Drawing an elliptical arc using polylines, quadratic or cubic Bézier curves

L. Maisonobe[*]

July 21, 2003

Contents

1 Introduction **2**

2 Definitions **3**
2.1 parametric curves 3
2.1.1 tangent 3
2.1.2 curvature 3
2.2 ellipse 4
2.2.1 parametric equation 5
2.3 Bézier curves 5
2.3.1 linear Bézier curves 6
2.3.2 quadratic Bézier curves 6
2.3.3 cubic Bézier curves 7

3 Approximation **7**
3.1 principles 7
3.2 linear Bézier curve 8
3.2.1 control points choice 8
3.2.2 error estimation 8
3.3 quadratic Bézier curve 10
3.3.1 control points choice 10
3.3.2 error estimation 11
3.4 cubic Bézier curve 15
3.4.1 control points choice 15
3.4.2 error estimation 18

4 Implementation **21**

[*]luc@spaceroots.org

1 Introduction

When dealing with three dimensional geometry, one often needs to draw some sketch with circles or arcs seen from an arbitrary point of view. They appear as ellipses and arcs of ellipses when displayed on a two dimensions plane such as paper or screen. The classical graphical languages or libraries provide primitives for drawing several objects like rectangles or circles. Lot of them can also handle ellipses, but only when their axes are aligned with the device axes, and sometimes only for complete ellipses (with the notable exception of SVG which has two commands, `A` and `a`, to add a general ellipse arc to the current path)

Hence, drawing these objects involve dealing with low level graphics. An algorithm based on Bresenham's principles has been published in the Foley and Van Dam book, it has been implemented by Andrew W. Fitzgibbon in C++ and is available with the PC Games Programmers Encyclopedia library (pcgpe) version 1.0[1], in the `conic.cc` file. The same file can be viewed online[2]. Unfortunately, one needs both good starting and ending points, i.e. exactly the same pixels the algorithm would have chosen by itself. Failing to do this result in strange spiral-like shapes. Another problem with such an algorithm is that it is at pixel level only and do not handle dashes or line width.

Another method is to use intermediate level objects like polylines or Bézier curves to approximate the ellipse. In this paper, we will describe how this can be done, depending on the available primitives and for any user defined accuracy.

Using these intermediate level curves has several advantages. The first one is that the user can often use his own coordinate system and use floating point numbers, he does not consider pixels at all. Another advantage is that the graphical packages handles high level features with such objects, like filling closed shapes, drawing with various pens (both pen shape and pen size can be customized) and drawing with various line styles (continuous lines, dashed lines with several dash patterns). Since each object can contain a lot of individual pixels, the description of the elliptical arc is also much shorter than specifying each pixel individually. Of course, for a given accuracy, polylines are less efficient than cubic Bézier curves for example, so the description size will depend on the available features of the underlying graphical package.

Three cases will be considered: lines (all graphical packages can handle them), quadratic Bézier curves (available for example in LaTeX 2_ε) and cubic Bézier curves (available in LaTeX 2_ε when the `bez123` extension is loaded, METAFONT, PDF, PostScript, Java API, SVG ...).

The last version of this document is always available in the spaceroots downloads page `http://www.spaceroots.org/downloads.html`. It can be browsed on-line or retrieved as a PDF, compressed PostScript or LaTeX source file.

[1]`ftp://x2ftp.oulu.fi/pub/msdos/programming/gpe/pcgpe10.zip`
[2]`http://www.flashdaddee.com/Books-Technical/win/Conics.htm`

2 Definitions

2.1 parametric curves

A parametric curve is a curve which is defined by a two dimensional equation P of one parameter t. The two coordinates of the vector $P(t)$ are the x and y coordinates of the point of the curve corresponding to a particular value of the parameter.

One curve can be defined by several different parametric equations like P_1 and P_2. This means that for each t_1 in the range of the first equation, another value t_2 in the range of the second equation can be found such that $P_1(t_1) = P_2(t_2)$. The relationship between t_1 and t_2 can be either simple or very complex depending on the equations. The f function that transforms t_1 into t_2, $t_2 = f(t_1)$, is monotonic. If t_2 increases when t_1 increases, the two equations define the same orientation for the curve, otherwise they define opposite orientations.

2.1.1 tangent

The unit tangent vector can be computed from the first derivative of the parametric equation: $\vec{T} = P'(t)/||P'(t)||$, which means its coordinates are:

$$\vec{T}\begin{cases}\dfrac{x'}{\sqrt{x'^2+y'^2}}\\[2ex]\dfrac{y'}{\sqrt{x'^2+y'^2}}\end{cases}$$

This unit tangent vector is an intrinsic property of the curve, it is independant of the parametric equations used as long as they define the same orientation. If two different orientations are used, they will define opposite unit tangent vectors.

