ICES REPORT 10-08

March 2010

Isogeometric Finite Element Data Structures based on Bezier Extraction of NURBS

by

Michael J. Borden, Michael A. Scott, John A. Evans, and Thomas J.R. Hughes

The Institute for Computational Engineering and Sciences
The University of Texas at Austin
Austin, Texas 78712

Reference: Michael J. Borden, Michael A. Scott, John A. Evans, and Thomas J.R. Hughes, "Isogeometric Finite Element Data Structures based on Bezier Extraction of NURBS", Institute for Computational Engineering and Sciences, The University of Texas at Austin, Austin, Texas, March 2010.

Isogeometric Finite Element Data Structures based on Bézier Extraction of NURBS*

Michael J. Borden[a,b] Michael A. Scott[a] John A. Evans[a]
Thomas J.R. Hughes[a]

[a]Institute for Computational Engineering and Sciences
The University of Texas at Austin
1 University Station C0200, Austin, Texas 78712, USA

[b]Sandia National Laboratories
Albuquerque, NM 87185, USA

Abstract

We present the Bézier extraction operator and isogeometric Bézier elements for NURBS-based isogeometric analysis. The Bézier extraction operator allows numerical integration of smooth functions to be performed on C^0 Bézier elements. We show how the Bézier extraction operator is computed for NURBS. We then show that the extraction operator and Bézier elements provide an element structure for isogeometric analysis that can be easily incorporated into existing finite element codes, without any changes to element form and assembly algorithms, and standard data processing arrays. All significant changes may be implemented in a shape function subroutine.

Keywords: Bézier extraction, Isogeometric analysis, NURBS, Finite elements

1 Introduction

Isogeometric analysis was introduced by Hughes et al. [11] as a generalization of standard finite elements and has been described in detail in [5]. The basic idea of the isogeometric concept is to use the same basis for analysis as is used to describe the geometry. The geometric representation (e.g., NURBS and T-splines [16]) is typically smooth whereas the representation for standard finite element analysis is typically continuous but not smooth. The ability to efficiently use a smooth basis in analysis has shown computational advantages over standard finite elements in many areas including turbulence [1, 4], incompressibility [2, 3, 8], structural analysis [6, 7], phase-field analysis [9], large deformation with mesh distortion [13], and shape optimization [17]. These results have motivated further investigations of isogeometric analysis.

In this paper we describe the construction of isogeometric Bézier elements and the Bézier extraction operator, which provide an element structure for isogeometric analysis that can be incorporated into existing finite element codes. We first discuss the construction of the Bézier elements and extraction operator. For simplicity, we restrict our discussion to NURBS for this development. We first show how Bézier decomposition can be used to compute a set of C^0 Bézier elements from a NURBS, and we then show how to compute

*Dedicated to the memory of O.C. Zienkiewicz

the Bézier extraction operator. Next, we remove the restriction to NURBS and show that any basis for which an extraction operator can be constructed can be incorporated into finite element codes with changes confined to shape function subroutines. Finally, we return to NURBS and walk through a simple two-dimensional example showing the construction of the extraction operator and its use in defining data processing arrays. We note that the same data processing arrays utilized in finite element analysis, namely, the IEN, ID, and LM arrays [10], are also sufficient for isogeometric Bézier elements.

2 Preliminaries

To establish definitions for later development, we provide a brief overview of the construction of B-splines and NURBS. For a more detailed description see [14, 15]. We index element (i.e., local) basis functions and control points with lower case indices $a, b, c, \ldots$ and patch (i.e., global) basis functions and control points with upper case indices $A, B, C, \ldots$ The indices i, j, k, are used for various things, the context making use clear. The global cases considered here consist of a single patch. However, the generalization to the multi-patch case is straightforward. It just involves a transformation between the control point indices of each patch and the corresponding global control points.

2.1 Bernstein polynomials and Bézier curves

A degree p Bézier curve is defined by a linear combination of $p+1$ Bernstein polynomial basis functions. We define the set of basis functions as $\mathbf{B}(\xi) = \{B_{a,p}(\xi)\}_{a=1}^{p+1}$, and the corresponding set of vector valued control points as $\mathbf{P} = \{\mathbf{P}_a\}_{a=1}^{p+1}$ where each $\mathbf{P}_a \in \mathbb{R}^d$, d being the number of spatial dimensions, and $\mathbf{P}$ is a matrix of dimension $n \times d$, viz.,

$$\mathbf{P} = \begin{bmatrix} P_1^1 & P_1^2 & \ldots & P_1^d \\ P_2^1 & P_2^2 & \ldots & P_2^d \\ \vdots & \vdots & & \vdots \\ P_n^1 & P_n^2 & \ldots & P_n^d \end{bmatrix}. \tag{1}$$

The Bézier curve can then be written as

$$C(\xi) = \sum_{a=1}^{p+1} \mathbf{P}_a B_{a,p}(\xi) = \mathbf{P}^T \mathbf{B}(\xi) \qquad \xi \in [0,1]. \tag{2}$$

The Bernstein polynomials can be defined recursively for $\xi \in [0,1]$ as

$$B_{a,p}(\xi) = (1-\xi) B_{a,p-1}(\xi) + \xi B_{a-1,p-1}(\xi) \tag{3}$$

where

$$B_{1,0}(\xi) \equiv 1 \tag{4}$$

and

$$B_{a,p}(\xi) \equiv 0 \quad \text{if } a < 1 \text{ or } a > p+1. \tag{5}$$

2.2 Knot vectors and B-splines

A univariate B-spline basis is defined by a knot vector. The knot vector is a set of non-decreasing parametric coordinates written as $\Xi = \{\xi_1, \xi_2, \ldots, \xi_{n+p+1}\}$ where $\xi_A \in \mathbb{R}$ is the A^{th} knot, p is the polynomial degree of the B-spline basis functions, and n is the number of basis functions. B-spline basis functions for a given degree, p, are defined recursively over the parametric domain by the knot vector. Piecewise constants are first defined as

$$N_{A,0}(\xi) = \begin{cases} 1 & \xi_A \leq \xi < \xi_{A+1} \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

For $p > 0$ the basis functions are define by the Cox-de Boor recursion formula

$$N_{A,p}(\xi) = \frac{\xi - \xi_A}{\xi_{A+p} - \xi_A} N_{A,p-1}(\xi) + \frac{\xi_{A+p+1} - \xi}{\xi_{A+p+1} - \xi_{A+1}} N_{A+1,p-1}(\xi). \tag{7}$$

A B-spline *curve* of degree p in $\mathbb{R}^d$ is defined by a set of B-spline basis functions, $\mathbf{N}(\xi) = \{N_{A,p}(\xi)\}_{A=1}^n$, and control points, $\mathbf{P} = \{\mathbf{P}_A\}_{A=1}^n$, as

$$T(\xi) = \sum_{A=1}^{n} \mathbf{P}_A N_{A,p}(\xi) = \mathbf{P}^T \mathbf{N}(\xi). \tag{8}$$

2.3 Knot insertion

Knots may be inserted into a knot vector without changing the geometric or parametric properties of the curve. Let $\Xi = \{\xi_1, \xi_2, \ldots, \xi_{n+p+1}\}$ be a given knot vector. Inserting a new knot $\bar{\xi} \in [\xi_k, \xi_{k+1}[$ with $k > p$ into the knot vector requires that $n+1$ new basis functions be defined using (6) and (7) with the new knot vector $\Xi = \{\xi_1, \xi_2, \ldots, \xi_k, \bar{\xi}, \xi_{k+1}, \xi_{n+p+1}\}$. The $m = n+1$ new control points, $\{\bar{\mathbf{P}}_A\}_{A=1}^m$, are formed from the original control points, $\{\mathbf{P}_A\}_{A=1}^n$, by

$$\bar{\mathbf{P}}_A = \begin{cases} \mathbf{P}_1 & A = 1 \\ \alpha_A \mathbf{P}_A + (1 - \alpha_A)\mathbf{P}_{A-1} & 1 < A < m \\ \mathbf{P}_n & A = m \end{cases} \tag{9}$$

where

$$\alpha_A = \begin{cases} 1 & 1 \leq A \leq k-p \\ \frac{\bar{\xi} - \xi_A}{\xi_{A+p} - \xi_A} & k-p+1 \leq A \leq k \\ 0 & A \geq k+1 \end{cases} \tag{10}$$

Knot values may be inserted multiple times but the continuity of the basis is reduced by one for each repetition of a give knot value. However, by choosing control variables as in (9) and (10) the continuity of the *curve* is preserved.

