Table of Contents

Drawing Lines

Pixels are referenced with a coordinate pair (x, y) . Each pixel has color and intensity attributes. Drawing lines is performed by turning pixels on in the frame buffer. The contents of the frame buffer is displayed at regular intervals (60Hz, usually).

Simple Line Drawing Algorithm

Given two points (x_1, y_1) and (x_2, y_2) and from the general equation of a 2D line $y = mx + b$, we can write the following:

$$\begin{aligned} y_1 &= mx_1 + b \\ y_2 &= mx_2 + b \\ b &= y_1 - mx_1 \\ b &= y_2 - mx_2 \end{aligned}$$

and, therefore:

$$\begin{aligned} y_1 - mx_1 &= y_2 - mx_2 \\ y_1 - y_2 &= m(x_1 - x_2) \\ \Delta y &= m\Delta x \end{aligned}$$

Consider the line $y = \left(\frac{1}{3}\right)x + \frac{2}{3}$. We work with x , starting the line at $(1,1)$.

x	y	$round(y)$
1	1	1
2	$4/3$	1
3	$5/3$	2
4	2	2

5	7/3	2
6	8/3	3
7	3	3

We observe that what is really being computed here is:

$$\begin{aligned} \Delta y &= m\Delta x \\ (y_{i+1}-y_i) &= m(x_{i+1}-x_i) \\ y_{i+1} &= m+y_i \end{aligned}$$

since $x_{i+1}-x_i=1$. Hence the naïve algorithm, for tracing a line between (x_1, y_1) and (x_n, y_n) is simply:

```
begin
    m=(y_n-y_1)/(x_n-x_1) ; /* delta x not zero
    x_1=round(x_1) ;
    y_1=round(y_1) ;
  plot (x_1, y_1) ;
  for (j = 2 to n by step of 1) do {
      (x_j, y_j)=(x_{j-1}+1, y_j-1+m) ;
    plot (x_j, round(y_j)) ;
  endfor
end
```

What happens if we try to use this algorithm to plot a line for which the slope is greater than 1? The line traced will not be solid, as in this case it is y that must vary by 1 and x to vary by m^{-1} .

Bresenham's Integer Arithmetic Line Algorithm

Let $0<m<1$ and $\Delta x=1$. At x_{i+1} , we need to choose between y_i and y_{i+1} . Given x_{i+1} , what is y ?

$$\begin{aligned} y &= m(x_{i+1})+b \\ d_1 &= y-y_i = m(x_i+1)+b-y_i \\ d_2 &= (y_i+1)-y = y_i+1-m(x_i+1)-b \end{aligned}$$

and we can write:

$$\begin{aligned} d_1-d_2 &= m(x_i+1)+b-y_i-(y_i+1-m(x_i+1)-b) \\ &= 2m(x_i+1)-2y_i+2b-1 \\ &= \frac{2\Delta y}{\Delta x}(x_i+1)-2y_i+2b-1 \end{aligned}$$

Multiplying with Δx , one obtains:

$$\begin{aligned}\Delta x(d_1-d_2) &= \frac{2\Delta y\Delta x}{\Delta x}(x_i+1)-2\Delta x\, y_i+2b\Delta x-\Delta x \\ &= 2\Delta y\, x_i+2\Delta y-2\Delta x\, y_i+2b\Delta x-\Delta x\end{aligned}$$

Pose $p_i=\Delta x(d_1-d_2)$, and write:

$$p_i=2\Delta y\, x_i-2\Delta x\, y_i+2\Delta y+\Delta x(2b-1)$$

if $p_i<0$ then $d_1<d_2$ and y_i is closer to y . Otherwise, $d_1\geq d_2$ and y_{1+1} is closer to y . We now rewrite this equation in terms of (x_{i+1}, y_{i+1}) :

$$p_{i+1}=2\Delta y\, x_{i+1}-2\Delta x\, y_{i+1}+2\Delta y+\Delta x(2b-1)$$

and find

$$p_{i+1}-p_i=2\Delta y\, x_{i+1}-2\Delta x\, y_{i+1}-(2\Delta y\, x_i-2\Delta x\, y_i)$$

