Arc Length Parameterization of Spline Curves

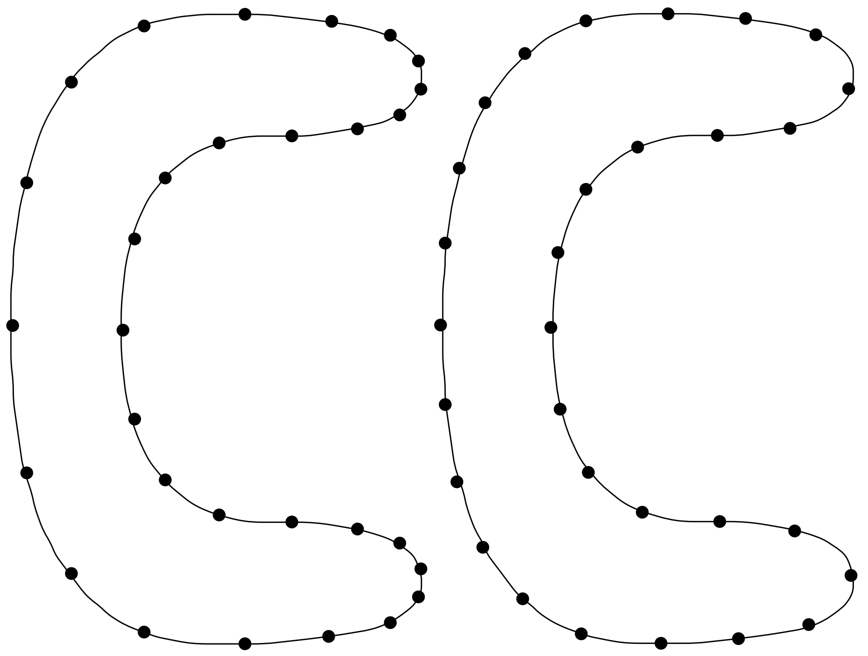
John W. Peterson
Taligent, Inc.
10725 N. DeAnza Blvd
Cupertino CA, 95014, USA
jp@taligent.com

Abstract

It is often desirable to evaluate parametric spline curves at points based on their arc-length instead of the curve's original parameter. Techniques are presented here for computing a reparameterization curve allowing approximate arc-length evaluation. This reparameterization curve is also expressed as a spline, allowing rapid evaluation as a function of arc-length. Using composition methods developed by DeRose et. al, the original curve and its reparameterization curve may be composed into a single, higher order curve that exhibits approximate arc-length parameterization.

Introduction

In many applications for spline curves, it is desirable to find points along a curve at intervals corresponding to the curve's arc-length. Examples include drawing a curve with dashed or patterned lines, placing text along a curved path, or accurately moving objects as part of an animated sequence. Except for linear (degree 1) curves, it is not possible to directly represent arc-length parameterization – it must be approximated. This paper presents an accurate approximation method for creating an auxiliary reparameterization curve. This reparameterization curve provides an efficient way to find points on the original curve corresponding to arc length.



Two parametric curves. The dots on the left curve are at equal parametric intervals. The dots on the right curve are at equal arc length intervals.

A parametric curve is defined as[1,2]:

$$\mathbf{Q}(u) = (Q_X(u), Q_Y(u))$$

A function $L(u)$ is defined as the arc length of the curve $\mathbf{Q}(u)$ at a particular value of u.. If $a = L(u)$ specifies an arc length along $\mathbf{Q}(u)$, an additional function is defined, $u = L^{-1}(a)$, that gives $\mathbf{Q}$'s parameter u for a particular arc length a. Thus evaluating $\mathbf{Q}(L^{-1}(a))$ determines the point on $\mathbf{Q}$ that is arc length a from the beginning of the curve.

Previous methods have concentrated on ways of evaluating $\mathbf{Q}(a)$ directly. Hartley and Judd[3] discuss methods for adjusting the knots of a B-Spline curve to get better parameterization, but found it was slow and unpredictable. Sharp and Thorn[4] present a method based on root finding similar to the process of evaluating $L^{-1}(a)$ presented here, but this must be done every time the curve is evaluated, which is computationally slow. Guenter and Parent[5] present an improvement to Sharp and Thorn's method that speeds up the integration step.

The approach taken here is to first approximate $L^{-1}(a)$ with an additional spline curve $\ell(u)$, thus allowing $\mathbf{Q}(a)$ to be found efficiently by evaluating $\mathbf{Q}(\ell(a))$. Since both $\mathbf{Q}(u)$ and $\ell(u)$ are defined as spline curves, it's also possible to define a new curve

$$\mathbf{P}(a) = \mathbf{Q}(\ell(a))$$

by composing the two spline functions together to generate a new, higher order, spline curve.