2.4 NURBS

A NURBS (Non-Uniform Rational B-Spline) is defined by a knot vector $\Xi = \{\xi_1, \xi_2, \ldots, \xi_{n+p+1}\}$, a set of rational basis functions $\mathbf{R} = \{R_{A,p}\}_{A=1}^n$, and a set of control points $\mathbf{P} = \{\mathbf{P}_A\}_{A=1}^n$ as

$$T(\xi) = \sum_{A=1}^{n} \mathbf{P}_A R_{A,p}(\xi). \tag{11}$$

The NURBS basis functions are defined as

$$R_{A,p}(\xi) = \frac{w_A N_{A,p}(\xi)}{W(\xi)} \tag{12}$$

where

$$W(\xi) = \sum_{B=1}^{n} w_B N_{B,p}(\xi) \tag{13}$$

is the weight function and w_B is the weight corresponding the the B^{th} basis function.

For more efficient computation, a rational curve in $\mathbb{R}^n$ can be represented by a polynomial curve in $\mathbb{R}^{n+1}$. The higher dimensional space is referred to as the projective space. As an example, if $\mathbf{P}_A$ is a control point of a NURBS curve then the corresponding homogeneous control point in projective space is $\tilde{\mathbf{P}}_A = \{w_A \mathbf{P}_A, w_A\}^T$. Thus, given a NURBS curve defined in $\mathbb{R}^n$ by (11) the corresponding B-spline curve defined in $\mathbb{R}^{n+1}$ is

$$T(\xi) = \sum_{A=1}^{n} \tilde{\mathbf{P}}_A N_{A,p}(\xi).$$

Working in the projective coordinate system allows the algorithms which operate on B-splines to be applied to NURBS. Once new control variables are computed for the B-spline in the projective coordinate system, simply dividing through by the weight yields the control variables for the NURBS.

3 The Bézier extraction operator

The Bézier extraction operator maps a piecewise Bernstein polynomial basis onto a B-spline basis. This transformation makes it possible to use piecewise C^0 Bézier elements as the finite element representation of a NURBS or T-spline. In this section we begin by showing how the Bézier element representation of a NURBS can be computed. We then define the Bézier extraction operator and show that it provides a mapping from a piecewise Bernstein polynomial basis onto a NURBS basis.

3.1 Bézier decomposition

To compute the Bézier elements of a NURBS we use Bézier decomposition. Technically, Bézier decomposition is accomplished by repeating all interior knots of a knot vector until they have a multiplicity equal to $p+1$. For our purposes, however, a multiplicity of p is sufficient. The result of this lower multiplicity is that neighboring Bézier elements will share control points. Since we are projecting to a smooth, continuous basis this has no effect and slightly reduces computation cost. We note that although multiple applications of knot insertion may be used for Bézier decomposition there are more efficient algorithms available (see [14] for details). Bézier decomposition of a univariate B-spline curve is illustrated as follows.

We begin with the cubic B-spline curve shown in Figure 1 and its associated knot vector

$$\Xi = \{0, 0, 0, 0, 1, 2, 3, 4, 4, 4, 4\}. \tag{14}$$

To decompose the curve into its Bézier elements we perform repeated knot insertion on all interior knots, beginning from the left, until they have a multiplicity equal to p, the degree of the curve. Thus, we will be performing knot refinement by inserting the knots $\{1, 1, 2, 2, 3, 3\}$ into the knot vector.

Figure 2 shows the sequence of basis functions and control variables created by inserting the new knots into the knot vector. Each inserted knot reduces the continuity of the basis by one at the knot location. The

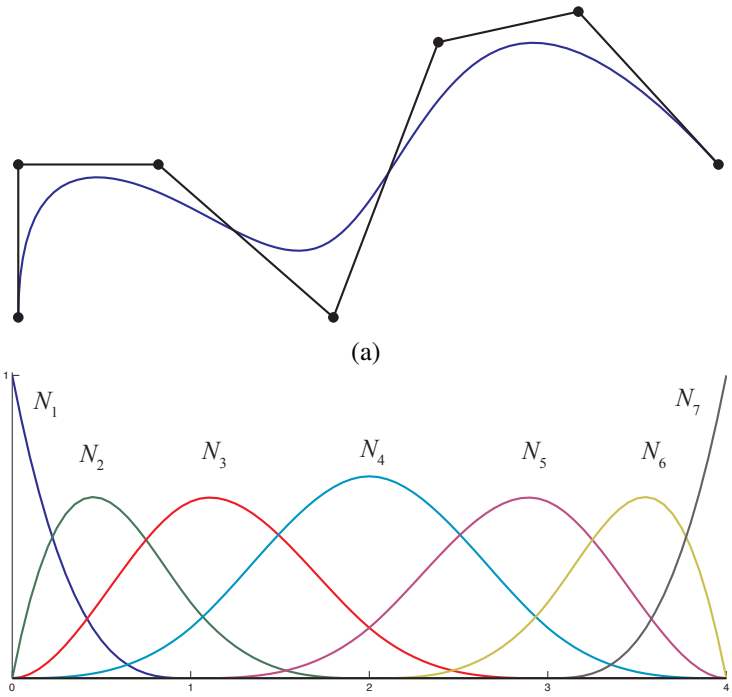


Figure 1: A cubic NURBS curve. (a) The curve and its control net. (b) The basis functions of the curve. The knot vector for the curve is $\{0, 0, 0, 0, 1, 2, 3, 4, 4, 4, 4\}$.

resulting basis has been decomposed into a set of C^0 Bézier elements with each element corresponding to a knot span in the original knot vector. The control points of the Bézier elements are computed by (9) and (10) each time a knot is inserted. Thus, the continuity of the curve is unchanged.

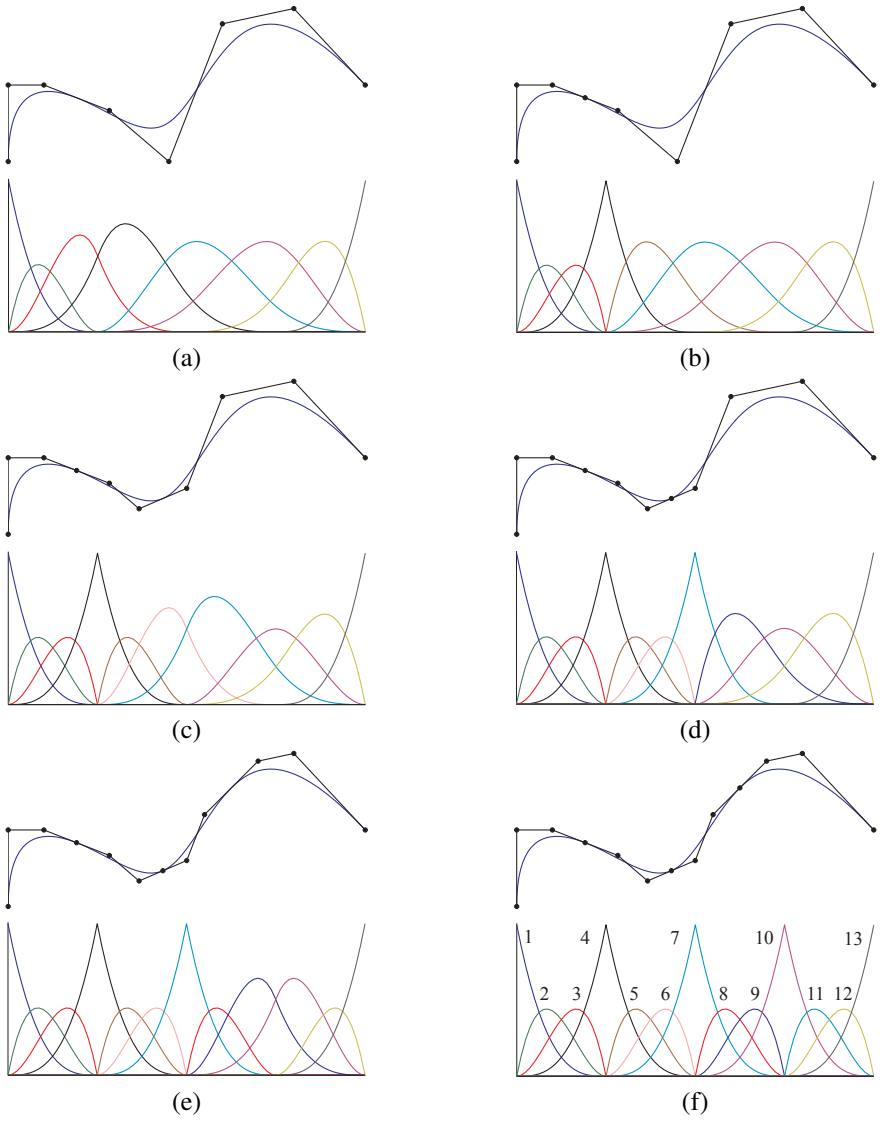


Figure 2: The sequence of basis functions created by inserting the knots $\{1, 1, 2, 2, 3, 3\}$ into the knot vector for the curve in Figure 1. The final set of basis functions in (f) is a collection of piecewise cubic Bézier basis functions. The numbers in (f) denote the numbering scheme of the Bézier basis functions.

