Approximating an Ellipse by Circular Arcs

David Eberly
Geometric Tools, LLC
http://www.geometrictools.com/

Created: January 13, 2002
Last Modified: February 12, 2008

Contents

1 Algorithm

The ellipses to be approximated are axis-aligned and centered at the origin,

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1 \tag{1}$$

where $a \neq b$. Only the portion of the ellipse in the first quadrant will be approximated by circular arcs. The approximations in the other quadrants can be obtained by reflections of those in the first quadrant.

The first part of the process involves selecting a set of ellipse points $\mathbf{P}_i = (x_i, y_i)$, $0 \leq i \leq n$, where $\mathbf{P}_0 = (a, 0)$, $\mathbf{P}_n = (0, b)$, and the points are counterclockwise ordered in the first quadrant. The selection is based on weighted averages of curvatures. Given a parameterized curve $(x(t), y(t))$, the curvature is

$$K(t) = \frac{x'(t)y''(t) - y'(t)x''(t)}{((x'(t))^2 + (y'(t))^2)^{3/2}}.$$

The ellipse is parameterized by $x(t) = a\cos t$ and $b\sin t$ for $t \in [0, 2\pi)$. The derivatives are $x'(t) = -a\sin t$, $y'(t) = b\cos t$, $x''(t) = -a\cos t$, and $y''(t) = -b\sin t$. The curvature is $K(t) = ab/(a^2\sin^2 t + b^2\cos^2 t)^{3/2}$. As a function of (x, y), the curvature is

$$K(x, y) = \frac{ab}{\left(\left(\frac{bx}{a}\right)^2 + \left(\frac{ay}{b}\right)^2\right)^{3/2}}, \tag{2}$$

where it is understood that (x, y) additionally satisfies equation (1). If the curvature K is specified, the corresponding (x, y) is obtained by solving equations (1) and (2). Setting $\lambda = (ab/K)^{2/3}$, the solution is

$$x = a\sqrt{\left|\frac{\lambda - a^2}{b^2 - a^2}\right|}, \quad y = b\sqrt{\left|\frac{\lambda - b^2}{a^2 - b^2}\right|}.$$

Define $K_0 = K(a, 0) = a/b^2$ and $K_n = K(0, b) = b/a^2$. The point $\mathbf{P}_i$ for $1 \leq i \leq n-1$ is selected so that the curvature is $K_i = (1 - i/n)K_0 + (i/n)K_n$.

The second part of the process involves selecting a circular arc with center $\mathbf{C}_i$ and radius r_i, $0 \leq i < n$, whose end points are $\mathbf{P}_i$ and $\mathbf{P}_{i+1}$. In order to obtain a vertical tangent at $(a, 0)$, the circle $(\mathbf{C}_0, r_0)$ is chosen that contains $\mathbf{P}_0$, $\mathbf{P}_1$, and $(x_1, -y_1)$ where the last point is the reflection of $\mathbf{P}_1$ through the x-axis. Similarly, in order to obtain a horizontal tangent at $(0, b)$, the circle $(\mathbf{C}_{n-1}, r_{n-1})$ is chosen that contains $\mathbf{P}_{n-1}$, $\mathbf{P}_n$, and $(-x_{n-1}, y_{n-1})$ where the last point is the reflection of $\mathbf{P}_{n-1}$ through the y-axis. The other circles $(\mathbf{C}_i, r_i)$ for $1 \leq i \leq n-2$ are chosen to contain $\mathbf{P}_{i-1}$, $\mathbf{P}_i$, and $\mathbf{P}_{i+1}$.