In the presentation here, the method for finding the arc length of a parametric curve is first discussed. Then methods for determining a curve's parameter given an arc length are shown. A method for approximating the reparameterization function with Chebyshev polynomials is discussed, along with a technique for converting this function into another spline curve. Finally, composing the reparameterization with the original curve is briefly summarized.

Finding the arc length

The length function $L(u)$ is defined by[5]

$$L(u) = \int_{u_0}^{u_1} \|\mathbf{Q}'(u)\| \, du$$

where

$$\|\mathbf{Q}'(u)\| = \sqrt{Q'_X(u)^2, Q'_Y(u)^2}$$

Calculating $L(u)$ requires evaluating the arc length integral. Guenter and Parent suggest using an adaptive application of Gaussian quadrature[5]. The results presented here are found by a similar method, using Gauss–Legendre integration. Instead of applying the integral to the entire curve, it is applied to each polynomial segment of the curve, and the results are summed. This gives very accurate results, since N point Gauss–Legendre is exact for a polynomial of degree $2N-1$ or less.[6]

If **Q** is a B-spline curve of order k, then it is defined with $m+1$ control points $\mathbf{V}_0 \ldots \mathbf{V}_m$ and $m+k+1$ knots $\bar{u}_0 \ldots \bar{u}_{m+k}$.[1] We define δ as an index into **Q**'s knot vector such that $\bar{u}_\delta < \bar{u}_{\delta+1}$ (i.e, δ indicates a breakpoint interval – a non-zero span of the knot vector defining a polynomial segment). If α is the highest δ where $\bar{u}_\alpha < u$, then the length is found with:

$$L(u) = \sum_{\text{for all }\delta}^{\delta<\alpha} \int_{\bar{u}_\delta}^{\bar{u}_{\delta+1}} \|\mathbf{Q}'(u)\|\, du + \int_{\bar{u}_\alpha}^{u} \|\mathbf{Q}'(u)\|\, du$$

where the integrals are computed with Gauss Legendre integration with the appropriate number of weights for the curve's polynomial degree. In the implementation it is useful to cache the lengths $L(\bar{u}_\delta)$ of all the curve's segments $\bar{u}_\delta < \bar{u}_{\delta+1}$ so that only one integration is necessary to find $L(u)$ for a particular value of u.

The next step is to find the parameter u for a particular arc length a of the curve (the function $L^{-1}(a)$). This is determined by using a root finding technique such as Newton's method to find a value u where $L(u) - a = 0$.

Newton's method uses the iteration sequence

$$u_{i+1} = u_i - \frac{f(u_i)}{f'(u_i)}$$

to find u such that $f(u) = 0$. In our case $f(u) = L(u) - a$, and $f'(u) = \|\mathbf{Q}'(u)\|$, the original integrand for $L(u)$. Newton's method converges very quickly, but it can fail in some pathological cases, either by cycling around the root (and never finding it) or rapidly diverging away from the root. In this algorithm however, these problems are not likely to occur, because $L^{-1}(a)$ is a smooth, monotonically increasing function, and is ideally suited for Newton iteration. Additional measures can be taken in implementing the root finder to avoid pathological cases, see Press et. al.[6]

Approximating the reparameterization curve

If the function $u = L^{-1}(a)$ is evaluated frequently for a particular $\mathbf{Q}(u)$, it is useful to create a spline curve that approximates $L^{-1}(a)$. Then evaluating $L^{-1}(a)$ is simply reduced to an additional spline evaluation, which is generally much less computation than performing the integration and root finding discussed above.

Our method for approximating this reparameterization curve is based on a technique developed by Watkins & Worsey[7] for reducing the degree of Bézier curves. The approach is to use Chebyshev polynomials to approximate the reparameterization curve. If the approximation is accurate enough, then the Chebyshev polynomial is converted directly to a B-spline (more specifically, a piecewise Bézier) by a simple change of basis.

Chebyshev approximation

A Chebyshev polynomial $T_i(u)$ is defined as:

$$T_i(u) = \cos(i \arccos u)$$

For small values of *i* the $T_i(u)$ are:

$$T_0(u) = 1$$

$$T_1(u) = u$$

$$T_2(u) = 2u^2 - 1$$

$$T_3(u) = 4u^3 - 3u$$

To approximate a function $f(u)$ with a *K* order Chebyshev polynomial, we compute a set of coefficients c_i:

$$c_i = \frac{2}{K} \sum_{j=1}^{K} f\left[\cos\left(\frac{\pi(j - \frac{1}{2})}{K}\right)\right] \cos\left(\frac{\pi i(j - \frac{1}{2})}{K}\right)$$

so that:

$$f(u) \approx \left[\sum_{i=0}^{K-1} c_i T_i(u)\right] - \frac{1}{2} c_0$$

giving a polynomial in terms of $T_i(u)$ that approximates $f(u)$. In the case of computing an approximation to the reparameterization function, $f(u)$ is set to $L^{-1}(u)$, where L^{-1} is the "inverse arc length" function described above.

