The conversion matrix between uniform B-spline and Bézier representations

L. Romani [a,1,*], M.A. Sabin [b]

[a] *University of Cambridge, Computer Laboratory, William Gates Building, JJ Thomson Avenue, Cambridge, CB3 0FD, England*

[b] *Numerical Geometry Ltd, 26 Abbey Lane, Lode, CB5 9EP, England*

Abstract

In computer-aided-design it is both convenient and practical to use the matrix form in representing parametric curves and surfaces. One of the reasons is that the matrix notation allows an easy conversion between different shape representations and provides a convenient implementation in either hardware or software with available matrix facilities.

In this work we propose a very simple and efficient procedure for computing in a recursive way the conversion matrices which provide the direct transformation between uniform B-spline and Bézier representations of arbitrary degrees.

Key words: matrix representation, direct transformations, uniform B-spline basis functions, explicit multi-Bézier form.

1 Introduction

This paper is essentially in two parts. In the first part (section 2) we consider the problem of converting a piecewise polynomial curve from a B-spline representation with equidistant knots to the corresponding multi-Bézier form; in the second one (section 3) we address our attention to the investigation of the inverse conversion.

* Corresponding author.
Email addresses: romani@dm.unibo.it (L. Romani), malcolm@geometry.demon.co.uk (M.A. Sabin).
[1] Present address: Dipartimento di Matematica, Universitá di Bologna, P.zza di Porta San Donato 5, 40127 Bologna, Italy.

During the last two decades several algorithms were developed for computing the transformation from B-spline to Bézier representation: these include Boehm's knot-insertion algorithm [1], the Oslo algorithm [5], Chui's scheme [2,3] and Grabowski's matrix procedure through monomials [7]. There is also the tetrahedral algorithm (see [9], page 72), based on the blossoming principle by Ramshaw [10] and de Casteljau [6], which allows us to convert a B-spline representation into a Bézier representation and vice versa. However, to our knowledge no author in the literature has considered a direct matrix transformation between B-spline and Bézier representations of arbitrary degrees.

The novelty of our proposal is therefore the idea of reducing the computation of the Bézier representation of each non-vanishing B-spline curve segment to a simple matrix multiplication with the column vector of the spline control points and computing the inverse transformation by multiplication of the inverse conversion matrix with the column vector of the Bézier control points. Hence the following sections are devoted to the recursive computation (with respect to the polynomial degree) of the entries of the B-spline-to-Bézier and the Bézier-to-B-spline conversion matrices. In particular, in section 2 two recursive computations of the B-spline-to-Bézier matrix entries, one derived from the Cox-de Boor recursion for uniform B-splines and the other (more efficient one) using the convolution formula for these functions, are presented. Since the convolution formula for uniform B-splines corresponds to the differentiation-integration formula on these functions, the matrix entries we derive via this approach will be described by a formula that is related to the computational scheme presented by Chui [2,3]. But, thanks to the matrix notation, our conversion method turns out to be more efficient and also more convenient for algebraic manipulation, since it allows us to symbolize the conversion just by a simple matrix multiplication. Additionally, thanks to the assumed notation, we are able to explain the origin of the fairly cryptic initial conditions assumed in Chui's algorithm and to avoid the rather involved use of indices his scheme requires, making our formulation much easier to follow. This will be of fundamental importance in the more complicated non-uniform case where additional indices are required and where the initial conditions are not so easy to be determined. In fact, as it appears by implementing the conversion formulae given in [2,3] for B-splines with non-uniform knots, it can be observed that such conditions are actually correct only in the case of splines with knots chosen in geometric progression and not for completely arbitrary configurations. Therefore, since the general setting of non-uniform B-splines requires a very careful investigation, we have decided to address our attention to the univariate uniform case only. The principles used are expected to be relevant to the multivariate Box-spline case and to the very general non-uniform case.

Although the multivariate extension turns out to be indeed very interesting, the direct B-spline-to-Bézier transformation in the univariate case can be extremely useful whenever we need to work out either algebraic or geometric operations on B-spline curves, since it provides a simplification of the mathematical treatment of B-splines allowing the use of popular Bézier techniques. Interesting applications of this

conversion have appeared very recently in some works concerning the computation of inner products of B-splines (which is needed in the theory of spline wavelets [8]) and the combination of texts in Bézier-based fonts with free-form surfaces in spline representations [12]. Applications of the Bézier-to-B-spline conversion procedure are also related to some problems that very often appear in CAGD, like modifying the control points of two assigned degree-n Bézier curves in order they can join C^{n-1}-continuously. In fact, without using the classical formulae given in the literature for determining the opportunely modified Bézier coefficients, we can exploit the Bézier-to-B-spline transformation to compute the B-spline representation (Fig. 1 right) of the two given curves (Fig. 1 left) on a common uniform knot-partition and determine the inner n control points of the C^{n-1} spline that represents their C^{n-1} join (Fig. 2 left) at the common parameter value, via n linear combinations of their inner n spline control points taken two by two. At this point, via the B-spline-to-Bézier conversion procedure we have the possibility to determine the Bézier point sets of the original curves opportunely modified in order they join C^{n-1}-continuously (Fig. 2 right).