2.1.2 curvature

The curvature of a curve is the inverse of the curvature radius. It is null for a straight line (infinite curvature radius). For a parametric curve, it can be computed from the first and second derivatives of the defining equation:

$$C = \frac{|x'y'' - y'x''|}{(x'^2+y'^2)^{3/2}}$$

In fact, we will be more interested in the signed curvature, which we define as being positive when the curve turns left and negative when it turns right when following the curve in the direction of parameter increase:

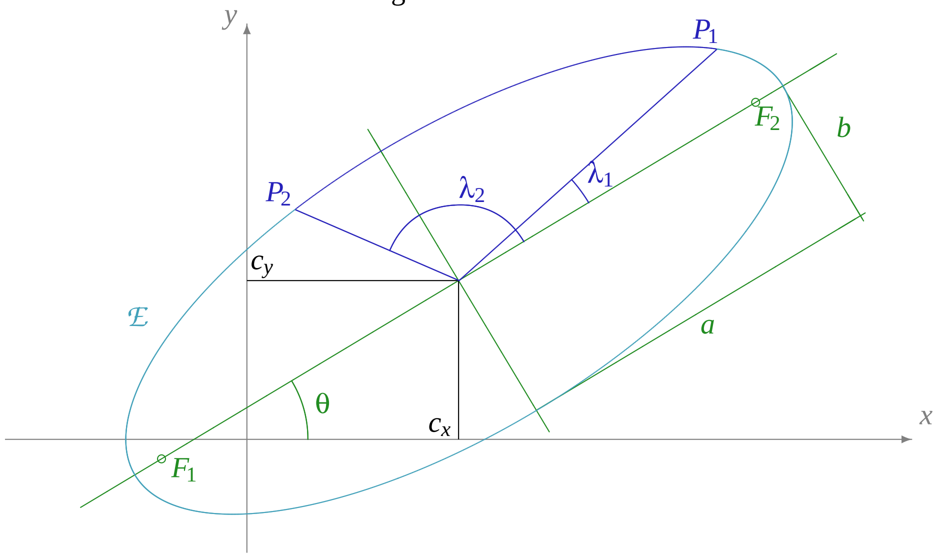
$$\widetilde{C} = \frac{x'y'' - y'x''}{(x'^2+y'^2)^{3/2}} \tag{1}$$

The signed curvature is also an intrinsic geometrical property of the curve at the point considered, it is independant of the parametric equations used as long as they define the same orientation. If two different orientations are used, they will define opposite signed curvatures.

2.2 ellipse

First, lets introduce some notations. Figure 1 shows an arc and the ellipse it belongs to. The ellipse $\mathcal{E}$ is defined by its center (c_x, c_y), its semi-major axis (a), its semi-minor axis (b) and its orientation (θ). The arc is defined by its start and end angles (λ_1 and λ_2, assuming $\lambda_1 < \lambda_2 \leq \lambda_1 + 2\pi$). If $a = b$, then the ellipse is a circle and the θ direction is irrelevant.

Figure 1: notations



The two points F_1 and F_2 are the focii of the ellipse. The distance between these points and the center of the ellipse is $\sqrt{a^2 - b^2}$. This means that with our notation, the coordinates of these points are:

$$F_1 \begin{cases} c_x - \sqrt{a^2 - b^2}\cos\theta \\ c_y - \sqrt{a^2 - b^2}\sin\theta \end{cases} \qquad F_2 \begin{cases} c_x + \sqrt{a^2 - b^2}\cos\theta \\ c_y + \sqrt{a^2 - b^2}\sin\theta \end{cases}$$

If the ellipse is a circle, then the two points F_1 and F_2 are both at the center. These points have the following interesting property:

$$d(P, F_1) + d(P, F_2) = 2a \quad \forall P \in \mathcal{E} \tag{2}$$

This property is one classical way to define the ellipse.

2.2.1 parametric equation

The elliptical arc can be seen as a parametric curve, each point is defined as an explicit vectorial function of one parameter $E(\eta)$ giving the two coordinates x and y of the point. We will use as a parameter the η angle computed such that:

$$\begin{cases} r\cos\lambda = a\cos\eta \\ r\sin\lambda = b\sin\eta \end{cases}$$
$$\Rightarrow \begin{cases} \eta = \arctan_2\left(\dfrac{\sin\lambda}{b}, \dfrac{\cos\lambda}{a}\right) \\ \lambda = \arctan_2(b\sin\eta, a\cos\eta) \end{cases}$$

where λ is the geometrical angle considered between one end of the semi-major axis and the current point, counted from the center of the ellipse (see λ_1 and λ_2 in the figure). This η angle is a theoretical angle, it has no representation on the preceding figure (except if the ellipse is really a circle, of course).

Using this angle, the ellipse parametric equation is:

$$E(\eta) \begin{cases} c_x + a\cos\theta\cos\eta - b\sin\theta\sin\eta \\ c_y + a\sin\theta\cos\eta + b\cos\theta\sin\eta \end{cases} \tag{3}$$

The derivatives of the parametric curve are:

$$E'(\eta) \begin{cases} -a\cos\theta\sin\eta - b\sin\theta\cos\eta \\ -a\sin\theta\sin\eta + b\cos\theta\cos\eta \end{cases} \tag{4}$$

and

$$E''(\eta) \begin{cases} -a\cos\theta\cos\eta + b\sin\theta\sin\eta \\ -a\sin\theta\cos\eta - b\cos\theta\sin\eta \end{cases} \tag{5}$$

2.3 Bézier curves

Bézier curves are parametric polynomial[3] curves that are widely used in graphical packages. They are often used to approximate another curve, the match being perfect at both endpoints. In order to match position, slope and curvature, a third degree polynomial is needed, these are the classical Bézier curves or cubic Bézier curves. If only position and slope need to match, a second degree polynomial is sufficient, the corresponding curves are quadratic Bézier curves. As an extension,

[3]it is also possible to define rational Bézier curves, however they are not widely supported by graphical packages, so they will not be discussed here, despite they allow to represent *exactly* elliptical arcs

we will also consider line segments to be linear Bézier curves defined by a first degree polynomial and matching only position of endpoints.