For very large values of *K*, the Chebyshev approximation is very accurate. An advantage of Chebyshev polynomials is the higher order terms may be dropped to give a lower order (and less accurate) approximation. The difference in the terms removed provides a good estimate of the accuracy of the lower *k*–order approximation. More specifically, if $k < K$ then error ε between $f(u)$ and the approximation to it is:

$$\varepsilon = \sum_{i=k}^{K-1} |c_i| \approx \left| f(u) - \left(\left[\sum_{i=0}^{k-1} c_i T_i(u) \right] - \frac{1}{2} c_0 \right) \right|$$

Converting to Bézier form

Chebyshev polynomials are originally defined over the range -1...1. However, it is a simple matter to convert them to the range 0...1. by substituting $2u - 1$ into $T_i(u)$ to get $\overline{T}_i(u)$ e.g.,, $\overline{T}_1(u) = 2u - 1$, $\overline{T}_2(u) = 8u^2 - 8u + 1$, etc.[1] The Chebyshev approximation can be represented in matrix form as $\mathbf{c}\overline{\mathbf{T}}\mathbf{u}$ where $\mathbf{u} = \begin{bmatrix} 1 & u & \cdots & u^k \end{bmatrix}^{\mathrm{T}}$ and **c** is the coefficient vector $\begin{bmatrix} c_0 & c_1 & \cdots & c_k \end{bmatrix}$. Likewise, it is possible to represent a Bézier curve in matrix form as **VBu** where **V** is the control point vector. The matrices **B** and $\overline{\mathbf{T}}$ can be thought of as basis conversion matrices. Thus converting Chebyshev coefficients into Bézier control points is simply:

$$\mathbf{V} = \left(\overline{\mathbf{T}}^{-1} \mathbf{B}^{-1} \right) \mathbf{c}$$

Explicit formulations for the change of basis matrices are given by Watkins et. al.[7,8]

Individual Bézier segments may be joined into a single B-spline curve, since Bézier curves are a special case of B-Splines.[2] Using multiple curve segments allows increasing the accuracy of the approximation without increasing the order of the curve. The algorithm presented in the following section takes advantage of this property.

When multiple segments of a Bézier curve are computed from a Chebyshev approximation, the endpoints of the individual segments are not equal, i.e., slight C_0 discontinuities occur. Fortunately the error introduced by this is minor and may be avoided by simply forcing the endpoints to be equal at the junction between Bézier segments. Watkins [Watkins88] gives a more detailed examination of the error introduced by this "enforced interpolation".

Approximation algorithm

[1]Note the substitution does not effect the equation for computing the coefficients c_i.

The combination of Chebyshev approximation and converting Chebyshev polynomials into Bézier form leads to the following algorithm for approximating a function $f(u)$ with a spline curve. The algorithm creates a Chebyshev approximation for $f(u)$ over the full range of the function. If the approximation is close enough, then it is converted into Bézier form. Otherwise the parameter range of $f(u)$ is subdivided and each half is approximated. The subdivision continues until the approximation is within tolerance or a maximum recursion depth is reached. During the subdivision process, an array of integers keeps track of the subdivision depth of each successful approximation. This is used to construct a knot vector for the final approximation curve that matches $f(u)$'s parameterization.

Inputs:

f	Function to be approximated
$u_{\min}, u_{\max}$	Parameter range *f* is approximated over
k	Desired order of the spline approximation
K	Order to use for Chebyshev approximation ($K > k$)
max_points	Maximum number of control points allowed
$\varepsilon_{\max}$	Maximum approximation error