3.2 Computing the Bézier extraction operator

We now show how the Bézier extraction operator for a NURBS curve may be computed in order to represent it in terms of a set of Bézier elements. Assume that we are given a knot vector $\Xi = \{\xi_1, \xi_2, \ldots, \xi_{n+p+1}\}$ and a set of control points, $\mathbf{P} = \{\mathbf{P}_A\}_{A=1}^{n}$, that define a B-spline curve (possibly in projective space). Let $\{\bar{\xi}_1, \bar{\xi}_2, \ldots, \bar{\xi}_m\}$ be the set of knots that are required to produce the Bézier decomposition of the B-spline. Then for each new knot, $\bar{\xi}_j$, $j = 1, 2, \ldots, m$, we define α_A^j, $A = 1, 2, \ldots, n+j$, to be the A^{th} alpha as defined in (10). Now, defining $\mathbf{C}^j \in \mathbb{R}^{(n+j-1)\times(n+j)}$ to be

$$\mathbf{C}^j = \begin{bmatrix} \alpha_1 & 1-\alpha_2 & 0 & \cdots & & & 0 \\ 0 & \alpha_2 & 1-\alpha_3 & 0 & \cdots & & 0 \\ 0 & 0 & \alpha_3 & 1-\alpha_4 & 0 & \cdots & 0 \\ \vdots & & & & & & \\ 0 & \cdots & & & 0 & \alpha_{(n+j-1)} & 1-\alpha_{(n+j)} \end{bmatrix} \tag{15}$$

and letting $\bar{\mathbf{P}}^1 = \mathbf{P}$ we can rewrite (9) in matrix form to represent the sequence of control variables created by knot refinement as

$$\bar{\mathbf{P}}^{j+1} = (\mathbf{C}^j)^T \bar{\mathbf{P}}^j. \tag{16}$$

The final set of control points, $\bar{\mathbf{P}}^{m+1}$, defines the Bézier elements of the decomposition. Letting $\mathbf{P}^b = \bar{\mathbf{P}}^{m+1}$ and defining $\mathbf{C}^T = (\mathbf{C}^m)^T(\mathbf{C}^{m-1})^T \ldots (\mathbf{C}^1)^T$ we get

$$\mathbf{P}^b = \mathbf{C}^T \mathbf{P}. \tag{17}$$

Recalling that $\mathbf{P}$ has dimension $n \times d$, we note that $\mathbf{C}$ has dimension $n \times (n+m)$, and $\mathbf{P}^b$ has dimension $(n+m) \times d$.

Recall that knot insertion causes no geometric or parametric change to a curve. Thus, if $\mathbf{B}(\xi) = \{\mathbf{B}_A(\xi)\}_{A=1}^{n+m}$ is the set of Bernstein basis functions defined by the final knot vector we have from (8) that

$$T(\xi) = (\mathbf{P}^b)^T \mathbf{B}(\xi) = (\mathbf{C}^T \mathbf{P})^T \mathbf{B}(\xi) = \mathbf{P}^T \mathbf{C}\mathbf{B}(\xi) = \mathbf{P}^T \mathbf{N}(\xi). \tag{18}$$

Since $\mathbf{P}$ is arbitrary, this shows we have constructed a new basis and linear operator such that

$$\mathbf{N}(\xi) = \mathbf{C}\mathbf{B}(\xi). \tag{19}$$

We call $\mathbf{C}$ the *Bézier extraction operator*.

It is important to note that the only input required to construct $\mathbf{C}$ is the knot vector. In other words, the extraction operator is an artifact of the parameterization and does not depend on the control points or basis functions. Thus, we can apply the extraction operator directly to NURBS as follows.

Define $\mathbf{W}$ to be the diagonal matrix of weights

$$\mathbf{W} = \begin{bmatrix} w_1 & & & \\ & w_2 & & \\ & & \ddots & \\ & & & w_n \end{bmatrix}. \tag{20}$$

Now, dropping the p subscript from (12) and writing it in matrix form we have

$$\mathbf{R}(\xi) = \frac{1}{W(\xi)} \mathbf{W}\mathbf{N}(\xi). \tag{21}$$

Using this relationship and (19) , we can write the NURBS domain from (11) in terms of the Bernstein basis as

$$\begin{aligned} T(\xi) &= \sum_{A=1}^{n} \mathbf{P}_A R_A(\xi) = \mathbf{P}^T \mathbf{R}(\xi) \\ &= \frac{1}{W(\xi)} \mathbf{P}^T \mathbf{W} \mathbf{N}(\xi) = \frac{1}{W(\xi)} \mathbf{P}^T \mathbf{W} \mathbf{C} \mathbf{B}(\xi) \\ &= \frac{1}{W(\xi)} (\mathbf{C}^T \mathbf{W} \mathbf{P})^T \mathbf{B}(\xi). \end{aligned} \tag{22}$$

Similary, letting $\mathbf{w} = \{w_A\}_{A=1}^n$ we can also rewrite the weight function, $W(\xi)$, from (13) in terms of the Bernstein basis as

$$\begin{aligned} W(\xi) &= \sum_{A=1}^{n} w_A N_A(\xi) = \mathbf{w}^T \mathbf{N}(\xi) \\ &= \mathbf{w}^T \mathbf{C} \mathbf{B}(\xi) = (\mathbf{C}^T \mathbf{w})^T \mathbf{B}(\xi) \\ &= (\mathbf{w}^b)^T \mathbf{B}(\xi) = W^b(\xi) \end{aligned} \tag{23}$$

where $\mathbf{w}^b = \mathbf{C}^T \mathbf{w}$ are the weights associated with the Bézier basis functions. To compute the Bézier element control points, $\mathbf{P}^b$, we first define $\mathbf{W}^b$ to be the diagonal matrix

$$\mathbf{W}^b = \begin{bmatrix} w_1^b & & & \\ & w_2^b & & \\ & & \ddots & \\ & & & w_{n+m}^b \end{bmatrix}. \tag{24}$$

The Bézier control points are now computed as

$$\mathbf{P}^b = (\mathbf{W}^b)^{-1} \mathbf{C}^T \mathbf{W} \mathbf{P}. \tag{25}$$

This equation can be interpreted as mapping the original control points into projective space, applying the extraction operator to compute the control points of the projected Bézier elements and then mapping these control points back from projective space.

If we multiply (25) by $\mathbf{W}^b$ we have the relationship

$$\mathbf{W}^b \mathbf{P}^b = \mathbf{C}^T \mathbf{W} \mathbf{P}, \tag{26}$$

but note that $\mathbf{W}^b \neq \mathbf{C}^T \mathbf{W}$. To get the final Bézier representation of the NURBS we substitute (23) and (26) into (22) to get

$$\begin{aligned} T(\xi) &= \frac{1}{W^b(\xi)} (\mathbf{W}^b \mathbf{P}^b)^T \mathbf{B}(\xi) \\ &= \sum_{A=1}^{n+m} \frac{\mathbf{P}_A^b w_A^b B_A(\xi)}{W^b(\xi)}. \end{aligned} \tag{27}$$

Thus, we have shown that a NURBS curve can be written equivalently in terms of a set of C^0 Bézier elements.

3.2.1 Localizing the extraction operator

If we compute $\mathbf{C}$ for the Bézier curve from Section 3.1 then equation (19) becomes

$$\begin{Bmatrix} N_1 \\ N_2 \\ N_3 \\ N_4 \\ N_5 \\ N_6 \\ N_7 \end{Bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1/2 & 1/4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/2 & 7/12 & 2/3 & 1/3 & 1/6 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/6 & 1/3 & 2/3 & 2/3 & 2/3 & 1/3 & 1/6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1/6 & 1/3 & 2/3 & 7/12 & 1/2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/4 & 1/2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} B_1 \\ B_2 \\ B_3 \\ B_4 \\ B_5 \\ B_6 \\ B_7 \\ B_8 \\ B_9 \\ B_{10} \\ B_{11} \\ B_{12} \\ B_{13} \end{Bmatrix}. \tag{28}$$

Bézier decomposition results in one Bézier element for each interval in the original knot vector. Thus, over each knot interval, the original NURBS basis can be represented as a linear combination of the basis functions of the Bézier element corresponding to that knot interval. For example, consider knot span $[0, 1[$ for the Bézier curve shown in Figure 1 and 2. The NURBS and Bernstein basis functions that are supported over this knot span are shown in Figure 3. Highlighting the corresponding entries in (28) for this knot span we get

$$\begin{Bmatrix} N_1 \\ N_2 \\ N_3 \\ N_4 \\ N_5 \\ N_6 \\ N_7 \end{Bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1/2 & 1/4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/2 & 7/12 & 2/3 & 1/3 & 1/6 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/6 & 1/3 & 2/3 & 2/3 & 2/3 & 1/3 & 1/6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1/6 & 1/3 & 2/3 & 7/12 & 1/2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/4 & 1/2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} B_1 \\ B_2 \\ B_3 \\ B_4 \\ B_5 \\ B_6 \\ B_7 \\ B_8 \\ B_9 \\ B_{10} \\ B_{11} \\ B_{12} \\ B_{13} \end{Bmatrix}. \tag{29}$$

Figure 4 visually shows this relationship.