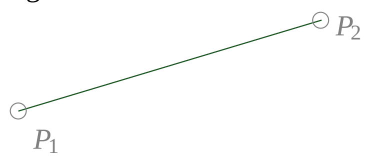
The polynomials underlying Bézier curves are very simple when their constants are defined in terms of control points. These polynomials are called Bernshteĭn polynomials. The range of the t parameter is between 0 and 1. When $t = 0$, we are at the start point of the Bézier curve which should match the start point of the approximated curve. When $t = 1$, we are at the end point of the Bézier curve which should match the end point of the approximated curve. All graphical packages that support Bézier curves define them by the control points only, the Bernshteĭn polynomials are used internally and are not available at user level.

2.3.1 linear Bézier curves

For linear Bézier curves, there are only two control points which are the endpoints of the curve, P_1 and P_2. The Bernshteĭn polynomial and its first derivatives are:

$$\begin{cases} B_1(t) = (1-t)P_1 + tP_2 \\ B_1'(t) = P_2 - P_1 \\ B_1''(t) = 0 \end{cases} \tag{6}$$

Figure 2: linear Bézier curve

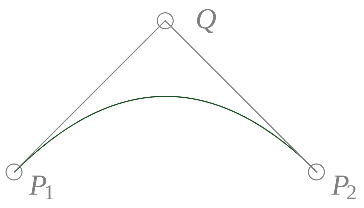


2.3.2 quadratic Bézier curves

For quadratic Bézier curves, there are three control points. The first two control points are the two endpoints of the curve, P_1 and P_2. The last control point is an intermediate point Q which controls the direction of the tangents of the curve at both ends. This point is generally away from the curve itself. The Bernshteĭn polynomial and its first derivatives are:

$$\begin{cases} B_2(t) = (1-t)^2P_1 + 2t(1-t)Q + t^2P_2 \\ B_2'(t) = -2\left[(1-t)P_1 - (1-2t)Q - tP_2\right] \\ B_2''(t) = 2\left[P_1 - 2Q + P_2\right] \end{cases} \tag{7}$$

Figure 3: quadratic Bézier curve



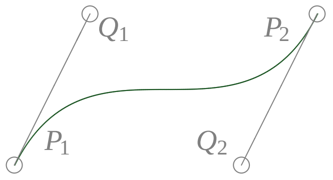
2.3.3 cubic Bézier curves

Cubic Bézier curves are the *classical* Bézier curves, the lower degree curves presented before should be seen as extensions of classical Bézier curves to low degrees.

For cubic Bézier curves, there are four control points. The first two are the two endpoints of the curve, P_1 and P_2. The two remaining ones are intermediate points Q_1 and Q_2. These intermediate points control the tangent and the curvature at both ends. The Bernshteĭn polynomial and its first derivatives are:

$$\begin{cases} B_3(t) = (1-t)^3 P_1 + 3t(1-t)^2 Q_1 + 3t^2(1-t) Q_2 + t^3 P_2 \\ B_3'(t) = -3\left[(1-t)^2 P_1 - (1-t)(1-3t) Q_1 - t(2-3t) Q_2 - t^2 P_2\right] \\ B_3''(t) = 6\left[(1-t) P_1 - (2-3t) Q_1 + (1-3t) Q_2 + t P_2\right] \end{cases} \tag{8}$$

Figure 4: cubic Bézier curve



3 Approximation

3.1 principles

The approximation principle is very simple: replace the real elliptical arc by a set of connected Bézier curves. The degree of the curves (linear, quadratic or cubic) is chosen according to the underlying graphical package and the number of curves and connecting points are chosen according to the required accuracy. Each curve will begin and end exactly on the elliptical arc.

We will describe algorithms to build Bézier curves matching an elliptical arc at both ends, and algorithms to compute quickly an upper bound of the approximation error for these curves. The approximation error estimation algorithms will allow to compute the error directly from the sub-arcs characteristics, without building the Bézier curve beforehand. In the linear case also, the exact error will be computed rather than an upper bound.

A simple iterative process can be used to build the appropriate curves set. Simply compute the error which would result from splitting the arc into 1, 2, 4 … 2^n equal lengths parts in terms of the curve parameter η, and stop the loop when this error drops below the user-defined threshold. Then, build the corresponding 2^n Bézier curves.

This process is not optimal in the sense that it can split the arc in more parts than strictly needed. However it is very simple to implement and very fast. Also since the error decreases a lot as the arc length decreases, very few iterations will be needed in practical cases.

3.2 linear Bézier curve

3.2.1 control points choice

Finding the two control points of the quadratic Bézier curve is trivial, since we want the Bézier curve endpoints to match the elliptical arc endpoints:

$$B_1(0) = E(\eta_1) \Rightarrow P_1 = E(\eta_1)$$
$$B_1(1) = E(\eta_2) \Rightarrow P_2 = E(\eta_2)$$

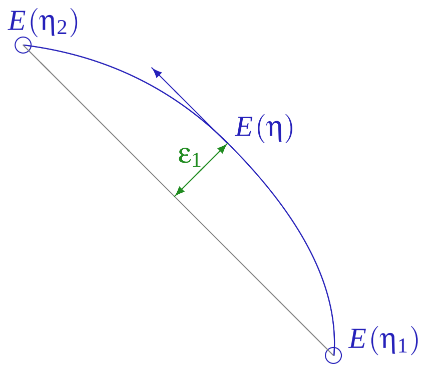
The control points P_1 and P_2 of a linear Bézier curve approximating an elliptical arc should be chosen as follows:

$$P_1 = E(\eta_1)$$
$$P_2 = E(\eta_2)$$

3.2.2 error estimation

Since the ellipse curve always has a positive curvature and since the linear Bézier curve is a straight line, the distance between the ellipse and its approximation is null at both arc ends and reaches its maximal value for one point $E(\eta)$ between the endpoints. At this point, the tangent to the ellipse is parallel to the line, as depicted in figure 5.

