A simple, efficient degree raising algorithm for B-spline curves

Wayne Liu

University of Waterloo, Department of Electrical & Computer Eng., Waterloo, Ontario N2L 3G1, Canada
Received February 1996; revised October 1996

Abstract

This paper presents a new algorithm to compute the degree-raised version of a spline. The new algorithm is as fast as the best existing algorithm, but is much easier to understand and to implement. The new control vertices of the degree-raised spline are obtained simply by a series of knot insertions followed by a series of knot deletions. © 1997 Elsevier Science B.V.

Keywords: B-splines; Splines; Degree elevation; Algorithm; Blossoming

1. Introduction

Several algorithms have been published that raise the degree of B-spline curves (Prautzsch and Piper, 1991; Prautzsch, 1984; Cohen et al., 1985; Piegl and Tiller, 1994). The fastest of these is by Prautzsch and Piper (1991). However, this algorithm is complex and difficult to understand. From a software engineering point of view, it is desirable to implement a simple and easy-to-understand algorithm. This approach was taken by Piegl and Tiller (1994), who implemented the simplest algorithm: they converted the B-spline curve to Bézier form, raised the degree of the Bézier curves, and then pieced together the Bézier curves into a B-spline curve.

This paper introduces a new algorithm to degree-raise a B-spline curve. This algorithm is as fast as Prautzsch and Piper's algorithm, while being as simple to understand and implement as Piegl and Tiller's algorithm. The new algorithm simply computes the control vertices of the degree-raised B-spline by a series of knot insertions followed by a series of knot deletions.

1E-mail: wbliu@vlsi.uwaterloo.ca.

0167-8396/97/$17.00 © 1997 Elsevier Science B.V. All rights reserved.
PII S0167-8396(96)00060-X
Prautzsch and Piper (1991) presented a table comparing the number of operations required by three previous algorithms. Piegł and Tiller’s result did not appear in this table as their work occurred after that of Prautzsch and Piper. Table 1 extends the table of Prautzsch and Piper to include both Piegł and Tiller’s algorithm and the algorithm presented in this paper. This table counts the number of operations required per knot of the B-spline curve, assuming all knots are of multiplicity one. Since the cost of linear combinations is much greater than that for additions and multiplication, the table shows the number required for these two types of operations separately.

The table shows that there is no real difference in speed between the new algorithm and the one by Prautzsch and Piper. These two algorithms are much better than the other three in that they require $O(n)$ operations per knot as opposed to $O(n^3)$. Note however that the table shows the cost of raising the degree of the curve by one; Piegł and Tiller’s algorithm is more efficient when increasing the degree by an arbitrary number.

### 2. Derivation of new algorithm

The new algorithm is based on the degree-raising formula for blossoms derived by Ramshaw (1987). A direct implementation of the formula would require $O(n^3)$ operations per knot, but the new algorithm organizes the computations to require only $O(n)$ operations per knot.

The derivation of the algorithm requires the two formulas given below. Readers are assumed to be familiar with the concepts and notation of blossoming analysis. (For an introduction to blossoming, see the article by Seidel (1989), which also gives the derivation of these formulas.)

**Theorem 1 (Degree Raising Formula).** Let $F$ be a B-spline curve, with knot sequence $t_0, \ldots, t_m$, and $f$ be its piecewise blossom. Let $G$ be the degree-raised version of $F$, and $g$ be its piecewise blossom. The knot sequence of $G$, $s_0, \ldots, s_m$, is the same as the

---

2 Mann and Liu (1995) derived formulas for degree-raising polynomial curves as a special case of blossom composition. Implementing their formulas for B-splines using their algorithms would yield the same computations as the Piegł and Tiller algorithm.
knot sequence for $F$, except that the multiplicity of each breakpoint is increased by one. The control vertices of $G$, $Q_0, \ldots, Q_{m'-n}$, are given by
\[
Q_i = g(s_i, \ldots, s_{i+n}) = \frac{1}{n+1} \sum_{j=i}^{i+n} f(s_j, s_{j-1}, s_{j+1}, \ldots, s_{i+n}).
\]

Theorem 2 (Knot Insertion Formula). Let $t_0', \ldots, t_{m'}'$ be a refinement of the knot sequence of $F$. Let $F'$ be the version of $F$ over this refined knot sequence. Then the control vertices, $P_0', \ldots, P_{m'-n+1}'$, of $F'$ are given by
\[
P_i' = f'(t_i', \ldots, t_{i+n-1}').
\]

A direct application of Eq. (1) yields a simple but inefficient implementation of degree-raising: use a modified de Boor algorithm to compute each blossom value required for a new control vertex. However, this direct algorithm would require $O(n^2)$ operations per control vertex because it computes some values unnecessarily, and other values several times. The new degree-raising algorithm avoids the inefficiencies of the direct approach.

