

# An Algorithm for Direct Multiplication of B-splines

Xianming Chen, Richard F. Riesenfeld, and Elaine Cohen,

**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. To address the difficulties mentioned heretofore, we present the *Sliding Windows Algorithm* (SWA), a new blossoming based algorithm for the multiplication of two B-spline curves, two B-spline surfaces, or any two general multivariate B-splines.

**Note to Practitioners:** Geometric kernels in commercial CAD systems typically use B-splines to represent smooth curves and surfaces. Geometric inquiry (such as curvature) on such curves and surfaces requires the fundamental mathematical operation of multiplying two B-splines. There are a few existing algorithms in the CAD community to perform B-spline multiplication. All of them are indirect methods, in the sense of either by some sampling and interpolation strategy, or leaving the domain of B-spline representation. The only direct multiplication, reported in early 1990s, actually only solved the problem from a purely mathematical perspective. It is so inefficient as to be not feasible for any practical usage. The presented paper re-exams this initial idea of direct B-spline multiplication, and finds some simple characteristics of the apparently combinatorial problem, and designs a set of efficient algorithms, known as Sliding Window Algorithm (SWA).

**Index Terms**—NURBS multiplication, sliding windows algorithm, blossoming.

## I. 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 [16] 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 [16]), and further derived recurrence relations (Proposition 4.1 in [16]) 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 [15], 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 [8] 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 [10] further used the algorithm as a fundamental tool to symbolically query and analyze second order differential surface properties.

Ueda [22] 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) [15] and Eq. (17) [22]) is very inefficient, Lee [15] 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 [5]. 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 [17], exploiting the algorithm for multiplying Bézier curves [4], 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

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.

Xianming Chen, Richard F. Riesenfeld and Elaine Cohen are with School of Computing, University of Utah

dimension (i.e., number of variables)

Several researchers (for example [7]) 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 [22]. 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 [21], [13] 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 [12], bi-tangent curves and flecnodal curves [9], accessible regions for 5-axis machining [11], [13], offset surface self-intersection [20], etc. 5D B-spline multiplication is required in the tracking of deforming surface/surface intersection [2], and even 7D B-spline multiplication has to be performed to find the triple-point singularity of deforming surface/surface intersection [1].

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 [22], [15] 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. 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.

## II. 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.

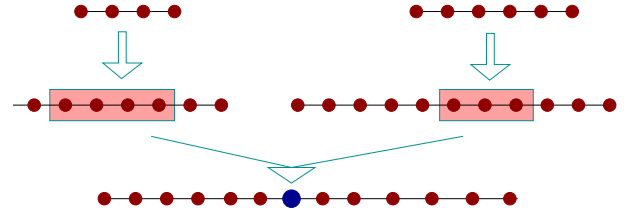


Fig. 1. Sliding Windows Algorithm Overview (univariate case)

Top row: Coefficient meshes of B-spline  $G, \hat{G}$ . Middle: Corresponding blossom meshes. Bottom: Coefficient meshes of  $F = G\hat{G}$ . The windows in the second row are used to compute the seventh control point of  $F$ . (cf. Examples 1 and 2).

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 III, Section IV presents a reformulation of the blossoming representation of a product B-spline. Section V develops an incremental algorithm of knot subsequence enumeration. Starting at Section VI 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 VII 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 VIII 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 IX presents concluding comments.

## III. B-SPLINE PRODUCTS VIA BLOSSOMING

This section reviews the basic blossoming principles used for multiplication of two B-spline functions, as noted in [15] and reported in detail in [22]. A general introduction to blossoming can be found in [18], [6], [19].

A degree  $d$  univariate B-spline function  $G(x)$  is defined by a knot vector of nondecreasing knot values

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

and a series of coefficients or control points. There is a unique symmetric multi-affine functions  $g(x_1, x_2, \dots, x_{d-1}, x_d)$ , called the blossom function of  $G$ , that diagonalise to  $G(x)$ , i.e.,  $g(x, x, \dots, x, x) = G(x)$ . By dual functional property [22], the control points of B-spline  $G$  are the blossom values of  $g$  evaluated on an order collection of knot sequences,

$$g(\underbrace{v_1, \dots, v_1}_d), g(\underbrace{v_1, \dots, v_1, v_2}_{d-1}), \dots, g(\underbrace{v_r, \dots, v_r}_d) \quad (2)$$

Each sequence at which  $g$  is evaluated in (2) has length  $d$  and is called a  $(\mathbf{d})$ -sequence or abbreviated as  $(\mathbf{d})$ -seq in this paper. Each  $(\mathbf{d})$ -seq in (2) is constructed from the knot vector (1) by deleting knots from its two ends only. To emphasize this characteristics, such a  $(\mathbf{d})$ -seq is called a  $(\mathbf{d})$ -seq of the knot

vector. For example, the (3)-seq  $bcc$  is not a (3)-seq of the knot vector  $aabcdee$ , but it is a (3)-seq of the refined knot vector  $aabccdee$  (by inserting another knot  $c$  into the original knot vector).

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

$$\underbrace{\widehat{v}_1, \dots, \widehat{v}_1}_{\widehat{n}_1 = \widehat{d}}, \underbrace{\widehat{v}_2, \dots, \widehat{v}_2}_{0 < \widehat{n}_2 \leq \widehat{d} + 1}, \dots, \underbrace{\widehat{v}_{s-1}, \dots, \widehat{v}_{s-1}}_{0 < \widehat{n}_{s-1} \leq \widehat{d} + 1}, \underbrace{\widehat{v}_s, \dots, \widehat{v}_s}_{\widehat{n}_s = \widehat{d}}, \quad (3)$$

and control points (i.e., blossom values of  $\widehat{g}$  evaluated at  $\widehat{\mathbf{d}}$ -seq of the knot vector),

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

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

$$\underbrace{u_1, \dots, u_1}_{m_1 = D}, \underbrace{u_2, \dots, u_2}_{0 < m_2 \leq D + 1}, \dots, \underbrace{u_{t-1}, \dots, u_{t-1}}_{0 < m_{t-1} \leq D + 1}, \underbrace{u_t, \dots, u_t}_{m_t = D} \quad (5)$$

where  $m_i = \max(n_j + \widehat{d}, \widehat{n}_k + d)$ , for some  $j$  and  $k$  such that  $u_i = v_j = \widehat{v}_k$ .

