ELSEVIER

Computer-Aided Design 35 (2003) 959–967

COMPUTER-AIDED DESIGN

www.elsevier.com/locate/cad

Algebraic manipulation in the Bernstein form made simple via convolutions

J. Sánchez-Reyes*

Department of Applied Mechanics, University of Castilla-La Mancha, ETS Ingenieros Industriales, Campus Universitario, 13071 Ciudad Real, Spain

Received 19 March 2002; received in revised form 13 January 2003; accepted 15 January 2003

Abstract

Traditional methods for algebraic manipulation of polynomials in Bernstein form try to obtain an explicit formula for each coefficient of the result of a given procedure, such us multiplication, arbitrarily high degree elevation, composition, or differentiation of rational functions. Whereas this strategy often furnishes involved expressions, these operations become trivial in terms of convolutions between coefficient lists if we employ the scaled Bernstein basis, which does not include binomial coefficients. We also carry over this scheme from the univariate case to multivariate polynomials, Bézier simplexes of any dimension and B-bases of other functional spaces. Examples of applications in geometry processing are provided, such as conversions between the triangular and tensor-product Bézier forms.

Keywords: Scaled Bernstein form; Convolution; Multivariate polynomials

1. Introduction

1.1. The Bernstein basis

In a CAGD context, a degree-n polynomial $b(u)$ over $u \in [0,1]$ is usually represented in Bernstein–Bézier form

$$b(u) = \sum_{i=0}^{n} b_i B_i^n(u), \qquad B_i^n = \binom{n}{i}(1-u)^{n-i}u^i, \tag{1.1}$$

where $B_i^n(u)$ denotes the degree-n Bernstein polynomials. This representation has become a standard, as it enjoys many advantages [4,10,17].

- Elegant geometric properties, including an intuitive geometric interpretation of the coefficients as control points.
- Existence of a de Casteljau-type algorithm: it is the only polynomial representation admitting a corner-cutting procedure that simultaneously provides subdivision, as proved by Barry and Goldman [1].
- Optimal stability: Farouki and Goodman [14] showed that we cannot find any other nonnegative polynomial basis on the unit interval with smaller condition numbers for evaluation and roots.

Formally speaking, the Bernstein basis can be identified as the *normalized B-basis* [6] of the space of algebraic polynomials. Given a space with normalized totally positive bases, among all them there exists a unique basis, called B-basis, with optimal shape-preserving properties and optimal stability [24]. In addition, such a B-basis is the only basis in the space furnishing a de Casteljau-type algorithm [23].

Despite all these advantageous properties, some systems convert their Bézier curves or surfaces to the monomial (power) form before performing certain operations, because, supposedly, such operations are more easily formulated. These conversions are not recommendable, as they sacrifice the stability advantages of the Bernstein form. In addition, as widely know, the transformation between the Bernstein and the power form is ill conditioned [13]. In conclusion, to take advantage of the intrinsic stability of the Bernstein basis, all procedures must be performed in this basis, without conversions. Thus, Tsay and Farouki

* Tel.: +34-926-295-463; fax: +34-926-295-361.
E-mail address: javier.sanchezreyes@uclm.es (J. Sánchez-Reyes).

0010-4485/03/$ - see front matter © 2003 Elsevier Science Ltd. All rights reserved.
doi:10.1016/S0010-4485(03)00021-6

[29] have recently developed specific algorithms for univariate polynomials in Bernstein form, and written an object-oriented library. An extension to tensor-product polynomials is due to Berchtold and Bowyer [2].

1.2. The scaled Bernstein basis

The scaled Bernstein form, first introduced by Farouki and Rajan [12], transfers the binomial numbers from the basis functions to the coefficients. Henceforth, we adopt the convention of denoting the scaled coefficients $\tilde{b}_i$ and basis functions $\tilde{B}_i^n(u)$ by a tilde, so that a degree-n polynomial $b(u)$ is written as

$$b(u) = \sum_{i=0}^{n} \tilde{b}_i \tilde{B}_i^n(u), \qquad \begin{aligned} \tilde{b}_i &= \binom{n}{i} b_i, \\ \tilde{B}_i^n &= (1-u)^{n-i} u^i \end{aligned} \quad . \tag{1.2}$$

The scaled basis is still a B-basis (although no longer normalized), thereby keeping the same condition numbers for evaluation or roots, because such numbers do not depend on the particular scaling adopted for the individual basis functions [11]. Hence, regarding stability there is no penalty if we carry out arithmetic in the scaled basis rather than in the customary Bernstein basis.

Farouki and Rajan [12] observed that several procedures are simplified in this scaled basis, because the often annoying binomial coefficients disappear. Indeed, the product between two polynomials reduces to the convolution between their lists of coefficients, a key property that has not been exploited in the literature. This property leads to a trivial formulation of generalized degree-elevation, addition of polynomials with different degrees and composition. In consequence, any polynomial expression involving basic algebraic operations is simply 'written out' in terms of convolutions and additions between lists of scaled coefficients. Several problems in geometry processing admit exact explicit algebraic expressions involving just these basic operations, in the customary case of polynomial curves and surfaces. Surprisingly, most works dealing with such problems try to obtain a (usually cumbersome) closed-form formula for a generic Bernstein coefficient of the result.

The paper is arranged as follows. Section 2 presents the product in the scaled Bernstein basis as a convolution and derives from this cornerstone other basic operations. In Section 3, the generalization of this idea to multivariate polynomials is outlined. Section 4 describes several examples of applications in geometry processing: conversions between rectangular and triangular Bézier representations, and obtaining the 3D representation of trimming curves or of surfaces after a free-form deformation. In Section 5 we carry over these ideas to other functional spaces that admit a B-basis with convolution structure, such as the space of trigonometric polynomials. Finally, conclusions are drawn in Chapter 6.