2 Implementation

The source code for the approximation is in files `Wm4ApprEllipseByArcs2.h` and `Wm4ApprEllipseByArcs2.cpp`. A test program is shown next.

```
#include "Wm4ApprEllipseByArcs2.h"
#include "Wm4Images.h"
```

```
#include "Wm4Matrix2.h"
#include "Wm4RasterDrawing.h"
using namespace Wm4;

static int gs_iB0 = 256, gs_iB1 = 256;
static int gs_iB0M = gs_iB0-1, gs_iB1M = gs_iB1-1;
static ImageRGB82D gs_kImage(gs_iB0,gs_iB0);
static int gs_iColor = 0;

//----------------------------------------------------------------------------
static void SetPixel (int iX, int iY)
{
    if (0 <= iX && iX < gs_kImage.GetBound(0)
    &&  0 <= iY && iY < gs_kImage.GetBound(1))
    {
        gs_kImage(iX,iY) = gs_iColor;
    }
}
//----------------------------------------------------------------------------
int main ()
{
    float fA = 2.0f, fB = 1.0f;
    int iNumArcs = 8;
    Vector2f* akPoint = new Vector2f[iNumArcs+1];
    Vector2f* akCenter = new Vector2f[iNumArcs];
    float* afRadius = new float[iNumArcs];

    ApproximateEllipseByArcs(fA,fB,iNumArcs,akPoint,akCenter,afRadius);

    // Draw approximate ellipse to see how good it looks.
    gs_kImage = 0x00FFFFFF;
    gs_iColor = 0x00FF0000;
    float fMax = 1.1f*fA, fMin = -fMax, fRange = fMax-fMin;
    int i;
    for (i = 0; i < iNumArcs; i++)
    {
        // Inefficiently draw circle just to see the results.
        Vector2f kP0 = akPoint[i] - akCenter[i];
        float fR0 = kP0.Length();
        Vector2 kP1 = akPoint[i+1] - akCenter[i];
        float fR1 = kP1.Length();
        float fAngle = Mathf::ACos(kP0.Dot(kP1)/(afRadius[i]*afRadius[i]));
        int jMax = 256;
        for (int j = 0; j <= jMax; j++)
        {
            Matrix2f kRot;
            kRot.FromAngle(j*fAngle/jMax);
            Vector2f kPm = akCenter[i] + kRot*kP0;

            int iX = int(gs_iB0M*(kPm.x-fMin)/fRange);
            int iY = int(gs_iB1M*(kPm.y-fMin)/fRange);
            SetPixel(iX,iY);
            SetPixel(iX,gs_iB1M-iY);
            SetPixel(gs_iB0M-iX,gs_iB1M-iY);
            SetPixel(gs_iB0M-iX,iY);
        }
    }

    // Plot the true ellipse.
    gs_iColor = 0x000000FF;
    Ellipse2D(128,128,128/1.1f,64/1.1f,SetPixel);

    // Plot the sample points
    gs_iColor = 0x00000000;
    for (i = 0; i <= iNumArcs; i++)
    {
        Vector2f kPm = akPoint[i];
        int iX = int((gs_kImage.GetBound(0)-1)*(kPm.x-fMin)/fRange);
        int iY = int((gs_kImage.GetBound(1)-1)*(kPm.y-fMin)/fRange);
        for (int iDY = -1; iDY <= 1; iDY++)
        {
            for (int iDX = -1; iDX <= 1; iDX++)
            {
                SetPixel(iX+iDX,iY+iDY);
                SetPixel(iX+iDX,gs_iB1M-iY+iDY);
                SetPixel(gs_iB0M-iX+iDX,gs_iB1M-iY+iDY);
                SetPixel(gs_iB0M-iX+iDX,iY+iDY);
            }
        }
    }

    gs_kImage.Save("appr.im");

    delete[] akPoint;
    delete[] akCenter;
    delete[] afRadius;

    return 0;
}
//----------------------------------------------------------------------------
```

The variable `iNumArcs` was set to 2, 4, and 8. The approximation by arcs is drawn in blue and the true ellipse, drawn with a Bresenham-style algorithm, is drawn in red. The arc end points are drawn as black dots. Figure 2.1 shows the drawings, 2 arcs in the first quadrant in the top image, 4 arcs for the middle image, and 8 arcs for the bottom image.

Figure 2.1 Top: A 2-arc approximation. Middle: A 4-arc approximation. Bottom: An 8-arc approximation.