Integer–based Algorithm for Drawing Ellipses

David Eberly
Magic Software, Inc.
http://www.magic-software.com

Created: March 2, 1999

1 Specifying the Ellipse

The algorithm described in this document is for drawing ellipses of any orientation on a 2D raster. The simplest way for an application to specify the ellipse is by choosing an oriented bounding box with center (x_c, y_c) and axes (x_a, y_a) and (x_b, y_b) where all components are integers. The axes must be perpendicular, so $x_a x_b + y_a y_b = 0$. I assume that (x_a, y_a) is in the first quadrant (not including the y–axis), so $x_a > 0$ and $y_a \geq 0$ are required. I also require that the other axis is in the second quadrant, so $x_b \leq 0$ and $y_b > 0$. There must be integers n_a and n_b such that $n_b(x_b, y_b) = n_a(-y_a, x_a)$, but the algorithm does not require knowledge of these. The ellipse axes are the box axes and has the same orientation as the box.

All pixel computations are based on the ellipse with center $(0, 0)$. These pixels are translated by (x_c, y_c) to obtain the ones for the original ellipse. A quadratic equation for the ellipse centered at the origin is

$$\frac{(x_a x + y_a y)^2}{(x_a^2 + y_a^2)^2} + \frac{(x_b x + y_b y)^2}{(x_b^2 + y_b^2)^2} = 1.$$

In this form it is easy to see that (x_a, y_a) and (x_b, y_b) are on the ellipse. Multiplying the matrices and multiplying through by denominators yields the quadratic equation

$$Ax^2 + 2Bxy + Cy^2 = D$$

where the integer coefficients are

$$\begin{aligned} A &= x_a^2(x_b^2 + y_b^2)^2 + x_b^2(x_a^2 + y_a^2)^2 \\ B &= x_a y_a(x_b^2 + y_b^2)^2 + x_b y_b(x_a^2 + y_a^2)^2 \\ C &= y_a^2(x_b^2 + y_b^2)^2 + y_b^2(x_a^2 + y_a^2)^2 \\ D &= (x_a^2 + y_a^2)^2(x_b^2 + y_b^2)^2 \end{aligned}$$

For standard size rasters, these integers can be quite large. The code uses 64–bit integers to accommodate this.

2 Axis–Aligned Ellipses

The algorithm for an axis–aligned ellipse draws that arc of the ellipse in the first quadrant and uses reflections about the coordinate axes to draw the other arcs. The ellipse centered at the origin is $b^2x^2 + a^2y^2 = a^2b^2$.

Starting at $(0, b)$, the arc is drawn in clockwise order. The initial slope of the arc is 0. As long as the arc has slope smaller than 1 in magnitude, the x value is incremented. The corresponding y value is selected based on a decision variable, just as in Bresenham's circle drawing algorithm. The remaining part of the arc in the first quadrant has slope larger than 1 in magnitude. That arc is drawn by starting at $(a, 0)$ and incrementing y at each step. The corresponding x value is selected based on a decision variable.

While drawing the arc starting at $(0, b)$, let (x, y) be the current pixel that has been drawn. A decision must be made to select the next pixel $(x + 1, y + \delta)$ to be drawn. Here δ is either 0 or -1. The ellipse is defined implicitly as $Q(x, y) = 0$ where $Q(x, y) = b^2x^2 + a^2y^2 - a^2b^2$. Each choice for next pixel lies on its own ellipse defined implicitly by $Q(x, y) = \lambda$ for some constant λ that is not necessarily zero. The idea is to choose δ so that the corresponding level curve has λ as close to zero as possible. This is the same idea that is used for Bresenham's circle algorithm. In the case of the circle algorithm, the choice relates to selecting the pixel that is closest to the true circle. For ellipses, the choice is based on level set value and not on distance between two ellipses (a much harder problem).

Given current pixel (x, y), for the next step the ellipse must do one of three things:

1. pass below $(x + 1, y)$ and $(x + 1, y - 1)$ in which case $Q(x + 1, y) \geq 0$ and $Q(x + 1, y - 1) \geq 0$,
2. pass between $(x + 1, y)$ and $(x + 1, y - 1)$ in which case $Q(x + 1, y) \geq 0$ and $Q(x + 1, y - 1) \leq 0$,
3. pass above $(x + 1, y)$ and $(x + 1, y - 1)$ in which case $Q(x + 1, y) \leq 0$ and $Q(x + 1, y - 1) \leq 0$.