Figure 5: error of a linear approximation



Using the ellipse parametric equation (3) and its derivative (4), we can find η:

$$\begin{aligned}
&\det\left(E'(\eta), E(\eta_2) - E(\eta_1)\right) = 0\\
\Leftrightarrow &x'_\eta(y_2 - y_1) - y'_\eta(x_2 - x_1) = 0\\
\Leftrightarrow &-ab\left(\cos\eta(\cos\eta_2 - \cos\eta_1) + \sin\eta(\sin\eta_2 - \sin\eta_1)\right) = 0\\
\Leftrightarrow &\cos(\eta_2 - \eta) = \cos(\eta - \eta_1)\\
\Leftrightarrow &\eta_1 = \eta_2 \quad \text{or} \quad \eta = \frac{\eta_1 + \eta_2}{2}
\end{aligned}$$

where x'_η and y'_η are the coordinates of $E'(\eta)$ and x_i and y_i are the coordinates of $E(\eta_i)$.

Obviously, the second solution is the one we were looking for. Given this value of η, we compute the error ε as the distance between $E(\eta)$ (coordinates x_η and y_η) and the line passing through $E(\eta_1)$ and $E(\eta_2)$.

The error resulting from the approximation of an ellipse arc by a linear Bézier curve is:

$$\varepsilon_1 = \frac{|x_\eta(y_2 - y_1) - y_\eta(x_2 - x_1) + x_2 y_1 - x_1 y_2|}{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}}$$

where

$$\eta = \frac{\eta_1 + \eta_2}{2}$$

3.3 quadratic Bézier curve

3.3.1 control points choice

Since we want the Bézier curve endpoints to match the elliptical arc endpoints, finding the first two control points of the quadratic Bézier curve is done as for linear Bézier curves, :

$$\begin{aligned} B_2(0) = E(\eta_1) &\Rightarrow P_1 = E(\eta_1) \\ B_2(1) = E(\eta_2) &\Rightarrow P_2 = E(\eta_2) \end{aligned}$$

Matching of the Bézier curve and elliptical arc slopes at start point allows us to define the remaining control point using one scalar parameter:

$$B_2'(0) = kE'(\eta_1) \Rightarrow 2(Q - P_1) = kE'(\eta_1)$$

In order to have the same orientation for the elliptical arc and its approximation, k should be positive. Replacing k by $\alpha = k/2$ leads to the simple definition of Q:

$$Q = P_1 + \alpha E'(\eta_1) \qquad \alpha > 0$$

We find α by stating the Bézier curve slope should also match the arc slope at end point:

$$\begin{aligned} & B_2'(1) = \tilde{k}E'(\eta_2) \\ \Rightarrow & 2(P_2 - Q) = \tilde{k}E'(\eta_2) \\ \Rightarrow & \det\left(P_2 - P_1 - \alpha E'(\eta_1), E'(\eta_2)\right) = 0 \\ \Rightarrow & (x_2 - x_1 - \alpha x_1')y_2' - (y_2 - y_1 - \alpha y_1')x_2' = 0 \\ \Rightarrow & \alpha = \frac{(x_2 - x_1)y_2' - (y_2 - y_1)x_2'}{x_1'y_2' - y_1'x_2'} \\ \Rightarrow & \alpha = \frac{1 - \cos(\eta_2 - \eta_1)}{\sin(\eta_2 - \eta_1)} \\ \Rightarrow & \alpha = \tan\left(\frac{\eta_2 - \eta_1}{2}\right) \end{aligned}$$

where x_{η_i}' and y_{η_i}' are the coordinates of $E'(\eta_i)$. It is easy to verify that this also implies the symmetrical expression:

$$Q = P_2 - \tan\left(\frac{\eta_2 - \eta_1}{2}\right) E'(\eta_2)$$

If we have preprocessed our arcs to ensure $\eta 1 < \eta_2 \leq \eta_1 + \pi/2$, we know that $0 <= \alpha <= 1$. This implies that the orientation is right at both endpoints and also that α can be computed without loss of accuracy. Equations (3) and (4) show that both $E(\eta)$ and $E'(\eta)$ are bounded by a, so this preprocessing also ensures Q does not escape far away from points P_1 and P_2 (this could generate numerical overflows in some graphical packages).

