Sliding Windows Algorithm for B-spline Multiplication*

Xianming Chen†
School of Computing
University of Utah

Richard F. Riesenfeld‡
School of Computing
University of Utah

Elaine Cohen§
School of Computing
University of Utah

Abstract

B-spline multiplication, that is, finding the coefficients of the product B-spline of two given B-splines, is useful as an end result, in addition to being an important prerequisite component to many other symbolic computation operations on B-splines. Algorithms for B-spline multiplication standardly use indirect approaches such as nodal interpolation or computing the product of each set of polynomial pieces using various bases. The original direct approach is complicated. B-spline blossoming provides another direct approach that can be straightforwardly translated from mathematical equation to implementation; however, the algorithm does not scale well with degree or dimension of the subject tensor product B-splines. We present the *Sliding Windows Algorithm* (SWA), a new blossoming based algorithm for B-spline multiplication that addresses the difficulties mentioned heretofore.

CR Categories: J.6 [Computer Applications]: Computer-Aided Engineering—Computer-aided design (CAD); I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Splines

Keywords: NURBS multiplication, sliding windows algorithm, blossoming.

1 Introduction

B-spline multiplication, that is, finding the coefficients of the product B-spline of two given B-splines, is useful as an end result, in addition to being an important prerequisite component to many other symbolic computation operations on B-splines. Several theoretically based direct algorithms and several indirect approaches have been proposed for performing this symbolic computation.

Using the discrete B-spline representation, Morken [Morken 1991] presented the first theoretically proven result for expressing the coefficients of a product B-spline in terms of the coefficients of its two factor B-splines (Theorem 3.1 in [Morken 1991]), and further derived recurrence relations (Proposition 4.1 in [Morken 1991]) that may be useful for developing an efficient algorithm for B-spline multiplication. However, these recurrence relations appear somewhat involved, and, as remarked in his paper, as well as in [Lee 1994], it is not obvious as how to obtain an efficient algorithm based on these recurrence relations. To the best of our knowledge, this discrete B-spline based approach, although theoretically appealing, has not resulted in any practical algorithm for B-spline multiplication in the CAD community.

The first practical B-spline multiplication algorithm proposed in [Elber 1992] is based on sampling the product by sampling each factor B-spline and indirectly forming the product B-spline using nodal interpolation. Elber and Cohen [Elber and Cohen 1993] further used the algorithm as a fundamental tool to symbolically query and analyze second order differential surface properties.

Ueda [Ueda 1994] reported a direct approach for B-spline multiplication based on a blossom representation of B-splines, and proved its equivalence to Morken's earlier discrete B-spline approach. However, observing that computing the product B-spline coefficients directly from the blossom representation of product B-spline (Eq. (22) [Lee 1994] and Eq. (17) [Ueda 1994]) is very inefficient, Lee [Lee 1994] proposed an indirect approach that converts a B-spline basis representation to a power basis representation, performs multiplication by convolving coefficients, and then converts back to B-spline basis representation via the de Boor-Fix formula [de Boor 1978]. As the whole process is computationally expensive, Lee developed a scheme to evaluate the coefficients of the product B-spline a group at a time by computing a chain of blossoms. Piegl and Tiller [Piegl and Tiller 1997], exploiting the algorithm for multiplying Bézier curves [Cohen et al. 2001; Farin 2002], provided another indirect approach that converts B-splines via knot insertion to piecewise Bézier curves, performs Bézier multiplication, and then employs knot removal methods to convert back to the B-spline representation.

Of various algorithms for B-spline multiplication, Ueda's blossoming-based approach does not involve a basis conversion, and only uses convex affine combination to construct new product B-spline coefficients. Although the authors prefer to use a direct method because it is constructive and stays within B-spline formulations, the original algorithm lacks efficiency and favorable scalability behavior with respect to degree and dimension (i.e., number of variables)

Several researchers (for example [DeRose et al. 1993]) have observed that straightforward implementations of many blossoming-based B-spline algorithms are inefficient when the involved recursive blossom evaluations exhibit combinatorial characteristics, which is true for B-spline multiplication. One strategy to speed up such algorithms is an associated look-up table to reuse previous partial results of recursive blossom evaluation [Ueda 1994]. However, partial result reuse alone offers limited efficiency benefits for multivariate high degree B-splines.

Tensor product splines with large numbers of variables arise in many analytical situations, and have particular use in B-spline subdivision based rational constraint solvers [Sherbrooke and Patrikalakis 1993; Elber and Kim 2001] to enable solution of many complex geometry problems. For example, 4-dimensional B-spline multiplication is carried out for the computation of various geometric entities including, bisector surfaces [Elber and Kim 2000], bi-tangent curves and flecnodal curves [Elber et al. 2005], accessible regions for 5-axis machining [Elber and Cohen 1999; Elber and Kim 2001], offset

*This work was supported in part by NSF IIS0218809 and NSF CCR0310705. All opinions, findings, conclusions or recommendations expressed in this document are those of the author and do not necessarily reflect the views of the sponsoring agencies.

†e-mail: xchen@cs.utah.edu
‡e-mail:rfr@cs.utah.edu
§e-mail:cohen@cs.utah.edu

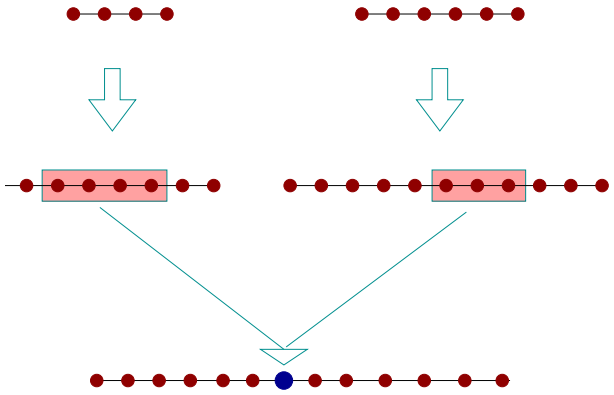

Figure 1: **Sliding Windows Algorithm Overview**

Top row: Coefficient meshes of factor B-splines. Middle row: Corresponding blossom meshes of curves in top row. Bottom row: Coefficient meshes of product B-spline. Notice that windows in the second row are used to compute the enlarged seventh control point of the product. (cf. Examples 1 and 2).

surface self-intersection [Seong et al. 2006a], and perspective silhouette of a general swept volume [Seong et al. 2006b], etc. 5D B-spline multiplication is required in the tracking of deforming surface/surface intersection [Chen et al. 2006], and even 7D B-spline multiplication has to be performed to find the triple-point singularity of deforming surface/surface intersection [Chen 2007].

Even with today's improved computer speeds compared to that of the 1990s when the blossoming-based direct B-spline multiplication algorithm was proposed, it would still be infeasible using that algorithm to compute the product B-spline at interactive speed for the difficult multiplication examples previously discussed. In this paper we present the Sliding Window Algorithm (SWA), an efficient algorithm for blossoming based B-spline multiplication. In order to develop this algorithm, we reformulate the blossom representation from the one presented in [Ueda 1994; Lee 1994] and carefully organize the overall computations of the coefficients of the product B-spline. Attaining interactive speeds and incurred no performance bottlenecks in all difficult situations like the previously mentioned examples, we have used this algorithm for computing coefficients of product splines. With only exception of the 7-variate multiplication, we have not encountered a slow-down in the computation. A rigorous analysis of the presented algorithm and other approaches on both efficiency and numerical stability issues is underway and will be discussed in a coming report. In this paper, we focus on presenting the form, structure, and details of the sliding windows algorithm.

2 Algorithm Overview

Given two tensor product B-spline factors, with their defining knot vectors and control meshes, the algorithm first constructs a pair of intermediate meshes of blossom values, one for each B-spline factor. It further maintains a sliding window sub-mesh of the constructed blossom mesh, one for each factor. The blossom values in a pair of sliding windows are used to compute a control point of the product B-spline. Each control point of the product is generated by a pair of windows and between computations the windows slide in an ordered way. Fig. 1 illustrates this with a pair of windows for a particular control point for univariate B-spline multiplication.

The rest of the paper is organized as follows. After a brief review of the basic principles of multiplying two B-splines via blossoming in Section 3, Section 4 presents a reformulation of the blossoming representation of a product B-spline. Section 5 develops an incremental algorithm of knot subsequence enumeration. Starting at Section 6 which reviews briefly general n-dimensional (i.e., n-variate) B-spline multiplication, we focus on the n-dimensional (n-variate) case for general n. Section 7 constructs, for the factor B-splines, a pair of n-dimensional hyper-arrays of blossom values called *blossom meshes*. A pair of windows of blossom submeshes is constructed in Section 8 to compute a corresponding control point of the product, and the collection of control points is computed by sliding this window pair along the blossom meshes. Finally, Section 9 presents concluding comments.

3 B-spline Products via Blossoming

This section reviews the basic blossoming principles used for multiplication of two B-spline functions, as noted in [Lee 1994] and reported in detail in [Ueda 1994]. A general introduction to blossoming can be found in [Ramshaw 1987; Ramshaw 1989; DeRose and Goldman 1991; Seidel 1993; Gallier 1998].

Suppose a degree d univariate B-spline function G defined by a knot vector and a series of coefficients (control points).

The knot vector is given by

$$v_1,\ \underbrace{v_1,\cdots,v_1}_{n_1=d},\underbrace{v_2,\cdots,v_2}_{0<n_2\le d+1},\cdots,\underbrace{v_{r-1},\cdots,v_{r-1}}_{0<n_{r-1}\le d+1},\underbrace{v_r,\cdots,v_r}_{n_r=d},\ v_r, \qquad (1)$$

where $v_1, v_2, \cdots, v_r$ is the *breakpoint sequence* of distinct knot values in ascending order. Note that knot vector (1) is augmented by extra knots v_1 and v_r because they are required at the two ends for the appropriate definition of basis B-splines. However, they do not appear in the blossoming representation, that is, if G is regarded as the diagonalization of a symmetric d-affine function g called the *blossom of G* (See [Ramshaw 1987; Ramshaw 1989] for how to construct the blossom g from G). Also note that the internal maximal multiplicity of $d+1$ allows possible $C^{(-1)}$ continuity at breakpoints.