2. Basic operations in the scaled Bernstein basis

2.1. Multiplication as convolution

The key advantage of the scaled Bernstein basis (1.2) is that multiplying polynomials is done exactly as in the power form, using the familiar Cauchy product rule: Given two polynomials $a(u)$, $b(u)$ of degrees m, n and lists of scaled coefficients $\tilde{\mathbf{a}} = \{\tilde{a}_0, \ldots, \tilde{a}_m\}$, $\tilde{\mathbf{b}} = \{\tilde{b}_0, \ldots, \tilde{b}_n\}$, respectively, their product has degree $m+n$ and a coefficient list $\tilde{\mathbf{c}}$ obtained via discrete convolution $*$

$$c(u) = a(u)b(u) \rightarrow \tilde{\mathbf{c}} = \tilde{\mathbf{a}} * \tilde{\mathbf{b}} = \sum_{j=0}^{n} \mathrm{shift}_j(\tilde{b}_j \tilde{\mathbf{a}}), \tag{2.1}$$

where the operation shift_j means shifting a list j positions to the right, filling the leading gaps with zeros. When two lists of different lengths are added in the summation (2.1), the shortest one is padded with trailing zeros. Note that multiplication by the ith monomial u^i, which coincides with the scaled Bernstein polynomial $\tilde{B}_i^i(u)$, reduces to shifting the coefficient list

$$c(u) = u^i b(u) \rightarrow \tilde{\mathbf{c}} = \mathrm{shift}_i(\tilde{\mathbf{b}}). \tag{2.2}$$

Convolutions are implemented as built-in functions in most software packages for symbolic computation, such as Mathematica [31].

We have written the convolution (2.1) with parallelization in mind, in terms of products of lists by scalars and additions of lists. For the sake of completeness, the sequential code for the convolution follows

```
c̃ = 0
for j := 0 to n do
    for k := j to m + j do
        c̃_k = c̃_k + b̃_k ã_{k−j}
    endfor
endfor
```

This algorithm has a cost of $(m+1)(n+1)$ products and additions. For the complete multiplication in the standard form, we require conversions between the standard and scaled forms, which incur $(m+n+2)$ additional products and $(m+n+1)$ divisions by binomial coefficients. We assume that the binomial coefficients are precalculated and hence obtained at no extra cost.

The traditional formula [12] for multiplying two polynomials, of standard Bernstein coefficients $\mathbf{a} = \{a_0, \ldots, a_m\}, \mathbf{b} = \{b_0, \ldots, b_n\}$, contains a non-trivial summation rule for the generic coefficient of the product

$$c(u) = a(u)b(u) \rightarrow$$

$$c_k = \sum_{j=\max(0,k-n)}^{\min(m,k)} \frac{\binom{m}{j}\binom{n}{k-j}}{\binom{m+n}{k}} a_j b_{k-j}. \tag{2.3}$$

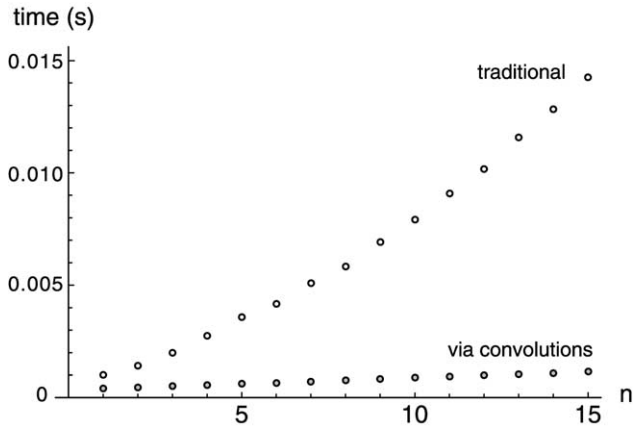

Fig. 1. CPU time for multiplying two degree-n polynomials: using the traditional summation rule versus convolutions.

Not surprisingly, the formula above has exactly the same cost than our approach, because both involve the same arithmetic operations, yet rearranged in a different way. Nevertheless, expression (2.1) is much simpler and suitable for parallelization.

Furthermore, many software packages for symbolic computation include convolutions as built-in functions. For instance, in Mathematica 4 the convolution (2.1) between two lists is simply written as `ListConvolve[a,b,{1, -1},0]`. Since these built-in functions run usually faster than any code created by the user, in such packages the implementation via convolutions results in substantial computational savings with respect to the traditional summation rule (2.3), especially for high degrees. This fact is shown in Fig. 1, which compares the computing times to multiply two degree-n polynomials, using Mathematica 4 [31] on a 400 MHz Power Mac G4 Computer.

2.2. Elevation to the kth power

Once established the equivalence between products in the scaled form and convolutions, the elevation to the kth power of a polynomial corresponds to the k-fold convolution

$$(b(u))^k \;\rightarrow\; \tilde{\mathbf{b}}^k = \underbrace{\tilde{\mathbf{b}} * \tilde{\mathbf{b}} * \cdots * \tilde{\mathbf{b}}}_{k}\,.$$

The resulting coefficients are those of the multinomial formula. For the special case of a list $\tilde{\mathbf{b}} = \{\tilde{b}_0, \tilde{b}_1\}$ ($n = 1$), elevation to the kth power yields the terms of the binomial formula

$$\tilde{\mathbf{b}}^k = \left\{ \binom{k}{i} \tilde{b}_0^{k-i} \tilde{b}_1^{i} \right\}_{i=0}^{k}, \qquad \tilde{\mathbf{b}} = \{\tilde{b}_0, \tilde{b}_1\}. \tag{2.4}$$

2.3. The unit function and generalized degree-elevation

To begin, consider how to calculate the degree-n scaled coefficients $\tilde{b}_i$ (1.2) for the unit function $b(u) = 1$. Clearly, they are the binomial coefficients from the nth row in Pascal's triangle. Moreover, it coincides with the n-fold convolution $\mathbf{1}^n$ (2.4) of the scaled (or unscaled) degree-1 coefficients $\mathbf{1}$ of the unit function

$$\left\{ \binom{n}{i} \right\}_{i=0}^{n} = \mathbf{1}^n = \mathbf{1} * \mathbf{1} * \cdots * \mathbf{1}, \qquad \mathbf{1} = \{1, 1\}. \tag{2.5}$$

Note that the expression above can be regarded as a rewriting of Pascal's formula for adjacent binomial coefficients. It also furnishes a compact notation for the conversion between standard $\mathbf{b}$ (1.1) and scaled $\tilde{\mathbf{b}}$ (1.2) coefficients

$$\tilde{\mathbf{b}} = \mathbf{1}^n \cdot \mathbf{b}, \qquad \mathbf{b} = \frac{\tilde{\mathbf{b}}}{\mathbf{1}^n}, \tag{2.6}$$

where the product or quotient between two lists is interpreted as a term-wise operation.