By the dual functional property, the control points of the product B-spline are the blossoms  $f$  of  $F$  evaluated on  $(\mathbf{D})$ -sequences of the product knot vector (5). The blossom of product B-spline is related to the blossoms of its two factor B-splines by (Eq.(17) of [22] and Eq.(22) of [15]),

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

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

In Eq. (6), one evaluation of the blossom  $f$  at a  $(\mathbf{D})$ -seq of the product B-spline, is expanded to  $2 \binom{D}{d}$  evaluations of blossom  $g$  at  $(\mathbf{d})$ -sequences, and of blossom  $\widehat{g}$  at  $(\widehat{\mathbf{d}})$ -sequences, where each pair of a  $(\mathbf{d})$ -seq and the complementary  $(\widehat{\mathbf{d}})$ -seq forms a partition (regarding sequences as sets) of the  $(\mathbf{D})$ -seq. Because the  $(\mathbf{d})$ -seq and the  $(\widehat{\mathbf{d}})$ -seq are both subsequences of the  $(\mathbf{D})$ -seq, they are also called  $(\mathbf{d})$ -subsequence and  $(\widehat{\mathbf{d}})$ -subsequence in the context of this partition.

Note that, in Eq. (6), the blossom  $g$  at a  $(\mathbf{d})$ -seq generally does not evaluate to the control point of the factor B-spline  $G$ , because the considered  $(\mathbf{d})$ -seq is generally not a  $(\mathbf{d})$ -seq of the knot vector of  $G$ . However, because of the multi-affine property of  $g$ , the evaluation can be recursively performed, ultimately resulting in certain affine combination of the blossom values of  $g$  at  $(\mathbf{d})$ -sequences of the original knot vector, that is, certain affine combination of control points of  $G$ . Similar recursive procedure applies to the evaluation of blossom  $\widehat{g}$  at any  $(\widehat{\mathbf{d}})$ -seq in Eq. (6) (cf. Algorithms 3 and 4).

This is basically the approach used by Ueda [22], which he augmented by using a strategy of partial result reuse with table look-up. However, as observed by Lee [15], this implementation is inefficient due to its combinatorial characteristics.

#### IV. REFORMULATION OF B-SPLINE MULTIPLICATION

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.

##### A. 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  $\widehat{d}$  to preserve the correct order of continuity. Hence, every breakpoint has multiplicity greater than 1. Therefore, Eq. (6) enumerates all *multisubsets of a multiset*. 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

$$u_1^{m_1}, u_2^{m_2}, \dots, u_i^{m_i}, u_{i+1}^{m_{i+1}}, \dots, u_{j-1}^{m_{j-1}}, u_j^{m_j}, \dots, u_t^{m_t}. \quad (7)$$

Eq. (6) can be reformulated by enumerating all the  $(\mathbf{d})$ -sequences (and the complementary  $(\widehat{\mathbf{d}})$ -sequences) as multisubsets of the given  $(\mathbf{D})$ -seq as a multiset; specifically  $(n_i \leq m_i$  and  $n_j \leq m_j$ , cf. Eq. (7))

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

where the weight  $w$  is the number of ways of partitioning the  $(\mathbf{D})$ -set

$$u_i^{n_i}, u_{i+1}^{m_{i+1}}, \dots, u_{j-1}^{m_{j-1}}, u_j^{n_j}, \quad \text{where } n_i + \sum_{l=i+1}^{j-1} m_l + n_j = D$$

into a  $(\mathbf{d})$ -subset

$$u_i^{\lambda_i}, u_{i+1}^{\lambda_{i+1}}, \dots, u_j^{\lambda_j}, \quad \text{where } \sum_{l=i}^j \lambda_l = d$$

and its complementary  $(\widehat{\mathbf{d}})$ -subset

$$u_i^{\widehat{\lambda}_i}, u_{i+1}^{\widehat{\lambda}_{i+1}}, \dots, u_j^{\widehat{\lambda}_j}, \quad \text{where } \sum_{l=i}^j \widehat{\lambda}_l = \widehat{d} = D - d;$$

that is,

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

The summation in Eq. (8) runs over all such partitions.

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

$$a^2 \ c \ d^2 \quad (10)$$

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

$$a^2, \ ac, \ cd, \ d^2. \quad (11)$$

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 (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 (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

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

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),

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

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),  $b^2$ ,  $bc$  and  $c^2$  are not (2)-sequences of the knot vector of  $G$ , and must be expanded as affine combinations of (2)-sequences of  $G$ 's knot vector, so that  $g(b^2), g(b, c)$ , and  $g(c^2)$  can ultimately evaluate to affine combinations of control points of  $G$ . Analogously,  $\widehat{g}(c^3), \widehat{g}(bc^2)$  and  $\widehat{g}(b^2c)$  evaluate to affine combinations of control points of  $\widehat{G}$ . For a detailed description, see Algorithms 3 and 4.  $\square$