The ordered coefficients are the blossom values of g evaluated on an order collection of knot sequences (this is the *dual functional property* of blossoming [Ueda 1994]),

$$g(\underbrace{v_1,\cdots,v_1}_{d}),g(\underbrace{v_1,\cdots,v_1}_{d-1},v_2),\cdots,g(\underbrace{v_r,\cdots,v_r}_{d}) \qquad (2)$$

Each sequence at which g is evaluated in (2) has length d and is called a (**d**)-knot-sequence, or simply (**d**)-sequence or abbreviated as (**d**)-seq in this paper. The (**d**)-sequences in (2) are special in the sense that they take **d** consecutive knots from the knot vector (1), and their corresponding blossom values are the control points. More formally,

> **Definition 1 (p)-sequence, dual (p)-sequence** *A knot sequence that has a total length of p is called a* (**p**)*-sequence. If a* (**p**)*-sequence consists of consecutive knots from the knot vector defining some B-spline, then, alluding to the property that it is dual to some control point of the B-spline, it is called a* dual (**p**)*-sequence.*

Dual (**p**)-sequences of a B-spline are ordered, with two sequential control points having their corresponding (**p**)-sequences shifted one position in the knot vector (cf. (1)).

Now, consider a second B-spline function $\widehat{G}$ (with blossom $\widehat{g}$) of

degree $\widehat{d}$, with knot vector,

$$\widehat{v}_1,\ \underbrace{\widehat{v}_1,\cdots,\widehat{v}_1}_{\widehat{n}_1=\widehat{d}},\underbrace{\widehat{v}_2,\cdots,\widehat{v}_2}_{0<\widehat{n}_2\le\widehat{d}+1},\cdots,\underbrace{\widehat{v}_{s-1},\cdots,\widehat{v}_{s-1}}_{0<\widehat{n}_{s-1}\le\widehat{d}+1},\underbrace{\widehat{v}_s,\cdots,\widehat{v}_s}_{\widehat{n}_s=\widehat{d}},\ \widehat{v}_s \tag{3}$$

and control points (i.e., blossom values of $\widehat{g}$ evaluated at dual $\widehat{\mathbf{d}}$-sequences),

$$\widehat{g}(\underbrace{\widehat{v}_1,\cdots,\widehat{v}_1}_{\widehat{d}}),\widehat{g}(\underbrace{\widehat{v}_1,\cdots,\widehat{v}_1}_{\widehat{d}-1},\widehat{v}_2),\cdots,\widehat{g}(\underbrace{\widehat{v}_s,\cdots,\widehat{v}_s}_{\widehat{d}}) \tag{4}$$

Assuming $v_1=\widehat{v}_1=u_1$ and $v_r=\widehat{v}_s=u_t$, the product of G and $\widehat{G}$ is another B-spline function F of degree $D=d+\widehat{d}$, with knot vector

$$u_1,\ \underbrace{u_1,\cdots,u_1}_{m_1=D},\underbrace{u_2,\cdots,u_2}_{0<m_2\le D+1},\cdots,\underbrace{u_{t-1},\cdots,u_{t-1}}_{0<m_{t-1}\le D+1},\underbrace{u_t,\cdots,u_t}_{m_t=D},\ u_t \tag{5}$$

where, $m_1=n_1+\widehat{n}_1=m_t=n_r+\widehat{n}_s=d+\widehat{d}=D$. For $0<i<t$, the multiplicity of u_i in the product knot vector is computed according to Table 1.

1	$m_i=n_j+\widehat{d}$	if $u_i=v_j$ for some j but is absent from $\widehat{G}$'s knot vector
2	$m_i=\widehat{n}_k+d$	if $u_i=\widehat{v}_k$ for some k but is absent from G's knot vector
3	$m_i=max(n_j+\widehat{d},\widehat{n}_k+d)$	if $u_i=v_j=\widehat{v}_k$ for some j and k

Table 1: **Multiplicity of Breakpoints in Product Knot Vector**

By the dual functional property, the control points of the product B-spline are the blossoms f of F evaluated on its dual $(\mathbf{D})$-*sequences*, that is, all sequences of $\mathbf{D}$ consecutive knots from the product knot vector (5). The blossom of product B-spline is related to the two blossoms of its two factor B-splines by (Eq.(17) of [Ueda 1994] and Eq.(22) of [Lee 1994]),

$$f(k_1,k_2,\cdots,k_{_D})=\frac{\sum g(k_{i_1},k_{i_2},\cdots,k_{i_d})\ \widehat{g}(k_{j_1},k_{j_2},\cdots,k_{j_{\widehat{d}}})}{\binom{d+\widehat{d}}{d}}, \tag{6}$$

where the summation runs over all $(\mathbf{d})$-subsets $\{i_1,i_2,\cdots,i_d\}$, and complementary $(\widehat{\mathbf{d}})$-subsets $\{j_1,j_2,\cdots,j_{\widehat{d}}\}$ of the set $\{1,2,\cdots,D-1,D\}$.

In Eq. (6), a single evaluation of the blossom f at a dual $(\mathbf{D})$-sequence of the product B-spline, is expanded to $2\binom{D}{d}$ evaluations of blossom g at $(\mathbf{d})$-subsequences, and of blossom $\widehat{g}$ at $(\widehat{\mathbf{d}})$-subsequences. The $(\mathbf{d})$-subsequence and $(\widehat{\mathbf{d}})$-subsequence are so named because they are subsets of the dual $(\mathbf{D})$-sequence from the product knot vector. Notice that, they are simply $(\mathbf{d})$ and $(\widehat{\mathbf{d}})$-sequences that consists of consecutive knots from some refined knot vectors of the two factor B-splines G and $\widehat{G}$, respectively. In other words, with respect to the original knot vector of the corresponding factor B-spline, they are not dual sequences in general. They are usually explicitly called *factor sequences*; more formally (cf. Definition 1),

Definition 2 factor $(\mathbf{d})$-sequence, factor $(\widehat{\mathbf{d}})$-sequence *Any dual $(\mathbf{D})$-sequence of a product B-spline of two factor B-spline of degree d and $\widehat{d}$, respectively, can be split, in multiple ways, into a pair of sub sequences of length d and $\widehat{d}$. They are called, respectively, factor $(\mathbf{d})$-sequence of the first factor B-spline and factor $(\widehat{\mathbf{d}})$-sequence of the second factor B-spline. They are also called $(\mathbf{d})$-subsequence and $(\widehat{\mathbf{d}})$-subsequence, respectively, alluding to the property that they are sub string of the $(\mathbf{D})$-sequence as a string.*

An evaluation of g at a given factor $(\mathbf{d})$-sequence in the summation of Eq. (6) can be further expanded to a convex affine combination of blossom values at two other $(\mathbf{d})$-sequences, each with reduced multiplicity at a single knot. The process can be performed recursively until the elements of the lowest level of convex combination blossom $(\mathbf{d})$-sequences consist of consecutive knots from G's original knot vector, that is, until the blossom g is evaluated only at dual $(\mathbf{d})$-sequences, i.e., using only the control points of G. Exactly the same procedure is applied on the recursive evaluation of the blossom values of $\widehat{g}$.

This is basically the approach used by Ueda [Ueda 1994], which he augmented by using a strategy of partial result reuse with table look-up. However, as observed by Lee [Lee 1994], this implementation is inefficient due to its combinatorial characteristics.

4 Reformulation of B-spline Multiplication in the Blossoming Representation

Now we provide a reformulation of the blossom representation of product B-splines that allows significant reduction in the number and complexity of blossoms computed and thus enables a faster algorithm. To our best knowledge, this has neither been observed and nor used in the specific context for B-spline multiplication.

4.1 Product B-spline Represented in Multisubsets of Multisets

Because each internal breakpoint comes from one of the factors, its multiplicity must be increased by d or $\widehat{d}$ (cf. Table 1) to preserve the correct order of continuity. Hence, it has multiplicity greater than 1 (cf. Table 1). Therefore, Eq. (6) enumerates all *multisubsets of a multiset*. Omitting the extra end knots and using superscript m of a knot value u to denote the same knot value u repeated m times, the product knot vector (5) can be rewritten as

$$u_1^{m_1},u_2^{m_2},\cdots,u_s^{m_s}. \tag{7}$$

Eq. (6) can be reformulated by enumerating all the factor sequences as multisubsets of the given dual knot sequence as a multiset; specifically

$$f(u_i^{n_i},u_{i+1}^{m_{i+1}},\cdots,u_{j-1}^{m_{j-1}},u_j^{n_j})=\frac{\sum\ w\ g(u_i^{\lambda_i},u_{i+1}^{\lambda_{i+1}},\cdots,u_j^{\lambda_j})\ \widehat{g}(u_i^{\widehat{\lambda}_i},u_{i+1}^{\widehat{\lambda}_{i+1}},\cdots,u_j^{\widehat{\lambda}_j})}{\binom{d+\widehat{d}}{d}}, \tag{8}$$

where the weight w depends on $(\lambda_i,\cdots,\lambda_j)$,

$$w=\prod_{i\le\ell\le j}\binom{\lambda_\ell+\widehat{\lambda}_\ell}{\lambda_\ell}=\binom{n_i}{\lambda_i}\prod_{i<\ell<j}\binom{m_\ell}{\lambda_\ell}\binom{n_j}{\lambda_j} \tag{9}$$

is the number of ways of choosing a specific $(\mathbf{d})$-subset

$$u_i^{\lambda_i}, u_{i+1}^{\lambda_{i+1}}, \cdots, u_j^{\lambda_j}$$

from the $(\mathbf{D})$-set

$$u_i^{n_i}, u_{i+1}^{m_{i+1}}, \cdots, u_{j-1}^{m_{j-1}}, u_j^{n_j}.$$

In other words, w is the number of ways to partition the $(\mathbf{D})$-set into the $(\mathbf{d})$-subset and its complementary $(\widehat{\mathbf{d}})$-subset

$$u_i^{\widehat{\lambda}_i}, u_{i+1}^{\widehat{\lambda}_{i+1}}, \cdots, u_j^{\widehat{\lambda}_j};$$

and where the summation is taken over all partitions $\{\lambda_i, \cdots, \lambda_j\}$ such that

$$\sum_{\ell=i}^{j} \lambda_\ell = d, \qquad \text{where } \lambda_\ell \geq 0 \quad \text{for } \ell = i, \cdots, j$$

$$\sum_{\ell=i}^{j} \widehat{\lambda}_\ell = \widehat{d}, \qquad \text{where } \widehat{\lambda}_\ell \geq 0 \quad \text{for } \ell = i, \cdots, j$$

$$\lambda_\ell + \widehat{\lambda}_\ell = n_\ell \text{ if } \ell = i \text{ or } j \text{ , or } m_\ell \text{ otherwise}$$