The control points P_1, Q and P_2 of a quadratic Bézier curve approximating an elliptical arc should be chosen as follows:

$$\begin{aligned} P_1 &= E(\eta_1) \\ P_2 &= E(\eta_2) \\ Q &= P_1 + \tan\left(\frac{\eta_2 - \eta_1}{2}\right) E'(\eta_1) \end{aligned}$$

which is equivalent to

$$Q = P_2 - \tan\left(\frac{\eta_2 - \eta_1}{2}\right) E'(\eta_2)$$

3.3.2 error estimation

Since both the arc and the approximation can be described by parametric equations $B_2(t)$ and $E(\eta)$, we could try to find for each point $B_2(t)$ the closest point $E(\eta_t)$ such that $d(B_2(t), E(\eta_t)) = \min_\eta\{d(B_2(t), E(\eta))\}$ and to use this $t \leftrightarrow \eta_t$ mapping to find the maximal error along the elliptical arc $\varepsilon_2 = \max_t\{d(B_2(t), E(\eta_t))\}$. This method is accurate, but involves lots of computation. A more practical method consist in building once and for all an analytical model $\tilde{\varepsilon}_2(a, b, \eta_1, \eta_2)$ approximating ε_2 and use it.

We first compute the true error by embedding two equation solvers. The first solver is used to compute η_t for any given t (in other words it performs the mapping), using a specific algorithm described in another paper. The second solver is used to find the value of the t parameter which leads to the maximal error. It is necessary to use very robust algorithms for this purpose, because the notion of *closest point* is not continuous, so η_t seen as a function of t is not continuous. A trivial example is given by a point $P(t)$ traveling along the minor axis. When this point crosses the ellipse center, η_t jumps from $-\pi/2$ to $+\pi/2$. The *distance* to the closest point, however, is continuous, only its first derivative is discontinuous. These embedded solvers allow us to compute for ε_2 for any arc, its value depend on a, b, η_1 and $\eta 2$.

The next step involves identifying the behaviour of the error when the various parameters change. Using the semi major axis as a scaling parameter and plotting some curves, we first identify canonical parameters: the error model will be :

$$\tilde{\varepsilon}_2 = a \times f(b/a, \eta, \Delta\eta) \qquad (9)$$

where

$$\eta = \frac{\eta_1 + \eta_2}{2} \quad \text{and} \quad \Delta\eta = \eta_2 - \eta_1$$

For $\Delta\eta$ between $1/20$ and $\pi/2$, the error appears to be almost a power of $\Delta\eta$, so we refine our model:

$$f(b/a, \eta, \Delta\eta) = e^{c_0(b/a,\eta) + c_1(b/a,\eta)\Delta\eta} \tag{10}$$

This behaviour seems to be valid also outside of the given interval. However, if $\Delta\eta$ gets too small, some numerical problems prevent from computing accurately the very tiny real errors. This interval should be compliant with classical needs.

For symmetry reasons, it is sufficient to study the behaviour of the origin and slope functions c_0 and c_1 for η between 0 and $\pi/2$. The functions decrease from a maximum value at 0 to a minimum value at $\pi/2$. The peak near the maximum can be very sharp for flat ellipses ($b \ll a$). This is because the curvature is very important and changes rapidly near the semi major axis ends. We will use several functions $\cos(2k\eta)$ to represent this behaviour. Our model becomes:

$$c_i(b/a, \eta) = \sum_{j=0}^{j=3} r_{i,j}(b/a) \cos(2j\eta) \tag{11}$$

Finally, the coefficients functions $r_{k,i}$ present various asymptotic behaviours when b/a varies from 0 to 1. In order to get a good fit, we describe the functions as rational functions with a quadratic numerator and linear denominator:

$$r_{i,j}(b/a) = \frac{\mu_{i,j,0}\left(\frac{b}{a}\right)^2 + \mu_{i,j,1}\left(\frac{b}{a}\right) + \mu_{i,j,2}}{\left(\frac{b}{a}\right) + \mu_{i,j,3}} \tag{12}$$

In fact, two sets of $\mu_{i,j,k}$ coefficients were used, one set for b/a between 0 and $1/4$, and one set for b/a between $1/4$ and 1. The coefficients are given by tables 1 and 2.

These coefficients were computed by curve fitting means. This imply that for some points $(b/a, \eta, \Delta\eta)$ this model gives errors larger than the real ones, and at other points it gives smaller errors ($\varepsilon_{2,\min} \le \tilde{\varepsilon}_2 \le \varepsilon_{2,\max}$). Multiplying this least square model by a suitable safety rational fraction $s(b/a)$, we obtain an upper bound of the error: $\varepsilon_2 \le s(b/a)\tilde{\varepsilon}_2$.

The error resulting from the approximation of an ellipse arc by a quadratic Bézier curve is bounded as follows:

$$\varepsilon_2 \leq \left(\frac{0.02 \left(\frac{b}{a}\right)^2 + 2.83 \left(\frac{b}{a}\right) + 0.125}{\left(\frac{b}{a}\right) + 0.01} \right) a e^{c_0 + c_1(\eta_2 - \eta_1)}$$

where

$$c_i = \sum_{j=0}^{j=3} \frac{\mu_{i,j,0} \left(\frac{b}{a}\right)^2 + \mu_{i,j,1} \left(\frac{b}{a}\right) + \mu_{i,j,2}}{\left(\frac{b}{a}\right) + \mu_{i,j,3}} \cos\left(j(\eta_1 + \eta_2)\right)$$

and the $\mu_{i,j,k}$ coefficients are given by tables 1 and 2

This bound correctly overestimates the error, but is not too conservative. The cumulative distribution function of $\varepsilon_2/(s(b/a)\tilde{\varepsilon}_2)$ is quite smooth and the mean value of the ratio is 0.538.