This simple representation of the unit function leads to a trivial formulation of arbitrary degree elevation $n \rightarrow N$ for a polynomial $b(u)$ in scaled form

$$\underset{\text{degree } n}{\tilde{\mathbf{b}}} \;\rightarrow\; \underset{\text{degree } N>n}{1^{N-n} * \tilde{\mathbf{b}}}\,, \tag{2.7}$$

which is much more compact than the formula found in standard books [10,17] or Farouki and Rajan's paper [12]. Degree-elevation is a fundamental operation when dealing with polynomials in Bernstein form. In particular, to add two polynomials $a(u)$, $b(u)$ of dissimilar degrees N, n, respectively, with $N > n$, we must perform degree-elevation up to the common maximum degree

$$c(u) = a(u) + b(u) \;\rightarrow\; \tilde{\mathbf{c}} = \tilde{\mathbf{a}} + \mathbf{1}^{N-n} * \tilde{\mathbf{b}}.$$

2.4. Composition

Suppose that we are given a certain polynomial expression $f(p_i(u))$ involving additions and products of polynomials $p_i(u)$, and that we want to compute the scaled coefficients of the result. From the fundamental results reached in the preceding sections, we simply rewrite f, following a three-step procedure

1. Replace in expression f the polynomials $p_i(u)$ by their lists of scaled coefficients.
2. Replace products/additions between polynomials with convolutions/additions between lists, performing degree-elevation when two polynomials of different degrees are added.
3. Evaluate the resulting expression using any suitable evaluation algorithm.

We can apply this procedure for the composition $c(v) = b(u(v))$ of two polynomials $b(u)$, $u(v)$ of degrees

n, p, respectively, in scaled form. To compute the scaled coefficients $\tilde{\mathbf{c}}$ of $c(v)$, first we substitute in the definition (1.2) of $b(u)$ the variable u by the scaled coefficients $\tilde{\mathbf{u}}$, and then make the replacements indicated in step 2

$$\tilde{\mathbf{c}} = \sum_{i=0}^{n} \tilde{b}_i (\mathbf{1}^p - \tilde{\mathbf{u}})^{n-i} * \tilde{\mathbf{u}}^i. \tag{2.8}$$

Finally, we evaluate the resulting expression. For instance, we could employ the de Casteljau algorithm: beginning with $\tilde{\mathbf{b}}_i^0 = \{b_i\}$, the recursive algorithm

$$\tilde{\mathbf{b}}_i^k = (\mathbf{1}^p - \tilde{\mathbf{u}}) * \tilde{\mathbf{b}}_i^{k-1} + \tilde{\mathbf{u}} * \tilde{\mathbf{b}}_{i+1}^{k-1}, \qquad \begin{matrix} k = 1, \ldots, n \\ i = 0, \ldots, n-k \end{matrix},$$

yields the sought coefficients $\tilde{\mathbf{c}} = \tilde{\mathbf{b}}_0^n$. In fact, we could employ any other strategy for evaluating standard Bézier curves, such as a more efficient nested algorithm, with the flavour of Horner's scheme, described in Farin's textbook [10].

We must remark the conceptual and formal simplicity of expression (2.8) compared with other existing approaches. Farouki and Rajan [12] employ the scaled form, but they try to obtain a explicit formula form each generic coefficient by applying the multinomial expansion, thereby leading to complex expressions that involve non-trivial cycling through a set of indices. By similar reasons, the method by Piegl and Tiller [25] for composing polynomial functions has no straightforward implementation. DeRose's product algorithm [8], implemented by Tsay and Farouki in their library [29], avoids closed forms for the final coefficients and computes them recursively filling a tetrahedral array. However, our approach is still considerably simpler thanks to the systematic use of the scaled basis. A variant [8] of the product algorithm, based on the blossoming theory, provides some geometric meaning, but at the cost of even more complex maths.

2.5. Differentiation

Obtaining the derivative $b'(u)$ of a polynomial $b(u)$ in the scaled basis (1.2) is only slightly more involved than in the standard basis

$$\tilde{b}_i' = (i+1)\tilde{b}_{i+1} - (n-i)\tilde{b}_i, \qquad i = 0, \ldots, n-1.$$

Needless to say, as an alternative, we could differentiate $b(u)$ in the Bernstein basis and then convert the resulting coefficients $\mathbf{b}'$ to the scaled basis $\tilde{\mathbf{b}}'$.

The superiority of the scaled basis becomes clear when we compute the derivative $r'(u)$ of a degree-n rational function $r(u)$:

$$r(u) = \frac{b(u)}{w(u)} \overset{d/du}{\longrightarrow} r'(u) = \frac{b'(u)w(u) - b(u)w'(u)}{w^2(u)}.$$

The derivative is a rational function of degree $2n$, whose numerator and denominator have scaled coefficients

$$\frac{(\tilde{\mathbf{b}}' * \tilde{\mathbf{w}} - \tilde{\mathbf{b}} * \tilde{\mathbf{w}}') * \mathbf{1}}{\tilde{\mathbf{w}} * \tilde{\mathbf{w}}},$$

respectively. Once again, this expression is dramatically more compact than that obtained in the standard Bernstein form for each individual coefficient by Kim et al. [19].

2.6. Conversion between power and Bernstein form

Consider first the conversion of a degree-n polynomial $a(u)$ in monomial (power) form

$$a(u) = \sum_{i=0}^{n} u^i a_i$$

to the scaled Bernstein representation $\tilde{\mathbf{b}}$ (1.2). Since $u^i = \tilde{B}_i^i(u)$, the solution is trivial in terms of degree-elevation. Just degree-raise (2.7) these functions up to degree n, taking into account the simplification (2.2), and collect the resulting terms

$$\tilde{\mathbf{b}} = \sum_{i=0}^{n} \mathrm{shift}_i(a_i \mathbf{1}^{n-i}).$$

Regarding the inverse conversion, from scaled Bernstein to monomial, expand in the definition (1.2) the powers of $(1-u)$, a polynomial with monomial coefficients $\{1, -1\}$, and apply that the monomial coefficients of a product are also computed via convolution

$$\{a_i\}_{i=0}^n = \sum_{i=0}^{n} \mathrm{shift}_i(\tilde{\mathbf{b}}_i \{1, -1\}^{n-i}).$$

3. Extension to multivariate polynomials

In this section the framework for univariate polynomials is carried over to multivariate polynomials, by applying two key ideas

1. Define the suitable multivariate scaled form by absorbing the binomial numbers into the coefficients.
2. Generalize the convolution $*$ (2.1) to multidimensional lists.