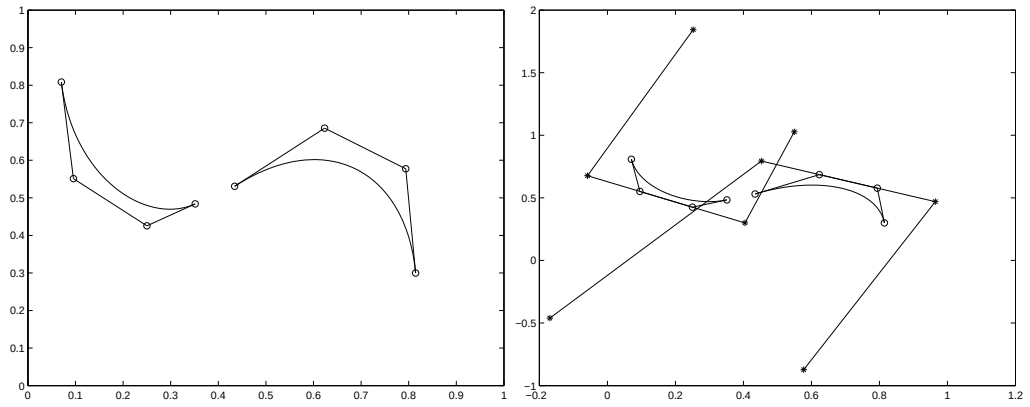

Fig. 1. Two cubic Bézier curves (left) and their B-spline representation (right).

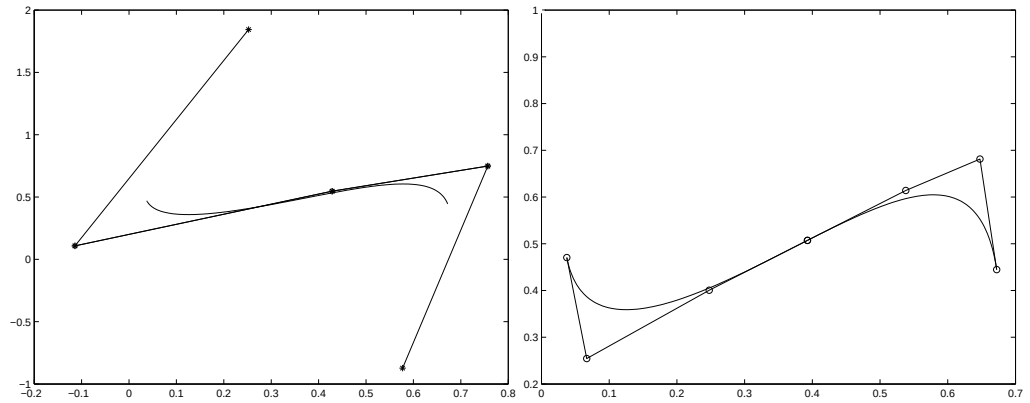

Fig. 2. B-spline (left) and Bézier (right) representation of the two cubics C^2-joined.

This is only one of the applications that motivate the results formulated in section 3. Here the main topic is the description of the recursive computation (with respect

to the polynomial degree) of the entries of the Bézier-to-B-spline conversion matrix, using both the recursion formula for Bernstein polynomials and the differentiation-integration formula for these functions. This time, both the approaches lead to a completely new proposal which has no comparisons in the literature.

2 The B-spline-to-Bézier conversion matrix

Let the B-spline-to-Bézier conversion matrix $\mathrm{S}^{(n)}$ be the $(n+1)\times(n+1)$ matrix by which we can multiply each consecutive $(n+1)$-long subsequence of control points in the control polygon C of a given B-spline curve of degree n, to give the Bézier control points D of the corresponding span.

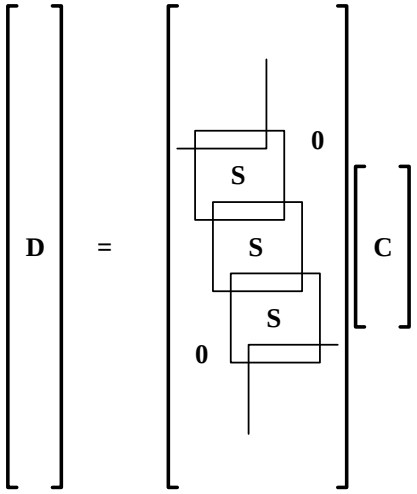

Fig. 3. Stacking to convert a complete B-spline into piecewise Bézier form.