## B. Reducing Combinatorial Complexity

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

$$\binom{d+\widehat{d}}{d} = \binom{D}{d} = \binom{D}{\widehat{d}} \quad (13)$$

with lower bounds [14],

$$(D/d)^d \quad \text{and} \quad (D/\widehat{d})^{\widehat{d}} \quad (14)$$

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

If Eq. (8) is used instead, the total number of distinct  $(\mathbf{d})$ -sequences (and complementary  $(\widehat{\mathbf{d}})$ -sequences) of a  $(\mathbf{D})$ -seq can be shown to be less than

$$(\sigma+1)^2 D^{(\sigma-1)}. \quad (15)$$

where  $\sigma$  is the maximal order of continuity across all breakpoints of the product B-spline of degree  $D$ . Notice that although the degrees of B-splines could increase for each application of B-spline multiplication ( $D = d + \widehat{d}$ ), the maximal continuity of B-splines involved will stay the same or decrease. Actually,  $\sigma$  is typically two or three as in automotive industry, and one in other manufacturing industry. Therefore, Eq. (15) effectively ensures a polynomial complexity. Furthermore, it can be shown that the upper bound is constant or linear for many commonly occurring cases of B-spline multiplication. Due to page limit, please refer to [3] for details. In this paper, we focus on presenting the form, structure, and details of the sliding windows algorithm.

## V. INCREMENTALLY ENUMERATE KNOT SUBSEQUENCES

In this section, we develop an algorithm for enumerating  $(\mathbf{d})$ -subsequences (and the complementary  $(\widehat{\mathbf{d}})$ -subsequences) of a  $(\mathbf{D})$ -seq of the product vector. The consecutive enumeration with respect to consecutive  $(\mathbf{D})$ -sequences of the product vector is generated incrementally. The incremental enumeration ultimately results in a pair of ordered lists, one for  $(\mathbf{d})$ -subsequences, and one for  $(\widehat{\mathbf{d}})$ -subsequences, which is used later for the computation of product B-spline control points.

### A. Enumeration of Subsequences of one $(\mathbf{D})$ -Sequence

Computing a control point of the product B-spline by Eq. (8), or equivalently, computing the blossom at the corresponding  $(\mathbf{D})$ -seq, the blossoms of the two factors must be evaluated at a collection of  $(\mathbf{d})$ -sequences and  $(\widehat{\mathbf{d}})$ -sequences, each of which is a subsequence of the  $(\mathbf{D})$ -seq of the product vector. Thus, subset enumeration is an essential operation. With the details of the ordering discussed in Section V-C, 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**  $(\mathbf{u}_i^{\lambda_i} \dots \mathbf{u}_j^{\lambda_j})$   $(\mathbf{q})$ -seq

**Output**  $\mathcal{L}_{(\mathbf{p}, \mathbf{q})}$  list of all  $(\mathbf{p})$ -subsequences of  $(\mathbf{u}_i^{\lambda_i} \dots \mathbf{u}_j^{\lambda_j})$

**Begin**

- 1)  $\mathcal{L}_{(\mathbf{p}, \mathbf{q})} \leftarrow \emptyset$
- 2) **If**  $(\mathbf{u}_i^{\lambda_i} \dots \mathbf{u}_j^{\lambda_j}) = ()$   
**If**  $\mathbf{p} = 0$ , add empty string  $()$  to  $\mathcal{L}_{(\mathbf{p}, \mathbf{q})}$
- 3) **Else, For**  $\mathbf{k} = 1, \dots, \lambda_j$ 
  - a)  $\mathcal{L}_{(\mathbf{p}-\mathbf{k}, \mathbf{q}-\mathbf{k})} \leftarrow$  enumeration of  $(\mathbf{p} - \mathbf{k})$ -subsequences of  $(\mathbf{q} - \mathbf{k})$ -seq  $(\mathbf{u}_i^{\lambda_i} \dots \mathbf{u}_{j-1}^{\lambda_{j-1}})$
  - b) **For each**  $(\mathbf{p} - \mathbf{k})$ -seq  $(\mathcal{X}) \in \mathcal{L}_{(\mathbf{p}-\mathbf{k}, \mathbf{q}-\mathbf{k})}$ , append  $(\mathcal{X} u_j^k)$  to the end of  $\mathcal{L}_{(\mathbf{p}, \mathbf{q})}$

**End**

*B. Enumeration of Subsequences of All  $(\mathbf{D})$ -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 subsequences. a performance enhancement is readily available by observing that the subsequence enumerations of two *neighboring*  $(\mathbf{D})$ -sequences share most of their subsequences, and their corresponding weights as well. This is true simply because the two neighboring  $(\mathbf{D})$ -sequences shift only one position with respect to the product knot vector. Specifically,

- 1) Consider any  $(\mathbf{d})$ -subsequence  $(u_i^{\lambda_i} \dots u_j^{\lambda_j})$  of the current  $(\mathbf{D})$ -seq  $(u_i^{n_i} u_{i+1}^{m_{i+1}} \dots u_{j-1}^{m_{j-1}} u_j^{n_j})$ . If  $\lambda_i < n_i$ , then it is still a valid  $(\mathbf{d})$ -subsequence of the next  $(\mathbf{D})$ -seq of the product vector, which is

$$\begin{aligned} & (u_i^{n_i-1} u_{i+1}^{m_{i+1}} \dots u_{j-1}^{m_{j-1}} u_j^{n_j+1}), \text{ if } n_j < m_j \\ & (u_i^{n_i-1} u_{i+1}^{m_{i+1}} \dots u_{j-1}^{m_{j-1}} u_j^m u_{j+1}^1), \text{ if } n_j = m_j \end{aligned} \quad (16)$$

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

$$\begin{aligned} & \binom{n_i-1}{\lambda_i} / \binom{n_i}{\lambda_i} = (n_i - \lambda_i) / n_i, \text{ if } n_j = m_j, \text{ otherwise} \\ & \binom{n_i-1}{\lambda_i} / \binom{n_i}{\lambda_i} \binom{n_j+1}{\lambda_j} / \binom{n_j}{\lambda_j} = \frac{n_i - \lambda_i}{n_i} \frac{n_j + 1}{n_j - \lambda_j + 1} \end{aligned} \quad (17)$$

Note that the next  $(\mathbf{D})$ -seq of the product vector in Eq. (16) actually starts with  $u_{i+1}^{m_{i+1}}$  if  $n_i = 1$ , but Eq. (17) holds as well in this special case.