3.1. Tensor-product polynomials

A bivariate polynomial $b(u, v)$ of degree (m, n), over a rectangular domain $(u, v) \in [0, 1] \times [0, 1]$, is written in tensor-product Bernstein form as

$$b(u, v) = \sum_{i=0}^{m} \sum_{j=0}^{n} b_{ij} B_{i,j}^{m,n}(u, v), \tag{3.1}$$

$$B_{i,j}^{m,n}(u, v) = B_i^m(u) B_j^n(v).$$

In the alternative scaled form

$$b(u,v)=\sum_{i=0}^{m}\sum_{j=0}^{n}\tilde{b}_{ij}\tilde{B}_{i,j}^{m,n}(u,v),$$
$$\tilde{b}_{ij}=\binom{m}{i}\binom{n}{j}b_{ij} \qquad (3.2)$$
$$\tilde{B}_{i,j}^{m,n}(u,v)=(1-u)^{m-i}u^{i}(1-v)^{n-j}v^{j}$$

Now the scaled coefficients are represented as a $m\times n$ matrix $\tilde{\mathbf{b}}$, and the product between two polynomials is computed through bivariate convolution

$$c(u,v)=a(u,v)b(u,v)\rightarrow$$
$$\tilde{\mathbf{c}}=\tilde{\mathbf{a}}*\tilde{\mathbf{b}}=\sum_{i=0}^{m}\sum_{j=0}^{n}\mathrm{shift}_{i,j}(b_{i,j}\mathbf{a}). \qquad (3.3)$$

We initialize to zero a matrix $\mathbf{c}$ and add to it, starting at the (i,j)th position, submatrices $b_{i,j}\mathbf{a}$ padded with zeros.

The bivariate convolution (3.3) furnishes the analogue of the univariate expressions (2.2) and (2.5)–(2.7)

- Product by (i,j)th power basis function

  $$c(u,v)=u^{i}v^{j}b(u,v)\rightarrow\tilde{\mathbf{c}}=\mathrm{shift}_{i,j}(\tilde{\mathbf{b}}).$$

- Degree-(m,n) scaled coefficients $\mathbf{1}^{m,n}$ of the unit function $b(u,v)=1$. If $\mathbf{1}^{m},(\mathbf{1}^{n})^{\perp}$ are column and row vectors, respectively, then

  $$\mathbf{1}^{m,n}=\mathbf{1}^{m}*(\mathbf{1}^{n})^{\perp}=\left[\binom{m}{i}\binom{n}{j}\right]_{i,j}$$

- Transformation between standard $\mathbf{b}$ and scaled $\tilde{\mathbf{b}}$ coefficients, via term-wise division

  $$\mathbf{b}=\frac{\tilde{\mathbf{b}}}{\mathbf{1}^{m,n}}.$$

- Arbitrary degree elevation $(m,n)\rightarrow(M,N)$

  $$\underset{\text{degree}(m,n)}{\tilde{\mathbf{b}}}\rightarrow\underset{\text{degree}(M,N)}{\mathbf{1}^{\Delta m,\Delta n}*\tilde{\mathbf{b}}},\qquad\begin{matrix}\Delta m=M-m>0\\ \Delta n=N-n>0\end{matrix}.$$

The generalization to multivariate polynomials of an arbitrary number k of variables is a straightforward exercise with k-dimensional lists.

3.2. Polynomials over simplexes

Bivariate polynomials can also be defined in barycentric coordinates $\mathbf{u}$ over a triangular domain

$$\mathbf{u}=(u,v,w),\qquad\begin{cases}u+v+w=1\\ u,v,w>0\end{cases}. \qquad (3.4)$$

Using this simplified notation, a degree-n polynomial $p(\mathbf{u})$ in the Bernstein basis takes the form

$$p(\mathbf{u})=\sum_{|i|=n}p_{\mathbf{i}}B_{\mathbf{i}}^{n}(\mathbf{u}),\qquad\begin{cases}\mathbf{i}=\{i,j,k\},i,j,k>0\\ |\mathbf{i}|=i+j+k\end{cases}, \qquad (3.5)$$

where $B_{\mathbf{i}}^{n}(\mathbf{u})$ denotes the Bernstein polynomials over triangles

$$B_{\mathbf{i}}^{n}(\mathbf{u})=\binom{n}{\mathbf{i}}u^{i}v^{j}w^{k},\qquad\binom{n}{\mathbf{i}}=\frac{n!}{i!j!k!}, \qquad (3.6)$$

and the coefficients $p_{\mathbf{i}}$ are arranged in a triangular fashion

$$\begin{matrix}&&&p_{0n0}&&&\\ &&p_{0,n-1,1}&&p_{1,n-1,0}&&\\ &&&\vdots&&&\\ p_{0,0,n}&p_{1,0,n-1}&\cdots&&p_{n-1,0,1}&p_{n,0,0}\end{matrix}.$$

In the scaled representation, $p(\mathbf{u})$ (3.5) is rewritten as

$$p(\mathbf{u})=\sum_{|\mathbf{i}|=n}\tilde{p}_{\mathbf{i}}\tilde{B}_{\mathbf{i}}^{n}(\mathbf{u}),\qquad\begin{cases}\tilde{p}_{\mathbf{i}}=\binom{n}{\mathbf{i}}p_{\mathbf{i}}\\ \tilde{B}_{\mathbf{i}}^{n}(\mathbf{u})=u^{i}v^{j}w^{k}\end{cases}. \qquad (3.7)$$

The product between two polynomials polynomials $p(\mathbf{u})$, $q(\mathbf{u})$ of degrees m,n, respectively, has degree $m+n$ and a triangular matrix $\tilde{\mathbf{c}}$ of scaled coefficients obtained via a triangular convolution

$$c(\mathbf{u})=p(\mathbf{u})q(\mathbf{u})\rightarrow\tilde{\mathbf{c}}=\tilde{\mathbf{p}}*\tilde{\mathbf{q}}=\sum_{|\mathbf{i}|=m}\mathrm{shift}_{\mathbf{i}}(p_{\mathbf{i}}\mathbf{q}).$$

We initialize to zero a $(m+n)$-sized matrix $\mathbf{c}$ and add to it, starting at the $\mathbf{i}$th position, triangular submatrices $p_{\mathbf{i}}\mathbf{q}$ padded with zeroes. The degree-n representation of unit function has now scaled coefficients

$$\mathbf{1}^{n}=\mathbf{1}*\mathbf{1}*\cdots*\mathbf{1},\qquad\mathbf{1}=\begin{matrix}&1&\\ 1&&1\end{matrix}.$$