Notice that the dual $(\mathbf{D})$-sequence, as a consecutive knot sequence from the product knot vector (7), does not necessarily have maximal multiplicity at each of its two end points, i.e., $0 < n_i \leq m_i$ and $0 < n_j \leq m_j$

Example 1 Let the first factor B-spline G (with blossom g) of degree 2 be defined by the knot vector,

$$a^2\ c\ d^2 \tag{10}$$

and a sequence of 4 control points, $\{P_i\}_{i=1}^4$, values of g on the corresponding dual (2)-sequences,

$$a^2,\ ac,\ cd,\ d^2. \tag{11}$$

That is, $P_1 = g(a^2), P_2 = g(a,c), P_3 = g(c,d), P_4 = g(d^2)$. Similarly, let the second factor B-spline $\widehat{G}$ (with blossom of $\widehat{g}$ and degree of 3) be defined by the knot vector,

$$a^3\ b\ c\ d^3$$

and a sequence of 6 control points, $\{Q_i\}_{i=1}^6$, values of $\widehat{g}$ on the corresponding dual (3)-sequences,

$$a^3,\ a^2b,\ abc,\ bcd,\ cd^2,\ d^3$$

The product, $F = G\widehat{G}$ (with blossom f), has degree $2+3=5$, and is defined by the knot vector

$$a^5\ b^3\ c^4\ d^5.$$

Its 13 control points, $\{R_i\}_{i=1}^{13}$, are computed by evaluating the blossom f on the corresponding dual (5)-sequences,

$$a^5, a^4b, a^3b^2, a^2b^3, ab^3c, b^3c^2, b^2c^3, bc^4, c^4d, c^3d^2, c^2d^3, cd^4, d^5.$$

For example, the 7th control point of the product is $R_7 = f(b^2, c^3)$. Using Eq. (6), it is evaluated as

$$\begin{aligned}
10R_7 &= \binom{5}{2} f(b^2,c^3) = 10\, f(b^2,c^3) = 10\, f(b_1,b_2,c_1,c_2,c_3) \\
&= g(b_1,b_2)\,\widehat{g}(c_1,c_2,c_3) + g(b_1,c_1)\,\widehat{g}(b_2,c_2,c_3) + g(b_1,c_2)\,\widehat{g}(b_2,c_1,c_3) + \\
&\quad g(b_1,c_3)\,\widehat{g}(b_2,c_1,c_2) + g(b_2,c_1)\,\widehat{g}(b_1,c_2,c_3) + g(b_2,c_2)\,\widehat{g}(b_1,c_1,c_3) + \\
&\quad g(b_2,c_3)\,\widehat{g}(b_1,c_1,c_2) + g(c_1,c_2)\,\widehat{g}(b_1,b_2,c_3) + \\
&\quad g(c_1,c_3)\,\widehat{g}(b_1,b_2,c_2) + g(c_2,c_3)\,\widehat{g}(b_1,b_2,c_1)
\end{aligned}$$

where all b_i's $(i=1,2)$ and all c_j's $(j=1,2,3)$ are the values b and c, respectively.

Using Eq. (8),

$$\begin{aligned}
10R_7 &= 10f(b^2,c^3) \\
&= g(b^2)\,\widehat{g}(c^3)\binom{2}{2}\binom{3}{0} + \\
&\quad g(b,c)\,\widehat{g}(b,c^2)\binom{2}{1}\binom{3}{1} + g(c^2)\,\widehat{g}(b^2,c)\binom{2}{0}\binom{3}{2} \\
&= g(b^2)\,\widehat{g}(c^3) + 6g(b,c)\,\widehat{g}(b,c^2) + 3g(c^2)\,\widehat{g}(b^2,c)
\end{aligned} \tag{12}$$

where the weights are computed from basic combinatorial formula. For example, the first term has a weight of $\binom{2}{2}\binom{3}{0}$ because the considered subsequence b^2 is formed by choosing 2 b's from a total of 2 b's in b^2c^3, and choosing no (i.e., 0) c's from a total of 3 c's in b^2c^3.

Notice that, on the right hand side of Eq. (12), factor sequences b^2, bc and c^2 of G must be expanded as affine combinations of dual (2)-sequences, and consequently $g(b^2), g(b,c)$, and $g(c^2)$ evaluate to affine combinations of control points of G. Analogously, the 3 factor (3)-sequences for $\widehat{G}$, i.e., c^3, bc^2 and b^2c, evaluate to affine combinations of control points of $\widehat{G}$. For a detailed description, see Algorithms 3 and 4. □

4.2 Reducing Combinatorial Complexity

The number of $(\mathbf{d})$-subsets of a set with cardinality $\mathbf{D}$ is

$$\binom{d+\widehat{d}}{d} = \binom{D}{d} = \binom{D}{\widehat{d}} \tag{13}$$

with lower bounds [Knuth 1997],

$$\left(\frac{D}{d}\right)^d \qquad \text{and} \qquad \left(\frac{D}{\widehat{d}}\right)^{\widehat{d}} \tag{14}$$

That is, if Eq. (6) is used directly to compute a control point of the product, blossoms of at least $max\left((D/d)^d,\ (D/\widehat{d})^{\widehat{d}}\right)$ factor sequences are evaluated. In addition, the sequences necessary for recursively evaluating the blossoms of the factor d-sequences must also be evaluated.

Note that if one of the factor is linear, then the number of subsets is D. Also, if the two factors have similar degrees, then by Eq. (14), the number of subsets, i.e., Eq. (13), will be exponential in D.

If Eq. (8) is used instead, the total number of distinct $(\mathbf{d})$-sequences (and complementary $(\widehat{\mathbf{d}})$-sequences) of a dual $(\mathbf{D})$-sequence is significantly reduced. The exponential complexity that occurs when d and $\widehat{d}$ are close to each other is reduced to polynomial complexity. Furthermore, the upper bound is constant or linear for the three commonly occurring cases discussed shortly. Very few computations are required in those cases. We show,

Theorem 1 *The number of partitions that split a $(\mathbf{D})$-sequence into a factor $(\mathbf{d})$-sequence and factor $(\widehat{\mathbf{d}})$-sequence pair ($d+\widehat{d}=D$), is less than*

$$(\sigma+1)^2\, D^{(\sigma-1)}$$

where σ is the maximal order of continuity across all breakpoints of the product B-spline of degree D.

We assume $\sigma \geq 1$ in the theorem; otherwise, we have a simple case of multiplying two piecewise Bézier functions, where the considered upper bound is trivially 1.

To prove Theorem. 1, we consider the general dual (**D**)-sequence

$$u_i^{n_i}, u_{i+1}^{m_{i+1}}, \cdots, u_{j-1}^{m_{j-1}}, u_j^{n_j} \tag{15}$$

from the product knot vector (7), where all interior knots must have the same multiplicity as that in the knot vector, while $n_i \in \{1,2,\cdots,m_i\}$ and $n_j \in \{1,2,\cdots,m_j\}$, subject to the requirement that

$$n_i + \sum_{\ell=i+1}^{j-1} m_\ell + n_j = D.$$

Because the product is at most C^σ at any breakpoint, each knot that is interior to the dual (**D**)-sequence (15) has multiplicity at least $D-\sigma$, and thus, the total multiplicity of all $(j-i-1)$ interior knots is at least

$$(j-i-1)(D-\sigma).$$

Consequently $n_i + n_j$ is at most

$$D-(j-i-1)(D-\sigma).$$

Therefore, to make the dual (**D**)-sequence (15) valid, the above expression must at least be 2 ($n_i \geq 1, n_j \geq 1$), or, equivalently,

$$(j-i-2) < \frac{\sigma-1}{D-\sigma} \leq \sigma - 1 \tag{16}$$

where the last inequality holds because $D \geq \sigma+1$, i.e., the order of continuity is at most 1 less the degree.

Now consider the possible subsequences of the dual (**D**)-sequence (15). Note that, on the one hand, the multiplicity of each interior breakpoint of the dual (**D**)-sequence (15) is at most D, while, on the other hand, at least $D-\sigma$. Thus the multiplicity at each of its two end knots is at most σ [1]. Since the multiplicity of a knot in any subsequence can be any value from 0 to the corresponding multiplicity in the dual (**D**)-sequence (i.e. D for the interior knots, and σ for each of its two end knots), the total number of ways of choosing distinct subsequences from the (**D**)-sequence is

$$\leq (\sigma+1)^2 (D+1)^{(j-i-2)} < (\sigma+1)^2 \, D^{(\sigma-1)} \tag{17}$$

where $(j-i-2)$ is one less the total number of interior knots [2], and we have used Eq. (16) to conclude the proof of Theorem 1.