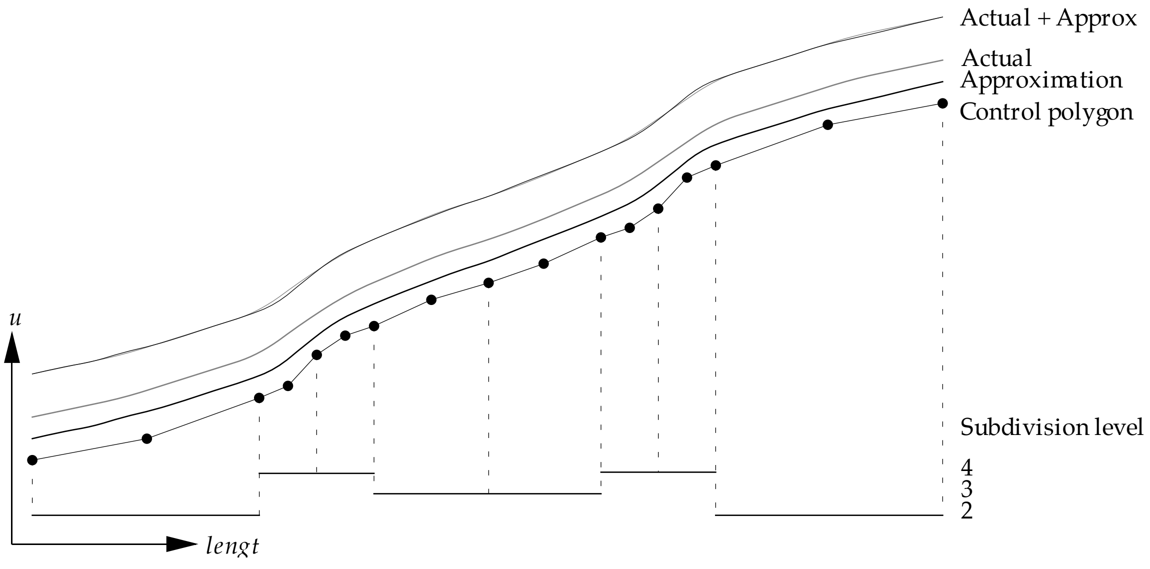
Outputs:

m	Highest control point index
$\mathbf{V}$	The B-Spline control points
$\bar{\boldsymbol{u}}$	The B-Spline knot vector

```
DoSegment( $t, v$, level )
begin
    $\mathbf{c} \leftarrow$ coefficients for a $K$ order Chebyshev approx to $f(u), u = t \ldots v$
    $\varepsilon \leftarrow \sum_{i=k}^{K-1} |c_i|$
    if ($\varepsilon < \varepsilon_{\max}$) or (level > max_level) then
        $\mathbf{V}_{N(k-1)\cdots(N+1)(k-1)} \leftarrow \left(\overline{\mathbf{T}}^{-1}\mathbf{B}^{-1}\right)\mathbf{c}_{0\cdots k-1}$
        $h_N \leftarrow$ level                /* Record recursion depth for creating the knot s */
        depth $\leftarrow$ max( level, depth )
        $N \leftarrow N + 1$
    else
        $s \leftarrow (t + v) / 2$
        DoSegment( $t, s$, level + 1 )
        DoSegment( $s, v$, level + 1 )
    endif
end
```

```
begin
    $N \leftarrow 0$                                   /* Number of segments in the approx. curve */
    depth $\leftarrow 0$
    max_level $\leftarrow \log_2 \frac{\text{max\_points}}{k}$
    DoSegement( $u_{\min}, u_{\max}$, 0 )
    $m \leftarrow N(k-1)$
    seg $\leftarrow 0$
    for $i \leftarrow 0$ to $N$ do                     /* Compute the knot vector based on the */
        for $j \leftarrow 1$ to $k$ - 1 do             /* subdivision depth of the particular segment */
            $\bar{u}_{j+i(k-1)} \leftarrow$ seg
        seg $\leftarrow$ seg + $2^{\text{depth}-h_i}$
    endfor
    $\bar{u}_{m+k} \leftarrow \bar{u}_{m+k-1}$
    for $i \leftarrow 0$ to $m + k$                    /* Scale the knot vector to the curve's arc length */
        $\bar{u}_i \leftarrow \bar{u}_i \left( \frac{u_{\max}}{\bar{u}_{m+k}} \right)$
end
```

This algorithm provides a way to create a spline curve ℓ that accurately approximates the inverse arc length function L^{-1}. Once this approximation is complete, the curve is evaluated at a particular arc length a by evaluating $\mathbf{Q}(\ell(a))$, as stated above.



Reparameterization curve for the curve shown in the Introduction.

