Comm. Korean Math. Soc. 9 (1°94), No. 1, pp. 253–259

BEZIER CURVE APPROXIMATION IN CAD/CAM SYSTEM

NAM-SOOK WEE

One of the fundamental ingredient of CAD/CAM system is the representation of curves by basic graphic primitives such as line-and-arc, Bezier curves, or various splines. In the numerical computation, one of the widely used is the use of the cubic spline curve. A cubic spline curve is a curve which is made up of small cubic polynomials each of which is joined to the neighboring ones in the C^2 manner. This C^2-ness is a big advantage in the numerical analysis because it makes it possible to better control the error. However, those spline curves are rather cumbersome to use in various applications in CAD/CAM systems. One of the biggest drawbacks of the use of splines in CAD/CAM system is that one needs excessive number of control points to accurately represent the curve, and to do so, the control points must be spaced more or less evenly. This leads to performance degradation. Therefore one prefers to use a more flexible system. One of the widely used is the so-called Bezier curve. Bezier curve was invented by Bezier in the Renault automobile manufacturing factory to facilitate the automobile body design [Bez].

Bezier curve $c : [0,1] \to \mathbb{R}^2$ is a curve in $\mathbb{R}^2$ given by

$$c(t) = (1-t)^3 P_0 + 3(1-t)^2 t P_1 + 3(1-t)t^2 P_2 + t^3 P_3,$$

where P_0, P_1, P_2, and P_3 are points in $\mathbb{R}^2$ and $t \in [0,1]$. To represent a more complicated curve, one makes use of those pieces connected in a suitable way. Let $B : [0,l] \to \mathbb{R}^2$ be a curve such that the restriction of B to each integer interval $[j, j+1]$ is a Bezier curve. Namely, the curve $B_j : [0,1] \to \mathbb{R}^2$ is a Bezier curve when defined by $B_j(t) = B(t -$

Received September 5, 1993.

This work was supported in part by Non-Directed Research Fund, Korea Research Foundation, 1992

$j+1)$ for $t \in [0,1]$. To make B a continuous curve, one requires that $B_j(1) = B_{j+1}(0)$ for $j = 1,\ldots,l-1$. In general, there may be breaks in the tangential direction of B at each integer point $j = 1,\ldots,l-1$. To make B look smoother, one generally requires $B_j'(1)$ and $B_{j+1}'(0)$ are proportional for $j = 1,\ldots,l-1$, Then B can be made a C^1 curve after suitable reparametrization, for example, by arclength. Such curve B is called a *Bezier spline curve*, and we call each curve B_j a Bezier segment of B. It is interesting to note that B cannot be, in general, a C^2 curve even after reparametrization. This fact can be easily verified by checking the curvature. This geometric property distinguishes the Bezier spline curve from the usual cubic spline curves.

In many applications, the curve representing the geometric figure is initially given by different graphic primitives. For example, it may be given as a polygon by connecting points with short line segments. In a smaller scale, this polygonal representation of a curve looks as if it is a smooth curve, but when expanded, the jaggedness shows. Also, this representation of a smooth curve by line segments requires lots of points, which contributes to performance degradation. To remedy this situation, one approximates this line-segment curve by a smooth curve. A reasonable choice for this purpose is a cubic spline variety, in particular, a B-spline curve. However, because of the aforementioned shortcomings of spline curves in their application in CAD/CAM, a more flexible system, typically a Bezier spline curve, is more widely used. Another situation that typically occurs in application is the need to convert the curve in one format, say, line-and-arc, to another format, say, Bezier spline.

In this paper, we deal with approximation of a given curve by a Bezier spline curve. In most case, one try to minimize the integral of the error squared, i.e., the L^2 norm of the error. We will set up a scheme which can be understood as a steepest gradient descent method or as a way of solving an ordinary differential equation. Furthermore, when the smoothness condition is imposed, the state space of possible configuration is not a linear space. The nonlinear constraint conditions make it a nonlinear space, typically, a differentiable manifold. Thus our approach is in general solving the problem in a manifold in the setting of a steepest gradient descent method or an ordinary differential equation on manifold. This method is practical in the sense that the algorithm

for solving this type of problem is generally fast and well established.

In what follows, let $L(t)$ be a curve in $\mathbb{R}^2$ which is to be converted to, i.e., approximated by, a Bezier spline curve. Suppose we want to approximate it with a Bezier spline curve $B : [0, l] \to \mathbb{R}^2$ consisting of l Bezier segments. First, after suitable reparametrization, we may assume that $L : [0, l] \to \mathbb{R}^2$. Let $B_j : [0, 1] \to \mathbb{R}^2$ be a Bezier curve each of which is represented by