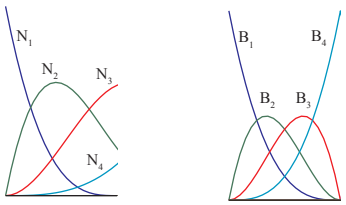


Figure 3: The basis functions over the knot span $[0, 1[$ from (a) the NURBS basis in Figure 1 and (b) the Bernstein basis in Figure 2f.

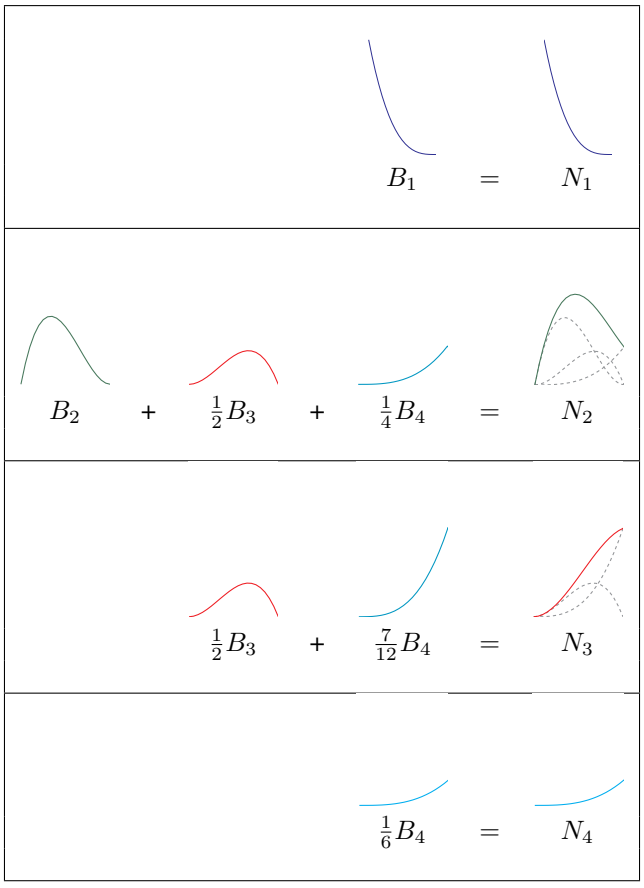


Figure 4: After knot insertion the original basis functions can be written as a linear combination of the basis functions for the Bézier elements.

Highlighing the relationship for the three remaining knot spans we have

$$\begin{Bmatrix} N_1 \\ N_2 \\ N_3 \\ N_4 \\ N_5 \\ N_6 \\ N_7 \end{Bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1/2 & 1/4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/2 & 7/12 & 2/3 & 1/3 & 1/6 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/6 & 1/3 & 2/3 & 2/3 & 2/3 & 1/3 & 1/6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1/6 & 1/3 & 2/3 & 7/12 & 1/2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/4 & 1/2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} B_1 \\ B_2 \\ B_3 \\ B_4 \\ B_5 \\ B_6 \\ B_7 \\ B_8 \\ B_9 \\ B_{10} \\ B_{11} \\ B_{12} \\ B_{13} \end{Bmatrix}, \tag{30}$$

$$\begin{Bmatrix} N_1 \\ N_2 \\ N_3 \\ N_4 \\ N_5 \\ N_6 \\ N_7 \end{Bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1/2 & 1/4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/2 & 7/12 & 2/3 & 1/3 & 1/6 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/6 & 1/3 & 2/3 & 2/3 & 2/3 & 1/3 & 1/6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1/6 & 1/3 & 2/3 & 7/12 & 1/2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/4 & 1/2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} B_1 \\ B_2 \\ B_3 \\ B_4 \\ B_5 \\ B_6 \\ B_7 \\ B_8 \\ B_9 \\ B_{10} \\ B_{11} \\ B_{12} \\ B_{13} \end{Bmatrix}, \tag{31}$$

and

$$\begin{Bmatrix} N_1 \\ N_2 \\ N_3 \\ N_4 \\ N_5 \\ N_6 \\ N_7 \end{Bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1/2 & 1/4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/2 & 7/12 & 2/3 & 1/3 & 1/6 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/6 & 1/3 & 2/3 & 2/3 & 2/3 & 1/3 & 1/6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1/6 & 1/3 & 2/3 & 7/12 & 1/2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/4 & 1/2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} B_1 \\ B_2 \\ B_3 \\ B_4 \\ B_5 \\ B_6 \\ B_7 \\ B_8 \\ B_9 \\ B_{10} \\ B_{11} \\ B_{12} \\ B_{13} \end{Bmatrix}. \tag{32}$$

To localize the extraction operator we first localize the basis functions to each element such that

$$\left\{\begin{array}{c} N_1^1 \\ N_2^1 \\ N_3^1 \\ N_4^1 \end{array}\right\} = \left\{\begin{array}{c} N_1 \\ N_2 \\ N_3 \\ N_4 \end{array}\right\}, \quad \left\{\begin{array}{c} N_1^2 \\ N_2^2 \\ N_3^2 \\ N_4^2 \end{array}\right\} = \left\{\begin{array}{c} N_2 \\ N_3 \\ N_4 \\ N_5 \end{array}\right\}, \tag{33}$$

$$\left\{\begin{array}{c} N_1^3 \\ N_2^3 \\ N_3^3 \\ N_4^3 \end{array}\right\} = \left\{\begin{array}{c} N_3 \\ N_4 \\ N_5 \\ N_6 \end{array}\right\}, \quad \left\{\begin{array}{c} N_1^4 \\ N_2^4 \\ N_3^4 \\ N_4^4 \end{array}\right\} = \left\{\begin{array}{c} N_4 \\ N_5 \\ N_6 \\ N_7 \end{array}\right\}, \tag{34}$$

and

$$\left\{\begin{array}{c} B_1^1 \\ B_2^1 \\ B_3^1 \\ B_4^1 \end{array}\right\} = \left\{\begin{array}{c} B_1 \\ B_2 \\ B_3 \\ B_4 \end{array}\right\}, \quad \left\{\begin{array}{c} B_1^2 \\ B_2^2 \\ B_3^2 \\ B_4^2 \end{array}\right\} = \left\{\begin{array}{c} B_4 \\ B_5 \\ B_6 \\ B_7 \end{array}\right\}, \tag{35}$$

$$\left\{\begin{array}{c} B_1^3 \\ B_2^3 \\ B_3^3 \\ B_4^3 \end{array}\right\} = \left\{\begin{array}{c} B_7 \\ B_8 \\ B_9 \\ B_{10} \end{array}\right\}, \quad \left\{\begin{array}{c} B_1^4 \\ B_2^4 \\ B_3^4 \\ B_4^4 \end{array}\right\} = \left\{\begin{array}{c} B_{10} \\ B_{11} \\ B_{12} \\ B_{13} \end{array}\right\} \tag{36}$$

where the superscript indicates the element number. Now, with these local quantities defined we can localize the coefficients from the global extraction operator in (28). Using (29) thru (32) to construct the element extraction operators and we get

$$\left\{\begin{array}{c} N_1^1 \\ N_2^1 \\ N_3^1 \\ N_4^1 \end{array}\right\} = \left[\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & 1 & 1/2 & 1/4 \\ 0 & 0 & 1/2 & 7/12 \\ 0 & 0 & 0 & 1/6 \end{array}\right] \left\{\begin{array}{c} B_1^1 \\ B_2^1 \\ B_3^1 \\ B_4^1 \end{array}\right\}, \tag{37}$$

$$\left\{\begin{array}{c} N_1^2 \\ N_2^2 \\ N_3^2 \\ N_4^2 \end{array}\right\} = \left[\begin{array}{cccc} 1/4 & 0 & 0 & 0 \\ 7/12 & 2/3 & 1/3 & 1/6 \\ 1/6 & 1/3 & 2/3 & 2/3 \\ 0 & 0 & 0 & 1/6 \end{array}\right] \left\{\begin{array}{c} B_1^2 \\ B_2^2 \\ B_3^2 \\ B_4^2 \end{array}\right\}, \tag{38}$$