To compute the piecewise Bézier representation of a complete curve, we can build a much larger matrix by stacking copies of $\mathrm{S}^{(n)}$ and then using the related facts that (i) two adjacent Bézier pieces share a common control point at the junction and (ii) the bottom row of $\mathrm{S}^{(n)}$ is just a shifted copy of the top row (see Fig. 3). In this paper we take the stacking for granted and focus on the determination of the elements of $\mathrm{S}^{(n)}$ for arbitrarily large n by recurrence (we do not concern ourselves here with the question of end-conditions).

Without any loss of generality, in order to work out the B-spline-to-Bézier conversion matrix for an arbitrary degree-n representation, we can compute the Bézier representation of every B-spline basis function $N_{j-n,n}(u)$, $j = 0, \cdots, n$ over the interval $[0, 1]$ (see Fig. 4).

This requires expressing each $N_{j-n,n}(u)$ as a linear combination of the Bernstein-Bézier polynomials of degree n, $B_{i,n}(u)$, $u \in [0, 1] \quad \forall i = 0, \cdots, n$, via a set of

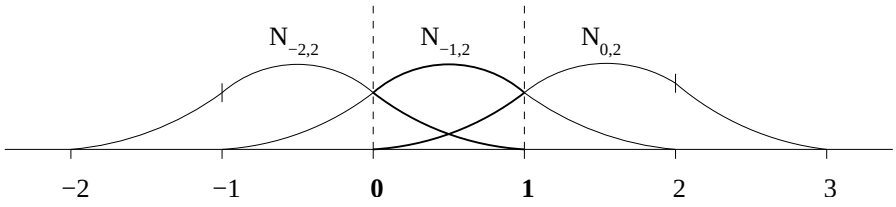

Fig. 4. The three quadratic uniform B-splines over the central interval [0, 1].

coefficients $s_{i,j}^{(n)}$, $i = 0, \cdots, n$:

$$\forall j = 0, \cdots, n \qquad N_{j-n,n}(u) = \sum_{i=0}^{n} s_{i,j}^{(n)} B_{i,n}(u) \qquad u \in [0,1]. \tag{1}$$

Therefore, writing (1) in matrix form we have that

$$[\, N_{-n,n}(u) \quad \cdots \quad N_{0,n}(u) \,] = [\, B_{0,n}(u) \quad \cdots \quad B_{n,n}(u) \,]\, \mathrm{S}^{(n)} \tag{2}$$

namely $\mathrm{S}^{(n)}$ is the so-called degree-n conversion matrix from B-spline to Bézier representation. Since both B-spline basis functions and Bernstein-Bézier polynomials satisfy the unit partition property, it follows that the rows of the conversion matrix $\mathrm{S}^{(n)}$ sum to 1: hence, converting a B-spline curve in Bézier form, every Bézier control point will be defined by a convex combination of the B-spline control points. The geometric meaning of the columns of $\mathrm{S}^{(n)}$ is instead related to the Bézier representation of the $n+1$ degree-n B-spline functions over the interval $[0, 1]$: the j-th column $\mathbf{s}_j^{(n)}$ of $\mathrm{S}^{(n)}$ represents the Bézier ordinates of the control polygon of the $(n-j)$-th polynomial piece of a degree-n uniform B-spline (see Fig. 5 for $\mathrm{S}^{(2)} = \frac{1}{2}\begin{bmatrix} 1 & 1 & 0 \\ 0 & 2 & 0 \\ 0 & 1 & 1 \end{bmatrix}$).

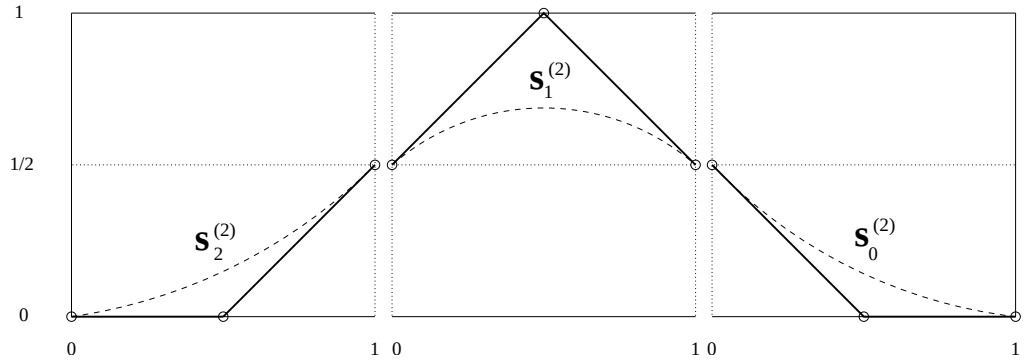

Fig. 5. Plot of the columns of $\mathrm{S}^{(2)}$ interpreted as the Bézier control point sets for each curve segment of a quadratic uniform B-spline.