Because $0<m<1$, it is certain that $x_{i+1}=x_i+1$ and we can write:

$$\begin{aligned}p_{i+1}-p_i &= 2\Delta y(x_i+1)-2\Delta x\, y_{i+1}-(2\Delta y\, x_i-2\Delta x\, y_i) \\ &= 2\Delta y-2\Delta x(y_{i+1}-y_i)\end{aligned}$$

We now have an iterative mechanism to determine which pixel to turn on, as we progress on the x axis by steps of 1.

At this point, we need to determine the value of p_1 such that the iteration may start. We can immediately write:

$$p_1=2\Delta y\, x_1-2\Delta x\, y_1+2\Delta y+\Delta x(2b-1)$$

and divide by Δx to obtain:

$$\begin{aligned}\frac{p_1}{\Delta x} &= 2(mx_1-y_1+b)+\frac{2\Delta y-\Delta x}{\Delta x} \\ p_1 &= 2\Delta y-\Delta x\end{aligned}$$

If $p_i<0$ then y_i is used for y_{i+1} and hence $p_{i+1}-p_i=2\Delta y-2\Delta x(y_{i+1}-y_i)$. Since $y_{i+1}-y_i=0$, then, in this case, $p_{i+1}=p_i+2\Delta y$. If $p_i\geq 0$ then y_1+1 is used as y_{i+1} and hence $y_{1+1}-y_i=1$. Consequently, $p_{i+1}=p_i+2\Delta y-2\Delta x$.

We can write Bresenham's algorithm in the following way:

```
begin
  plot (x1, y1) ;
  for (i= x1  to x2 by step of 1) {
```

```
    if (i== x1 ) {
        pi=2Δy−Δx ;
    }
    else {
        if ( pi<0 ) {
              pi= pi+2Δy ;
        }
        else {
              pi= pi+2Δy−2Δx ;
              y1 ++ ;
        }
         x1 ++ ;
         plot (x1, y1) ;
    }
  }
end
```

Symmetric Cases

It is important to note that this algorithm performs correctly when the slope of the line to draw is comprised between 0 and 1. There are 7 other cases that need to be addressed and symmetries are put to use to perform this (by exchanging x with y , and/or exchanging Δx with Δy , and incrementing/decrementing the x and y coordinates accordingly).

Drawing Circles

The equation of a circle centered at the origin is given by:

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

where r is the radius. Because of symmetries, we just need to compute the pixel coordinates of one eighth of the circle to draw. The technique is to start at the top of the circle $(0,r)$ and proceed until $x=y$ (we assume the circle is centered at the origin).

Bresenham's Circle Drawing Algorithm

Assume $(x_c, y_c)=(0,0)$ and start at the top of the circle: $(0,r)$. Hence the current pixel is (x_i, y_i) and the next pixel to turn on must be either (x_i+1, y_i) or (x_i+1, y_i-1) . The exact value for y is obtained as:

$$y^2=r^2-(x_i+1)^2$$

and we define distances d_1 and d_2 in the following manner:

$$\begin{aligned} d_1 &= y_i^2-y^2 &= y_i^2-(r^2-(x_i+1)^2) \\ d_2 &= y^2-(y_i-1)^2 &= r^2-(x_i+1)^2-(y_i-1)^2 \end{aligned}$$

and

$$p_i = d_1 - d_2 = 2(x_i+1)^2 + y_i^2 + (y_i-1)^2 - 2r^2$$

If $p_i < 0$, then we choose y_i for y_{i+1} . Else we choose $y_i - 1$ for y_{i+1} . As with the line algorithm, we define p_{i+1} in terms of p_i :

$$\begin{aligned} p_{i+1} &= 2((x_i+1)+1)^2 + y_{i+1}^2 + (y_{i+1}-1)^2 - 2r^2 \\ &= p_i + 4x_i + 6 + 2(y_{i+1}^2 - y_i^2) - 2(y_{i+1} - y_i) \end{aligned}$$

where y_{i+1} is either y_i or $y_i - 1$, depending on the sign of p_i . At the start, p_1 is obtained by setting (x_1, y_1) to $(0, r)$ in the first equation obtained for p_i , yielding $p_1 = 3 - 2r$.