- 2) New  $(\mathbf{d})$ -subsequences of the next  $(\mathbf{D})$ -seq (16), must be of the form

$$\begin{aligned} & (\mathcal{X}_1 u_j^{n_j+1}), \text{ if } n_j < m_j, \\ & (\mathcal{X}_2 u_{j+1}^1), \text{ if } n_j = m_j. \end{aligned} \quad (18)$$

where  $\mathcal{X}_1$  and  $\mathcal{X}_2$  are computed recursively by Algorithm 1 for subsets problems with reduced size.

Specifically,  $\mathcal{X}_1$  is any  $(\mathbf{d} - \mathbf{n}_j - \mathbf{1})$ -subsequence of the  $(\mathbf{D} - \mathbf{n}_j - \mathbf{1})$ -seq

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

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

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

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  $(\mathbf{D})$ -seq of the product knot vector, where each enumerated subsequence is further tagged with an associated weight. Fig. 2 illustrates a snapshot of output of the algorithm.

**Algorithm 2: Enumerate subsequences of all sequences of a product B-spline**

**Input**

$\mathbf{d}, \widehat{\mathbf{d}}, \mathbf{D}$  degrees of the factors and the product  
 $\mathbf{u}_1^{m_1} \dots \mathbf{u}_s^{m_s}$  product knot vector, where  $\mathbf{m}_1 = \mathbf{m}_s = \mathbf{d} + \widehat{\mathbf{d}} = \mathbf{D}$

**Output**

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

**Begin**

- 1) Initialization

- a)  $\mathcal{L}\mathbf{seq} \leftarrow \{ (u_1^{\mathbf{d}}) \}$ ,  $\mathcal{L}\widehat{\mathbf{seq}} \leftarrow \{ (u_1^{\widehat{\mathbf{d}}}) \}$
- b) **currentSEQ**  $\leftarrow \mathbf{u}_1^{\mathbf{D}}$
- c) Initialize the current pair of weighted intervals,  $(\widehat{\mathbf{I}}, \widehat{\mathbf{I}})$ , by setting the first interval to be the whole  $\mathcal{L}\mathbf{seq}$  with its only sequence tagged with weight  $\binom{\mathbf{D}}{\mathbf{d}}$ , and the second interval to be the whole  $\mathcal{L}\widehat{\mathbf{seq}}$  with its only sequence tagged with weight 1, respectively.

- 2) Append **currentSEQ** to  $\mathcal{L}\mathbf{SEQ}$
- 3) Append  $(\widehat{\mathbf{I}}, \widehat{\mathbf{I}})$  to  $\mathcal{L}\mathbf{P}$ .
- 4) **While** (**currentSEQ**  $\neq \mathbf{u}_s^{\mathbf{D}}$ )

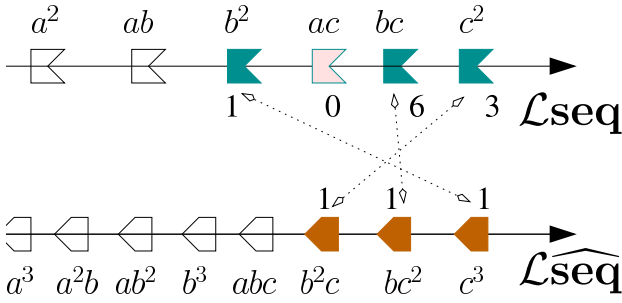


Fig. 2. **Illustrating Subsequences Enumeration of Algo. 2** (cf. Example 2) *Shown is the output of Algo. 2 at Iteration 7 for Example 1. For example, the (2)-subsequence  $bc$  (resp. the complementary (3)-subsequence  $bc^2$ ) occurs 6 times out of all 10 possible (2)-subsequences (resp. (3)-subsequences)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

- Update **currentSEQ** using Eq. (16);
- For each (**d**)-seq in the weighted interval  $\vdash$ , set its associated weight  $w$  to 0 if it is not a valid (**d**)-subsequences of **currentSEQ**; else, scale  $w$  by Eq. (17).
- Append to  $\mathcal{L}\text{seq}$  all new (**d**)-sequences, as subsequences of **currentSEQ** and generated by Eq. (18).
- Expand the interval  $\vdash$  by sliding forward its right end to that of  $\mathcal{L}\text{seq}$ . Each new element in  $\vdash$  is also tagged by a weight that is computed by Eq.(9).
- Shrink the interval  $\vdash$  by sliding forward its left end to the first tagged  $w \neq 0$
- Apply the same procedure (steps (b), (c), (d), and (e)) to  $\mathcal{L}\widehat{\text{seq}}$  and  $\widehat{\vdash}$ . However, weights are not scaled, only zeroed if necessary.

**End**

### C. Reverse Lexicographic Order of Enumerated Subsequences

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  $(\mathcal{X} u_k^i)$  (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 *reverse lexicographic order*. Furthermore, Algorithm 2 iterates (**D**)-sequences using Eq. (16), 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 (**D**)-sequences and the two subsequences in reverse lexicographic order (cf. Fig. 2 and Example 2).

### D. 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 assume values of either 1 or 0, indicating whether or not the sequence is really a subsequence of the (**D**)-seq of the product knot vector, respectively.