Now, in order to work out a recursion formula for computing the conversion matrix for an arbitrary degree-n spline curve, we substitute in the Cox-de Boor recursion

formula (see [9], page 61) for a uniform degree-n B-spline, the expression in (1), obtaining

$$\sum_{i=0}^{n} s_{i,j}^{(n)} B_{i,n}(u) = \sum_{i=0}^{n-1} \left(\frac{u-j+n}{n}\ s_{i,j-1}^{(n-1)} \ + \ \frac{j+1-u}{n}\ s_{i,j}^{(n-1)} \right) B_{i,n-1}(u). \tag{3}$$

Then, if in place of $B_{i,n}(u)$, $u \in [0,1]$ we substitute the recursion relation on Bernstein polynomials (see [9], page 10), we can impose the equality of the coefficients in both members of equation (3), obtaining that $\forall j = 0, \cdots, n \ \ i = 0, \cdots, n-1$:

$$u\ s_{i+1,j}^{(n)} + (1-u)\ s_{i,j}^{(n)} = \frac{u-j+n}{n}\ s_{i,j-1}^{(n-1)} \ + \ \frac{j+1-u}{n}\ s_{i,j}^{(n-1)} \quad u \in [0,1] \tag{4}$$

where $\forall i = 0, \cdots, n-1\ s_{i,-1}^{(n-1)} = s_{i,n}^{(n-1)} = 0$. Hence evaluating (4) at $u = 0$ we have that

$$s_{i,j}^{(n)} = \frac{n-j}{n}\ s_{i,j-1}^{(n-1)} \ + \ \frac{j+1}{n}\ s_{i,j}^{(n-1)} \ \ \forall j = 0, \cdots, n \ \ i = 0, \cdots, n-1 \tag{5}$$

while evaluating (4) at $u = 1$ it follows:

$$s_{i+1,j}^{(n)} = \frac{n+1-j}{n}\ s_{i,j-1}^{(n-1)} \ + \ \frac{j}{n}\ s_{i,j}^{(n-1)} \ \ \forall j = 0, \cdots, n \ \ i = 0, \cdots, n-1. \tag{6}$$

Then, starting from the 1×1 matrix $\mathrm{S}^{(0)} = s_{0,0}^{(0)} = 1$ which relates $N_{0,0}(u) = 1$ and $B_{0,0}(u) = 1\ \forall u \in [0,1]$, the $(n+1) \times (n+1)$ conversion matrix $\mathrm{S}^{(n)} = \left\{ s_{i,j}^{(n)} \right\}_{i,j=0,\cdots,n}$ with $n \geq 1$, can be defined recursively $\forall j = 0, \cdots, n$ either by

$$\begin{array}{rcll} s_{i,j}^{(n)} & = & \frac{n-j}{n}\ s_{i,j-1}^{(n-1)} \ + \ \frac{j+1}{n}\ s_{i,j}^{(n-1)} & \forall i = 0, \cdots, n-1 \\ s_{n,j}^{(n)} & = & \frac{n+1-j}{n}\ s_{n-1,j-1}^{(n-1)} \ + \ \frac{j}{n}\ s_{n-1,j}^{(n-1)} & \end{array} \tag{7}$$

or by

$$\begin{array}{rcll} s_{0,j}^{(n)} & = & \frac{n-j}{n}\ s_{0,j-1}^{(n-1)} \ + \ \frac{j+1}{n}\ s_{0,j}^{(n-1)} & \\ s_{i,j}^{(n)} & = & \frac{n+1-j}{n}\ s_{i-1,j-1}^{(n-1)} \ + \ \frac{j}{n}\ s_{i-1,j}^{(n-1)} & \forall i = 1, \cdots, n. \end{array} \tag{8}$$

Through equations (7) and (8) the generation of the conversion matrix turns out to be very simple, but in order to make its computation more efficient (only integer additions and subtractions), we propose an alternative definition via the convolutional relation of uniform B-splines. In this way, by substituting (1) in the recursion

formula of a degree-n uniform B-spline (see [9], page 109), we obtain the following expression

$$N_{j-n,n}(u) = \int_{-\infty}^{u} \sum_{i=0}^{n-1} \left(s_{i,j-1}^{(n-1)} - s_{i,j}^{(n-1)} \right) B_{i,n-1}(v)\, dv \qquad u \in [0,1] \tag{9}$$

where $\forall i = 0, \cdots, n-1 \quad s_{i,-1}^{(n-1)} = s_{i,n}^{(n-1)} = 0$. Equation (9) can be regarded as the combination of two operators, each of which can be handled numerically: a shift-and-subtraction, an integration. The shift-and-subtraction is relatively obvious: the shift merely involves applying the Bézier coefficients $s_{i,j-1}^{(n-1)}$ of $N_{j-n,n-1}(v) \equiv N_{j-1-(n-1),n-1}(v)$ to the part of the parametric domain displaced by one lattice unit; the subtraction of the basis functions $N_{j-n,n-1}(v) - N_{j-n+1,n-1}(v) \equiv N_{j-1-(n-1),n-1}(v) - N_{j-(n-1),n-1}(v)$, corresponds to the subtraction of the coefficients $s_{i,j-1}^{(n-1)} - s_{i,j}^{(n-1)}$, according to the fact that (1) is linear in the coefficients. Like the shift-and-subtraction, the integration operator has a simple numerical form, involving antidifferences on the Bézier coefficients set (see [9], page 19). Hence, by solving the Bézier integral in (9) and by comparing the resulting expression with (1), we get that for a given j between n and 0

