (r - 0.5)^2 + (0 + 1)^2 - r^2 \tag{14}$$

$$F(x_n, y_n) = 1.25 - r \tag{15}$$

When working with integers 1.25 can be rounded to just 1. The equations that we actually need then are the following.

The equation for starting 'error value':

$$F(x_n, y_n) = 1.25 - r \tag{16}$$

Next 'error value' when we are still inside a circle '≤ 0 case':

$$F(x_{n+1}, y_{n+1}) = F(x_n, y_n) + 2(y_n + 1) + 1 \tag{17}$$

Next 'error value' when we are <u>not</u> inside a circle '> 0 case':

$$F(x_{n+1}, y_{n+1}) = F(x_n, y_n) + 2(y_n + 1) - 2(x_n - 1) + 1 \tag{18}$$

For further simplification, we also don't need to consider all the points, we can make use of the fact that cicles have nice symmetry to them and just calculate all the points in the first octant. After that we can just 'mirror' them accordingly:

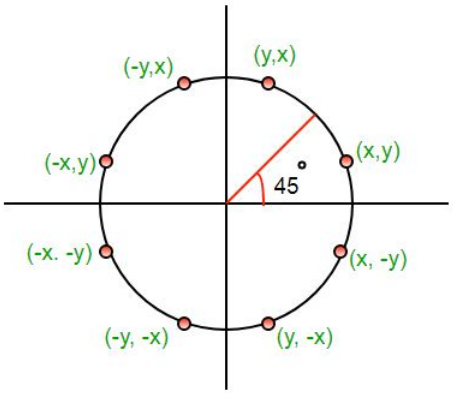

Figure 2: Circle octants

2 Simple C/C++ Code Block (SDL2)

```
void draw_circle(SDL_Renderer *renderer, int cx, int cy, int r)
{
    int x = r;
    int y = 0;
    int p = 1 - r;

    SDL_SetRenderDrawColor(renderer, 255, 0, 0, 255);
    SDL_RenderDrawLine(renderer, cx - r, cy, cx + r, cy);

    while (x > y) {
        y += 1;

        if (p <= 0) {
            p += 2 * y + 1;
        } else {
            x -= 1;
            p += 2 * y - 2 * x + 1;
        }

        // @Note: Using circle octants to not go over every point.
        SDL_RenderDrawLine(renderer, cx+x, cy-y, cx-x, cy-y);
        SDL_RenderDrawLine(renderer, cx+x, cy+y, cx-x, cy+y);
        SDL_RenderDrawLine(renderer, cx+y, cy-x, cx-y, cy-x);
        SDL_RenderDrawLine(renderer, cx+y, cy+x, cx-y, cy+x);
    }
}
```