In consequence, formulae for the transformation between standard $\mathbf{p}$ and scaled $\tilde{\mathbf{p}}$ coefficients and for arbitrary degree elevation are exactly those (2.6),(2.7) of the univariate case

$$\mathbf{p}=\frac{\tilde{\mathbf{p}}}{\mathbf{1}^{n}},\qquad\underset{\text{degree }n}{\tilde{\mathbf{p}}}\rightarrow\underset{\text{degree }N>n}{\mathbf{1}^{N-n}*\tilde{\mathbf{p}}}. \qquad (3.8)$$

Triangles are the simplexes in two-dimensional space. We can generalize the framework to Bézier simplexes in 3D (over tetrahedral domains) or in spaces of arbitrary dimension, by defining general barycentric coordinates. deBoor [7] coined the term *B-forms* to denote such Bézier simplexes. Once again, expression (3.8) applies, which is considerably simpler than the scheme by Trump and Prautzsch [30] for arbitrary high degree elevation.

4. Examples of applications in geometry processing

The composition of polynomials is a fundamental tool in many operations in geometry processing [9], such as conversions between tensor-product and Bézier simplex forms and computing exact Bézier representations of trimming curves or Bézier curves and surfaces that have undergone free-form Bézier deformations. The same three-step procedure described in Section 2.3 for the univariate case holds for compositions involving Bézier rectangles $b(u,v)$ or triangles $p(\mathbf{u})$, as illustrated in this section.

4.1. Conversion from Bézier rectangle to Bézier triangles

We address now the conversion from a Bézier rectangle $b(u,v)$ (3.2) of degree (m,n) to two degree-$(m+n)$ triangles, $(u+v)\le 1$ and $(u+v)\ge 1$ (Fig. 2). For instance, consider the triangle $(u+v)\le 1$. As a simpler alternative to the conversion formula by Goldman and Filip [16], just rewrite $b(u,v)$ in barycentric coordinates $\mathbf{u}$ (3.4)

$$p(\mathbf{u})=\sum_{i=0}^{m}\sum_{j=0}^{n}\tilde{b}_{ij}c(\mathbf{u})^{m-i}u^i d(\mathbf{u})^{n-j}v^j,$$

$$\begin{cases} c(\mathbf{u})=1-u=v+w \\ d(\mathbf{u})=1-v=u+w \end{cases}.$$

The scaled coefficients $\tilde{\mathbf{p}}$ of the triangular representation are hence given by

$$\tilde{\mathbf{p}}=\sum_{i=0}^{m}\sum_{j=0}^{n}\mathrm{shift}_{i,j,0}(\tilde{b}_{ij}\mathbf{c}^{m-i}*\mathbf{d}^{n-j}),$$

$$\mathbf{c}=\begin{matrix}1 & \\ 1 & 0\end{matrix},\quad \mathbf{d}=\begin{matrix}0 & \\ 1 & 1\end{matrix}.$$

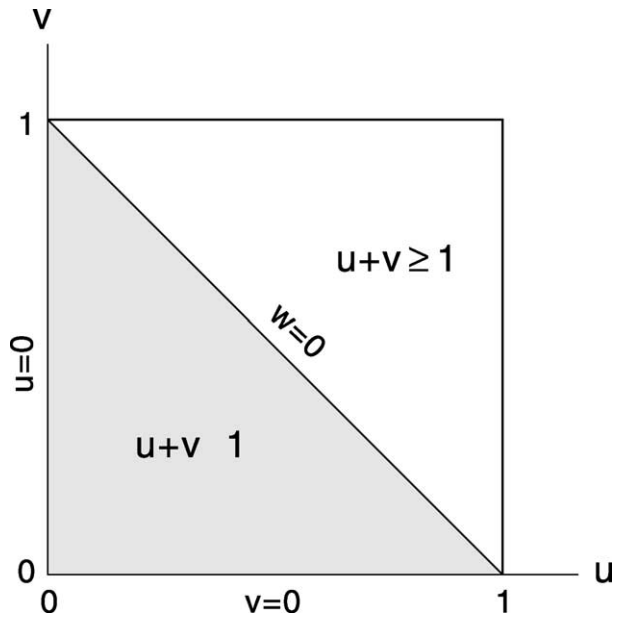

Fig. 2. Splitting a Bézier rectangle into two triangles.

4.2. Conversion from Bézier triangles to Bézier rectangles

To convert a degree-n polynomial $p(\mathbf{u})$ (3.7) over a triangle to the tensor-product form (3.2) of degree (n,n), simply rewrite $p(\mathbf{u})$ as a bivariate polynomial $b(u,v)$:

$$b(u,v)=\sum_{|\mathbf{i}|=n}\tilde{p}_{\mathbf{i}}u^i v^j(w(u,v))^k,$$
$$w(u,v)=(1-u-v), \tag{4.1}$$

by expressing $w(u,v)$ as a degree-(1,1) bivariate function. Therefore, the scaled coefficients $\tilde{\mathbf{b}}$ are

$$\tilde{\mathbf{b}}=\sum_{|\mathbf{i}|=n}\tilde{p}_{\mathbf{i}}\mathbf{1}^{j,i}*\mathrm{shift}_{i,j}(\mathbf{w}^k),\qquad \mathbf{w}=\begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix},$$

where the unit $\mathbf{1}^{i,j}$ is needed to degree-elevate the terms of the summation to the common degree (n,n). The expression above is much more compact than the formula by Brueckner [5] for each individual coefficient. Note that, to keep the original triangular domain (3.4), we must trim the rectangular domain (u,v) to the region $(u+v)\le 1$.