It is necessary to pair (**d**)-sequences with ( $\widehat{\mathbf{d}}$ )-sequences so each pair forms a partition of the (**D**)-seq. By Algorithm 2, each (**D**)-seq of the product knot vector is partitioned into a (**d**)-seq and a complementary  $\widehat{\mathbf{d}}$ -seq in multiple ways, with all the possible (**d**)-sequences forming an interval  $\vdash$  of  $\mathcal{L}\text{seq}$ , and all the possible  $\widehat{\mathbf{d}}$ -sequences forming another interval  $\widehat{\vdash}$  of  $\mathcal{L}\widehat{\text{seq}}$ . Because of the reverse lexicographic order of both  $\mathcal{L}\text{seq}$  and  $\mathcal{L}\widehat{\text{seq}}$ , the first (**d**)-seq in  $\vdash$  must be paired with the last  $\widehat{\mathbf{d}}$ -seq in  $\widehat{\vdash}$  to make a (**D**)-seq, and the process proceeds recursively with the rest (**d**)-sequences and  $\widehat{\mathbf{d}}$ -sequences that are not paired yet while skipping invalid subsequences (i.e., those with 0 weights). Fig. 2 illustrates this reverse pairing. Finally, 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  $F = G\widehat{G}$  are  $(a^2 c d^2)$ ,  $(a^3 b c d^3)$ , and  $(a^5 b^3 c^4 d^5)$ , respectively.

Iterations 1, 2, 6 and 7 are shown with output (cf. Fig. 2),

- list  $\mathcal{L}\text{SEQ}$  of (5)-seq generated so far
- list  $\mathcal{L}\text{seq}$  of (2)-seq generated so far
- current weighted interval  $\vdash$  of  $\mathcal{L}\text{seq}$ , for the current (5)-seq that is the last element of  $\mathcal{L}\text{SEQ}$
- list  $\mathcal{L}\widehat{\text{seq}}$  of (3)-seq generated so far
- the current weighted interval  $\widehat{\vdash}$  of  $\mathcal{L}\widehat{\text{seq}}$ , for the current (5)-seq

Iteration 1:

$$\begin{array}{ll} \mathcal{L}\text{SEQ} & a^5 \\ \mathcal{L}\text{seq} & a^2 \\ \vdash & 10 \end{array}$$

$$\begin{array}{ll} \mathcal{L}\widehat{\text{seq}} & a^3 \\ \widehat{\vdash} & 1 \end{array}$$

Iteration 6:

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

Iteration 2:

$$\begin{array}{ll} \mathcal{L}\text{SEQ} & a^5 \quad a^4b \\ \mathcal{L}\text{seq} & a^2 \quad ab \\ \vdash & 6 \quad 4 \\ \mathcal{L}\widehat{\text{seq}} & a^3 \quad a^2b \\ \widehat{\vdash} & 1 \quad 1 \end{array}$$

Iteration 7:

$$\begin{array}{l} \mathcal{L}_{\text{SEQ}} \quad a^5 \quad a^4b \quad a^3b^2 \quad a^2b^3 \quad ab^3c \quad b^3c^2 \quad \underline{b^2c^3} \\ \mathcal{L}_{\text{seq}} \quad a^2 \quad ab \quad b^2 \quad ac \quad bc \quad c^2 \\ \text{H} \quad \quad \quad \quad \quad 1 \quad 0 \quad 6 \quad 3 \\ \mathcal{L}_{\widehat{\text{seq}}} \quad a^3 \quad a^2b \quad ab^2 \quad b^3 \quad abc \quad b^2c \quad bc^2 \quad \underline{c^3} \\ \widehat{\text{H}} \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad 1 \quad 1 \quad 1 \end{array}$$

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

**Step 4(a)** of the algorithm updates the current (5)-seq from  $b^3c^2$  to  $b^2c^3$ . Refer to the first case in Eq. (16) where  $n_i = 3$  and  $n_j = 2$ .

**Step 4(b)** does not zero any weights, as the valid (2)-subsequences from iteration 6 are still valid; however, **Step 4(b)** updates all three weights, e.g., for the (2)-seq  $b^2$ ,  $\lambda_i = 2$  and  $\lambda_j = 0$ , so by the second case of Eq. (17),  $(n_i - \lambda_i)/n_i \cdot (n_j + 1)/(n_j - \lambda_j + 1) = 1/3$ , and the new weight is scaled to 1.

**Step 4(c)& 4(d)** generate no new (2)-subsequences as a valid (2)-subsequence cannot have its end knot with multiplicity 3 (cf. the first case of Eq. (18)). Thus, the right end of the interval stay the same.

By Step 4(b), the left end of the interval stay the same as well.

**Step 4(b), 4(c), 4(d) & 4(e)** are repeated for the (3)-sequences. The (3)-seq  $b^3$  from iteration 6 is not a valid subsequence of the current (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)-seq  $c^3$  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,

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

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  $g(b, c)$ .  $\square$

## VI. MULTIPLICATION OF $n$ -DIMENSIONAL B-SPLINES

Algorithm 2 in Section V-B is best understood for univariate B-spline functions. However, it works for  $n$ -variate or  $n$ -dimensional tensor product B-spline functions as well, when the focus is on any *single* direction or dimension. In this more general setting, a 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 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

$$v_1^{p_1}, v_2^{p_2}, \dots, v_l^{p_l}, \quad (19)$$

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

$$\begin{aligned} &f(u_i^{n_i} u_{i+1}^{m_{i+1}}, \dots, u_{j-1}^{m_{j-1}}, u_j^{n_j}; v_k^{q_k} v_{k+1}^{p_{k+1}}, \dots, v_{l-1}^{p_{l-1}}, v_l^{q_l}) \\ &= \left\{ \sum w_1 w_2 g \otimes \widehat{g} \right\} / \left\{ \binom{d_1 + \widehat{d}_1}{d_1} \binom{d_2 + \widehat{d}_2}{d_2} \right\}, \end{aligned}$$

where

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

$$w_1 = \binom{n_i}{\widehat{\eta}_i} \prod_{i < r < j} \binom{m_r}{\lambda_r} \binom{n_j}{\widehat{\eta}_j}, \quad w_2 = \binom{q_k}{\eta_k} \prod_{k < r < l} \binom{p_r}{\eta_r} \binom{q_l}{\eta_l}$$

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

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

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

and the summation takes over all  $\{\lambda_i, \dots, \lambda_j\}$ ,  $\{\eta_k, \dots, \eta_l\}$ .