$$s_{i,j}^{(n)} = s_{i-1,j}^{(n)} + \frac{1}{n} \left(s_{i-1,j-1}^{(n-1)} - s_{i-1,j}^{(n-1)} \right) \qquad \forall i = 1, \cdots, n \tag{10}$$

where $\forall i = 1, \cdots, n \quad s_{i-1,-1}^{(n-1)} = s_{i-1,n}^{(n-1)} = 0$ and $s_{0,j}^{(n)} \quad j = n, \cdots, 0$ is the integration constant, computed through (1) in the following way: $s_{0,n}^{(n)} = N_{0,n}(0) = 0, \quad s_{0,j}^{(n)} = N_{j-n,n}(0) = N_{j+1-n,n}(1) = s_{n,j+1}^{(n)} \quad \forall j = n-1, \cdots, 0$ (see Fig. 4). Therefore, the conversion matrix $\mathrm{S}^{(n)} = \left\{ s_{i,j}^{(n)} \right\}_{i,j=0,\cdots,n}$ with $n \geq 1$, can be defined recursively $\forall j = n, \cdots, 0$ by (10) choosing the integration constant in the following way:

$$s_{0,n}^{(n)} = 0, \qquad s_{0,j}^{(n)} = s_{n,j+1}^{(n)} \quad j = n-1, \cdots, 0. \tag{11}$$

Note that the $\frac{1}{n}$ factors in (10) do combine into a single $\frac{1}{n!}$ which can be carried out at the end of a sequence of operations which are indeed only integer additions and subtractions.

The following MATLAB function takes as input the degree n and efficiently generates the conversion matrix $\mathrm{S}^{(n)}$ (save for the factor $\frac{1}{n!}$) via the shift-subtraction-integration procedure, exploiting the symmetry of the columns:

```
function S=conv(n)
S=eye(n+1);
for k = n-1:-1:1
    nc=round((n - k)/2);
```

1. Shift-and-Subtraction
$fc = n + 1 - nc$;
for $j = fc : n$
  for $i = k + 1 : n + 1$
    S(i, j)=S(i, j)-S$(i, j + 1)$;
  end
end
2. Integration
for $j = n : -1 : fc$
  S(k, j)=S$(n + 1, j + 1)$;
  for $i = k + 1 : n + 1$
    S(i, j)=S$(i - 1, j)$+S(i, j);
  end
end
if mod$(n + 1 - k$,2)==1
  S$(k : n + 1, n - nc)$=S$(n + 1 : -1 : k, fc)$;
end
end
3. Replication of columns
for j=1:round$(n/2)$
  S(1:$n + 1, j$)=S(n+1:-1:1,$n + 2 - j$);
end

Remark 1 *Since the Cox-de Boor formula and the convolutional relation are equivalent strategies by which the univariate uniform B-spline basis functions may be generated, it is trivially verified that considering together two of the three recursions proposed above in (7) , (8) and (10), the third one can be easily worked out.*

Remark 2 *There is also a way to formulate the recursions for the matrix entries in (7)-(8) by means of the following matrix recursion between the matrices* $\mathrm{S}^{(n)}$ *and* $\mathrm{S}^{(n-1)}$*:*

$$\mathrm{S}^{(n)} = \begin{bmatrix} \mathrm{S}_u^{(n)} & 0 \\ 0 & \mathrm{S}_d^{(n)} \end{bmatrix}$$

$$\mathrm{S}_u^{(n)} = \mathrm{S}^{(n-1)}\mathrm{L}^{(n-1)} \tag{12}$$

$$\mathrm{S}_d^{(n)} = \mathrm{S}^{(n-1)}\mathrm{M}^{(n-1)} \tag{13}$$

where $\mathrm{S}_u^{(n)} = \{s_{i,j}^{(n)}\}_{i,j=0,\cdots,n-1}$, $\mathrm{S}_d^{(n)} = \{s_{i,j}^{(n)}\}_{i,j=1,\cdots,n}$ *and* $\mathrm{L}^{(n-1)}$, $\mathrm{M}^{(n-1)}$ *are respectively the following* $n \times n$ *bidiagonal matrices:*

$$\mathrm{L}^{(n-1)} = \begin{bmatrix} \frac{1}{n} & \frac{n-1}{n} & 0 & 0 & 0 & \cdots & 0 \\ 0 & \frac{2}{n} & \frac{n-2}{n} & 0 & 0 & \cdots & 0 \\ \vdots & & \ddots & \ddots & & & \vdots \\ 0 & 0 & \cdots & \frac{j}{n} & \frac{n-j}{n} & \cdots & 0 \\ \vdots & & & & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & \frac{n-1}{n} & \frac{1}{n} \\ 0 & 0 & \cdots & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathrm{M}^{(n-1)} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ \frac{1}{n} & \frac{n-1}{n} & 0 & 0 & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & & & & & \vdots \\ 0 & \cdots & 0 & \frac{n-j}{n} & \frac{j}{n} & 0 & \cdots & 0 \\ \vdots & & & & \ddots & \ddots & & \vdots \\ 0 & \cdots & 0 & 0 & 0 & \frac{n-2}{n} & \frac{2}{n} & 0 \\ 0 & \cdots & 0 & 0 & 0 & 0 & \frac{n-1}{n} & \frac{1}{n} \end{bmatrix}.$$