$$\left\{\begin{array}{c} N_1^3 \\ N_2^3 \\ N_3^3 \\ N_4^3 \end{array}\right\} = \left[\begin{array}{cccc} 1/6 & 0 & 0 & 0 \\ 2/3 & 2/3 & 1/3 & 1/6 \\ 1/6 & 1/3 & 2/3 & 7/12 \\ 0 & 0 & 0 & 1/4 \end{array}\right] \left\{\begin{array}{c} B_1^3 \\ B_2^3 \\ B_3^3 \\ B_4^3 \end{array}\right\}, \tag{39}$$

$$\left\{\begin{array}{c} N_1^4 \\ N_2^4 \\ N_3^4 \\ N_4^4 \end{array}\right\} = \left[\begin{array}{cccc} 1/6 & 0 & 0 & 0 \\ 7/12 & 1/2 & 0 & 0 \\ 1/4 & 1/2 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array}\right] \left\{\begin{array}{c} B_1^4 \\ B_2^4 \\ B_3^4 \\ B_4^4 \end{array}\right\}. \tag{40}$$

We can now write (19) in its localized element form as

$$\mathbf{N}^e = \mathbf{C}^e \mathbf{B}^e. \tag{41}$$

3.2.2 Computing the localized extraction operator

In practice the global extraction operator, $\mathbf{C}$, is never computed. The local extraction operators, $\mathbf{C}^e$, can be computed directly by modifying existing Bézier decomposition algorithms. Algorithm 1 is based on the Bézier decomposition algorithm presented in [14]. In its original form, this algorithm used (9) and (10) to compute the control points of the Bézier elements of a curve. We have modified this algorithm so that it uses only (10) to compute the coefficients of the element extraction operators. Note that by inserting knots from left to right we are able to reduce the total computational cost by updating overlapping coefficients between neighboring elements.

Algorithm 1 An algorithm to compute the local extraction operators for a one-dimensional B-spline parametric domain. Note that the input does not require the geometric information of the control variables.

input Knot vector, $\mathtt{U} = \{u_1, \cdots, u_m\}$
Number of knots, `m`
Curve degree, `p`
output Number of elements, `nb`
Element extraction operators, $\mathtt{C}^e$, $e = 1, 2, \ldots,$`nb`

```
// Initializations:
a = p+1;
b = a+1;
nb = 1;
C^1 = I;
while  b < m  do
   C^(nb+1) = I; // Initialize the next extraction operator.
   i = b;

   // Count multiplicity of the knot at location b.
   while  b < m && U(b+1) == U(b)  do  b = b+1;
   mult = b-i+1;

   if  mult < p  do
     // Use (10) to compute the alphas.
     numer = U(b)-U(a);
     for  j = p,p-1,...,mult+1  do
        alphas(j-mult) = numer / (U(a+j)-U(a));
     end
     r = p-mult;
     // Update the matrix coefficients for r new knots
     for  j=1,2,...,r  do
        save = r-j+1;
        s = mult+j;
        for  k=p+1,p,...,s+1  do
           alpha = alphas(k-s);
```

```
            // The following line corresponds to (9).
            C^nb(:,k) = alpha*C^nb(:,k) + (1.0-alpha)*C^nb(:,k-1);
          end
          if  b < m  do
            // Update overlapping coefficients of the next operator.
            C^nb+1(save:j+save,save) = C^nb(p-j+1:p+1,p+1);
          end
        end
        nb = nb + 1; // Finished with the current operator.
        if  b < m   do
          // Update indices for the next operator.
          a = b;
          b = b+1;
        end
      end
    end
```

3.3 NURBS surfaces and solids

It is convenient to take advantage of the tensor product structure of higher dimension NURBS to compute the Bézier extraction operators for higher dimension. For consistency, we introduce a mapping, $\tilde{A}$, between the tensor product space and the global indexing of the basis functions and control points. Let $i = 1, 2, \ldots, n$, $j = 1, 2, \ldots, m$, and $k = 1, 2, \ldots, l$ then in two dimensions we define

$$\tilde{A}(i, j) = m(i - 1) + j \tag{42}$$

and in three dimensions

$$\tilde{A}(i, j, k) = (l \times m)(i - 1) + l(j - 1) + k. \tag{43}$$

NURBS basis functions for surfaces and solids are defined by the tensor product of univariate B-spline basis functions. If $N_{i,p}(\xi)$, $M_{j,q}(\eta)$, and $L_{l,r}(\zeta)$ are univariate B-spline basis functions, then in two dimensions with $A = \tilde{A}(i, j)$ and $\hat{A} = \tilde{A}(\hat{i}, \hat{j})$

$$R^{p,q}_A(\xi, \eta) = \frac{M_{i,q}(\eta)N_{j,p}(\xi)w_A}{\sum_{\hat{i}=1}^{n}\sum_{\hat{j}=1}^{m} M_{\hat{i},q}(\eta)N_{\hat{j},p}(\xi)w_{\hat{A}}} \tag{44}$$

and in three dimensions with $A = \tilde{A}(i, j, k)$ and $\hat{A} = \tilde{A}(\hat{i}, \hat{j}, \hat{k})$ depending on the spatial dimension

$$R^{p,q,r}_A(\xi, \eta, \zeta) = \frac{L_{i,r}(\zeta)M_{j,q}(\eta)N_{k,p}(\xi)w_A}{\sum_{\hat{i}=1}^{n}\sum_{\hat{j}=1}^{m}\sum_{\hat{k}=1}^{l} L_{\hat{i},r}(\zeta)M_{\hat{j},q}(\eta)N_{\hat{k},p}(\xi)w_{\hat{A}}} \tag{45}$$

are the surface and solid NURBS basis functions respectively. For $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, m$, a NURBS surface is defined by a given control net $\{\mathbf{P}_A\}$, $A = 1, 2, \ldots, (n \times m)$, and knot vectors $\Xi = \{\xi_1, \xi_2, \ldots, \xi_{n+p+1}\}$ and $\mathcal{H} = \{\eta_1, \eta_2, \ldots, \eta_{m+q+1}\}$ as

$$S(\xi, \eta) = \sum_{A=1}^{n\times m} R^{p,q}_A(\xi, \eta)\mathbf{P}_A. \tag{46}$$

Similarly for a NURBS solid with $i = 1, 2, \ldots, n$, $j = 1, 2, \ldots, m$, $k = 1, 2, \ldots, l$, $\{\mathbf{P}_A\}$, $A = 1, 2, \ldots, (n \times m \times l)$, and knot vectors $\Xi = \{\xi_1, \xi_2, \ldots, \xi_{n+p+1}\}$, $\mathcal{H} = \{\eta_1, \eta_2, \ldots, \eta_{m+q+1}\}$, and $\mathcal{Z} = \{\zeta_1, \zeta_2, \ldots, \zeta_{l+r+1}\}$ we have

$$S(\xi, \eta, \zeta) = \sum_{A=1}^{n \times m \times l} R_A^{p,q,r}(\xi, \eta, \zeta)\mathbf{P}_A. \tag{47}$$

To define the surface and solid element extraction operators we let $\mathbf{C}_\xi^i$, $\mathbf{C}_\eta^j$, and $\mathbf{C}_\zeta^k$ be the i^{th}, j^{th}, and k^{th} univariate element extraction operators in the ξ, η, and ζ direction. Then we have for a surface and solid respectively

$$\mathbf{C}_A^e = \mathbf{C}_\eta^i \otimes \mathbf{C}_\xi^j \tag{48}$$

and

$$\mathbf{C}_A^e = \mathbf{C}_\zeta^i \otimes \mathbf{C}_\eta^j \otimes \mathbf{C}_\xi^k \tag{49}$$

where $\otimes$ is defined for two matrices $\mathbf{A}$ and $\mathbf{B}$, which may have different dimensions, as

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} A_{11}\mathbf{B} & A_{12}\mathbf{B} & \cdots \\ A_{21}\mathbf{B} & A_{22}\mathbf{B} & \\ \vdots & & \ddots \end{bmatrix}. \tag{50}$$

4 Bézier extraction and the finite element framework

Bézier extraction provides an element structure that can be incorporated into existing finite element frameworks. In the sections which follow we explore this application of Bézier extraction.