An alternative method proposed by Hu [18], which avoids trimming and its associated problems, is based on splitting the original Bézier triangle $p(\mathbf{u})$ into three rectangles, in a star-like configuration (Fig. 3). We hence choose an inner common corner point $\mathbf{c}$ for these three rectangles and three additional corners $\mathbf{c}_{u=0},\mathbf{c}_{v=0},\mathbf{c}_{w=0}$ on the three edges, denoted by the subscript, of the triangular domain. For instance, consider how to express in tensor-product form $b(s,t)$, $(s,t)\in[0,1]\times[0,1]$, the polynomial over the quadrilateral with corners

$$\begin{matrix}\mathbf{O}=(0,0), & \mathbf{c}_{u=0}=(0,v_0)\\ \mathbf{c}_{v=0}=(u_0,0) & \mathbf{c}=(u_1,v_1)\end{matrix},$$

where we have employed coordinates (u,v). The key idea is to parameterize the quadrilateral using the bilinear map

$$(u,v)=\{1-s,s\}\begin{bmatrix}\mathbf{O} & \mathbf{c}_{u=0}\\ \mathbf{c}_{v=0} & \mathbf{c}\end{bmatrix}\begin{Bmatrix}1-t\\ t\end{Bmatrix},$$

rewrite $p(\mathbf{u})$ in tensor-product form and finally compute the composition $b(s,t)=p(u(s,t),v(s,t))$. The resulting scaled coefficients $\tilde{\mathbf{b}}$ are

$$\tilde{\mathbf{b}}=\sum_{|\mathbf{i}|=n}\tilde{p}_i\mathbf{u}^i*\mathbf{v}^j*(\mathbf{1}^{1,1}-\mathbf{u}-\mathbf{v})^k,$$

$$\mathbf{u}=\begin{bmatrix}0 & 0\\ u_0 & u_1\end{bmatrix},\quad \mathbf{v}=\begin{bmatrix}0 & v_0\\ 0 & v_1\end{bmatrix}.$$

The remaining two quadrilaterals would be handled analogously.

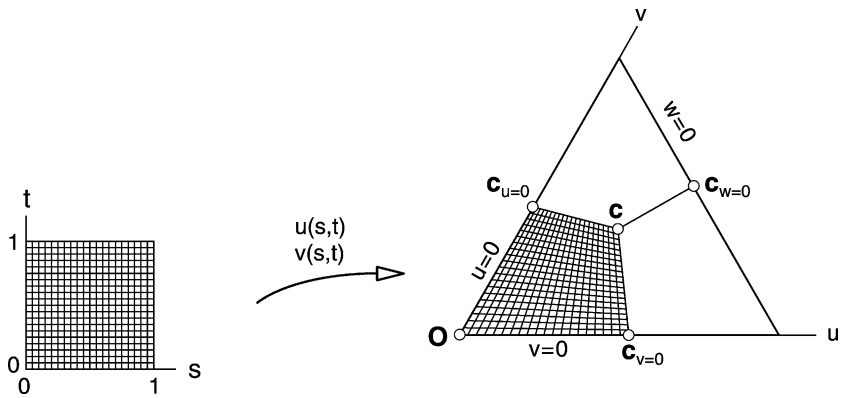

Fig. 3. Splitting a Bézier triangle into three rectangles.

4.3. 3D expression of trimming curves

As sketched in Fig. 4, composition is needed to obtain the 3D representation $\mathbf{c}_{3D}(t)$ of a trimming curve $\mathbf{c}_{2D}(t) = (u(t), v(t))$. This 2D curve is defined on the domain $(u, v) \in [0, 1] \times [0, 1]$ of a degree-(m, n) rectangular Bézier patch $\mathbf{b}(u, v)$ with control points $\mathbf{b}_{i,j}$:

$$\mathbf{b}(u, v) = \sum_{i=0}^{m} \sum_{j=0}^{n} \mathbf{b}_{ij} B_{i,j}^{m,n}(u, v). \tag{4.2}$$

The 3D curve stems from the composition $\mathbf{c}_{3D}(t) = \mathbf{b}(u(t), v(t))$ and has degree $(m + n)p$, where p denotes the degree of the original curve $\mathbf{c}_{2D}(t)$. Hence, if $\mathbf{c}_{2D}(t)$ is given in scaled Bézier form, with components $(\tilde{\mathbf{u}}, \tilde{\mathbf{v}})$, $\mathbf{c}_{3D}(t)$ has scaled points

$$\tilde{\mathbf{c}}_{3D} = \sum_{i=0}^{m} \sum_{j=0}^{n} \tilde{\mathbf{b}}_{ij} (\mathbf{1}^p - \tilde{\mathbf{u}})^{m-i} * \tilde{\mathbf{u}}^i * (\mathbf{1}^p - \tilde{\mathbf{v}})^{n-j} * \tilde{\mathbf{v}}^j.$$

Once again, observe the simplicity of the expression above, compared with the formulae for each coefficient given by Hoschek and Lasser [17] or Lasser and Bonneau [20].

4.4. Free-form deformations

Another remarkable application involving composition is free-form deformation of curves and surfaces, a method first introduced by Bézier [3]. If we apply a polynomial space deformation to a region, those polynomial curves or surfaces embedded in the region transform to new curves or surfaces whose expressions are obtained by composition. Suppose that the deformation $\mathbf{d}(x, y, z)$ is defined in trivariate tensor-product Bézier form and, by choosing a suitable coordinate system, in the unit cube

$$\mathbf{d}(x, y, z) = \sum_{i=0}^{p} \sum_{j=0}^{q} \sum_{k=0}^{r} \mathbf{d}_{ij} B_{i,j,k}^{p,q,r}(x, y, z). \tag{4.3}$$

A tensor-product Bézier patch $\mathbf{b}(u, v) = (x(u, v), y(u, v), z(u, v))$ (4.2) transforms to a higher-degree patch $\mathbf{b}_{FFD}(u, v)$ obtained as the composition $\mathbf{b}_{FFD}(u, v) = \mathbf{d}(\mathbf{b}(u, v))$. If $\mathbf{b}(u, v)$ has scaled components $(\tilde{\mathbf{x}}, \tilde{\mathbf{y}}, \tilde{\mathbf{z}})$, the resulting scaled points of $\mathbf{b}_{FFD}(u, v)$ are

$$\tilde{\mathbf{b}}_{FFD} = \sum_{i=0}^{p} \sum_{j=0}^{q} \sum_{k=0}^{r} \tilde{\mathbf{d}}_{ij} (\mathbf{1}^{m,n} - \tilde{\mathbf{x}})^{p-i} * \tilde{\mathbf{x}}^i * (\mathbf{1}^{m,n} - \tilde{\mathbf{y}})^{q-j} * \tilde{\mathbf{y}}^j * (\mathbf{1}^{m,n} - \tilde{\mathbf{z}})^{r-k} * \tilde{\mathbf{z}}^k.$$

The simplicial case would be handled in a similar way.