$$B_j(t) = (1-t)^3 P_0^j + 3(1-t)^2 t P_1^j + 3(1-t)t^2 P_2^j + t^3 P_3^j,$$

for $j = 1, \ldots, l$. The continuity condition is that $P_3^j = P_0^{j+1}$ for $j = 1, \ldots, l-1$. Thus the problem of finding the curve B is finding suitable points P_0^j, P_1^j, P_2^j, and P_3^j for $j = 1, \ldots l$ subject to the above continuity condition. Thus the set of all possible positions of such control points P_i^j is a $6l + 2$ dimensional vector space, which is denoted by $\mathbb{F}$. For the rest of this paper, we denote $n = 6l + 2$. Let's introduce the variables $x_1, \ldots, x_n$ such that $P_0^1 = (x_1, x_2), \ldots, P_3^l = (x_{n-1}, x_n)$. To be precise, $P_i^j = (x_{6j+2i-5}, x_{6j+2i-4})$ for $j = 1, \ldots l$ and $i = 0, \ldots 2$ Now the condition that $B_j'(1)$ and $B_{j+1}'(0)$ are proportional for all $j = 1, \ldots, l-1$ imposes extra $l - 1$ conditions. These conditions are expressible as

$$f_j(x_1, \ldots, x_n) = 0, \tag{1}$$

for $j = 2, \ldots, l$, where

$$\begin{aligned} f_j(x_1, \ldots, x_n) =& (x_{6j-7} - x_{6j-5})(x_{6j-2} - x_{6j-4}) \\ & - (x_{6j-6} - x_{6j-4})(x_{6j-3} - x_{6j-5}). \end{aligned} \tag{2}$$

Let $\mathbf{M}$ be the set of such possible positions under the above constraints. Namely, it is given by

$$\mathbf{M} = \{(x_1, \ldots, x_n) \in \mathbb{F} \mid f_j((x_1, \ldots, x_n) = 0 \quad \text{for} \quad j = 1, \ldots, n\}$$

Such $\mathbf{M}$ is called the *configuration space* which has the following nice structure:

THEOREM 1. *The configuration space* $\mathbb{M}$ *is a* $5l + 3$ *dimensional smooth submanifold of* $\mathbb{F}$, *as long as* P_2^{j-1}, P_0^j, *and* P_1^j *are distinct for each* $j = 2, \ldots l$.

Proof. We have demonstrated that the set of all possible configuration without the condition on the derivatives of the Bezier segments is a $6l+2$ dimensional vector space $\mathbb{F}$. Thus $\mathbb{M}$ is a subset of $\mathbb{F}$ given by $l-1$ equations $f_j = 0$ for $j = 2, \ldots, l$ in Condition (1). It is easy to check ∇f_j are linearly independent for $j = 2, \ldots, l$, as long as P_2^{j-1}, P_0^j, and P_1^j are distinct for each $j = 2, \ldots l$. Therefore, the proof follows from the implicit function theorem.

A point $x = (x_1, \ldots, x_n)$ in $\mathbb{F}$ defines a Bezier spline curve B. Let us now introduce the mean square error E which measures how close B is to the given curve L. E is defined by

$$E = \int_0^l \big(B(t) - L(t)\big)^2 dt.$$

Clearly, E can be thought of as a function of $x = (x_1, \ldots, x_n)$. The basic idea of finding the best B is to improve the error by evolving in the direction in which the error decreases fastest. First it is easy to see that E is a quadratics expression in $x_1, \ldots, x_n$. Therefore the gradient ∇E is an n-tuple of linear expressions in $x_1, \ldots, x_n$. One of the easy was of setting up the evolution strategy is to set it up as a differential equation which is the negative gradient flow. Namely let $x(t) = (x_1(t), \ldots, x_n(t))$ be a curve in $\mathbb{F}$ which represent the evolutionary path of the Bezier spline curve. Then we defines the ordinary differential equation as

$$x'(t) = -\eta \nabla E(x(t)), \tag{3}$$

where η is a suitable constant. This Equation (3) can be rewritten as an ordinary differential equation in the form

$$x' = Ax, \tag{4}$$

where A is an $n \times n$ matrix of entries of constant real numbers. To summarize, we have

THEOREM 2. *The least error solution B can be found by solving an ordinary differential equation* (4) *which comes from the negative gradient flow of E.*

It is useful to note that the solution of Equation (4) can be computed fast and it is even possible to write down the solution explicitly. Although the result of Theorem 2 is nice, it has one drawback because it has breaks in the tangential direction at the points where two Bezier segments are joined. Let us now discuss how to handle this case. To do so, one first needs the following Lemmas.

LEMMA 1. *The gradients ∇f_j defined in* (2) *are mutually perpendicular. Namely $\langle \nabla f_j, \nabla f_k \rangle = 0$ for $j \neq k$.*

Proof. When $j \neq k$, the variables occurring in f_j do not occur in f_k. Therefore, the standard inner product structure in $\mathbf{F}$ implies that ∇f_j and ∇f_k are perpendicular when $j \neq k$.

