Circle Algorithms

Emily Gunawan

August 9, 2004

1 Euler's Method

Simple harmonic motion is derived from the following differential equation,

$$x''(t) + x(t) = 0, \tag{1}$$

which can also be written as $x''(t) = -x(t)$.
This derives

$$\begin{aligned} y(t) &= x'(t) \quad (2)\\ y'(t) &= x''(t) = -x(t), \end{aligned}$$

and rewritten as

$$\begin{aligned} x'(t) &= y\\ y'(t) &= -x'(t). \end{aligned}$$

As explained in Math 198 class notes, in order to derive a circle algorithm using Euler's method, first switch to Leibniz notation,

$$x'(t) = \frac{dx}{dt}.$$

Second,

$$\begin{aligned} \frac{dx}{dt} &= y & &\Longrightarrow dx = ydt\\ \frac{dy}{dt} &= -x & & dy = -xdt \end{aligned}$$

Third, replace dt by a finite, small number δ, which makes $dx = y\delta$ and $dy = -x\delta$.

Fourth, replace dx and dy by a finite displacement,

$$dx = x_{\text{next}} - x.$$

By substitution, get a set of difference equations $x_{next} - x = y\delta$ and $y_{next} - y = -x\delta$, which can be written as

$$x_{next} = x + y\delta \tag{3}$$

$$y_{\text{next}} = y - x\delta$$

Translated into Python code, it should look something like
h=.015
x=1 ; y=0
for i in range(2 $*$ π/h):
xn=x+y$*$h
y=−x$*$h+y
x=xn
draw(x,y)
Blinn explains that, for each iteration, (3) is calculated, which process can be written as the following product,

$$\begin{bmatrix} x_{\text{next}} \\ y_{\text{next}} \end{bmatrix} = \begin{bmatrix} 1 & -h \\ h & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

The determinant of the matrix $\left[\begin{smallmatrix} 1 & -h \\ h & 1 \end{smallmatrix}\right]$ equals $1 + h^2$. It means that $[x, y]$ is magnified by this amount each time through the loop, which creates a net spiraling effect. Each iteration carries $[x, y]$ to another circular path of greater radius than the previous one. In conclusion, no matter how small δ is made to be, Euler's method does not give satisfactory results.

2 Gauss-Seidel's Method

Modify the difference equations (3) by using the new x value over the current x value to evaluate y_{next}. The result is the following difference equation,

$$\begin{aligned} x_{\text{next}} &= x + y\delta \\ y_{\text{next}} &= -x_{next}\delta + y, \end{aligned} \tag{4}$$

which is equivalent to

$$\begin{aligned} x_{\text{next}} &= x + y\delta \\ y_{\text{next}} &= -(x + y\delta)\delta + y \\ &= -x\delta - y\delta^2 + y \\ &= -x\delta + y(1 - \delta^2). \end{aligned}$$

Hence,

$$\begin{bmatrix} x_{\text{next}} \\ y_{\text{next}} \end{bmatrix} = \begin{bmatrix} 1 & -\delta \\ \delta & 1 - \delta^2 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

The determinant of the above matrix is 1, and this algorithm creates an ellipse stretched in the northeast-southwest and squeezed in the northwest-southeast direction that resembles a circle more than the Euler's method does. According to Blinn, the maximum radius error is approximately $\delta/4$. Prove if this is so.

3 Rational Polynomials

If

$$x = \frac{1-t^2}{1+t^2} \tag{5}$$

and

$$y = \frac{2t}{1+t^2}, \tag{6}$$

then

$$\begin{aligned} x^2 + y^2 &= (\frac{1-t^2}{1+t^2})^2 + (\frac{2t}{1+t^2})^2 \\ &= \frac{1-2t^2+t^4+4t^2}{(1+t^2)^2} \\ &= \frac{(1+t^2)^2}{(1+t^2)^2} \\ &= 1 \end{aligned}$$

4 Bresenham Circle

The expression $r^2 = x^2+y^2$ represents the equation that parameterize a circle with radius r. Blinn's Bresenham's circle algorithm utilizes what he calls an " error " function. If $P(x, y)$ represents the current pixel on the graph, then let the error be how far$P$ is from the correct point on the circle. Hence,

$$E_{\text{now}} = r^2 - x^2 - y^2. \tag{7}$$

If $E_{\text{now}} \geq 0$, which means that the pixel is either on or inside the circle, a pixel is moved one pixel to the right by

$$x_{\text{next}} = x + 1,$$

which makes

$$\begin{aligned} E_{\text{next}} &= r^2 - x_{\text{next}}^2 - y^2 \\ &= r^2 - (x+1)^2 - y^2 \\ &= E_{\text{now}} - 2x - 1. \end{aligned}$$

On the other hand, if $E_{now} < 0$, the pixel is outside the circle and needs to be moved diagonally by

$$x_{\text{next}} = x + 1 and y_{\text{next}} = y - 1,$$

which makes

$$\begin{aligned} E_{\text{next}} &= r^2 - x_{\text{next}}^2 - y_{\text{next}}^2 \\ &= r^2 - (x+1)^2 - (y-1)^2 \\ &= E_{\text{now}} - 2x - 1 + 2y - 1. \end{aligned}$$

Starting from $(0, 1)$ and moving to the right or right diagonal, the following conditions will always be true during each iteration,

$$x <= y; x >= 0; y > 0.$$

Because of these, a step to the right decreases E and a diagonal step increases E. Blinn decribes an algorithm that keeps track of the current sign of E and, at each iteration, drive the next pixel toward a direction that gets the next E closer to its opposite sign. Written in Python, starting from $(0, 100)$ with radius of 100, the code looks something like x=0; y=100
E=0
while x¡= y:
(tab) if E¡0:
(tab tab) E=E+y+y-1
(tab tab) y=y-1
(tab) E=E-x-x-1
(tab) x=x+1
pixset(x,y)

5 REFERENCES

J.Blinn, Jim Blinn's Corner: A Trip Down the Graphics Pipeline chapter 1 (1996).
J.K.Francis, Math 198: Hypergraphics Lab and Class Notes (2003).
J. Kennedy, blah