3 The Bézier-to-B-spline conversion matrix

Now let the Bézier-to-B-spline conversion matrix $\mathrm{R}^{(n)}$ be the $(n+1)\times(n+1)$ matrix by which we can multiply the vector of $n+1$ control points of a Bézier curve, to give a sequence of $n+1$ control points which will be a subsequence of any B-spline which contains that curve as a span (there is no equivalent of stacking because in general a sequence of Bézier curves is not an equal interval B-spline).

Although the Bézier-to-B-spline conversion matrix $\mathrm{R}^{(n)}$ coincides with the inverse of the B-spline-to-Bézier conversion matrix $\mathrm{S}^{(n)}$, we can compute the entries $\left\{r_{j,i}^{(n)}\right\}_{j,i=0,\cdots,n}$ of $\mathrm{R}^{(n)}$ more efficiently, and without using the entries of $\mathrm{S}^{(n)}$, repeating the process

described in the previous section just substituting equation (1) by

$$\forall i = 0, \cdots, n \qquad B_{i,n}(u) = \sum_{j=0}^{n} r_{j,i}^{(n)} N_{j-n,n}(u) \qquad u \in [0,1]. \tag{14}$$

Using matrix notation, equation (14) can be written as

$$[\, B_{0,n}(u) \quad \cdots \quad B_{n,n}(u) \,] = [\, N_{-n,n}(u) \quad \cdots \quad N_{0,n}(u) \,]\, \mathrm{R}^{(n)} \tag{15}$$

in such a way that the columns of $\mathrm{R}^{(n)}$ express the B-spline representation of the degree-n Bernstein polynomials $B_{0,n}(u), \cdots, B_{n,n}(u)$. In other words, if we plot the $n+1$ columns of the matrix $\mathrm{R}^{(n)}$, we have the B-spline ordinates of the control polygon of each degree-n Bernstein polynomial (see Fig. 6 for $\mathrm{R}^{(2)} = \begin{bmatrix} 2 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 2 \end{bmatrix}$).

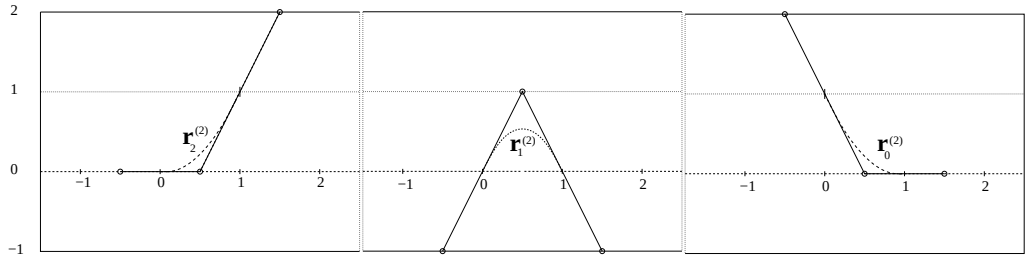

Fig. 6. Plot of the columns of $\mathrm{R}^{(2)}$ interpreted as the B-spline control point sets of each quadratic Bernstein polynomial.

This time, in order to work out a recursion formula for computing the conversion matrix for an arbitrary degree-n Bézier curve, we substitute in the recursion formula for Bernstein polynomials (see [9], page 10), the expression in (14), obtaining

$$\sum_{j=0}^{n} r_{j,i}^{(n)} N_{j-n,n}(u) = \sum_{j=0}^{n-1} \Big(\; u \; r_{j,i-1}^{(n-1)} \; + \; (1-u) \; r_{j,i}^{(n-1)} \; \Big) \, N_{j-(n-1),n-1}(u). \tag{16}$$