Suppose $p_i < 0$. Then $y_{i+1} = y_i$ and we have

$$p_{i+1} = p_i + 4x_i + 6 + 2(y_i^2 - y_i^2) - 2(y_i - y_i) = p_i + 4x_i + 6$$

Conversely, if $p_i \geq 0$ then $y_{i+1} = y_i - 1$ and

$$p_{i+1} = p_i + 4x_i + 6 + 2((y_i-1)^2 - y_i^2) - 2(y_i - 1 - y_i) = p_i + 4(x_i - y_i) + 10$$

The algorithm can be expressed as:

```
begin
    x=0 ;
    y=r ;
    plot (x,y) ;
    d=3-2r ;
  while ( x<y ) {
      x ++ ;
    if ( d<0 ) {
        d=d+4x+6 ;
    }
    else {
        y -- ;
        d=d+4(x-y)+10 ;
    }
    plot (x,y) ;
  }
end
```

Symmetric Cases

This algorithm only draws one eighth of the circle. Symmetries are used to draw the rest of it. The way to do this is that each time a pixel (x, y) is turned on, we then also turn on the following ones: (y, x), $(x, -y)$, $(y, -x)$, $(-y, -x)$, $(-x, -y)$, $(-x, y)$, $(-y, x)$.

Drawing Ellipses

The previous circle drawing algorithm can be extended to plot ellipses. The ellipse equation can written as

$$\left(\frac{x}{r_1}\right)^2 + \left(\frac{y}{r_2}\right)^2 = 1$$

where r_1 and r_2 are the semi-major and semi-minor axes, respectively. Bresenham's algorithm can trace ellipses by using this equation instead of that of a circle in the evaluation of the parameter p_i . For an ellipse centered at the origin, we can express y values as

$$y^2 = r_2^2\left(1-\frac{x^2}{r_1^2}\right)$$

and modify the computation of p_i accordingly. Let's develop the required mathematics and the algorithm to trace ellipses with integer arithmetic only. In the above equation r_1 and r_2 can be referred to as the width and height of the ellipse respectively. As with the circle we need to develop an expression that gives us the exact y value at x_i+1 . This is obtained with the above equation, yielding

$$y^2=r_2^2\left(1-\frac{(x_i+1)^2}{r_1^2}\right)$$

We proceed as with the circle, to define squares of distances as

$$d_1 = y_i^2-y^2 = y_i^2-r_2^2\left(1-\frac{(x_i+1)^2}{r_1^2}\right)$$

and multiply by r_1^2 on both sides to eliminate the denominator:

$$r_1^2 d_1 = r_1^2(y_i^2-y^2) = r_1^2 y_i^2-r_2^2\left(r_1^2-(x_i+1)^2\right) = r_1^2 y_i^2-r_1^2 r_2^2+r_2^2(x_i+1)^2$$