We assume that we have a geometric domain defined by the mapping

$$\mathbf{x}(\boldsymbol{\xi}) = \sum_{A=1}^{n} \mathbf{P}_A R_A(\boldsymbol{\xi}) \tag{51}$$

where $\boldsymbol{\xi} = (\xi_1, \xi_2, \xi_3) = (\xi, \eta, \zeta)$ is the parametric coordinate, $\mathbf{P} = \{\mathbf{P}_A\}_{A=1}^n$ is a set of control points and $\mathbf{R} = \{R_A\}_{A=1}^n$ is a NURBS basis for which a Bézier extraction operator, $\mathbf{C}$, and Bernstein basis, $\mathbf{B} = \{B_A\}_{A=1}^m$, can be computed.

Remark: We note that T-splines satisfy this assumption. Thus, all that follows in this section can be applied to T-splines.

4.1 Incorporating $\mathbf{C}^e$ into the finite element formulation

We now show how the element extraction operator, $\mathbf{C}^e$, can be incorporated into the finite element formulation. We begin with an abstract weak formulation. Letting $\mathcal{S}$ be the trail solution space and $\mathcal{V}$ be the space of weighting functions we have

$$(W) \begin{cases} \text{Given } f, \text{ find } u \in \mathcal{S} \text{ such that for all } w \in \mathcal{V} \\ a(w, u) = (w, f) \end{cases} \tag{52}$$

where $a(\cdot,\cdot)$ is a bilinear form and $(\cdot,\cdot)$ is the L^2 inner-product. Both $\mathcal{S}$ and $\mathcal{V}$ are assumed to be subspaces of the Sobolev space H^1 (see [10] for further details). Galerkin's method consists of constructing finite-dimensional approximations of $\mathcal{S}$ and $\mathcal{V}$. In an isogeometric setting we construct the finite-dimensional subspaces $\mathcal{S}^h \subset \mathcal{S}$ and $\mathcal{V}^h \subset \mathcal{V}$ from the basis which describes the geometry. The Galerkin formulation is then

$$(G)\begin{cases} \text{Given } f, \text{ find } u^h \in \mathcal{S}^h \text{ such that for all } w^h \in \mathcal{V}^h \\ \qquad a(w^h, u^h) = (w^h, f) \end{cases} \tag{53}$$

In isogeometric analysis, the isoparametric concept is invoked, that is, the field in question is represented in terms of the geometric basis. We can write u^h and w^h as

$$w^h = \sum_{A=1}^{n} c_A R_A \tag{54}$$

$$u^h = \sum_{B=1}^{n} d_B R_B \tag{55}$$

where c_A and d_B are control variables. Substituting these into (53) yields the matrix form of the problem

$$\mathbf{K}\mathbf{d} = \mathbf{F} \tag{56}$$

where

$$\mathbf{K} = [K_{AB}], \tag{57}$$

$$\mathbf{F} = \{F_A\}, \tag{58}$$

$$\mathbf{d} = \{d_B\}, \tag{59}$$

$$K_{AB} = a\left(R_A, R_B\right), \tag{60}$$

$$F_A = \left(R_A, f\right). \tag{61}$$

The preceding formulation applies to scalar-valued partial differential equations, such as the heat conduction equation. The generalization to vector-valued partial differential equations, such as elasticity, follows standard procedures described in [10].

4.1.1 The element shape function routine

As in standard finite elements, the global stiffness matrix, $\mathbf{K}$, and force vector, $\mathbf{F}$, can be computed by performing integration over the Bézier elements to form the element stiffness matrices and force vectors—$\mathbf{k}^e$ and $\mathbf{f}^e$ respectively—and assembling these into their global counterparts. The element form of (60) and (61) is

$$k^e_{ab} = a_e(R^e_a, R^e_b), \tag{62}$$

$$f^e_a = (R^e_a, f)_e \tag{63}$$

where $a_e(\cdot,\cdot)$ denotes the bilinear form restricted to the element, $(\cdot,\cdot)_e$ is the L^2 inner-product restricted to the element, and R^e_a are the element shape functions. The integration is usually performed by Gaussian quadrature. As shown in Figure 5 for the two dimensional case, the integrals are pulled back, first onto the parametric element and then onto a bi-unit parent element. This requires the evaluation of the global

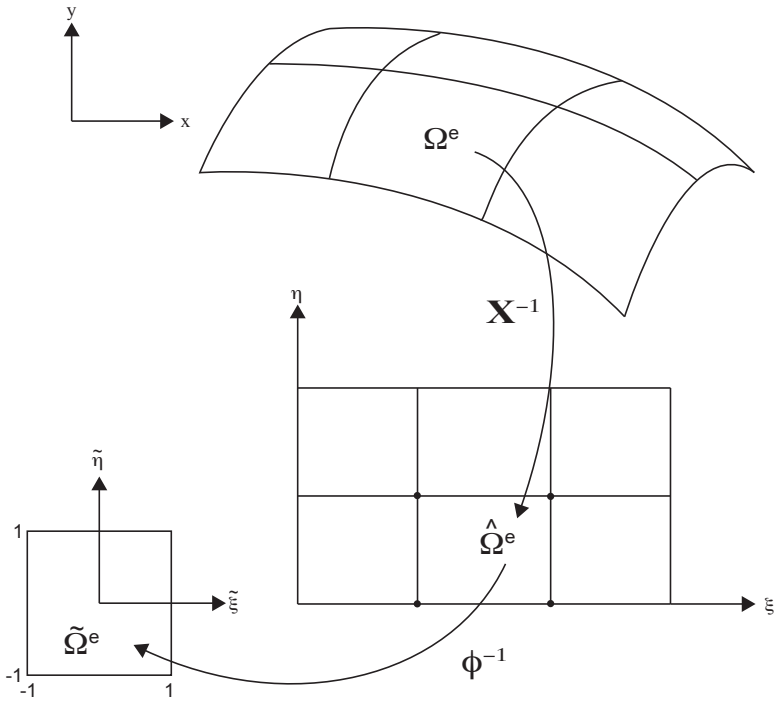


Figure 5: Integration is performed by Gaussian quadrature on each element. The physical element is pulled back first to the parametric domain through the geometrical mapping and then to the parent element through an affine mapping. For NURBS more efficient quadrature rules may be devised (see [12]).

basis functions, their derivatives, and the Jacobian determinate of the pullback from the physical space to the parent element at each quadrature point in the parent element. These evaluations are done in an element shape function routine.

In order to perform the shape function routine evaluations we recall from Section 3.2 that

$$\mathbf{R}(\boldsymbol{\xi}) = \mathbf{W}\frac{\mathbf{N}(\boldsymbol{\xi})}{W(\boldsymbol{\xi})} = \mathbf{W}\mathbf{C}\frac{\mathbf{B}(\boldsymbol{\xi})}{W^b(\boldsymbol{\xi})}. \tag{64}$$

Localizing everything to the element, this becomes

$$\mathbf{R}^e(\boldsymbol{\xi}) = \mathbf{W}^e\mathbf{C}^e\frac{\mathbf{B}^e(\boldsymbol{\xi})}{W^b(\boldsymbol{\xi})}. \tag{65}$$

Thus, the derivatives of $\mathbf{R}^e$ with respect to the parametric coordinates, ξ_i, are

$$\frac{\partial \mathbf{R}^e(\boldsymbol{\xi})}{\partial \xi_i} = \mathbf{W}^e\mathbf{C}^e\frac{\partial}{\partial \xi_i}\left(\frac{\mathbf{B}^e(\boldsymbol{\xi})}{W^b(\boldsymbol{\xi})}\right) = \mathbf{W}^e\mathbf{C}^e\left(\frac{1}{W^b(\boldsymbol{\xi})}\frac{\partial \mathbf{B}^e(\boldsymbol{\xi})}{\partial \xi_i} - \frac{\partial W^b(\boldsymbol{\xi})}{\partial \xi_i}\frac{\mathbf{B}^e(\boldsymbol{\xi})}{(W^b(\boldsymbol{\xi}))^2}\right). \tag{66}$$

To compute the derivatives with respect to the physical coordinates, (x_1, x_2, x_3), we apply the chain rule to

get

$$\frac{\partial \mathbf{R}^e(\boldsymbol{\xi})}{\partial x_i} = \sum_{j=1}^{3} \frac{\partial \mathbf{R}^e(\boldsymbol{\xi})}{\partial \xi_j} \frac{\partial \xi_j}{\partial x_i}. \tag{67}$$

To compute $\partial\boldsymbol{\xi}/\partial\mathbf{x}$ we first compute $\partial\mathbf{x}/\partial\boldsymbol{\xi}$ using (51) and (66) and then take its inverse. Since we are integrating over the parent element we must also compute the Jacobian determinant of the mapping from the parent element to the physical space, J. It is computed as

$$J = \left| \frac{\partial \mathbf{x}}{\partial \tilde{\boldsymbol{\xi}}} \right| = \left| \frac{\partial \mathbf{x}}{\partial \boldsymbol{\xi}} \frac{\partial \boldsymbol{\xi}}{\partial \tilde{\boldsymbol{\xi}}} \right|. \tag{68}$$

Higher-order derivatives can also be computed as described in [5]. An element shape function routines in given in Appendix A.