Similar equations exist for the multiplication of tensor product B-splines of any general dimensions  $\mathbf{n}$ .

## VII. COMPUTE 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  $i \in \{1, \dots, n\}$ , the algorithm generates  $\mathcal{L}_{\text{seq}_i}$ , 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  $n$  subsequences  $((\mathbf{d}_1)\text{-seq}, \dots, (\mathbf{d}_n)\text{-seq})$ . 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.

### A. Knot Sequences as Convex Affine Combinations

A method to recursively evaluate a blossom value of  $g$  on a  $(\mathbf{d})$ -seq, **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  $(\mathbf{d})$ -sequences of  $G$ 's knot vector and

thus 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  $\mathbf{seq}$  is a sequence of  $G$ 's knot vector, then no interpolation is required. Otherwise, let  $\mathbf{seq} = \mathcal{X}b\mathcal{Y}\mathcal{Z}$ , where  $\mathcal{X}$  and  $\mathcal{Z}$  consist of consecutive knots from the original knot vector of  $G$ , while  $\mathcal{X}b$  does not. Further, let  $a$  be the left neighbor knot to  $\mathcal{X}$  in  $G$ 's original knot vector of non-descending knots, and respectively,  $c$  be the right neighbor knot to  $\mathcal{Z}$ , 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  $a\mathcal{X}\mathcal{Y}\mathcal{Z}$  and  $\mathcal{X}\mathcal{Y}\mathcal{Z}c$ . 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 [22], where, in our notation, the interpolating right knot  $c$  is the right neighbor of the  $\mathcal{X}$ , i.e., the leftmost matched string, instead of  $\mathcal{Z}$ , 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  $\hat{G}$  inserted with 0 multiplicity

##### Output

$\mathbb{L}, \mathbb{R}, \rho$  2  $(\mathbf{d})$ -sequences interpolating to  $(\mathbf{u}_i^{\lambda_i} \cdots \mathbf{u}_j^{\lambda_j})$  by ratio

##### Begin

- 1)  $k \leftarrow$  first index that  $\lambda_k > m_k$
- 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  $\mathbb{L} \leftarrow (\mathbf{u}_r^1 \mathbf{u}_{r+1}^0 \cdots \mathbf{u}_{i-1}^0 \mathbf{u}_i^{\lambda_i} \cdots \mathbf{u}_k^{\lambda_k-1} \cdots \mathbf{u}_j^{\lambda_j})$ , where  $r < i$  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  $\mathbb{R} \leftarrow (\mathbf{u}_i^{\lambda_i} \cdots \mathbf{u}_k^{\lambda_k-1} \cdots \mathbf{u}_j^{\lambda_j} \mathbf{u}_{j+1}^0 \cdots \mathbf{u}_{s-1}^0 \cdots \mathbf{u}_t^1)$ , where  $t > j$  is the first such index that  $m_t \neq 0$ .
- 4)  $\rho \leftarrow (u_k - u_s) / (u_t - u_s)$

##### 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  $(2)$ -seq 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)).  $\square$

## B. Constructing Blossom Meshes

In the previous sub-section, a  $(\mathbf{d})$ -seq 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  $(\mathbf{d})$ -sequences of 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  $g$ ,

$$g(*; \mathcal{X}b\mathcal{Y}; *) = (1-\rho)g(*; a\mathcal{X}\mathcal{Y}; *) + \rho g(*; \mathcal{X}\mathcal{Y}c; *) \quad (20)$$

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. (20) represents an interpolation of two slices – that is,  $(n-1)$  dimensional arrays of blossom values – into the one slice corresponding to the knot sequence  $\mathcal{X}b\mathcal{Y}$  (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 subsequences 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

$\mathcal{C}$  Control mesh of  $G$   
 $\mathcal{L}seq_i^0$  List of  $(\mathbf{d}_i)$ -seq in direction  $i \in \{1, \dots, n\}$   
 $\mathcal{L}seq_i$  List of  $(\mathbf{d}_i)$ -seq in direction  $i$  from Algo. 2

##### Output

$\mathcal{B}$   $n$ -dimensional array of  $g$  evaluated at  $n$ -tuples of subsequences,  $((\mathbf{d}_1)$ -seq,  $\dots$ ,  $(\mathbf{d}_n)$ -seq)

##### Begin

- 1)  $\mathbf{SrcMesh} \leftarrow \mathcal{C}$
- 2) For each direction  $i = 1, \dots, n$ 
  - a)  $\mathbf{DstMesh} \leftarrow n$ -dimensional empty mesh
  - b) For  $k = 1, \dots, m$ , where  $m$  is the total elements in  $\mathcal{L}seq_i$ 
    - i Using Eq. (20), recursively evaluate

$$g(\mathcal{L}seq_1[*]; \dots; \mathcal{L}seq_{i-1}[*]; \mathcal{L}seq_i[k]; \mathcal{L}seq_{i+1}[*]; \dots; \mathcal{L}seq_n^0[*])$$

to some affine combination of slices (crossing direction  $i$ ) from  $\mathbf{SrcMesh}$

- ii Append the evaluated slice to  $\mathbf{DstMesh}$  along direction  $i$ .