We gain an intution for the new algorithm by writing down some of the values that must be computed using the example B-spline of Fig. 1.

\[
\begin{align*}
Q_0 &= g(0, 0, 0, 0) = 1/4 (f(0, 0, 0) + f(0, 0, 0) + f(0, 0, 0) + f(0, 0, 0)) \\
Q_1 &= g(0, 0, 0, 1) = 1/4 (f(0, 0, 0) + f(0, 0, 0) + f(0, 1, 0) + f(0, 1, 0)) \\
Q_2 &= g(0, 0, 1, 1) = 1/4 (f(0, 0, 1) + f(0, 0, 1) + f(0, 1, 1) + f(0, 1, 1)) \\
Q_3 &= g(0, 1, 1, 2) = 1/4 (f(0, 1, 1) + f(0, 1, 2) + f(0, 1, 2) + f(1, 1, 2)) \\
Q_4 &= g(1, 1, 2, 2) = 1/4 (f(1, 1, 2) + f(1, 1, 2) + f(1, 2, 2) + f(1, 1, 2)) \\
Q_5 &= g(1, 2, 2, 3) = 1/4 (f(1, 2, 2) + f(1, 2, 3) + f(1, 2, 3) + f(2, 2, 3)) \\
\ldots
\end{align*}
\]
Notice, by formula (2), the values marked by an underline correspond to the first four control vertices of a knot-inserted version of $F$, that is, a representation of $F$ that has knot sequence

$$0, 0, 0, 1, 1, 2, 2, 3, 3, 4, 4, 6, 6, 6, 6.$$  

Next notice the values marked by double underlines correspond to the second four control vertices of the version of $F$ with knot sequence

$$0, 0, 0, 0, 1, 2, 2, 3, 3, 4, 4, 6, 6, 6, 6.$$  

In general, the terms in the summations are taken from the control vertices of knot-inserted version of $F$ obtained by increasing the multiplicity of each breakpoint by one, and then deleting one knot.

The new algorithm would perform the required computations as follows: first, set all control points $Q_i$ initially to 0; next, create a version $F'$ of $F$ where the multiplicity of each breakpoint is increased by 1 (knots are 0, 0, 0, 0, 1, 1, 2, 2, 3, 3, 4, 4, 6, 6, 6, 6); next, create a version $F''$ of $F'$ by deleting the first knot 0 from $F''$ (knots 0, 0, 0, 1, 1, 2, 2, 3, 3, 4, 4, 6, 6, 6, 6, 6, 6), then add the first control vertex of $F'$ to $Q_0$; next, create another version, $F''$, of $F'$ by deleting the second knot 0 from $F''$ (knots the same as $F''$), then add the first two control vertices to $Q_0$ and $Q_1$; continue in this way, deleting each knot of $F''$, and adding the appropriate control vertices to the $Q_i$; finally, divide each $Q_i$ by $n + 1$.

Fig. 2 shows the degree-raised representation of the example B-spline curve. The hollow circles show the control vertices of the knot-inserted representation $F'$.

Here is the complete algorithm, along with a proof of the correctness of the computations:

**Input:** The knot sequence, $t_0, \ldots, t_n$, and control vertices, $P_0, \ldots, P_{m-n}$, of a B-spline $F$.

**Output:** The the knot sequence, $s_0, \ldots, s_{m'}$, and control vertices, $Q_0, \ldots, Q_{m'-n}$, of the degree-raised version.

1. Set all control vertices $Q_0, \ldots, Q_{m'-n}$ to zero. At this point, $Q_0 = \ldots = Q_{m'-n} = 0$. 
2. Create \( F' \) by inserting knots into \( F \) to increase the multiplicity of each breakpoint by one. The knot sequence of \( F' \) is \( s_0, \ldots, s_{m'} \). The control vertices of \( F' \) are \( P_i' = f(s_i, \ldots, s_{i+n-1}) \).