For many common situations, though, the upper bound is actually either constant or linear in D because the dual (**D**)-sequences for these common situations have at most 3 distinct breakpoints. If the considered dual (**D**)-sequence has three distinct breakpoints, then $j-i-2 \equiv 1$ (cf. Eq. 15 and Eq. 16), and so the left hand side of Eq. (17) yields a linear bound in D. Similarly, if only two distinct breakpoints are involved, the upper bound in D is constant.

To show that a dual (**D**)-sequence has typically at most three distinct breakpoints, let us assume, on the contrary, that

$$u^n v^{D-\sigma_1} w^{D-\sigma_2} x^{\sigma_1+\sigma_2-D-n}$$

is a (**D**)-sequence with 4 breakpoints, u, v, w and x, sequential values from the breakpoint sequence, and σ_1 and σ_2 are the order of continuity at breaks v and w, respectively. For this to be a valid dual (**D**)-sequence,

$$\sigma_1 + \sigma_2 - D - n > 0, \quad \text{and } n > 0$$

Since $\sigma_1 \leq \sigma$ and $\sigma_2 \leq \sigma$, then $D < \sigma_1 + \sigma_2 - n < \sigma_1 + \sigma_2 \leq 2\sigma$, or

$$D < 2\sigma \tag{18}$$

i.e., the degree D of the product is less than 2σ, twice the maximum order of continuity. This turns out to be false for many common situations.

We illustrate this by considering three specific cases. Suppose $\widehat{d} = d$. Then the maximum smoothness possible for the product curve $\sigma = d-1$. Now, $D = 2d > 2(d-1)$, so there cannot be 4 knots in the (**D**)-sequence. If $\widehat{d} = d-1$ or $d-2$, the maximum smoothness of the product curve is $d-1$ again, so now $D = 2d-1$ or $2d-2$ is not less than $2(d-1)$, so again the hypothesis is contradicted, and the (**D**)-sequence has fewer than 4 distinct knots. Hence when the degrees of the two factors differ by 2 or less, then (**D**)-sequences cannot have 4 or more elements, and the number of distinct factor sequence pairs is at most linear as shown in Table 2. Note that this situation occurs exactly when $\binom{D}{d}$ (cf. Eq. (13)) exhibits its worst combinatorial complexity.

Eq. (18) is also false for the common situation where both factors have the same breakpoints. Since, in that case, $d > \sigma$ and $\widehat{d} > \sigma$. Many applications, including rational differentiation and finding cross products of first and second derivatives of curves (in curvature calculations), all have this characteristics; and thus, the involved product B-splines all have fewer than 4 distinct knots in their (**D**)-sequences.

Summarizing, we have

Observation 1 *Three common cases of dual* (**D**)*-sequences and the corresponding total number of ways of choosing distinct subsequences from these dual sequences, respectively, are as in Table 2.*

	D-Sequence	Pairs of Distinct Factor Sequences
case 1	u^D	1
case 2	$u^n v^{D-n}$	$min(d+1, \widehat{d}+1, 1+n, 1+D-n)$
case 3	$u^n v^{D-\sigma} w^{\sigma-n}$	$\leq (1+n)(1+\sigma-n) \leq (1+\frac{\sigma}{2})^2$

Table 2: **Number of Pairs of Distinct Factor Sequences**

5 Incremental Algorithm for Knot Subsequence Enumeration

In this section, we develop an algorithm for enumerating subsequences, i.e., factor sequences, of dual knot sequences of the product vector. The consecutive enumeration with respect to consecutive dual sequences of the product vector is generated incrementally; furthermore, it results in a pair of ordered lists of factor sequences, which is used later for the computation of product B-spline control points.

[1] The total length of the D-seq could not be more than D.

[2] "one less" because we can choose any interior knot multiplicity in the subsequence to be determined by the constraint that the total multiplicity of all knots, i.e. the length of the factor (**d**)-subsequence, has to be fixed value d

5.1 Enumeration of Subsequences of a Single Dual Sequence

Computing a control point of the product B-spline by Eq. (8), or equivalently, computing the blossom at the corresponding dual (**D**)-sequence, the blossoms of the two factors must be evaluated at a collection of factor knot sequences, each of which is a subset of the dual (**D**)-sequence of the product vector. Thus, subset enumeration is an essential operation. With the details of the ordering discussed in Section 5.3, the following algorithm generates subsequences of a given sequence in an ordered way. The algorithm is presented for easy illustration.

Algorithm 1 Enumerate subsequences of a sequence

Input $(\mathbf{u_i^{\lambda_i}} \cdots \mathbf{u_j^{\lambda_j}})$ (**q**)-seq

Output $\mathscr{L}_{(\mathbf{p},\,\mathbf{q})}$ list of all (**p**)-subsequences of $(\mathbf{u_i^{\lambda_i}} \cdots \mathbf{u_j^{\lambda_j}})$

Begin

1. $\mathscr{L}_{(\mathbf{p},\,\mathbf{q})} \longleftarrow \varnothing$
2. **If** $(\mathbf{u_i^{\lambda_i}} \cdots \mathbf{u_j^{\lambda_j}}) = ()$

   If $\mathbf{p} = 0$, *add empty string* $()$ *to* $\mathscr{L}_{(\mathbf{p},\,\mathbf{q})}$
3. **Else**

   For $\mathbf{k} = \mathbf{1}, \cdots, \lambda_{\mathbf{j}}$

   (a) $\mathscr{L}_{(\mathbf{p-k},\,\mathbf{q-k})} \longleftarrow$ *enumeration of* $(\mathbf{p}-\mathbf{k})$-*subseqs of* $(\mathbf{q}-\mathbf{k})$-*seq* $(\mathbf{u_i^{\lambda_i}} \cdots \mathbf{u_{j-1}^{\lambda_{j-1}}})$

   (b) **For each** $(\mathbf{p}-\mathbf{k})$-*seq* $(\mathscr{X}) \in \mathscr{L}_{(\mathbf{p-k},\,\mathbf{q-k})}$, *append* $(\mathscr{X}\ u_j^k)$ *to the end of* $\mathscr{L}_{(\mathbf{p},\,\mathbf{q})}$

End

5.2 Incremental Subsequence Enumeration for All Dual Sequences

Even though at first glance it seems that each time Eq. (8) is used to compute a new control point of the product, Algorithm 1 must be applied to enumerate all the factor sequences. a performance enhancement is readily available by observing that the subsequence enumerations of two *neighboring* dual (**D**)-sequences share most of their subsequences, and their corresponding weights as well. This is true simply because the two neighboring dual (**D**)-sequences shift only one position with respect to the product knot vector. Specifically,

1. Consider any (**d**)-subsequence

$$(u_i^{\lambda_i} \cdots u_j^{\lambda_j})$$

of the current (**D**)-sequence

$$(u_i^{n_i} u_{i+1}^{m_{i+1}} \cdots u_{j-1}^{m_{j-1}} u_j^{n_j}).$$

If $\lambda_i < n_i$, then it is still a valid (**d**)-subsequence of the next dual (**D**)-sequence of the product vector, which is

$$\begin{aligned} &(u_i^{n_i-1} u_{i+1}^{m_{i+1}} \cdots u_{j-1}^{m_{j-1}} u_j^{n_j+1}), \text{ if } n_j < m_j \\ &(u_i^{n_i-1} u_{i+1}^{m_{i+1}} \cdots u_{j-1}^{m_{j-1}} u_j^{m_j} u_{j+1}^{1}), \text{ if } n_j = m_j \end{aligned} \tag{19}$$

Furthermore, by Eq. (9), the associated weight is updated simply by a scale factor of

$$\begin{aligned} &\frac{\binom{n_i-1}{\lambda_i}}{\binom{n_i}{\lambda_i}} = \frac{n_i-\lambda_i}{n_i}, \quad \text{if } n_j = m_j, \text{ otherwise} \\ &\frac{\binom{n_i-1}{\lambda_i}}{\binom{n_i}{\lambda_i}} \frac{\binom{n_j+1}{\lambda_j}}{\binom{n_j}{\lambda_j}} = \frac{n_i-\lambda_i}{n_i} \frac{n_j+1}{n_j-\lambda_j+1} \end{aligned} \tag{20}$$

Note that the next dual (**D**)-sequence of the product vector in Eq. (19) actually starts with $u_{i+1}^{m_{i+1}}$ if $n_i = 1$, but Eq. (20) holds as well in this special case.

2. New (**d**)-subsequences of the next dual (**D**)-sequence (19), must be of the form

$$\begin{aligned} &(\mathscr{X}_1\ u_j^{n_j+1}), \text{ if } n_j < m_j, \\ &(\mathscr{X}_2\ u_{j+1}^{1}), \text{ if } n_j = m_j. \end{aligned} \tag{21}$$

where $\mathscr{X}_1$ and $\mathscr{X}_2$ are computed recursively by Algorithm 1 for subsets problems with reduced size. Specifically, $\mathscr{X}_1$ is any $(\mathbf{d}-\mathbf{n_j}-\mathbf{1})$-subsequence of the $(\mathbf{D}-\mathbf{n_j}-\mathbf{1})$-sequence

$$(u_i^{n_i-1} u_{i+1}^{m_{i+1}} \cdots u_{j-1}^{m_{j-1}}),$$

and $\mathscr{X}_2$ is any $(\mathbf{d}-\mathbf{1})$-subsequences of the $(\mathbf{D}-\mathbf{1})$-sequence

$$(u_i^{n_i-1} u_{i+1}^{m_{i+1}} \cdots u_{j-1}^{m_{j-1}} u_j^{n_j}).$$

The associated weight is initialized by a direct computation of Eq. (9)

Finally, Algorithm 2 below gives the details on incrementally enumerating all subsequence pairs (one for each factor) for each dual (**D**)-sequence of the product knot vector, where each enumerated subsequence is further tagged with an associated weight. We create ordered sub lists of the list of the ordered enumerated weighted subsequences and name them *weighted intervals*. Fig. 2 illustrates a snapshot of output of the algorithm.