Then, if in place of $N_{j-n,n}(u)$ we substitute the Cox-de Boor recursion formula for uniform B-splines, we can impose the equality of the coefficients in both members of equation (16), obtaining that $\forall i = 0, \cdots, n \;\; j = 0, \cdots, n-1$:

$$\frac{u-j+n-1}{n} \; r_{j+1,i}^{(n)} + \frac{j+1-u}{n} \; r_{j,i}^{(n)} = u \; r_{j,i-1}^{(n-1)} \; + \; (1-u) \; r_{j,i}^{(n-1)} \tag{17}$$

where $u \in [0,1]$ and $\forall j = 0, \cdots, n-1 \; r_{j,-1}^{(n-1)} = r_{j,n}^{(n-1)} = 0$. But, since (17) is linear in u and is satisfied $\forall u \in [0,1]$, we can affirm that it is satisfied $\forall u \in \mathbb{R}$. Hence

evaluating (17) at $u = j - n + 1$ we have that

$$r_{j,i}^{(n)} = (j - n + 1)\ r_{j,i-1}^{(n-1)}\ +\ (n - j)\ r_{j,i}^{(n-1)}\ \ \forall i = 0, \cdots, n\ \ j = 0, \cdots, n-1 \tag{18}$$

while evaluating (17) at $u = j + 1$ it follows:

$$r_{j+1,i}^{(n)} = (j + 1)\ r_{j,i-1}^{(n-1)}\ -\ j\ r_{j,i}^{(n-1)}\ \ \forall i = 0, \cdots, n\ \ j = 0, \cdots, n-1. \tag{19}$$

Then, starting from the 1×1 matrix $\mathrm{R}^{(0)} = r_{0,0}^{(0)} = 1$ which relates $B_{0,0}(u) = N_{0,0}(u) = 1$, the $(n+1) \times (n+1)$ conversion matrix $\mathrm{R}^{(n)} = \left\{ r_{j,i}^{(n)} \right\}_{j,i=0,\cdots,n}$, with $n \geq 1$, can be defined recursively $\forall i = 0, \cdots, n$ either by

$$\begin{aligned} r_{j,i}^{(n)} &= (j + 1 - n)\ r_{j,i-1}^{(n-1)}\ +\ (n - j)\ r_{j,i}^{(n-1)} \quad \forall j = 0, \cdots, n-1 \\ r_{n,i}^{(n)} &= n\ r_{n-1,i-1}^{(n-1)}\ +\ (1 - n)\ r_{n-1,i}^{(n-1)} \end{aligned} \tag{20}$$

or by

$$\begin{aligned} r_{0,i}^{(n)} &= (1 - n)\ r_{0,i-1}^{(n-1)}\ +\ n\ r_{0,i}^{(n-1)} \\ r_{j,i}^{(n)} &= j\ r_{j-1,i-1}^{(n-1)}\ +\ (1 - j)\ r_{j-1,i}^{(n-1)} \quad \forall j = 1, \cdots, n. \end{aligned} \tag{21}$$

Through equations (20) and (21) the generation of the conversion matrix turns out to be very simple, but in order to make its computation more efficient, we propose an alternative definition via the integral relation of Bernstein polynomials. In this way, by substituting (14) in the derivative formula of a degree-n Bernstein polynomial (see [9], page 15), we obtain the following expression

$$B_{i,n}(u) = \int_{-\infty}^{u} n \sum_{j=0}^{n-1} \left(r_{j,i-1}^{(n-1)} - r_{j,i}^{(n-1)} \right) N_{j-(n-1),n-1}(v)\ dv \qquad u \in [0, 1] \tag{22}$$

where $\forall j = 0, \cdots, n-1\ \ r_{j,-1}^{(n-1)} = r_{j,n}^{(n-1)} = 0$. Hence, by solving the integral in (22) and by comparing the resulting expression with equation (14), we get that, for a given i between n and 0,

$$r_{j,i}^{(n)} = r_{j-1,i}^{(n)} + n \left(r_{j-1,i-1}^{(n-1)} - r_{j-1,i}^{(n-1)} \right) \qquad \forall j = 1, \cdots, n \tag{23}$$

where $\forall j = 1, \cdots, n\ \ r_{j-1,-1}^{(n-1)} = r_{j-1,n}^{(n-1)} = 0$ and $r_{0,i}^{(n)}\ \ i = n, \cdots, 0$ is the integration constant computed through (21). Note that while in the B-spline-to-Bézier conversion algorithm it's easy to compute the integration constant, here we need to exploit the first method to compute it. Therefore, the conversion matrix $\mathrm{R}^{(n)} = \left\{ r_{j,i}^{(n)} \right\}_{j,i=0,\cdots,n}$, with $n \geq 1$, can be defined recursively by (23) choosing

$r_{0,n}^{(n)} = 0,\ r_{0,i}^{(n)} = (1-n)r_{0,i-1}^{(n-1)} + nr_{0,i}^{(n-1)}\ \ \forall i = n-1, \cdots, 0$ as integration constants.