In a similar way we define the second distance as

$$d_2 = y^2-(y_i-1)^2 = r_2^2\left(1-\frac{(x_1+1)^2}{r_1^2}\right)-(y_i-1)^2$$

and multiply by r_1^2 on both sides:

$$r_1^2 d_2 = r_2^2\left(r_1^2-(x_i+1)^2\right)-r_1^2(y_i-1)^2$$

We define p_i as

$$p_i = r_1^2 d_1 - r_1^2 d_2 = r_1^2 y_i^2 - r_1^2 r_2^2 + r_2^2 (x_i+1)^2 - \left[r_2^2 \left(r_1^2 - (x_i+1)^2 \right) - r_1^2 (y_i-1)^2 \right]$$

which simplifies to

$$p_i = r_1^2 y_i^2 - 2 r_1^2 r_2^2 + 2 r_2^2 (x_i+1)^2 + r_1^2 (y_i-1)^2$$

We now express p_{i+1} in terms of p_i :

$$p_{i+1} = r_1^2 y_{i+1}^2 - 2 r_1^2 r_2^2 + 2 r_2^2 \left((x_i+1)+1 \right)^2 + r_1^2 (y_{i+1}-1)^2$$

and write

$$p_{i+1} - p_i = r_1^2 y_{i+1}^2 - 2 r_1^2 r_2^2 + 2 r_2^2 \left((x_i+1)+1 \right)^2 + r_1^2 (y_{i+1}-1)^2 - \left[r_1^2 y_i^2 - 2 r_1^2 r_2^2 + 2 r_2^2 (x_i+1)^2 + r_1^2 (y_i-1)^2 \right]$$

which simplifies to

$$p_{i+1} = p_1 + r_1^2 (y_{i+1}^2 - y_i^2) + r_2^2 (4 x_i + 6) + r_1^2 (y_{i+1}-1)^2 - r_1^2 (y_i-1)^2$$

If $p_i < 0$ then we choose y_i for y_{i+1} and p_{i+1} becomes

$$p_{i+1} = p_i + r_2^2 (4 x_i + 6)$$

If $p_i \geq 0$ then we choose $y_i - 1$ for y_{i+1} and p_{i+1} becomes

$$p_{i+1} = p_i + r_2^2 (4 x_i + 6) + 4 r_1^2 (1 - y_i)$$

We begin drawing at $(0, r_2)$ and p_1 is thus

$$p_1 = 2 r_2^2 + r_1^2 (1 - 2 r_2)$$

We need to keep drawing until

$$\frac{x^2}{r_1^2} = \frac{y^2}{r_2^2}$$

where the slope of the tangent to the ellipse is $m = -1$. In order to keep this stop condition in integer form, we multiply it on both sides with $r_1^2 r_2^2$ to obtain $(r_2 x)^2 = (r_1 y)^2$, which is equivalent to $r_2 x = r_1 y$ when used as a stop condition.

The next step is to trace the remaining part of the ellipse (in the first quadrant) after $r_2 x = r_1 y$. It is easy to see that if we interchange x and y and trace from $(r_1, 0)$ in

steps along y we will obtain the remaining part of the ellipse in the first quadrant. This transforms p_1 into

$$p_1 = 2r_1^2 + r_2^2(1 - 2r_1)$$

and p_{i+1} into

$$p_{i+1} = p_i + r_1^2(4y_i + 6) + 4r_2^2(1 - x_i)$$

We can now write the algorithm that traces the part of the ellipse in the first quadrant. The rest of the curve is obtained by symmetry.

Bresenham's Ellipse Drawing Algorithm

```
begin
    x=0 ;
    y=r2 ;
    plot (x,y) ;
    d=2r2^2+r1^2(1-2r2) ;
  while ( r2x≤r1y ) {
      x ++ ;
    if ( d<0 ) {
        d=d+r2^2(4x+6) ;
    }
    else {
        y -- ;
        d=d+r2^2(4x+6)+4r2^2(1-y) ;
    }
    plot (x,y) ;
  }

    x=r1 ;
    y=0 ;
    plot (x,y) ;
    d=2r1^2+r2^2(1-2r1) ;
  while ( r1y≤r2x ) {
      y ++ ;
    if ( d<0 ) {
        d=d+r1^2(4y+6) ;
    }
    else {
        x -- ;
        d=d+r1^2(4y+6)+4r1^2(1-x) ;
    }
```

```
    plot (x,y) ;
  }
end
```