Remark If $\mathbf{R}^e$ and $\mathbf{C}^e$ are computed as the tensor product of univariate components—as described in Section 3.3 for NURBS—their tensor product structure can be exploited to reduce computational cost when computing the matrix products in (65) and (66). The following procedure can be used: (i) pre-compute the univariate Bernstein basis functions and derivatives at the prescribed quadrature points in each direction, (ii) at each call of the shape function routine, compute the univariate B-spline results for (65) and (66), and (iii) depending on dimension, use (46) or (47) to compute tensor product values of these results.

5 A two-dimensional quadratic NURBS example

We now present an elasticity example to illustrate how the Bézier extraction operator can be used to compute a Bézier mesh and associated data processing arrays for a NURBS. The example geometry, consisting of one quarter of an annulus, is shown in Figure 6. We assume homogeneous Dirichlet boundary conditions on the left edge of the domain. The geometry can be represented exactly by a quadratic NURBS surface, so we will select $p = 2$.

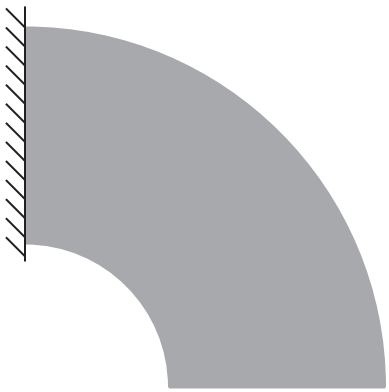


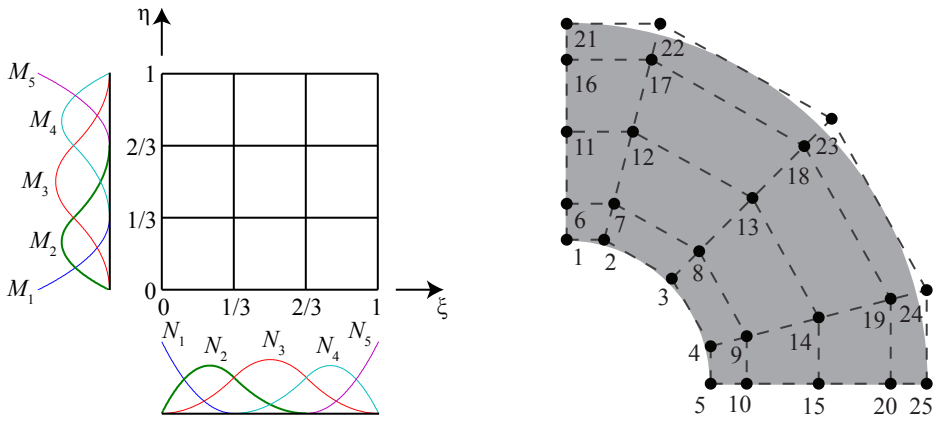
Figure 6: A bivariate NURBS example. The hash marks on the left boundary indicate homogeneous Dirichlet conditions.

The parametric domain for this example, shown in Figure 7a, is defined by the two uniform open knot vectors

$$\Xi = \left\{0, 0, 0, \frac{1}{3}, \frac{2}{3}, 1, 1, 1\right\} \tag{69}$$

$$\mathcal{H} = \left\{0, 0, 0, \frac{1}{3}, \frac{2}{3}, 1, 1, 1\right\}. \tag{70}$$

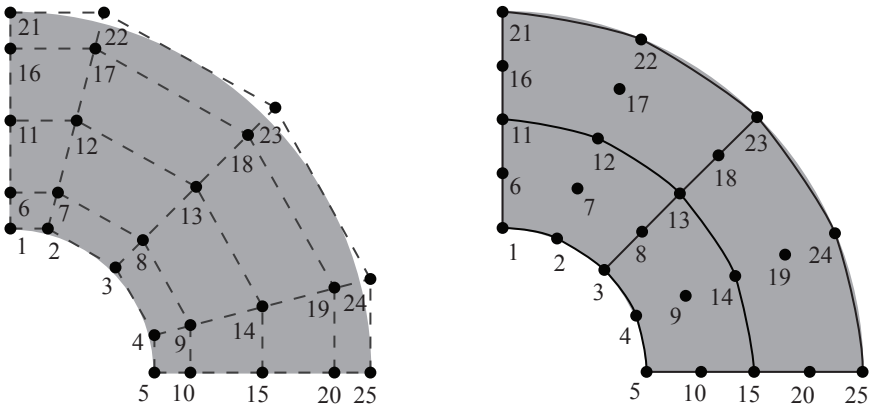
The knot vectors define the univariate B-spline basis functions that are also shown in Figure 7a. These functions define the tensor product basis functions that will be used in the analysis. The control mesh along with the numbering of the control variables is shown in Figure 7b. The weights and control variables listed in Appendix B Table 3 have been chosen such that the geometry is represented exactly by the NURBS surface.



(a) (b)

Figure 7: The parametric and control mesh for the NURBS in Figure 6. (a) The parametric mesh is defined by the global knot vectors. The bivariate basis functions for the NURBS patch are constructed as the tensor product of the one-dimensional basis functions in each parametric direction. The one-dimensional basis functions are shown for each direction. (b) Assigning control variables to each basis function generates the physical NURBS domain. The control points form the control mesh which is a piecewise bilinear interpolation of the control points.

Figure 8 shows the control mesh next to a standard C^0 quadratic isoparametric finite element mesh that has the same number of degrees of freedom. We note that while the NURBS represents the geometry exactly it is interpolatory only at the corners. The finite element mesh, on the other hand, is interpolatory at all nodes but only approximates the geometry. The ID array shown in Figure 8c maps the degree-of-freedom number (i.e., direction index of the displacement component) and global control point number to the corresponding equation number in the global system. A zero value indicates a degree of freedom that is constrained by the boundary condition and for which the equation has been removed from the global system. The numbering of both meshes is such that the ID array is identical for both.



NURBS control mesh
(a)

Finite element mesh
(b)

ID array:

		Global control point number (A)													
		1	2	3	4	5	6	7	8	9	10	11	12	13	14
Degree-of-freedom number (i)	1	0	1	3	5	7	0	9	11	13	15	0	17	19	21
	2	0	2	4	6	8	0	10	12	14	16	0	18	20	22

15	16	17	18	19	20	21	22	23	24	25
23	0	25	27	29	31	0	33	35	37	39
24	0	26	28	30	32	0	34	36	38	40

$$P = \mathrm{ID}(i, A)$$

(c)

Figure 8: A comparison between a quadratic NURBS control mesh and an approximately equivalent (geometrically) C^0 finite element mesh. (a) The control mesh for the bivariate NURBS example. The NURBS control mesh only interpolates the corner points but produces an exact geometry. (b) An approximate C^0 finite element mesh. The elements are interpolatory but only approximate the geometry. (c) The ID array maps degree-of-freedom numbers, i, and global control point numbers, A, to global equation numbers, P. A zero value indicates the degree-of-freedom has been removed from the global system due to the specification of an essential Dirichlet boundary condition, as in standard finite element analysis [10]. Note that this array is the same for both meshes.