The restriction of E to the configuration space $\mathbf{M}$ defines a smooth function on $\mathbf{M}$. Let us denote by $\nabla^{\mathbf{M}} E$ the gradient on $\mathbf{M}$ of E, which can be expressed as follows:

LEMMA 2. *The gradient on $\mathbf{M}$ of E is given by*

$$\nabla^{\mathbf{M}} E = \nabla E - \sum_{j=2}^{l} \langle \nabla f_j, \nabla E \rangle \frac{\nabla f_j}{|\nabla f_j|^2}.$$

Proof. $\nabla^{\mathbf{M}} E$ is the tangential component of ∇E. Since ∇f_j is normal to $\mathbf{M}$ and ∇f_j's are mutually perpendicular by Lemma 1, $\nabla^{\mathbf{M}} E$ is computed simply by subtracting from ∇E the components ∇E in ∇f_j directions.

Thus the negative gradient flow $x(t)$ on $\mathbf{M}$ can be set up as:

THEOREM 3. *The least error solution B while maintaining the smoothness condition* (1) *can be found by solving the negative gradient flow equation on $\mathbf{M}$ which is expressed as*

$$x'(t) = -\eta \nabla^{\mathbf{M}} E(x(t)), \tag{5}$$

This is a differential equation on $\mathbf{M}$ which is conceptually clean and simple, but is harder computationally. Therefore, we present an algorithmically better way of handling this problem below.

Steepest Gradient Descent Algorithm. Let us now discuss the algorithm that is easy to implement in terms of computation, and which also works reasonably fast. This algorithm is essentially the steepest gradient descent algorithm on manifold. The standard implementation of this algorithm is first to go in the negative direction which is proportional to $\nabla^{\mathbf{M}}E$ then project the new point to $\mathbf{M}$ orthogonally because this new point lies outside $\mathbf{M}$. But the cost of finding the orthogonal projection at the end of each step is too expensive, thus we replace it with a projection which approximates the orthogonal projection by taking advantage of the geometry of the problem.

Define two sequences $\{x^{(k)}\} = \{(x_1^{(k)}, \ldots, x_n^{(k)})\}$ and $\{y^{(k)}\} = \{(y_1^{(k)}, \ldots, y_n^{(k)})\}$ of points in $\mathbf{M}$ as follows: First define $\{y^{(k)}\}$ by

$$y^{(k+1)} = -\eta \nabla^{\mathbf{M}} E(x^{(k)}). \tag{6}$$

Because of the curved nature of $\mathbf{M}$, $y^{(k+1)}$ is, in general, outside of $\mathbf{M}$. Thus in order to stay on $\mathbf{M}$, one has to take the nearest point $x^{(k+1)}$ in $\mathbf{M}$. Because of the reason mentioned above, we define $x^{(k+1)}$ in the following manner: Suppose now that $y^{(k+1)}$ is obtained by (6), it thus defines the control points P_i^j of B, hence a Bezier spline curve. The requirement of $x^{(k+1)}$ being in $\mathbf{M}$ means that P_2^{j-1}, P_0^j, and P_1^j are collinear for $j = 2, \ldots, l$. Thus, for $j = 2, \ldots, l$, let P_2^{j-1}, P_0^j, and P_1^j be control points defined by $y^{(k+1)}$. Then rotate P_2^{j-1} and P_1^j with respect to P_0^j which serves as the center of rotation so that P_2^{j-1}, P_0^j, and P_1^j becomes collinear. To be more specific, let θ be the (signed) angle between two vectors $P_0^j - P_2^{j-1}$ and $P_1^j - P_0^j$. Then rotate P_1^j with respect to P_0^j by angle $-\theta/2$, and rotate P_2^{j-1} with respect to P_0^j by angle $\theta/2$. This process defines new P_2^{j-1}, P_0^j, and P_1^j, therefore a new point $x^{(k+1)}$ in $\mathbf{M}$. Then repeat (6) and the process of projecting to $\mathbf{M}$.

It is also worth noticing that η is a constant suitably chosen so that it has to be large enough for fast convergence while small enough not to miss the minimum too far away. In practice, it is best to experiment

with η to figure out the best compromise. The choice of initial point also contribute somewhat to the performance of this method, so it must be judiciously chosen.

References

[Bez] P. Bezier, *Emploi des Machines a commande numerique*, Masson et Cie, Paris, 1970.

[deB] C. de Boor, *A Practical Guide to Splines*, Springer-Verlag, New York, 1978.

[F] G. Farin, *Triangular Bernstein-Bezier Patches*, Computer Aided Geometric Design **3(2)** (1986), 83–127.

[FDFH] J. Foley, A. van Dam, S. Feiner. J. Hughes, *Computer Graphics, Principles and Practice, 2nd Edition*, Addison-Wesley, Reading Massachusetts, 1991.

[FB] A. Forsey and R. Bartels, *Hierarchical B-Spline Refinement*, SIGGRAPH88 (1988), 205–212.

[SB] J. Stoer and R. Bulirsch, *Introduction to Numerical Analysis, 2nd Edition*, Springer-Verlag, New York-Berlin, 1992.

Department of Industrial Engineering
Hansung University
Seoul 136-792, Korea