- c)  $\mathbf{SrcMesh} \leftarrow \mathbf{DstMesh}$

- 3)  $\mathcal{B} \leftarrow \mathbf{DstMesh}$

##### End

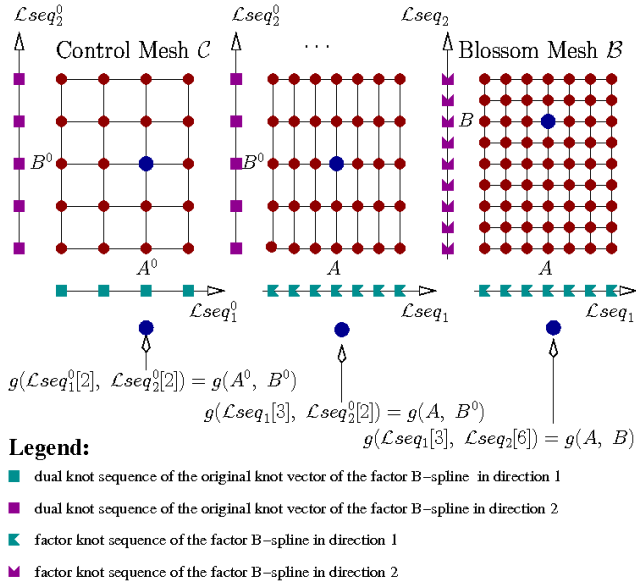


Fig. 3. Compute Blossom Mesh (cf. Algo. 4)

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  $(\mathbf{d}_1)$ -seq is interpolated from appropriate  $(\mathbf{d}_1)$ -sequences of the original knot vector in direction 1. Notice that the horizontal slices of blossom values in the intermediate mesh are still dual to  $(\mathbf{d}_2)$ -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  $(\mathbf{d}_1)$ -sequences and  $(\mathbf{d}_2)$ -sequences. Notice that both  $Lseq_1$  and  $Lseq_2$  are computed by Algorithm 2.

## VIII. THE SLIDING WINDOWS ALGORITHM

The algorithms presented construct a pair of  $n$ -dimensional arrays, i.e., meshes of blossom values of  $g$  and  $\hat{g}$  that are evaluated at  $(\mathbf{d})$ -sequences and  $(\hat{\mathbf{d}})$ -sequences, respectively. Since these subsequences 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 reverse lexicographic orderings of  $(\mathbf{D})$ -sequences of the product knot vector,  $(\mathbf{d})$ -subsequences, and  $(\hat{\mathbf{d}})$ -subsequences, in conjunction with the associated weighted sequence intervals corresponding to each  $(\mathbf{D})$ -seq, 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 V-D 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 shown in Algorithm 5, and illustrated in Fig. 4.

### Algorithm 5: Sliding Windows Algorithm

#### Input

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

#### Output

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

#### Notation

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

#### Begin

#### For each $\mathbf{J}$

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

- a) Go to next  $\mathbf{b}$  and  $\mathbf{w}$  until  $\mathbf{w} \neq \mathbf{0}$
- b) Go to next  $\hat{\mathbf{b}}$  and  $\hat{\mathbf{w}}$  until  $\hat{\mathbf{w}} = \mathbf{1}$
- c)  $\mathcal{C}[\mathbf{J}] \leftarrow \mathcal{C}[\mathbf{J}] + \mathbf{b} * \hat{\mathbf{b}} * \mathbf{w}$

#### End

We conclude this section with a simple numerical example of running SWA.

*Example 4:* Consider multi-linear functions

$$F(u) = u, \quad G(u, v) = uv, \quad H(u, v, w) = uvw.$$

Taking all the  $2^{nd}$  order partial derivatives of the squared function will yield constant functions as follow,

$$\frac{\partial^2}{\partial u^2}(H^2) = 2, \quad \frac{\partial^4}{\partial u^2 \partial v^2}(G^2) = 4, \quad \frac{\partial^8}{\partial u^2 \partial v^2 \partial w^2}(H^2) = 8. \quad (21)$$

Using blossoming principle, these multi-linear functions  $(f, g, h, \cdots)$  are easily represented by multi-dimensional B-splines of any degrees, with appropriate knot vectors and

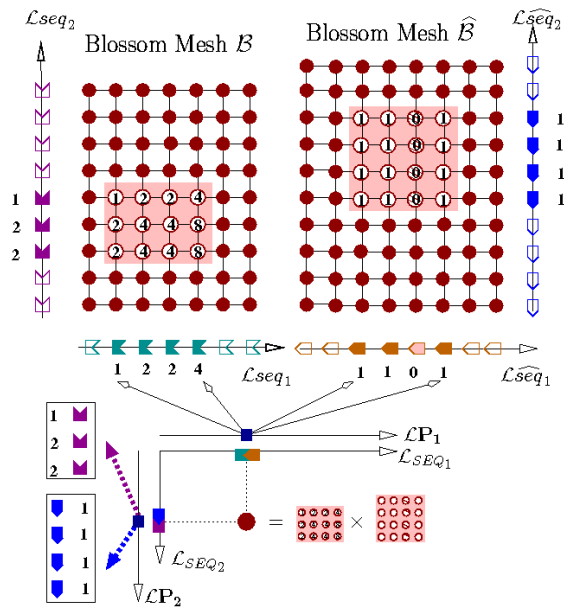


Fig. 4. Illustrating Sliding Windows Algorithm

A control point of product B-spline  $F$  is computed by finding the pair of windows of the two blossom meshes of  $G$  and  $\hat{G}$ , 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.

control points. For example, the trilinear function  $H(u, v, w) = uvw$  is the diagonalization of

$$h(u_1, u_2; v_1, v_2; w_1, w_2) = \frac{u_1 + u_2}{2} \frac{v_1 + v_2}{2} \frac{w_1 + w_2}{2}$$