Table 1: error coefficients for quadratic Bézier $(0 < b/a < 1/4)$

i	j	$\mu_{i,j,0}$	$\mu_{i,j,1}$	$\mu_{i,j,2}$	$\mu_{i,j,3}$
0	0	3.92478	-13.5822	-0.233377	0.0128206
0	1	-1.08814	0.859987	0.000362265	0.000229036
0	2	-0.942512	0.390456	0.0080909	0.00723895
0	3	-0.736228	0.20998	0.0129867	0.0103456
1	0	-0.395018	6.82464	0.0995293	0.0122198
1	1	-0.545608	0.0774863	0.0267327	0.0132482
1	2	0.0534754	-0.0884167	0.012595	0.0343396
1	3	0.209052	-0.0599987	-0.00723897	0.00789976

Table 2: error coefficients for quadratic Bézier $(1/4 \leq b/a \leq 1)$

i	j	$\mu_{i,j,0}$	$\mu_{i,j,1}$	$\mu_{i,j,2}$	$\mu_{i,j,3}$
0	0	0.0863805	-11.5595	-2.68765	0.181224
0	1	0.242856	-1.81073	1.56876	1.68544
0	2	0.233337	-0.455621	0.222856	0.403469
0	3	0.0612978	-0.104879	0.0446799	0.00867312
1	0	0.028973	6.68407	0.171472	0.0211706
1	1	0.0307674	-0.0517815	0.0216803	-0.0749348
1	2	-0.0471179	0.1288	-0.0781702	2.0
1	3	-0.0309683	0.0531557	-0.0227191	0.0434511

3.4 cubic Bézier curve

3.4.1 control points choice

Since we want the Bézier curve endpoints to match the elliptical arc endpoints, finding the first two control points of the cubic Bézier curve is done as for linear Bézier curves:

$$\begin{aligned} B_3(0) = E(\eta_1) \Rightarrow P_1 = E(\eta_1) \\ B_3(1) = E(\eta_2) \Rightarrow P_2 = E(\eta_2) \end{aligned}$$

Matching of the Bézier curve and elliptical arc slopes at endpoints allows us to define the two remaining control points using only two scalar parameters:

$$\begin{aligned} B_3'(0) = k_1 E'(\eta_1) \Rightarrow 3(Q_1 - P_1) = k_1 E'(\eta_1) \\ B_3'(1) = k_2 E'(\eta_2) \Rightarrow 3(P_2 - Q_2) = k_2 E'(\eta_2) \end{aligned}$$

In order to have the same orientation for the elliptical arc and its approximation, k_1 and k_2 should both be positive. Replacing the k_1 and k_2 scalars by $\alpha_1 = k_1/3$ and $\alpha_2 = k_2/3$ leads to the simple definition of Q_1 and Q_2:

$$\begin{aligned} Q_1 &= P_1 + \alpha_1 E'(\eta_1) \qquad & \alpha_1 > 0 \\ Q_2 &= P_2 - \alpha_2 E'(\eta_2) \qquad & \alpha_2 > 0 \end{aligned}$$

With these definitions of the control points, the first and second derivatives of the Bézier curves are:

$$\begin{aligned} B_3'(0) = 3(Q_1 - P_1) &= 3\alpha_1 E'(\eta_1) \\ B_3''(0) = 6[Q_2 - 2Q_1 + P_1] &= 6[E(\eta_2) - E(\eta_1) - 2\alpha_1 E'(\eta_1) - \alpha_2 E'(\eta_2)] \\ B_3'(1) = 3(P_2 - Q_2) &= 3\alpha_2 E'(\eta_2) \\ B_3''(1) = 6[P_2 - 2Q_2 + Q_1] &= -6[E(\eta_2) - E(\eta_1) - \alpha_1 E'(\eta_1) - 2\alpha_2 E'(\eta_2)] \end{aligned}$$

introducing these derivatives in equation (1) gives the signed curvatures at Bézier curve endpoints:

$$(13) \qquad \begin{cases} \widetilde{C}_B(0) = \dfrac{2[x_1'(y_2 - y_1) - y_1'(x_2 - x_1)] + 2\alpha_2[x_2' y_1' - x_1' y_2']}{3\alpha_1^2 ({x_1'}^2 + {y_1'}^2)^{3/2}} \\[2ex] \widetilde{C}_B(1) = \dfrac{-2[x_2'(y_2 - y_1) - y_2'(x_2 - x_1)] + 2\alpha_1[x_2' y_1' - x_1' y_2']}{3\alpha_2^2 ({x_2'}^2 + {y_2'}^2)^{3/2}} \end{cases}$$

The curvatures of the elliptical arc at endpoints are computed the same way, introducing the ellipse parametric equation derivatives (4) and (5) into equation (1):

$$(14)\qquad \begin{cases} \widetilde{C}_E(\eta_1) = \dfrac{x_1' y_1'' - y_1' x_1''}{({x_1'}^2 + {y_1'}^2)^{3/2}} \\ \widetilde{C}_E(\eta_2) = \dfrac{x_2' y_2'' - y_2' x_2''}{({x_2'}^2 + {y_2'}^2)^{3/2}} \end{cases}$$

where x''_{η_i} and y''_{η_i} are the coordinates of $E''(\eta_i)$.