3. For each \( j = 0, \ldots, m' \):
   (a) Create \( F^j \) by deleting knot \( s_j \) from \( F' \). The knots of \( F^j \) are \( s_0, \ldots, s_{j-1}, s_{j+1}, \ldots, s_{m'} \). For \( i = \max(j - n + 1, 0), \ldots, \min(j, m' - n) \), the control vertices of \( F^j \) are
   \[
P_i^j = f(s_i, \ldots, s_{j-1}, s_{j+1}, \ldots, s_{i+n}) ,
   \]
   while control vertices outside this range stay the same as in \( F' \).
   Note that if \( s_j = s_{j-1} \), the knot deletion should not be recomputed: the control vertices \( P_i^{j-1} \) will be the same as the control vertices \( P_i^j \), and can be reused.
   (b) For each \( i = \max(j - n + 1, 0), \ldots, \min(j, m' - n) \), add \( P_i^j \) to \( Q_i \). This step ensures that, at the end of the iterations, each \( Q_i \) will have the value
   \[
   Q_i = \sum_{j=i}^{i+n} P_i^j = \sum_{j=i}^{i+n} f(s_i, \ldots, s_{j-1}, s_{j+1}, \ldots, s_{i+n}).
   \]

4. Divide each \( Q_i \) by \( n + 1 \). Thus, \( Q_i \) has the value required by formula (1):
   \[
   Q_i = \frac{1}{n + 1} \sum_{j=i}^{i+n} f(s_i, \ldots, s_{j-1}, s_{j+1}, \ldots, s_{i+n}).
   \]

3. Run-time analysis of new algorithm

In order to calculate the run-time, a choice must be made for which knot insertion and knot deletion algorithms to use. A good choice in terms of efficiency is to use Boehm’s algorithm (1980) for knot insertion, and an “inverted” Boehm’s algorithm for knot deletion. If a new knot \( s \) is inserted between \( t_j \) and \( t_{j+1} \), Boehm’s algorithm computes for \( j - n + 2 \leq i \leq j + 1 \),
   \[
P_i' \leftarrow \alpha_i P_i - (1 - \alpha_i) P_i,
   \]
   where \( s = \alpha_i t_{i-1} + (1 - \alpha_i) t_{i+n-1} \). On the other hand, to delete the knot \( s \), the deletion algorithm computes
   \[
P_i \leftarrow \alpha_i P_i - (1 - \alpha_i) P_i',
   \]
   where \( t_{i+n-1} = \alpha_i t_{i-1} + (1 - \alpha_i) s \).

The cost for the run-time of the new algorithm comes from three places: inserting the knots, deleting the knots, and adding the points. The cost of a knot insertion is \( n - \mu + 1 \) linear combinations, where \( \mu \) is the multiplicity of the knot. A knot deletion also costs \( n - \mu + 1 \) linear combinations. Thus, the actual cost of the algorithm depends on the multiplicity of the knots. A summation requires \( n + 1 \) additions and a multiplication (by \( \frac{1}{n+1} \)).
In the worst case, each knot has multiplicity one: \( m - 2n + 3 \) knots must be inserted, \( m \) knots must be deleted, and summations for \( 2m - 3n + 4 \) control vertices must be computed. In total, the algorithm requires \( n(2m - 2n + 3) \) linear combinations and \( (n + 1)(2m - 3n + 4) \) addition and \( 2m - 3n + 4 \) multiplications. In the best case, all knots have full multiplicity, so knot insertion and deletion require no linear combinations. The algorithm simply becomes a less efficient version of Bézier degree-raising. In both the best and worst case, the order of the algorithm is \( O(nm) \).

Note that because of the knot deletion step, the algorithm uses some non-convex combinations. In total, a resulting control vertex is separated from the initial control vertices by at most \( n/2 \) successive convex combinations, followed by \( n \) successive non-convex combinations, followed by one more convex combination.

Acknowledgements

I want to thank Stephen Mann for valuable advice regarding this paper.

References

Prautzsch, H. and Piper, B. (1991), A fast algorithm to raise the degree of B-spline curves, Computer Aided Geometric Design 8, 253–266.