5. Extension to other B-bases

5.1. The convolution nature of Bernstein polynomials

As Stefanus and Goldman [28] observed, the Bernstein basis (1.1) has a convolution nature, since the set of

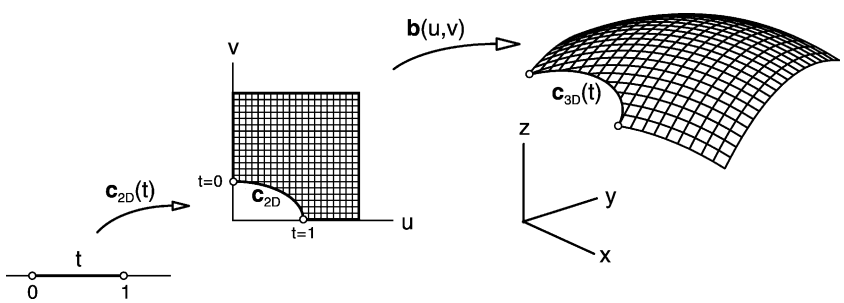

Fig. 4. Computing the 3D image $\mathbf{c}_{3D}(t)$ of a trimming curve $\mathbf{c}_{2D}(t)$.

degree-n basis functions can be generated from the pair of degree-1 functions $\{B_0^1(u), B_1^1(u)\}$ by n-fold convolution (2.4)

$$\{B_i^n(u)\}_{i=0}^n = \{B_0^1, B_1^1\}^n = \left\{ \binom{n}{i} (B_0^1)^{n-i} (B_1^1)^i \right\}_{i=0}^n,$$

$$\{B_0^1, B_1^1\} = \{1-u, u\}.$$

This is tantamount to the well-known recursive formula satisfied by Bernstein polynomials. The set of degree-n Bernstein polynomials $B_{\mathbf{i}}^n(\mathbf{u})$ (3.6) over triangles are generated in a similar way from the degree-1 functions, whereas the bivariate tensor-product Bernstein polynomials $B_{i,j}^{m,n}(u,v)$ (3.1) are computed as the convolution (3.3) between their univariate counterparts

$$\begin{bmatrix} B_{0,0}^{m,n} & & \\ \cdots & & \\ B_{0,n}^{m,n} & & \\ \vdots & \vdots & \\ B_{m,0}^{m,n} & \cdots & B_{m,n}^{m,n} \end{bmatrix} = \begin{bmatrix} B_0^m \\ \vdots \\ B_m^m \end{bmatrix} * [B_0^n, \ldots, B_n^n].$$

5.2. Spaces of trigonometric and hyperbolic polynomials

The convolution structure of the Bernstein basis is precisely what makes possible to express the product in the scaled form in terms of convolutions. In consequence, the framework developed for the Bernstein basis extends to other spaces of functions provided that their B-bases are generated similarly by repeated convolution.

A remarkable example is provided by the space of degree-n trigonometric polynomials [15]

$$\mathscr{T}_n = \text{span}\{\sin^{n-i}(t)\cos^i(t)\}_{i=0}^n, \qquad t \in [-\Delta, \Delta], \qquad (5.1)$$

which has dimension n. As demonstrated by the author [27], if $\Delta < \pi/2$ the space $\mathscr{T}_n$ admits the B-basis

$$A_i^n(t) = \frac{1}{S^n} \binom{n}{i} \sin^{n-i}(\Delta - t)\sin^i(\Delta + t), \qquad S = \sin(2\Delta).$$

Analogously to the Bernstein basis, this family of functions $A_i^n(t)$ is generated via n-fold convolution

$$\{A_i^n(u)\}_{i=0}^n = \{A_0^1, A_1^1\}^n,$$

$$\{A_0^1, A_1^1\} = \frac{1}{S}\{\sin(\Delta - t), \sin(\Delta + t)\},$$

and we can define the corresponding scaled basis $\tilde{A}_i^n(t)$ without binomial coefficients

$$\tilde{A}_i^n(t) = \frac{1}{S^n} \sin^{n-i}(\Delta - t)\sin^i(\Delta + t), \qquad S = \sin(2\Delta).$$

Thus, given two trigonometric polynomials of degrees m, n and scaled coefficients $\tilde{\mathbf{a}} = \{\tilde{a}_0, \ldots, \tilde{a}_m\}$, $\tilde{\mathbf{b}} = \{\tilde{b}_0, \ldots, \tilde{b}_n\}$, respectively, their product has degree $m+n$ and coefficients $\tilde{\mathbf{a}} * \tilde{\mathbf{b}}$.

The fundamental difference with respect to the space of algebraic polynomials is that only for even degrees n the space $\mathscr{T}_n$ (5.1) contains the unit function, whose quadratic scaled coefficients are

$$\tilde{\mathbf{1}} = \{1, 2C, 1\}, \qquad C = \cos(2\Delta).$$

Hence, degree elevation $n \rightarrow N$ is possible only if $N - n$ is an even number

$$\underset{\text{degree}n}{\tilde{\mathbf{a}}} \rightarrow \underset{\text{degree } N>n}{\tilde{\mathbf{1}}^{(N-n)/2} * \tilde{\mathbf{a}}}.$$

These results for the space T_n extend in a straightforward manner to the space of degree-n hyperbolic polynomials

$$\mathscr{H}_n = \text{span}\{\sinh^{n-i}(t)\cosh^i(t)\}_{i=0}^n, \qquad t \in [-\Delta, \Delta],$$

by replacing in the formulae above the trigonometric functions $\sin t$, $\cos t$ with their hyperbolic counterparts.

6. Conclusions

Standard techniques for algebraic manipulation of polynomials in Bernstein form try to obtain closed form expressions for a generic coefficient of the result for each particular procedure. This approach leads to multiple 'ad-hoc' expressions, with summations that often require non-trivial cycling through a set of indices, specially in certain operations in geometry processing. Informally speaking, it would be like trying to obtain a closed formula for each coefficient of a result involving matrix algebra (additions, products, exponentiation of matrices). Obviously, we rather express the result in terms of basic matrix operations.

Similarly, for algebraic manipulation of polynomials in Bernstein form we should express a procedure in terms of fundamental operations between lists of coefficients. This is achieved if we employ the scaled Bernstein basis that does not include binomial numbers, because in this basis the product corresponds to convolution of lists, and composition and generalized degree-raising become trivial. In consequence, given a formula involving polynomials, we simply rewrite it replacing addition or products between polynomials with addition or convolutions between lists, performing degree elevation when adding terms of dissimilar degrees.

We have not as goal optimal computation times, but simplicity. In a software system supporting convolutions and component-wise basic operations (addition, product, division) between lists, all procedures reduce to a few lines of code, and thus may run faster. An optimal runtime analysis, as done by Liu and Mann [22] for composition, is beyond the scope of this paper. In addition, this analysis

would be highly depending on the hardware and compiler available, since convolution involves addition of vectors and multiplying them by scalars, thereby becoming a perfect candidate for parallelization.