Note the adaptive nature of the spline approximation

Creating an approximately arc length parameterized curve

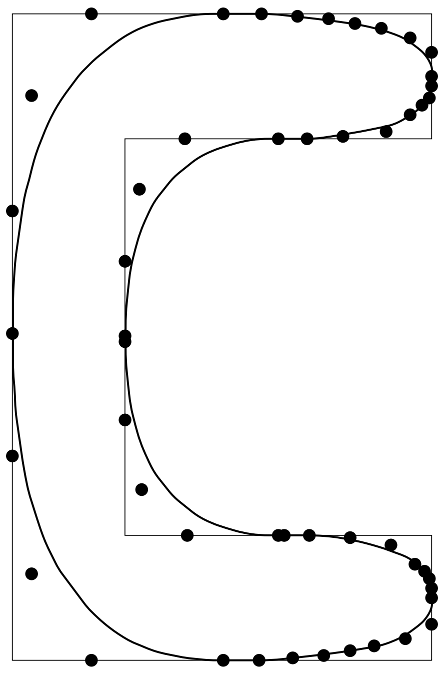
As stated in the introduction, it is possible to create a single curve ,

$\mathbf{P}(a) = \mathbf{Q}(\ell(a))$

by composing the two polynomials defining $\mathbf{Q}(u)$ and $\ell(u)$.

DeRose et. al. describes how to compose two spline curves to create a new curve of higher order.[9] To briefly summarize, the knots of $\mathbf{Q}(u)$ must be merged into the knot vector of $\ell(u)$, so that $\ell(u)$ is defined as a piecewise Bézier curve with each segment contained within a parametric interval of $\mathbf{Q}(u)$'s knot vector. Each of these segments is composed with $\mathbf{Q}(u)$, creating a new piecewise Bézier curve, $\mathbf{P}(a)$. The knot vector for $\mathbf{P}(a)$ is reconstructed from the knots created for $\ell(u)$.

The new curve $\mathbf{P}(a)$ will have a degree equal to the degree of $\ell(u)$ times the degree $\mathbf{Q}(u)$ (e.g., a cubic reparameterized by a quadratic is degree six). The high order significantly increases the number of control points required to represent the reparameterized curve. In the example above, the original curve has 10 control points, and the reparameterization curve has 17. The composition of the two curves, however, requires over 50 control points. Thus it usually more efficient to store the reparameterization curve separately and evaluate both $\ell(u)$ and $\mathbf{Q}(u)$ to find a point on the curve. The computational cost of two low-order spline evaluations is not significantly different from a single high order evaluation.



Curve generated by composing the curve with the reparameterization function. The straight lines are the original control polygon, the dots are the control points for the composed curve.

Discussion

The method presented for approximating the reparameteriztion curve probably has uses beyond the problem presented here. It may be useful for other situations where it is desirable to represent a function with a spline curve.

The technique works with rational curves (e.g., NURBS), as long as the derivative function for the curve ($\mathbf{Q}'(u)$) is correctly defined by using the quotient rule from elementary calculus. See Farin[2] for details.

The techniques presented here are most useful for situations where an arc length parameterization of a curve is evaluated frequently after the curve is created. Storing the reparameterization function as an additional spline curve allows rapid evaluation of a curve by its arc length, since spline evaluation is much faster than computing the integral length of the curve. Additionally, it is possible to compose the reparameterization curve with the original, generating a new higher order curve with arc length parameterization.

Acknowledgements
I would like to thank Tony DeRose and Yinsun Feng for helpful discussion during the course of this work.

References

1. Bartels, R.H., Beatty, J.C., and Barsky, B.A., *An Introduction to Splines for use in Computer Graphics and Geometric Modelling*, Morgan Kaufmann (Los Altos), 1987.

2. Farin, G., *Curves and Surfaces for Computer Aided Geometric Design*,, (2nd Ed.) Academic Press (San Diego), 1990.

3. Hartley, P.J., and Judd, C.J., "Parameterization and Shape of B-Spline Curves for CAD," *CAD,* 12(5), September 1980, p.235

4. Sharpe, R.J., and Thorne R.W., "Numerical Method for Extracting an Arc Length Parameterization from Parametric Curves," *CAD*, 14(2), March 1982, p.79

5. Guenter, B., and Parent R., "Computing the Arc Length of Parametric Curves," *IEEE Computer Graphics and Applications,* May 1990, p. 72

6. Press, W.H., Flannery, B.P., Teukolsky, S.A., and Vetterling, W.T., *Numerical Recipes in C,* Cambridge University Press (Cambridge), 1988.

7. Watkins, M.A., and Worsey, A.J., "Degree Reduction of Bézier Curves", *CAD*, 20(7), September 1988, p. 398,

8. Peterson, J.W., "Letter to the Editor", *CAD*, 23(6), August 1991, p.460

9. DeRose, T.D., Goldman, R.N., Hagen, H. and Mann, S. "Functional Composition Algorithms via Blossoming", May 1992, to appear.