Algorithm 2 Enumerate factor sequences of all dual sequences of a product B-spline

Input

$\mathbf{d}, \widehat{\mathbf{d}}, \mathbf{D}$	degrees of the factors and the product
$\mathbf{u_1^{m_1}} \cdots \mathbf{u_s^{m_s}}$	product knot vector, where $\mathbf{m_1} = \mathbf{m_s} = \mathbf{d} + \widehat{\mathbf{d}} = \mathbf{D}$

Output

$\mathscr{L}$**seq**	List of factor (**d**)**-seq** of the first factor
$\mathscr{L}\widehat{\textbf{seq}}$	List of factor $(\widehat{\mathbf{d}})$**-seq** of the second factor
$\mathscr{L}_{\mathbf{SEQ}}$	List of dual (**D**)**-seq** of the product
$\mathscr{L}$**P**	List of pairs of weighted intervals of $\mathscr{L}$**seq** and $\mathscr{L}\widehat{\textbf{seq}}$. Each such weighted interval specifies a sublist of $\mathscr{L}$**seq** or $\mathscr{L}\widehat{\textbf{seq}}$, consisting of consecutive elements, each of which is tagged with a weight. A pair of weighted intervals is created for each corresponding dual (**D**)**-seq** of the product, i.e., each corresponding element in $\mathscr{L}_{\mathbf{SEQ}}$

Begin

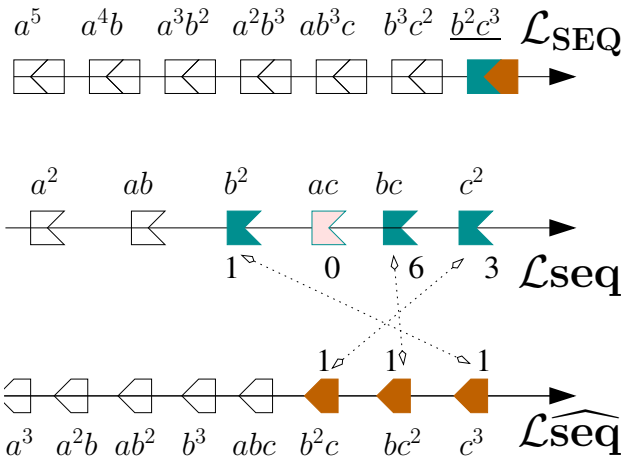

Figure 2: **Illustrating Subsequences Enumeration of Algo. 2 (cf. Example 2)**

Shown is the output of Algo. 2 at Iteration 7 for Example 1. For example, the (2)-subsequence bc (resp. the complementary (3)-subsequence bc^2) occurs 6 times out of all 10 possible (2)-subsequences (resp. (3)-subsequences) of the current (underlined) dual (5)-seq b^2c^3. Refer to Example 2 for more details.

1. *Initialization*
   (a) $\mathscr{L}seq \leftarrow \{(u_1^d)\}$, $\mathscr{L}\widehat{seq} \leftarrow \{(u_1^{\hat{d}})\}$
   (b) $\mathbf{currentSEQ} \leftarrow \mathbf{u_1^D}$
   (c) *Initialize the current pair of weighted intervals,* $(\mapsto, \widehat{\mapsto})$*, by setting the first interval to be the whole* $\mathscr{L}\mathbf{seq}$ *with its only sequence tagged with weight* $\binom{\mathbf{D}}{\mathbf{d}}$*, and the second interval to be the whole* $\mathscr{L}\widehat{\mathbf{seq}}$ *with its only sequence tagged with weight 1, respectively.*
2. *Append* **currentSEQ** *to* $\mathscr{L}_{\mathbf{SEQ}}$
3. *Append* $(\mapsto, \widehat{\mapsto})$ *to* $\mathscr{L}\mathbf{P}$.
4. **While** $(\mathbf{currentSEQ} \neq \mathbf{u_s^D})$
   (a) *Update* **currentSEQ** *using Eq.* (19)*;*
   (b) *For each factor* $(\mathbf{d})$*-sequence in the weighted interval* $\mapsto$*, set its associated weight* $\mathbf{w}$ *to* 0 *if it is not a valid* $(\mathbf{d})$*-subsequences of* **currentSEQ***; otherwise, scale* $\mathbf{w}$ *according to Eq.* (20)*.*
   (c) *Append to* $\mathscr{L}\mathbf{seq}$ *all new factor* $(\mathbf{d})$*-sequences, as subsequences of* **currentSEQ** *and generated by Eq.* (21)*.*
   (d) *Expand the interval* $\mapsto$ *by sliding forward its right end to that of* $\mathscr{L}\mathbf{seq}$ *. Each new element in* $\mapsto$ *is also tagged by a weight that is computed by Eq.*(9)*.*
   (e) *Shrink the interval* $\mapsto$ *by sliding forward its left end to the first tagged* $\mathbf{w} \neq \mathbf{0}$
   (f) *Apply the same procedure (steps (b), (c), (d), and (e)) to* $\mathscr{L}\widehat{\mathbf{seq}}$ *and* $\widehat{\mapsto}$*. However, weights are not scaled, only zeroed if necessary.*

End

5.3 Backward Lexicographic Order of Subsequence Enumeration

Although subset enumeration is a classical topic in combinatorial algorithms; the detailed Algorithm 1 is given to illustrate the specific order of the resulting enumeration.

In the recursive algorithm Algorithm 1, the loop control variable k is in ascending order, and the corresponding output is in the form of $(\mathscr{X}\ u_j^k)$ (cf. the last line of Algorithm 1; therefore, if each of the enumerated sequence is reversed (i.e., for example $b^2c = bbc$ turned into cbb), the enumeration would be in lexicographic order with respect to the alphabet that consists of distinct ascending knots. We call this *backward lexicographic order*. Furthermore, Algorithm 2 iterates dual $(\mathbf{D})$-sequences using Eq. (19), which also adds knots of higher alphabet values to the rightmost side instead of to the leftmost side, and thus has the same property. In conclusion, Algorithm 1 and Algorithm 2 generate enumerations of both the dual $(\mathbf{D})$-sequences and the two factor sequences in backward lexicographic order (cf. Fig. 2 and Example 2).

5.4 Reverse Pairing of Knot Sequences of Two Factor B-splines

In Algorithm 2, subsequence enumeration for $\widehat{G}$ is independent from that for G. It follows the same procedure, except that the associated weights serve to only indicate whether or not the sequence is really a subsequence of the $(\mathbf{D})$-sequence of the product knot vector.

It is necessary to pair $(\mathbf{d})$-sequences with $(\widehat{\mathbf{d}})$-sequences so each pair forms a partition of the $(\mathbf{D})$-sequence. By Algorithm 2, each dual $(\mathbf{D})$-sequence of the product knot vector is partitioned into a factor $(\mathbf{d})$-sequence and a complementary factor $\widehat{\mathbf{d}}$-sequence in multiple ways, with all the possible factor $(\mathbf{d})$-sequences forming an interval $\mapsto$ of $\mathscr{L}\mathbf{seq}$, and all the possible factor $\widehat{\mathbf{d}}$-sequences forming another interval $\widehat{\mapsto}$ of $\mathscr{L}\widehat{\mathbf{seq}}$. Because of the backward lexicographic order of both $\mathscr{L}\mathbf{seq}$ and $\mathscr{L}\widehat{\mathbf{seq}}$, the first $(\mathbf{d})$-sequence in $\mapsto$ must be paired with the last $\widehat{\mathbf{d}}$-sequence in $\widehat{\mapsto}$ to make a dual $(\mathbf{D})$-seq, and the process proceeds recursively with the rest $(\mathbf{d})$-sequences and $\widehat{\mathbf{d}}$-sequences that are not paired yet while skipping invalid factor sequences (i.e., those with 0 weights). Fig. 2 illustrates the reverse pairing.

We conclude this section with a detailed example.

Example 2 Algorithm 2 is applied to Example 1. Recall that the knot vectors of G, $\widehat{G}$, and F are $(a^2\ c\ d^2)$, $(a^3\ b\ c\ d^3)$, and $(a^5\ b^3\ c^4\ d^5)$, respectively.