In order to find the values of α_1 and α_2, we state that the signed curvature of the Bézier curve and elliptical arc at endpoints should be the same, so we combine systems (13) and (14):

$$\begin{cases} 2[x_1'(y_2-y_1) - y_1'(x_2-x_1)] + 2\alpha_2[x_2'y_1' - x_1'y_2'] - 3\alpha_1^2[x_1'y_1'' - y_1'x_1''] = 0 \\ -2[x_2'(y_2-y_1) - y_2'(x_2-x_1)] + 2\alpha_1[x_2'y_1' - x_1'y_2'] - 3\alpha_2^2[x_2'y_2'' - y_2'x_2''] = 0 \end{cases}$$

This system can be further simplified by using the ellipse parametric equation (3) and its derivatives (4) and (5) which imply the following relations:

$$\begin{cases} x_1'(y_2-y_1) - y_1'(x_2-x_1) = ab(1-\cos(\eta_2-\eta_1)) \\ x_2'y_1' - x_1'y_2' = -ab\sin(\eta_2-\eta_1) \\ x_1'y_1'' - y_1'x_1'' = ab \\ x_2'(y_2-y_1) - y_2'(x_2-x_1) = -ab(1-\cos(\eta_2-\eta_1)) \\ x_2'y_2'' - y_2'x_2'' = ab \end{cases}$$

So after simplifying by ab, the system becomes:

$$\begin{cases} 2(1-\cos(\eta_2-\eta_1)) - 2\alpha_2\sin(\eta_2-\eta_1) - 3\alpha_1^2 = 0 \\ 2(1-\cos(\eta_2-\eta_1)) - 2\alpha_1\sin(\eta_2-\eta_1) - 3\alpha_2^2 = 0 \end{cases}$$

Introducing $\tau = \tan\left(\frac{\eta_2-\eta_1}{2}\right)$ and multiplying by $1+\tau^2$ this can be written:

$$\begin{cases} 4\tau^2 - 4\tau\alpha_2 - 3(1+\tau^2)\alpha_1^2 = 0 \\ 4\tau^2 - 4\tau\alpha_1 - 3(1+\tau^2)\alpha_2^2 = 0 \end{cases}$$

Subtracting one equation from the other, we obtain:

$$(\alpha_2-\alpha_1)\big(3(1+\tau^2)(\alpha_1+\alpha_2) - 4\tau\big) = 0$$

This implies the system has two sets of solutions. Since we have preprocessed our arcs to ensure $\eta 1 < \eta_2 \leq \eta_1 + \pi/2$, we know that $0 <= \tau <= 1$, so τ can be computed without loss of accuracy and τ^2 can be extracted from square roots without sign change. The first set of solutions is:

$$\begin{cases} \alpha_2 = \alpha_1 \\ 3(1+\tau^2)\alpha_1^2 + 4\tau\alpha_1 - 4\tau^2 = 0 \end{cases}$$

$$\Rightarrow \begin{cases} \alpha_1 = \left(\dfrac{2\tau}{1+\tau^2}\right)\left(\dfrac{-1 \pm \sqrt{4+3\tau^2}}{3}\right) \\ \alpha_2 = \left(\dfrac{2\tau}{1+\tau^2}\right)\left(\dfrac{-1 \pm \sqrt{4+3\tau^2}}{3}\right) \end{cases}$$

and the second set is:

$$\begin{cases} \alpha_2 = \dfrac{4\tau}{3(1+\tau^2)} - \alpha_1 \\ 3(1+\tau^2)\alpha_1^2 - 4\tau\alpha_1 + \dfrac{4\tau^2(1-3\tau^2)}{3(1+\tau^2)} = 0 \end{cases}$$

$$\Rightarrow \begin{cases} \alpha_1 = \left(\dfrac{2\tau}{1+\tau^2}\right)\left(\dfrac{1 \pm \tau\sqrt{3}}{3}\right) \\ \alpha_2 = \left(\dfrac{2\tau}{1+\tau^2}\right)\left(\dfrac{1 \mp \tau\sqrt{3}}{3}\right) \end{cases}$$

It is obvious that all these solutions are real ones. For the first set of solutions it is also obvious that one solution is always positive while the other one is always negative. Remembering we are only interested in positive solutions, we can choose the positive one. For the second set of solutions, one solution becomes negative or null when $\tau \geq 1/\sqrt{3}$, i.e. if $\eta_2 - \eta_1 \geq \frac{\pi}{3}$.

The full set of positive solutions is therefore:

$$\begin{cases} \alpha_1 = \sin(\eta_2 - \eta_1)\dfrac{\sqrt{4 + 3\tan^2\left(\frac{\eta_2-\eta_1}{2}\right)} - 1}{3} \\ \alpha_2 = \sin(\eta_2 - \eta_1)\dfrac{\sqrt{4 + 3\tan^2\left(\frac{\eta_2-\eta_1}{2}\right)} - 1}{3} \end{cases} \tag{15}$$

or (only if $\eta_2 - \eta_1 < \frac{\pi}{3}$):

$$\begin{cases} \alpha_1 = \sin(\eta_2 - \eta_1)\dfrac{1 + \tan\left(\frac{\eta_2-\eta_1}{2}\right)\sqrt{3}}{3} \\ \alpha_2 = \sin(\eta_2 - \eta_1)\dfrac{1 - \tan\left(\frac{\eta_2-\eta_1}{2}\right)\sqrt{3}}{3} \end{cases} \tag{16}$$

or (only if $\eta_2 - \eta_1 < \frac{\pi}{3}$):

$$
\begin{cases}
\alpha_1 = \sin(\eta_2 - \eta_1)\dfrac{1 - \tan\left(\frac{\eta_2-\eta_1}{2}\right)\sqrt{3}}{3} \\
\alpha_2 = \sin(\eta_2 - \eta_1)\dfrac{1 + \tan\left(\frac{\eta_2-\eta_1}{2}\right)\sqrt{3}}{3}
\end{cases} \tag{17}
$$

Some tests show that all these solutions are acceptable, there is not a unique way to set up a cubic Bézier curve with the matching criteria we have chosen. Since the first solution is both the simplest one ($\alpha_1 = \alpha_2$) and has the wider applicability domain, we will use this one only and forget the other solutions.