We have addressed in detail algebraic manipulation of univariate polynomials. The results achieved carry over easily to multivariate polynomials, by defining the corresponding scaled form and generalizing the convolution to multidimensional lists. Another feasible extension is from the space of algebraic polynomials to other functional spaces, such as the space of trigonometric polynomials, whose basis share the convolution nature of Bernstein polynomials.

This work can be also extended to piecewise polynomials defined in the B-spline basis, following the idea by Piegl and Tiller for degree-elevation [25] or products [26]. Simply convert the spline to the Bézier piecewise form, perform the operation in Bézier form and finally obtain the B-spline representation through knot-removal. Such a strategy provides a practical alternative to the explicit computation of products of B-spline functions via blossoming [21].

Acknowledgements

This work is supported by the Spanish *Ministerio de Ciencia y Tecnología*, under research grant DPI2000-0676.

References

[1] Barry PJ, Goldman RN. De Casteljau-type subdivision is peculiar to Bézier curves. Comput-Aided Des 1988;20(3):114–6.
[2] Berchtold J, Bowyer A. Robust arithmetic for multivariate Bernstein-form polynomials. Comput-Aided Des 2000;32:681–9.
[3] Bézier P. General distortion of an ensemble of biparametric surfaces. Comput-Aided Des 1977;10(2):116–20.
[4] Bowyer A, Woodwark J. Computing with geometry. Winchester, UK: Information Geometers; 1993.
[5] Brueckner I. Construction of Bézier points of quadrilaterals from those of triangles. Comput-Aided Des 1980;12(1):21–4.
[6] Carnicer JM, Peña JM. Shape preserving representations and optimality of the Bernstein basis. Adv Comput Math 1993;1:173–96.
[7] deBoor C. B-form basics. In: Farin G, editor. Geometric modeling: algorithms and new trends. Philadelphia, PA: SIAM; 1987. p. 131–48.
[8] DeRose TD. Composing Bézier simplexes. ACM Trans Graph 1988; 7(3):198–221.
[9] DeRose TD, Goldman RN, Hagen H, Mann S. Functional composition algorithms via blossoming. ACM Trans Graph 1993;12(2):113–35.
[10] Farin G, 5th ed. Curves and surfaces for computer aided geometric design, Los Altos, CA: Morgan Kaufmann; 2001.
[11] Farouki RT, Rajan VT. On the numerical condition of polynomials in Bernstein form. Comput-Aided Geomet Des 1987;4(3):191–216.
[12] Farouki RT, Rajan VT. Algorithms for polynomials in Bernstein form. Comput-Aided Geomet Des 1988;5(1):1–26.
[13] Farouki RT. On the stability of transformations between power and Bernstein polynomial forms. Comput-Aided Geomet Des 1991;8(1): 29–36.
[14] Farouki RT, Goodman TNT. On the optimal stability of the Bernstein basis. Math Comput 1996;65:1553–66.
[15] Goodman TNT, Lee SL. B-spline on the circle and trigonometric B-splines. In: Singh SP, editor. Approximation theory and spline function. Reidel D. Publishing Company; 1984. p. 297–325.
[16] Goldman RN, Filip DJ. Conversion from Bézier rectangles to Bézier triangles. Comput-Aided Des 1987;19(1):25–7.
[17] Hoschek J, Lasser D, Fundamentals of computer aided geometric design, Wellesley, MA: AK Peters; 1993.
[18] Hu S-M. Conversion of a triangular Bézer patch into three rectangular Bézier patches. Comput-Aided Geomet Des 1996; 13(3):219–26.
[19] Kim DS, Jang T, Shin H, Park JY. Rational Bézier form of hodographs of rational Bézier curves and surfaces. Comput-Aided Des 2001;33: 321–30.
[20] Lasser D, Bonneau GP. Bézier representation of trim curves. In: Hagen H, Farin G, Noltemeier H, editors. Geometric modelling, Dagstuhl; 1993. p. 227–42. Springer, 1995.
[21] Lee ETY. Computing a chain of blossoms, with application to products of splines. Comput-Aided Geomet Des 1994;11(6): 562–97.
[22] Liu W, Mann S. An optimal algorithm for expanding the composition of polynomials. ACM Trans Graph 1993;16(2):155–78.
[23] Mainar E, Peña JM. Corner cutting algorithms associated with optimal shape preserving representations. Comput-Aided Geomet Des 1999; 16(9):883–906.
[24] Peña JM. Stability and error analysis of shape preserving representations. In: Peña JM, editor. Shape preserving representations in Computer-aided geometric design. New York: Nova Science; 1999. p. 85–97.
[25] Piegl L, Tiller W. The NURBS Book, 2nd ed. Springer; 1997.
[26] Piegl L, Tiller W. Algorithm for computing the product of two B-splines. In: Méhaute A, Rabut C, Schumaker LL, editors. Curves and surfaces with applications in CAGD. Vanderbilt University Press; 1997. p. 337–44.
[27] Sánchez-Reyes J. Harmonic rational Bézier curves, p-Bézier curves and trigonometric polynomials. Comput-Aided Geomet Des 1998; 11(9):909–24.
[28] Stefanus Y, Goldman RN. Discrete convolutions schemes. In: Lyche T, Schumaker LL, editors. Mathematical methods in computer aided geometric design II. USA: Academic Press; 1992. p. 585–96.
[29] Tsay Y-F, Farouki RT. BPOLY: an object-oriented library of numerical algorithms for polynomials in Bernstein form. ACM Trans Math Software 2001;27(2):267–96.
[30] Trump W, Prautzsch H. Arbitrarily high degree elevation of Bézier representations. Comput-Aided Geomet Des 1996;13(4):387–98.
[31] Wolfram S. The Mathematica book, 4th ed. Wolfram Media/ Cambridge University Press; 1999.

Javier Sánchez-Reyes is currently a professor in the Department of Applied Mechanics, University of Castilla-La Mancha (Spain). He received his MS and PhD degrees in Mechanical Engineering from the Polytechnic University of Catalonia (Spain) in 1985 and 1988, respectively. In 1989, he worked as a visiting scholar in the Department of Mechanical Engineering, University of California, Berkeley, USA.His research interests include computer graphics and computer-aided geometric design, with particular emphasis on NURBS, trigonometric representations and geometry processing.