The first few iterations are shown with output (cf. Fig. 2),

1. list $\mathscr{L}_{\mathbf{SEQ}}$ of dual (5)-seq of the product generated so far
2. list $\mathscr{L}\mathbf{seq}$ of factor (2)-seq of the factor G generated so far
3. current weighted interval $\mapsto$ of $\mathscr{L}\mathbf{seq}$, for the current dual (5)-seq that is the last (underlined) element of $\mathscr{L}_{\mathbf{SEQ}}$
4. list $\mathscr{L}\widehat{\mathbf{seq}}$ of factor (3)-seq of the factor $\widehat{G}$ generated so far
5. the current weighted interval $\widehat{\mapsto}$ of $\mathscr{L}\widehat{\mathbf{seq}}$, for the current dual (5)-seq

Iteration 1:

$$\begin{array}{ll} \mathscr{L}_{\mathbf{SEQ}} & \underline{a^5} \\ \mathscr{L}\mathbf{seq} & a^2 \\ \vdash & 10 \\ \mathscr{L}\widehat{\mathbf{seq}} & a^3 \\ \widehat{\vdash} & 1 \end{array}$$

Iteration 2:

$$\begin{array}{lll} \mathscr{L}_{\mathbf{SEQ}} & a^5 & \underline{a^4 b} \\ \mathscr{L}\mathbf{seq} & a^2 & ab \\ \vdash & 6 & 4 \\ \mathscr{L}\widehat{\mathbf{seq}} & a^3 & a^2 b \\ \widehat{\vdash} & 1 & 1 \end{array}$$

$\vdots$

Iteration 6:

$$\begin{array}{lllllll} \mathscr{L}_{\mathbf{SEQ}} & a^5 & a^4 b & a^3 b^2 & a^2 b^3 & ab^3 c & \underline{b^3 c^2} \\ \mathscr{L}\mathbf{seq} & a^2 & ab & b^2 & ac & bc & c^2 \\ \vdash & & & 3 & 0 & 6 & 1 \\ \mathscr{L}\widehat{\mathbf{seq}} & a^3 & a^2 b & ab^2 & b^3 & abc & b^2 c & bc^2 \\ \widehat{\vdash} & & & & 1 & 0 & 1 & 1 \end{array}$$

Iteration 7:

$$\begin{array}{llllllll} \mathscr{L}_{\mathbf{SEQ}} & a^5 & a^4 b & a^3 b^2 & a^2 b^3 & ab^3 c & b^3 c^2 & \underline{b^2 c^3} \\ \mathscr{L}\mathbf{seq} & a^2 & ab & b^2 & ac & bc & c^2 \\ \vdash & & & 1 & 0 & 6 & 3 \\ \mathscr{L}\widehat{\mathbf{seq}} & a^3 & a^2 b & ab^2 & b^3 & abc & b^2 c & bc^2 & \underline{c^3} \\ \widehat{\vdash} & & & & & & 1 & 1 & 1 \end{array}$$

For example, Algorithm 2 proceeds to Iteration 7 from Iteration 6 as follows.

1. **Step 4(a)** of the algorithm updates the current dual (5)-seq from b^3c^2 to b^2c^3. Refer to the first case in Eq. (19) where $n_i = 3$ and $n_j = 2$.
2. **Step 4(b)** does not zero any weights, as the valid factor (2)-sequences from iteration 6 are still valid; however, **Step 4(b)** updates all three weights, e.g., for the factor (2)-seq b^2, $\lambda_i = 2$ and $\lambda_j = 0$, so by the second case of Eq. (20),

$$\frac{n_i - \lambda_i}{n_i} \, \frac{n_j + 1}{n_j - \lambda_j + 1} = 1/3,$$

   and the new weight is scaled to 1.
3. **Step 4(c)& 4(d)** generate no new factor (2)-sequences as a valid (2)-sequence cannot have its end knot with multiplicity 3 (cf. the first case of Eq. (21)). Thus, the right end of the interval stay the same.
4. By Step 4(b), the left end of the interval stay the same as well.
5. **Step 4(b), 4(c), 4(d) & 4(e)** are repeated for the factor (3)-sequences. The (3)-seq b^3 from iteration 6 is not a valid subsequence of the current dual (5)-seq b^2c^3 and thus its weight is zeroed, and consequently the left end of the interval slides forward by two steps to the first valid (3)-seq b^2c (Notice that no scaling of the weight is required this time). Finally, a new (3)-sequence c^3 is added, and consequently, the right end of the interval slides forward by one step.

These output lists are used to compute the control points of the product. For example, according to the output of iteration 7, and by the reverse pairing property,

$$R_7 = f(b^2, c^3) = 1\, g(b^2)\, \widehat{g}(c^3) + 6\, g(b,c)\, \widehat{g}(b,c^2) + 3\, g(c^2)\, \widehat{g}(b^2,c)$$

which is seen in Eq. (12) in Example 1. The right hand side expression is further computed by Algorithms 3 and 4. See Example 3 for the computation of $g(b,c)$. □

6 Multiplication of n-Dimensional Tensor Product B-splines

Algorithm 2 in Section 5.2 is best understood for univariate B-spline functions. However, it works for n-variate or n-dimensional B-spline functions as well, when the focus is on any *single* direction or dimension. In this more general setting, a dual sequence of the knot vector in the considered dimension, corresponds to a *slice* of control points. Algorithm. 2 must be applied n times, once for each dimension, and then the resulting n list pairs are combined for the purpose of computing the blossoms at various *n-tuples* of dual knot sequences of the product knot vector. But first, in this section, we need to briefly review multiplication of 2 n-dimensions B-splines of general dimension $n > 1$.

Suppose G and $\widehat{G}$ are 2-D (i.e., bivariate) tensor product B-splines. Assume that the product knot vector in the second dimension is

$$v_1^{p_1}, v_2^{p_2}, \cdots, v_t^{p_t}, \tag{22}$$

and the degrees of the first factor, the second factor and the product are d_i, $\widehat{d}_i$ and D_i, respectively for $i = 1, 2$. The multi-dimensional analogy to Eq. (8) is

$$f(u_i^{n_i} u_{i+1}^{m_{i+1}}, \cdots, u_{j-1}^{m_{j-1}}, u_j^{n_j}; \quad v_k^{q_k} v_{k+1}^{p_{k+1}}, \cdots, v_{l-1}^{p_{l-1}}, v_l^{q_l})$$
$$= \frac{\sum w_1\, w_2 \;\; g \otimes \widehat{g}}{\binom{d_1 + \widehat{d}_1}{d_1} \binom{d_2 + \widehat{d}_2}{d_2}},$$

where

$$g \otimes \widehat{g} = g(u_i^{\lambda_i}, \cdots, u_j^{\lambda_j};\ v_k^{\eta_k}, \cdots, v_l^{\eta_l})\ \widehat{g}(u_i^{\widehat{\lambda}_i}, \cdots, u_j^{\widehat{\lambda}_j};\ v_k^{\widehat{\eta}_k}, \cdots, v_l^{\widehat{\eta}_l})$$

$$w_1 = \binom{n_i}{\widehat{\eta}_i} \prod_{i<r<j} \binom{m_r}{\lambda_r} \binom{n_j}{\widehat{\eta}_j}$$

$$w_2 = \binom{q_k}{\eta_k} \prod_{k<r<l} \binom{p_r}{\eta_r} \binom{q_l}{\eta_l}$$

$$\sum_{r=i,\cdots,j} \lambda_r = d_1, \qquad \sum_{r=k,\cdots,l} \eta_r = d_2$$

$$\begin{aligned} &(\lambda_i,\ \lambda_{i+1},\ \cdots,\ \lambda_{j-1},\ \lambda_j) \\ &+ (\widehat{\lambda}_i,\ \widehat{\lambda}_{i+1},\ \cdots,\ \widehat{\lambda}_{j-1},\ \widehat{\lambda}_j) \\ &= (n_i,\ m_{i+1},\ \cdots,\ m_{j-1},\ n_j), \end{aligned}$$

$$\begin{aligned} &(\eta_k,\ \eta_{k+1},\ \cdots,\ \eta_{l-1},\ \eta_l) \\ &+ (\widehat{\eta}_k,\ \widehat{\eta}_{k+1},\ \cdots,\ \widehat{\eta}_{l-1},\ \widehat{\eta}_l) \\ &= (q_k,\ p_{k+1},\ \cdots,\ p_{l-1},\ q_l), \end{aligned}$$

and the summation takes over all

$$\{\lambda_i, \cdots, \lambda_j\}, \quad \{\eta_k, \cdots, \eta_l\}$$

Similar equations exist for the multiplication of tensor product B-splines of any general dimensions **n**.

7 Computing Blossom Meshes of Factor B-splines

To multiply 2 n-dimensional B-splines, Algorithm 2 must be applied n times, once for each dimension. For each $i \in \{1, \cdots, n\}$, the algorithm generates $\mathscr{L}seq_i$, a list of $(\mathbf{d_i})$-sequences with respect to the i-th dimension knot vector of the first factor G where $\mathbf{d_i}$ is G's degree in the i-th direction. Taking the n such lists as vectors and iteratively forming tensor product yields an n-dimensional array, each element of which is a tuple of n factor sequences ($(\mathbf{d_1})$-seq, $\cdots$, $(\mathbf{d_n})$-seq). In this section we evaluate the blossom g of G at all such sequence tuples, therefore construct an n-dimensional array of blossom values, called a blossom mesh of G. Of course, another blossom mesh of $\widehat{G}$ is analogously constructed.

7.1 Knot Sequences as Convex Affine Combinations

A method to recursively evaluate a blossom value of g on a $(\mathbf{d})$-sequence, ***seq***, of B-spline G is to recursively expand ***seq*** into an affine combination of other 2 $(\mathbf{d})$-sequences until all the final $(\mathbf{d})$-sequences are dual $(\mathbf{d})$-sequences that correspond to control points of G. For the sake of the discussion that follows, we also call an affine combination of 2 knot $(\mathbf{d})$-sequences into another one *interpolation*.

If ***seq*** is a dual sequence of G, then no interpolation is required. Otherwise, let ***seq*** $= \mathscr{X}\, b\, \mathscr{Y}\mathscr{Z}$, where $\mathscr{X}$ and $\mathscr{Z}$ consist of consecutive knots from the original knot vector of G, while $\mathscr{X}b$ does not. Further, let a be the left neighbor knot to $\mathscr{X}$ in G's original knot vector of non-descending knots, and respectively, c be the right neighbor knot to $\mathscr{Z}$, then b can be expressed as an convex affine combination of a and c,

$$b = \frac{c-b}{c-a}a + \frac{b-a}{c-a}c = (1-\rho)a + \rho c, \text{ where } \rho = \frac{b-a}{c-a},$$

Consequently, the left and right interpolating $(\mathbf{d})$-sequences $\mathbb{L}$ and $\mathbb{R}$ are $a\mathscr{X}\mathscr{Y}\mathscr{Z}$ and $\mathscr{X}\mathscr{Y}\mathscr{Z}c$. The detailed algorithm, based on multiplicity knot vector representation, is shown in Algorithm 3 below.