The control points P_1, Q_1, Q_2 and P_2 of a cubic Bézier curve approximating an elliptical arc should be chosen as follows:

$$
\begin{aligned}
P_1 &= E(\eta_1) \\
P_2 &= E(\eta_2) \\
Q_1 &= P_1 + \alpha E'(\eta_1) \\
Q_2 &= P_2 - \alpha E'(\eta_2)
\end{aligned}
$$

where

$$
\alpha = \sin(\eta_2 - \eta_1)\frac{\sqrt{4 + 3\tan^2\left(\frac{\eta_2-\eta_1}{2}\right)} - 1}{3}
$$

3.4.2 error estimation

The method used to estimate the error of cubic Bézier is exactly the same as the one explained in section 3.3.2 for quadratic Bézier curves. The coefficients are given by tables 3 and 4.

As before, we introduce a safety rational fraction $s(b/a)$ to obtain an upper bound of the error: $\varepsilon_3 \leq s(b/a)\tilde{\varepsilon}_3$.

The error resulting from the approximation of an ellipse arc by a cubic Bézier curve is bounded as follows:

$$\varepsilon_3 \leq \left(\frac{0.001\left(\frac{b}{a}\right)^2 + 4.98\left(\frac{b}{a}\right) + 0.207}{\left(\frac{b}{a}\right) + 0.0067}\right) a e^{c_0 + c_1(\eta_2 - \eta_1)}$$

where

$$c_i = \sum_{j=0}^{j=3} \frac{\mu_{i,j,0}\left(\frac{b}{a}\right)^2 + \mu_{i,j,1}\left(\frac{b}{a}\right) + \mu_{i,j,2}}{\left(\frac{b}{a}\right) + \mu_{i,j,3}} \cos\left(j(\eta_1 + \eta_2)\right)$$

Where the $\mu_{i,j,k}$ coefficients are given by tables 3 and 4.

This bound correctly overestimates the error, but is not too conservative, the cumulative distribution function of $\varepsilon/(s(b/a)\tilde{\varepsilon})$ is quite smooth and the corresponding mean value is 0.623.

Table 3: error coefficients for cubic Bézier $(0 < b/a < 1/4)$

i	j	$\mu_{i,j,0}$	$\mu_{i,j,1}$	$\mu_{i,j,2}$	$\mu_{i,j,3}$
0	0	3.85268	-21.229	-0.330434	0.0127842
0	1	-1.61486	0.706564	0.225945	0.263682
0	2	-0.910164	0.388383	0.00551445	0.00671814
0	3	-0.630184	0.192402	0.0098871	0.0102527
1	0	-0.162211	9.94329	0.13723	0.0124084
1	1	-0.253135	0.00187735	0.0230286	0.01264
1	2	-0.0695069	-0.0437594	0.0120636	0.0163087
1	3	-0.0328856	-0.00926032	-0.00173573	0.00527385

Table 4: error coefficients for cubic Bézier $(1/4 \leq b/a \leq 1)$

i	j	$\mu_{i,j,0}$	$\mu_{i,j,1}$	$\mu_{i,j,2}$	$\mu_{i,j,3}$
0	0	0.0899116	-19.2349	-4.11711	0.183362
0	1	0.138148	-1.45804	1.32044	1.38474
0	2	0.230903	-0.450262	0.219963	0.414038
0	3	0.0590565	-0.101062	0.0430592	0.0204699
1	0	0.0164649	9.89394	0.0919496	0.00760802
1	1	0.0191603	-0.0322058	0.0134667	-0.0825018
1	2	0.0156192	-0.017535	0.00326508	-0.228157
1	3	-0.0236752	0.0405821	-0.0173086	0.176187

4 Implementation

An implementation in the Java language of the algorithms explained in this paper is available in source form as a stand-alone file at `http://www.spaceroots.org/documents/ellipse/EllipticalArc.java`.

It is designed as an implementation of the `java.awt.Shape` interface and can therefore be drawn easily as any of the more traditional shapes provided by the standard Java API.

This class differs from the `java.awt.geom.Ellipse2D` in the fact it can handles parts of ellipse in addition to full ellipses and it can handle ellipses which are not aligned with the x and y reference axes of the plane.

Another improvement is that this class can handle degenerated cases like for example very flat ellipses (semi-minor axis much smaller than semi-major axis) and drawing of very small parts of such ellipses at very high magnification scales. This imply monitoring the drawing approximation error for extremely small values. Such cases occur for example while drawing orbits of comets near the perihelion.

When the arc does not cover the complete ellipse, the lines joining the center of the ellipse to the endpoints can optionally be included or not in the outline, hence allowing to use it for pie-charts rendering. If these lines are not included, the curve is not naturally closed.