Exploiting the symmetry of its columns, the $(n+1) \times (n+1)$ Bézier-to-B-spline conversion matrix $\mathrm{R}^{(n)}$, for an arbitrary degree n can be efficiently generated by the following MATLAB function:

```
function R=invconv(n)
R=eye(n+1);
for k = n-1:-1:1
    kk = n + 1 - k;
    nc=round((n - k)/2);
    1. Shift-and-Subtraction
    fc = n + 1 - nc;
    for j = fc : n
        R(k, j)=kk*(R(k + 1, j)-R(k + 1, j + 1));
        for i = k + 2 : n + 1
            R(i, j)=kk*(R(i, j)-R(i, j + 1));
        end
    end
    R(n + 1, n + 1)=kk*R(n + 1, n + 1);
    2. Integration
    for j = n : −1 : fc
        R(k, j)=R(k + 1, j)-R(k, j);
        for i = k + 2 : n + 1
            R(i, j)=R(i − 1, j)+R(i, j);
        end
    end
    if mod(kk,2)==1
        R(k : n + 1, n − nc)=R(n + 1 : −1 : k, fc);
    end
end
3. Replication of columns
for j=1:round(n/2)
    R(1:n + 1, j)=R(n+1:-1:1,n + 2 − j);
end
```

Remark 3 *As for the recursive computations of the B-spline-to-Bézier matrix entries, it is trivially verified that considering together two of the three recursions proposed in (20), (21) and (23) for computing the entries of the Bézier-to-B-spline conversion matrix, the third one can be easily worked out.*

Conversely, this time there is no way to formulate the recursions for the matrix entries in (20)-(21) by means of a matrix recursion between the matrices $\mathrm{R}^{(n)}$ *and*

$\mathrm{R}^{(n-1)}$.

4 Conclusions and future work

In this work we have proposed a simple and efficient procedure for generating recursively the B-spline-to-Bézier and Bézier-to-B-spline conversion matrices in the uniform univariate case. They give us a practical direct transformation between arbitrary degree-n B-spline and Bézier representations just multiplying the conversion matrix by the column vector of control points. Since the tensor-product bivariate case is just a tensor-product of the univariate results, the next step will be extending our matrix conversion formulae to three- and four- directional Box-splines, providing an algorithm competitive with those proposed in [4] and [11] for computing the Bézier coefficients of each polynomial piece of a Box-spline.

Additionally we would like to propose an analogous matrix conversion procedure for transforming Bézier patches into Box-splines: in fact, while the possibility of representing a Box-spline surface by its multi-Bézier representation provides an acceleration in rendering processes and facilitates the analysis around extraordinary points of all Box-spline subdivision schemes, the inverse transformation turns out to be very interesting in solving the problem of modifying the sets of control points of given triangular Bézier patches in order to guarantee their maximum join (with respect to the corresponding Box-spline order of continuity).

Acknowledgements

This research was supported by the European Union research project *Multiresolution in Geometric Modelling* (MINGLE for short, EU Contract No. HPRN-CT-1999-00117) and by University of Cambridge Computer Laboratory. We like to thank Giulio Casciola for his helpful suggestions. Special thanks also go to the referees.

References

[1] W. Boehm, *Generating the Bézier points of B-spline curves and surfaces*, Computer Aided Design **13**(6) (1981), 365-366

[2] C.K. Chui, M.J. Lai, *Computation of box-splines and B-splines on triangulations of nonuniform rectangular partitions*, Approximation Theory & its Applications **3** (1987), 37-62

[3] C.K. Chui, *Multivariate Splines*, CBMS Lectures Series 54, SIAM, Philadelphia (1988)

[4] C.K. Chui, M.J. Lai, *Algorithms for generating B-nets and graphically displaying spline surfaces on three- and four- directional meshes*, Computer Aided Geometric Design **8** (1991), 479-493

[5] E. Cohen, T. Lyche, R. Riesenfeld, *Discrete B-splines and subdivision tecniques in computer aided geometric design and computer graphics*, Computer Graphics and Image Processing **14**(2) (1980), 87-111

[6] P. de Casteljau, Formes á pôles, in: *Mathematiques et CAO 2* (Hermés Publishing, Paris, 1985)

[7] H. Grabowski, X. Li, General matrix representation for NURBS curves and surfaces for interfaces, in: J. Hoschek (Ed.), *Freeform Tools in CAD Systems - A Comparison* (Teubner, 1991), 219-232

[8] T. Lyche, K. Mørken, E. Quak, Theory and Algorithms for non-uniform spline wavelets, in: N. Dyn, D. Leviatan, D. Levin and A. Pinkus (Eds.), *Multivariate Approximation and Applications* (Cambridge University Press, 2001), 152-187

[9] H. Prautzsch, W. Boehm, M. Paluszny, *Bézier and B-spline techniques* (Springer, Berlin, 2002)

[10] L. Ramshaw, *Blossoming: A Connect-the-Dots Approach to Splines*, Technical Report 19, Digital System Research Center (Palo Alto, 1987)

[11] M.A. Sabin, *The use of piecewise forms for the numerical representation of shape*, Tanulmanyok 60/1977, MTA Szamitastechnikai es Automatizalasi Kutato Intezet (Budapest, 1976)

[12] T. Surazhsky, G. Elber, *Arbitrary Precise Orientation Specification for Layout of Text*, Proceedings of the Eighth Pacific Conference on Computer Graphics and Applications (2000), 80-86