In the first case the next pixel to draw is $(x+1, y)$. In the third case the next pixel to draw is $Q(x+1, y-1)$. In the second case, the pixel with Q value closest to zero is chosen. The decision in all three cases can be made by using the sign of $\sigma = Q(x + 1, y) + Q(x + 1, y - 1)$. If $\sigma < 0$, then next pixel is $(x + 1, y - 1)$. If $\sigma > 0$, then next pixel is $(x + 1, y)$. For $\sigma = 0$ either choice is allowed, so I choose $(x + 1, y)$.

The decision variable σ can be updated incrementally. The initial value is $\sigma_0 = Q(1, b) + Q(1, b - 1) = 2b^2 + a^2(1 - 2b)$. Given current pixel (x, y) and decision variable σ_i, the next decision is

$$\sigma_{i+1} = \left\{ \begin{array}{ll} Q(x + 2, y) + Q(x + 2, y - 1), & \sigma_i \geq 0 \\ Q(x + 2, y - 1) + Q(x + 2, y - 2), & \sigma_i < 0 \end{array} \right\}.$$

The choice is based on whether or not the chosen pixel after (x, y) is $(x+1, y)$ [when $\sigma_i > 0$] or $(x+1, y-1)$ [when $\sigma_i \leq 0$]. Some algebra leads to

$$\sigma_{i+1} = \sigma_i + \left\{ \begin{array}{ll} 2b^2(2x + 3), & \sigma_i \geq 0 \\ 2b^2(2x + 3) + 4a^2(1 - y), & \sigma_i < 0 \end{array} \right\}.$$

On this arc x is always incremented at each step. The processing stops when slope becomes 1 in magnitude. The slope dy/dx of the ellipse can be computed implicitly from $Q(x, y) = 0$ as $Q_x + Q_y dy/dx = 0$ where Q_x and Q_y are the partial derivatives of Q with respect to x and y. Therefore, $dy/dx = -Q_x/Q_y = -(2b^2x)/(2a^2y) = -(b^2x)/(a^2y)$. The iteration on x continues as long as $-(b^2x)/(a^2y) \geq -1$. The termination condition of the iteration using only integer arithmetic is $b^2x \leq a^2y$.

Code for the iteration is

```
int a2 = a*a, b2 = b*b, fa2 = 4*a2;
int x, y, sigma;

for (x = 0, y = b, sigma = 2*b2+a2*(1-2*b); b2*x <= a2*y; x++)
{
    DrawPixel(xc+x,yc+y);
    DrawPixel(xc-x,yc+y);
    DrawPixel(xc+x,yc-y);
    DrawPixel(xc-x,yc-y);

    if ( sigma >= 0 )
    {
        sigma += fa2*(1-y);
        y--;
    }
    sigma += b2*(4*x+6);
}
```

The code for the other half of the arc in the first quadrant is symmetric in x and y and in a and b:

```
int a2 = a*a, b2 = b*b, fb2 = 4*b2;
int x, y, sigma;

for (x = a, y = 0, sigma = 2*a2+b2*(1-2*a); a2*y <= b2*x; y++)
{
    DrawPixel(xc+x,yc+y);
    DrawPixel(xc-x,yc+y);
    DrawPixel(xc+x,yc-y);
    DrawPixel(xc-x,yc-y);

    if ( sigma >= 0 )
    {
        sigma += fb2*(1-x);
        x--;
    }
    sigma += a2*(4*y+6);
}
```

3 General Ellipses

We could attempt to mimic the case of axis–aligned ellipses by drawing the arc from (x_b, y_b) to (x_a, y_a) and reflecting each pixel (x, y) through the appropriate lines. For example, given pixel $\vec{u} = (x, y)$, we would also

draw the pixel reflected through $\vec{v} = (x_b, y_b)$ given by

$$(x', y') = \vec{u} - 2\left(\frac{\vec{u}\cdot\vec{v}}{\vec{v}\cdot\vec{v}}\right)\vec{v} = (x, y) - 2\left(\frac{x_b x + y_b y}{x_b^2 + y_b^2}\right)(x_b, y_b).$$

The right–hand side requires a division. Moreover, even if the division is performed (whether as float or integer), the resulting pixels are not always contiguous and noticeable gaps occur. The general orientation of the ellipse requires a better method for selecting the pixels. Instead I generate the arc from $(-x_a, -y_a)$ to (x_a, y_a) and plot pixels $(x_c + x, y_c + y)$ and their reflections through the origin $(x_c - x, y_c - y)$.

The algorithm is divided into two cases.