Without delving into a detailed description, one final comment on the comparison of the above algorithm with the approach in [Ueda 1994], where, in our notation, the interpolating right knot c is the right neighbor of the $\mathscr{X}$, i.e., the leftmost matched string, instead of $\mathscr{Z}$, i.e., the rightmost matched string. Although there will not be any significant performance difference if associated table is used to store and retrieve the intermediate result, our method typically does have fewer levels of recursion.

Algorithm 3 Compute Interpolating Knot Sequences

Input

$(\mathbf{u_i^{\lambda_i}} \cdots \mathbf{u_j^{\lambda_j}})$	$(\mathbf{d})$-seq to be recursively interpolated
$\mathbf{u_1^{m_1}} \cdots \mathbf{u_s^{m_s}}$	Original knot vector of G, with new knots from $\widehat{G}$ inserted with 0 multiplicity

Output

$\mathbb{L}, \mathbb{R}, \rho$ 2 $(\mathbf{d})$-seqs interpolating to $(u_i^{\lambda_i} \cdots u_j^{\lambda_j})$ by ratio ρ

Begin

1. $\boldsymbol{k} \leftarrow$ *first index that* $\boldsymbol{\lambda_k > m_k}$
2. **If** $\lambda_i < m_i$ $\mathbb{L} \leftarrow (\mathbf{u_i^{\lambda_i+1}} \cdots \mathbf{u_k^{\lambda_k-1}} \cdots \mathbf{u_j^{\lambda_j}})$
   Else $\mathbb{L} \leftarrow (\mathbf{u_r^1 u_{r+1}^0} \cdots \mathbf{u_{i-1}^0 u_i^{\lambda_i}} \cdots \mathbf{u_k^{\lambda_k-1}} \cdots \mathbf{u_j^{\lambda_j}})$ *where* $\boldsymbol{r < i}$ *is the first such index that* $m_r \neq 0$.
3. **If** $\lambda_j < m_j$ $\mathbb{R} \leftarrow (\mathbf{u_i^{\lambda_i}} \cdots \mathbf{u_k^{\lambda_k-1}} \cdots \mathbf{u_j^{\lambda_j+1}})$
   Else $\mathbb{R} \leftarrow (\mathbf{u_i^{\lambda_i}} \cdots \mathbf{u_k^{\lambda_k-1}} \cdots \mathbf{u_j^{\lambda_j} u_{j+1}^0} \cdots \mathbf{u_{s-1}^0} \cdots \mathbf{u_t^1})$ *where* $\boldsymbol{t > j}$ *is the first such index that* $m_t \neq 0$.
4. $\rho \leftarrow \dfrac{u_k - u_s}{u_t - u_s}$

End

Example 3 Consider again Examples 1 and 2. Evaluation of $g(bc)$ is required for the computation of R_7 of the product. As bc is not a dual (2)-sequence of G, Algorithms 3 and 4 are used to compute $g(bc)$. For the sake of discussion, let $a = 0, b = 1, c = 2, d = 3$, then b is the affine combination of a and d with ratio of $\frac{b-a}{d-a} = 1/3$, therefore, $g(bc)$ is the affine combination of $g(ac)$ and $g(cd)$ with the same ratio of $1/3$, or equivalently, $g(bc) = 2/3P_2 + 1/3P_3$, where P_2 and P_3 are the second and the third control points of G (cf. Eq. (11)). □

7.2 Constructing Blossom Meshes

In the previous sub-section, a factor $(\mathbf{d})$-sequence is expanded to a convex affine combination of two other $(\mathbf{d})$-sequences, which are recursively expanded until reaching an expression of convex affine combinations of *dual* $(\mathbf{d})$-sequences from the original knot vector of G. There is a dual statement of evaluating blossoms to an expression of affine combinations of control points of G. Specifically, by the affine property of blossom g,

$$\begin{aligned} &g(*;\ \cdots;\ \mathscr{X}\, b\, \mathscr{Y};\ \cdots;\ *) \\ &= (1-\rho)\ g(*;\ \cdots;\ a\mathscr{X}\mathscr{Y};\ \cdots;\ *) + \\ &\qquad \rho\ g(*;\ \cdots;\ \ \mathscr{X}\mathscr{Y}c;\ \cdots;*) \end{aligned} \tag{23}$$

where $*$ denotes any knot sequences in all dimensions other than the one being considered. Notice that, for the 1-dimensional case, the equation represents an interpolation of two blossom values into the one to be evaluated, and the two interpolating blossom values have to be recursively evaluated, ultimately from the control points of G. For a general n-dimensional case, Eq. (23) represents an interpolation of two slices – that is, $(n-1)$ dimensional arrays of blossom values – into the one slice corresponding to the knot sequence $\mathscr{X}\,b\,\mathscr{Y}$ (cf. Fig 2), which means that the same interpolation is applied to each corresponding triple of blossom values of the 3 involved slices. Such an interpolation of slices are carried out iteratively for all dimensions, ultimately resulting an n-dimensional array of blossom values, all of which are evaluated at n-tuples of factor sequences as generated by Algorithm. 2.

Fig. 3 illustrates this idea and Algorithm 4 gives the details.

Algorithm 4 Computing Blossom Mesh of a Factor B-spline

Input

$\mathfrak{C}$	Control mesh of G
$\mathscr{L}\mathbf{seq}_i^0$	List of dual $(\mathbf{d}_i)$-seq in direction $i \in \{1,\cdots,n\}$
$\mathscr{L}\mathbf{seq}_i$	List of factor $(\mathbf{d}_i)$-seq in direction i from Algo. 2

Output

$\mathscr{B}$	**n**-dimensional array of g evaluated at n-tuples of factor sequences, $((\mathbf{d_1})$-seq, $\cdots$, $(\mathbf{d_n})$-seq)

Begin

1. **SrcMesh** $\leftarrow \mathfrak{C}$
2. *For each direction* $\boldsymbol{i = 1,\cdots,n}$
   (a) **DstMesh** $\leftarrow$ ***n**-dimensional empty mesh*
   (b) *For* $\boldsymbol{k = 1,\cdots,m}$, *where* $\boldsymbol{m}$ *is the total elements in* $\mathscr{L}seq_i$
      i. *Using Eq.* (23), *recursively evaluate*

$$g(\mathscr{L}seq_1[*];\ \cdots;\ \mathscr{L}seq_{i-1}[*];\ \mathscr{L}seq_i[k];\ \mathscr{L}seq^0_{i+1}[*];\ \cdots;\ \mathscr{L}seq^0_n[*])$$

      to some affine combination of slices (crossing direction $\boldsymbol{i}$*) from* **SrcMesh**
      ii. *Append the evaluated slice to* **DstMesh** *along direction* $\boldsymbol{i}$.
   (c) **SrcMesh** $\leftarrow$ **DstMesh**
3. $\mathscr{B} \leftarrow$ **DstMesh**

End

8 The Sliding Windows Algorithm

The algorithms presented construct a pair of n-dimensional arrays, i.e., meshes of blossom values corresponding to factor $(\mathbf{d})$-sequences and $(\widehat{\mathbf{d}})$-sequences of the G and $\widehat{G}$, respectively. Since these factor sequences are the ones that appear in the right hand side of Eq. (8) for computation of control points of the product F, it is now possible to compute each control point of the product directly by Eq. (8). Furthermore, because of the consistent backward lexicographic orderings of dual $(\mathbf{D})$-sequences of the product knot

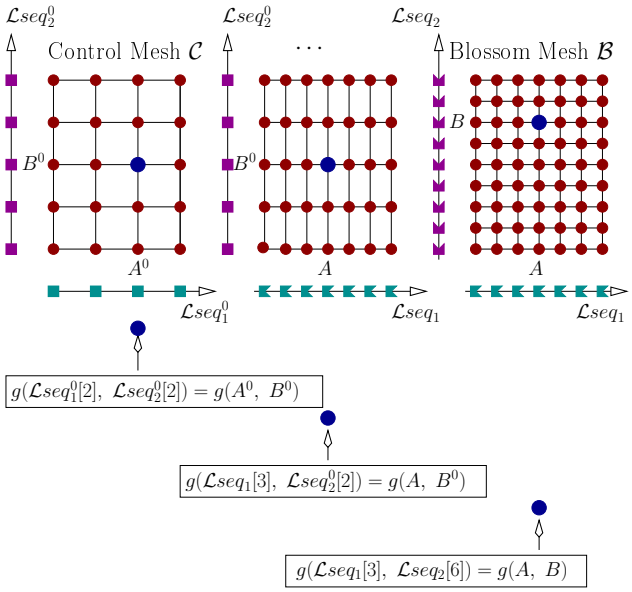

Figure 3: **Compute Blossom Mesh (cf. Algo. 4)**
Each vertical slice of blossom points in the intermediate mesh (the middle one) is interpolated from the appropriate vertical slices of control points in the original control mesh (the left one); dually, the factor $(\mathbf{d_1})$*-sequence is interpolated from appropriate dual* $(\mathbf{d_1})$*-sequences from the original knot vector in direction 1. Notice that the horizontal slices of blossom values in the intermediate mesh are still dual to dual* $(\mathbf{d_2})$*-sequences from the original knot vector in dimension 2. After applying another interpolation at direction 2, these slices are interpolated into horizontal slices in the final blossom mesh (the right one), all elements of which are now dual to both factor* $(\mathbf{d_1})$*-sequences and factor* $(\mathbf{d_2})$*-sequences. Notice that both* $Lseq_1$ *and* $Lseq_2$ *are computed by Algorithm 2.*

vector, factor $(\mathbf{d})$-sequences of G, and factor $(\widehat{\mathbf{d}})$-sequences of $\widehat{G}$, in conjunction with the associated weighted factor sequence intervals corresponding to each $(\mathbf{D})$-sequence, we are able to compute the product B-spline control points one by one in a natural linear order while iterating correspondingly in a linear way on the blossom meshes.

