Approximating an Ellipse by Circular Arcs

David Eberly, Geometric Tools, Redmond WA 98052
https://www.geometrictools.com/

Created: January 13, 2002
Last Modified: April 24, 2016

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

Pseudocode for an implementation is shown in Listing 1.

Listing 1. An implementation of the algorithm for approximating an axis-aligned ellipse by a sequence of circular arcs. The number of arcs must be 2 or more and $a \neq b$ is required for the ellipse (the ellipse is not a circle). The number of elements for points is numArcs + 1. The number of elements for centers and radii is numArcs.

```
void ApproximateEllipseByArcs(Real a, Real b, int numArcs,
    Vector2 points[], Vector2 centers[], Real radii[])
{
    // Compute intermediate ellipse quantities.
    Real a2 = a * a, b2 = b * b, ab = a * b;
    Real invB2mA2 = 1 / (b2 - a2);  // denominator not zero because a and b differ

    // Compute the endpoints of the ellipse in the first quadrant.
    points[0] = { a, 0 };
    points[numArcs] = { 0, b };

    // Compute the curvature at the endpoints.
    Real curv0 = a / b2;
    Real curv1 = b / a2;

    // Select the ellipse points based on curvature properties.
    Real invNumArcs = 1 / numArcs;
    for (int i = 1; i < numArcs; ++i)
    {
        // The curvature at a new point is a weighted average of curvature at the endpoints.
        Real weight1 = i * invNumArcs;
        Real weight0 = 1 - weight1;
        Real curv = weight0 * curv0 + weight1 * curv1;

        // Compute point having this curvature.
        Real tmp = pow(ab / curv, 2 / 3);
        points[i][0] = a * sqrt(fabs((tmp - a2) * invB2mA2));
        points[i][1] = b * sqrt(fabs((tmp - b2) * invB2mA2));
    }

    // Compute the arc at (a,0).
    Vector2 p0 = points[0];
    Vector2 p1 = points[1];
    Vector2 reflectionP1 = { p1[0], -p1[1] };
    Circumscribe(reflectionP1, p0, p1, centers[0], radii[0]);

    // Compute arc at (0,b).
    Vector2 pNm1 = points[numArcs - 1];
    Vector2 pN = points[numArcs];
    Vector2 reflectionPNm1 = { -pNm1[0], pNm1[1] };
    Circumscribe(reflectionPNm1, pN, pNm1, centers[numArcs - 1], radii[numArcs - 1]);

    // Compute arcs at intermediate points between (a,0) and (0,b).
    for (int iM = 0, i = 1, iP = 2; i < numArcs - 1; ++iM, ++i, ++iP)
    {
        Circumscribe(points[iM], points[i], points[iP], centers[i], radii[i]);
    }
}
```

Figure 1 shows the approximations for 2, 4, and 8 arcs. The approximation by arcs is drawn in blue and the true ellipse, using a Bresenham-style algorithm, is drawn in red. The arc endpoints are drawn as black dots.

Figure 1. Top: A 2-arc approximation. Middle: A 4-arc approximation. Bottom: An 8-arc approximation.