1. Slope at $(-x_a, -y_a)$ is larger than 1 in magnitude. Five subarcs are drawn.
   (a) Arc from $(-x_a, y_a)$ to a point (x_0, y_0) whose slope is infinite. For all points between, the ellipse has slope larger than 1 in magnitude, so y is always incremented at each step.
   (b) Arc from (x_0, y_0) to a point (x_1, y_1) whose slope is 1. For all points between, the ellipse has slope larger than 1 in magnitude, so y is always incremented at each step.
   (c) Arc from (x_1, y_1) to a point (x_2, y_2) whose slope is 0. For all points between, the ellipse has slope less than 1 in magnitude, so x is always incremented at each step.
   (d) Arc from (x_2, y_2) to a point (x_3, y_3) whose slope is -1. For all points between, the ellipse has slope less than 1 in magnitude, so x is always incremented at each step.
   (e) Arc from (x_3, y_3) to (x_a, y_a). For all points between, the ellipse has slope larger than 1 in magnitude, so y is always decremented at each step.
2. Slope at $(-x_a, -y_a)$ is smaller than 1 in magnitude. Five subarcs are drawn.
   (a) Arc from $(-x_a, -y_a)$ to a point (x_0, y_0) whose slope is -1. For all points between, the ellipse has slope less than 1 in magnitude, so x is always decremented.
   (b) Arc from (x_0, y_0) to a point (x_1, y_1) whose slope is infinite. For all points between, the ellipse has slope larger than 1, so y is always incremented.
   (c) Arc from (x_1, y_1) to a point (x_2, y_2) whose slope is 1. For all points between, the ellipse has slope larger than 1 in magnitude, so y is always incremented at each step.
   (d) Arc from (x_2, y_2) to a point (x_3, y_3) whose slope is 0. For all points between, the ellipse has slope less than 1 in magnitude, so x is always incremented at each step.
   (e) Arc from (x_3, y_3) to (x_a, y_a). For all points between, the ellipse has slope less than 1 in magnitude, so x is always incremented at each step.

Each subarc is computed using a decision variable as in the case of an axis–aligned ellipse. The decision to switch between the three subarcs is based on slope of the ellipse. The ellipse is implicitly defined by $Q(x, y) = 0$ where $Q(x, y) = Ax^2 + 2Bxy + Cy^2 - D = 0$. The derivative $dy/dx = -(Ax + By)/(Bx + Cy)$ is obtained by implicit differentiation. The numerator and denominator of the derivative can be maintained incrementally. Initially the current pixel $(x, y) = (-x_a, -y_a)$ and the numerator and denominator of the slope are $dy = Ax_a + By_a$ and $dx = -(Bx_a + Cy_a)$.

The decision variable σ is handled slightly differently than in the case of an axis–aligned ellipse. In the latter case, the decision was made to use the pixel whose own level curve is closest to the zero level curve. In the

current case, a general ellipse handled in the same way can lead to gaps at the end points of the arc and the reflected arc. To avoid the gaps, the decision is made to always select the ellipse with *smallest positive* level curve value rather than smallest magnitude level curve value. The selected pixels are always outside the true ellipse. I do not incrementally maintain the decision variable as it does not gain any cycles, although it is possible to maintain it so.

Each of the algorithms for the ten subarcs are similar in structure. I describe the case 1a here. The initial values are $x = -x_a$, $y = -y_a$, $dx = Bx_a + Cy_a$, and $dy = -(Ax_a + By_a)$. As y is incremented, eventually the left–most point in the x–direction is encountered where the slope of the ellipse is infinite. At each step the two pixels to test are $(x, y+1)$ and $(x-1, y+1)$. It is enough to test $\sigma = Ax^2 + 2Bx(y+1) + C(y+1)^2 - D < 0$ to see if $(x, y+1)$ is inside the true ellipse. If it is, then $(x-1, y+1)$ is the next pixel to draw. If $\sigma \geq 0$, then $(x, y+1)$ is outside the true ellipse and closer to it than $(x-1, y+1)$, so the next pixel is $(x, y+1)$. Code is given below.

```
while ( dx <= 0 )  // loop until point with infinite slope occurs
{
    DrawPixel(xc+x,yc+y);
    DrawPixel(xc-x,yc-y);
    y++;
    sigma = a*x*x+2*b*x*y+c*y*y-d;
    if ( sigma < 0 )
    {
        dx -= b;
        dy += a;
        x--;
    }
    dx += c;
    dy -= b;
}
```

The other nine cases are structured similarly.