Therefore, if the same knot vector 0012345... is chosen for all three dimensions, the quadratic trivariate B-spline that represents  $H(u, v, w)$  would have control points

$$P_{234} = f(0, 1; 1, 2; 2, 3) = (0 + 1)/2 * (1 + 2)/2 * (2 + 3)/2 = 15/8 = 1.875$$

and etc. The multiplication (i.e.,  $F * F$ ,  $G * G$  and  $H * H$ ) can be consequently implemented by SWA. The differentiation in Eq. (21) is another typical B-spline symbolic operation. If there were not numerical error, the control points of the final B-spline that represents  $\frac{\partial^2}{\partial u^2}(F^2)$  should all be exactly 2, or 4 and 8 for the other two cases. With uniform integer knot vectors and degrees that are powers of 2, the mathematical function  $F, G$  and  $H$  can be exactly represented by B-spline functions (for example, the machine representation of the control point 1.875 is exact), therefore, the maximal deviation of the final control points from 2 (or 4 and 8) measures the numerical error bound (the error bounding is because of the convex hull property of B-splines) caused by the multiplication algorithm SWA and the differential operations. The table below lists such error bound caused by multiplication and differentiation operations involved in Eq. (21) when  $F, G$  and  $H$  are represented by B-splines of different degrees. Also shown is running time recorded on a laptop with 1G RAM and Pentium M 1.7 GHz processor.  $\square$

degree	univariate		bivariate		trivariate	
	error	time	error	time	error	time
2	5.684e-14	0.01	7.275e-11	0.02	5.438e-07	0.58
4	3.410e-13	0.01	1.469e-09	0.06	8.651e-06	4.67
8	1.136e-13	0.01	3.715e-10	0.07	5.378e-06	15.15

## IX. CONCLUSION

We have presented the Sliding Windows Algorithm (SWA) for direct B-spline multiplication, based on blossoming representation of B-splines. 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. Future works include detailed efficiency and numerical stability comparison of the various B-spline multiplication algorithms, and possible hardware acceleration strategies for the actual implementation of the sliding windows algorithm.

## REFERENCES

- [1] Xianming Chen. An application of singularity theory to robust geometric calculation of intersections among dynamically deforming geometric objects. *School of Computing, University of Utah, Ph.D. Thesis*, 2008.
- [2] Xianming Chen, Elaine Cohen, Elaine Cohen, and James Damon. Theoretically-based algorithms for robustly tracking intersection curves of deforming surfaces. *Computer-Aided Design*, 39(5):389–397, 2007.
- [3] Xianming Chen, Richard Riesenfeld, and Elaine Cohen. Sliding windows algorithm for B-spline multiplication. *ACM Proceedings of Solid and Physical Modeling*, pages 265 – 276, 2007.
- [4] Elaine Cohen, Richard F. Riesenfeld, and Gershon Elber. *Geometric Modeling with Splines: An Introduction*. A K Peters, 1 edition, 2001.
- [5] Carl de Boor. *A Practical Guide to Splines*. Springer-Verlag, 1978.
- [6] Tony DeRose and Ron Goldman. A tutorial introduction to blossoming. In H. Hagen and D. Roller, editors, *Geometric Modeling: Methods and Applications*, pages 267–286. Springer-Verlag, 1991.
- [7] Tony DeRose, Ronald N. Goldman, Hans Hagen, and Stephen Mann. Functional composition algorithms via blossoming. *ACM Trans. Graph.*, 12(2):113–135, 1993.
- [8] Gershon Elber. Free form surface analysis using a hybrid of symbolic and numeric computation. *Ph.D. thesis, University of Utah, Computer Science Department*, 1992.
- [9] Gershon Elber, Xianming Chen, and Elaine Cohen. Mold accessibility via gauss map analysis. *ASME Transactions, Journal of Computing & Information Science in Engineering*, June 2005:79-85, 2005.
- [10] Gershon Elber and Elaine Cohen. Second order surface analysis using hybrid symbolic and numeric operators. *ACM Transactions on Graphics*, 12(2):160–178, April 1993.
- [11] Gershon Elber and Elaine Cohen. A unified approach to verification in 5-axis freeform milling environments. *Computer-Aided Design*, 31(13):795–804, 1999.

- [12] Gershon Elber and Myung-Soo Kim. A computational model for non-rational bisector surfaces: Curve-surface and surface-surface bisectors. *GMP*, pages 364–372, 2000.
- [13] Gershon Elber and Myung-Soo Kim. Geometric constraint solver using multivariate rational spline functions. *ACM Symposium on Solid Modeling and Applications*, pages 1–10, 2001.
- [14] Donald Knuth. *The Art of Computer Programming, Volume 1: Fundamental Algorithms*. Addison-Wesley, 3 edition, 1997.
- [15] E.T.Y Lee. Computing a chain of blossoms, with application to products of splines. *Computer Aided Geometric Design*, 11(6):597–620, 1994.
- [16] Knut M. Mørken. Some identities for products and degree raising of splines. *Constructive Approximation*, 7:195–208, 1991.
- [17] Les A. Piegl and Wayne Tiller. Symbolic operators for nurbs. *Computer-Aided Design*, 29(5):361–368, 1997.
- [18] Lyle Ramshaw. Blossoms are polar forms. *Computer Aided Geometric Design*, 6(4):323–359, 1989.
- [19] Hans-Peter Seidel. An introduction to polar forms. *IEEE Computer Graphics and Applications*, 13:38–46, 1993.
- [20] Joon-Kyung Seong, Gershon Elber, and Myung-Soo Kim. Trimming local and global self-intersections in offset curves/surfaces using distance maps. *Computer-Aided Design*, 38(3):183–193, March 2006.
- [21] Evan C. Sherbrooke and Nicholas M. Patrikalakis. Computation of the solutions of nonlinear polynomial systems. *Computer Aided Geometric Design*, 10(5):379–405, 1993.
- [22] Kenji Ueda. Multiplication as a general operation for splines. *Curves and Surfaces in Geometric Design*, pages 475–482, 1994.