First we order elements in various n-dimensional arrays considered in this paper, in a natural way, by numbers $i_1 i_2 \cdots i_n$ that correspond to their multi-indices $(i_1, i_2, \cdots, i_n)$. Then, each control point can be computed from a pair of windows, that is, constructed from n copies of 1-dimensional interval pairs as computed by Algorithm 2, and that is used to access sub-arrays of the blossom meshes, respectively. Due to the reverse pairing property as discussed in Section 5.4 for 1-dimensional case, blossom values in the first window are paired with those in the second window in a reverse linear order where the linearity in the n-dimensional case is specified as above.

Details are show in Algorithm 5 below; see also illustrations in Fig. 4.

Algorithm 5 Sliding Windows Algorithm

Input

$\mathscr{B}$	Blossom Mesh from Algo. (4) applied on G
$\widehat{\mathscr{B}}$	Blossom Mesh from Algo. (4) applied on $\widehat{G}$
$\mathscr{L}\mathbf{P}_i$	List of interval pairs from Algo. (2) on i-th dimension for$i = 1, \cdots, n$.

Output

$\mathbb{C}$	n-dimensional control mesh of product B-spline

Notation

$\mathbf{sz}_i$	Total pairs in $\mathscr{L}\mathbf{P}_i$, $i = 1, \cdots, n$
$\mathbf{J}$	n-dimensional multi-index, where $1 \le \mathbf{J}_i \le \mathbf{sz}_i$

Begin

For each J

1. $\mathbb{C}[J] \leftarrow 0$
2. *Use n interval pairs $\mathscr{L}\mathbf{P}_i[\mathbf{J}_i]$, one per dimension $i \in \{1, \cdots, n\}$, to construct $\boxplus_{\mathbf{B}}$ and $\widehat{\boxplus}_{\mathbf{B}}$, two sub-arrays of $\mathscr{B}$ and $\widehat{\mathscr{B}}$, resp.*
3. *Use the associated weights of $\mathscr{L}\mathbf{P}_i[\mathbf{J}_i]$ to construct a pair of n-dimensional weight arrays $\boxplus_{\mathbf{W}}$ and $\widehat{\boxplus}_{\mathbf{W}}$, each element of which is simply the product of the corresponding n copies of tagged weights, one per dimension.*
4. *Linear iterate $\boxplus_{\mathbf{B}}$ and $\boxplus_{\mathbf{W}}$.*
   Linear reverse iterate $\widehat{\boxplus}_{\mathbf{B}}$ and $\widehat{\boxplus}_{\mathbf{W}}$.
   Let the iterated to be $\mathbf{b}$ and $\mathbf{w}$, and $\widehat{\mathbf{b}}$ and $\widehat{\mathbf{w}}$, respectively
   (a) *Go to next $\mathbf{b}$ and $\mathbf{w}$ until $\mathbf{w} \neq \mathbf{0}$*
   (b) *Go to next $\widehat{\mathbf{b}}$ and $\widehat{\mathbf{w}}$ until $\widehat{\mathbf{w}} = \mathbf{1}$*
   (c) $\mathbb{C}[J] \leftarrow C[J] + b * \widehat{b} * w$

End

9 Conclusion

We have presented in this paper the Sliding Windows Algorithm (SWA) for direct B-spline multiplication. The Sliding Windows Algorithm is based on blossoming representation of B-splines, and is conceptually straightforward. By constructing two n-dimensional meshes of blossom values of the factor n-variate B-splines, the control points of the product B-spline can be computed simply by sliding a pair of windows (as sub-arrays) on the two blossom meshes, and the blossom values in the windows are paired, multiplied, and finally affine combined into the control points. The algorithm is motivated by the efficiency issue of NURBS symbolic computation involving B-splines of high degrees and especially high dimensions, which we believe is a current trend in the CAD community due to the increasing demand on tasks beyond simple modeling, including especially inquiry, analysis and verification of the modeled curves/surfaces.

We are currently working on detailed efficiency and numerical stability comparison of the various B-spline multiplication algorithms especially including the one presented in this paper, and is also considering any possible hardware acceleration strategies for the actual implementation of the sliding windows algorithm.

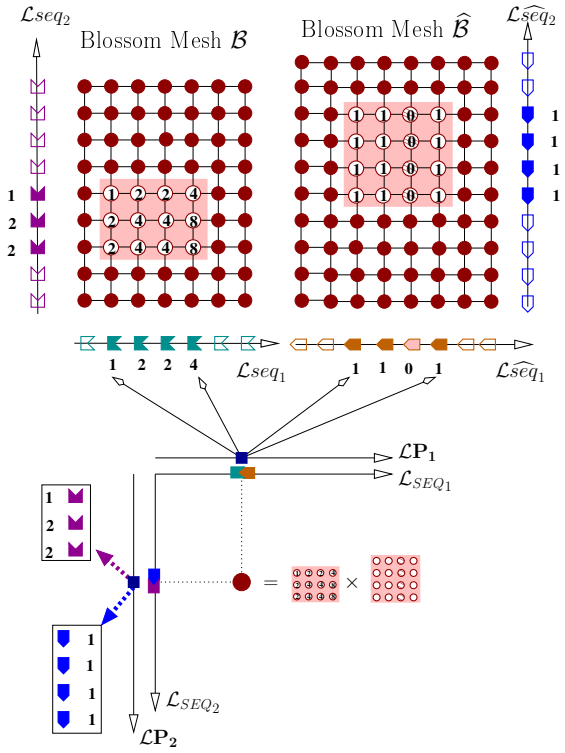

Figure 4: **Illustrating Sliding Windows Algorithm**
A control point of product B-spline F is computed by finding the pair of windows of the two blossom meshes of G and $\widehat{G}$, pairing elements in two windows in reverse order, multiplying each paired elements, and then affine combining the result with associated weights to yield the desired control point.

References

CHEN, X., COHEN, E., DAMON, J., AND COHEN, E. 2006. Tracking intersection curves of two deforming surfaces. *Springer-Verlag Lecture Notes in Computer Science 4077 (GMP 2006): 101-114*.

CHEN, X. 2007. Dynamic geometric computation by singularity detection and shape analysis. *Ph.D. Thesis Manuscript*.

COHEN, E., RIESENFELD, R. F., AND ELBER, G. 2001. *Geometric Modeling with Splines:An Introduction*, 1 ed. A K Peters.

DE BOOR, C. 1978. *A Practical Guide to Splines*, 1 ed. Springer-Verlag.

DEROSE, T., AND GOLDMAN, R. 1991. A tutorial introduction to blossoming. In *Geometric Modeling: Methods and Applications*, Springer-Verlag, H. Hagen and D. Roller, Eds., 267–286.

DEROSE, T., GOLDMAN, R. N., HAGEN, H., AND MANN, S. 1993. Functional composition algorithms via blossoming. *ACM Trans. Graph. 12*, 2, 113–135.

ELBER, G., AND COHEN, E. 1993. Second order surface analysis using hybrid symbolic and numeric operators. *ACM Transactions on Graphics 12*, 2 (April), 160–178.

ELBER, G., AND COHEN, E. 1999. A unified approach to verification in 5-axis freeform milling environments. *Computer-Aided Design 31*, 13, 795–804.

ELBER, G., AND KIM, M.-S. 2000. A computational model for nonrational bisector surfaces: Curve-surface and surface-surface bisectors. *GMP*, 364–372.

ELBER, G., AND KIM, M.-S. 2001. Geometric constraint solver using multivariate rational spline functions. *ACM Symposium on Solid Modeling and Applications*, 1–10.

ELBER, G., CHEN, X., AND COHEN, E. 2005. Mold accessibility via gauss map analysis. *ASME Transactions, Journal of Computing & Information Science in Engineering, June 2005:79-85*.

ELBER, G. 1992. Free form surface analysis using a hybrid of symbolic and numeric computation. *Ph.D. thesis, University of Utah, Computer Science Department*.

FARIN, G. 2002. *Curves and Surfaces for CAGD: A Practical Guide*, 5 ed. Academic Press.

GALLIER, J. 1998. *Curves and Surfaces in Geometric Modeling: Theory and Algorithms*, 2 ed. Morgan Kaufman.

KNUTH, D. 1997. *The Art of Computer Programming, Volume 1: Fundamental Algorithms*, 3 ed. Addison-Wesley.

LEE, E. 1994. Computing a chain of blossoms, with application to products of splines. *Computer Aided Geometric Design 11*, 6, 597–620.

MORKEN, K. M. 1991. Some identities for products and degree raising of splines. *Constructive Approximation 7*, 195–208.

PIEGL, L. A., AND TILLER, W. 1997. Symbolic operators for nurbs. *Computer-Aided Design 29*, 5, 361–368.

RAMSHAW, L. 1987. Blossoming: A connect-the-dots approach to splines. *System Research Center, DEC*.

RAMSHAW, L. 1989. Blossoms are polar forms. *Computer Aided Geometric Design 6*, 4, 323–359.

SEIDEL, H.-P. 1993. An introduction to polar forms. *IEEE Computer Graphics and Applications 13*, 38–46.

SEONG, J.-K., ELBER, G., AND KIM, M.-S. 2006. Trimming local and global self-intersections in offset curves/surfaces using distance maps. *Computer-Aided Design 38*, 3 (March), 183–193.

SEONG, J.-K., KIM, K.-J., KIM, M.-S., AND ELBER, G. 2006. Perspective silhouette of a general swept volume. *The Visual Computer 22*, 2, 109–116.

SHERBROOKE, E. C., AND PATRIKALAKIS, N. M. 1993. Computation of the solutions of nonlinear polynomial systems. *Computer Aided Geometric Design 10*, 5, 379–405.

UEDA, K. 1994. Multiplication as a general operation for splines. *Curves and Surfaces in Geometric Design*, 475–482.