5.1 Computing the extraction operators

As was discussed in Section 3 we can use the element extraction operators to compute the control points of the Bézier elements. Bézier extraction is performed on the univariate basis functions in the ξ and η directions for the parametric mesh in Figure 7a to compute the corresponding univariate element extraction operators $\mathbf{C}_\xi^i$, $i = 1, 2, 3$, and $\mathbf{C}_\eta^j$, $j = 1, 2, 3$. The bivariate element extraction operator is then computed as

$$\mathbf{C}^e = \mathbf{C}_\eta^i \otimes \mathbf{C}_\xi^j \tag{71}$$

where the multi-index i, j has been mapped to the element number such that $e = 3(i-1) + j$ and $\otimes$ is defined in (50). In this example the knot vectors in each parametric direction are uniform and open. This simplifies the computation of the extraction operators. In fact, other than the first and last extraction operator, which must account for the Bézier end conditions, all other operators are exactly the same. In general, if Ξ is a quadratic uniform open knot vector with no repeated interior knots and n non-zero intervals (i.e., elements) the univariate extraction operators are simply

$$\mathbf{C}_\xi^1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1/2 \\ 0 & 0 & 1/2 \end{bmatrix} \tag{72}$$

$$\mathbf{C}_\xi^2 = \mathbf{C}_\xi^3 = \cdots = \mathbf{C}_\xi^{n-1} = \begin{bmatrix} 1/2 & 0 & 0 \\ 1/2 & 1 & 1/2 \\ 0 & 0 & 1/2 \end{bmatrix} \tag{73}$$

$$\mathbf{C}_\xi^n = \begin{bmatrix} 1/2 & 0 & 0 \\ 1/2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{74}$$

and $\mathbf{C}_\eta^i = \mathbf{C}_\xi^i$, $i = 1, 2, \ldots, n$. For the current example $n = 3$ in both the ξ and η direction. The extraction operators for the tensor product element domains are

$$\begin{aligned} \mathbf{C}^1 = \mathbf{C}_\eta^1 \otimes \mathbf{C}_\xi^1 &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1/2 \\ 0 & 0 & 1/2 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1/2 \\ 0 & 0 & 1/2 \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1/2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1/2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1/2 & 0 & 1/2 & 1/4 \\ 0 & 0 & 0 & 0 & 0 & 1/2 & 0 & 0 & 1/4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1/2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/2 & 1/4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/4 \end{bmatrix}, \end{aligned} \tag{75}$$

$$\mathbf{C}^2 = \mathbf{C}^1_\eta \otimes \mathbf{C}^2_\xi = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1/2 \\ 0 & 0 & 1/2 \end{bmatrix} \otimes \begin{bmatrix} 1/2 & 0 & 0 \\ 1/2 & 1 & 1/2 \\ 0 & 0 & 1/2 \end{bmatrix}$$

$$= \begin{bmatrix} 1/2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1/2 & 1 & 1/2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/2 & 0 & 0 & 1/4 & 0 & 0 \\ 0 & 0 & 0 & 1/2 & 1 & 1/2 & 1/4 & 1/2 & 1/4 \\ 0 & 0 & 0 & 0 & 0 & 1/2 & 0 & 0 & 1/4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1/4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1/4 & 1/2 & 1/4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/4 \end{bmatrix}, \tag{76}$$

$$\vdots$$

$$\mathbf{C}^9 = \mathbf{C}^3_\eta \otimes \mathbf{C}^3_\xi = \begin{bmatrix} 1/2 & 0 & 0 \\ 1/2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1/2 & 0 & 0 \\ 1/2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1/4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1/4 & 1/2 & & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1/4 & 0 & 0 & 1/2 & 0 & 0 & 0 & 0 & 0 \\ 1/4 & 1/2 & 0 & 1/2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/2 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1/2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1/2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1/2 \end{bmatrix}. \tag{77}$$

5.2 Constructing the IEN array

The IEN array is a map between the local and global numbering of the NURBS basis functions. The IEN array must be constructed before computing the Bézier elements for a NURBS. In standard finite elements the local numbering typically corresponds to the local nodes [10]. In the setting of the Bézier extraction operator, however, a local control point may not coincide with a global control point.

To construct the IEN array we first establish the NURBS basis function numbering scheme. In the bivariate case, each global NURBS basis function is the product of two univariate basis functions. Thus, for the current example, the NURBS basis functions can be written as

$$R_A(\xi, \eta) = M_i(\eta) N_j(\xi) \tag{78}$$

where the global numbering is defined as $A = 5(i-1) + j$, where $1 \leq i, j \leq 5$.

The next step in constructing the IEN array is to determine which NURBS basis functions will be supported over each Bézier element. There is a one-to-one relationship between the tensor product elements of the parametric space and the Bézier elements that are being computed. Thus, by determining which functions are supported over each univariate knot span and taking the appropriate tensor product of these functions we can determine which functions will be supported by each Bézier element. Figure 9 shows each tensor product in the parametric mesh and the associated one-dimensional basis functions they support. Using this information, the IEN array is constructed as shown in Table 1. The LM array can then be constructed as the composition of the ID and IEN arrays as in standard finite element analysis (see Table 2).

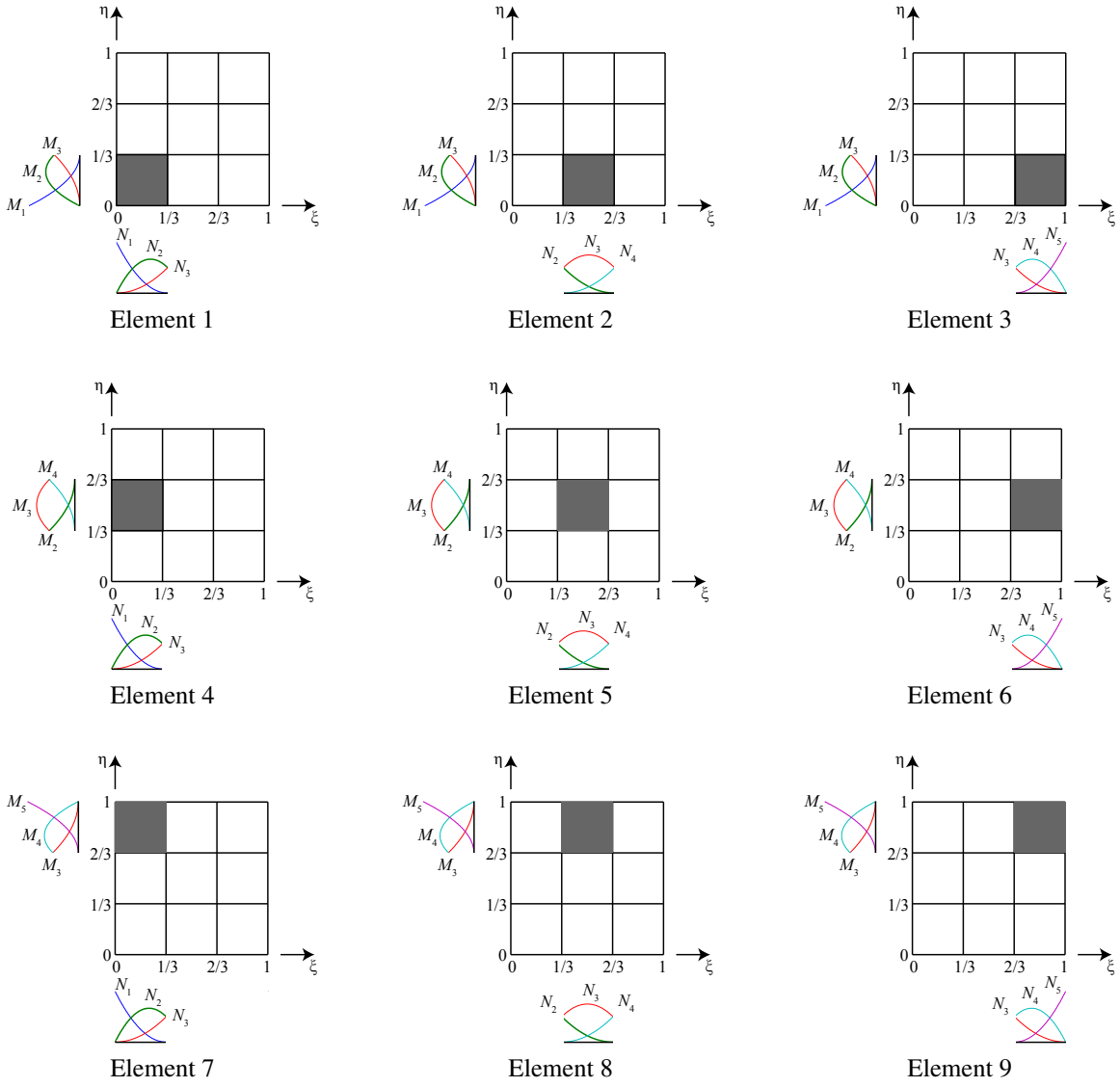


Figure 9: The univariate B-spline basis functions supported by each Bézier element. There is a one-to-one correspondence between Bézier elements and the parametric elements in the tensor product partitioning of the parametric mesh created by the global knot vectors. This figure shows each parametric element and the associated one-dimensional basis functions it supports. This information is used to construct the IEN array.

	Element number (e)								
a	1	2	3	4	5	6	7	8	9
1	1	2	3	6	7	8	11	12	13
2	2	3	4	7	8	9	12	13	14
3	3	4	5	8	9	10	13	14	15
4	6	7	8	11	12	13	16	17	18
5	7	8	9	12	13	14	17	18	19
6	8	9	10	13	14	15	18	19	20
7	11	12	13	16	17	18	21	22	23
8	12	13	14	17	18	19	22	23	24
9	13	14	15	18	19	20	23	24	25

$$A = \text{IEN}(a, e)$$

Table 1: The IEN array is constructed using the information from Figure 9. The IEN array maps the local basis function number (a) and the element number (e) to the corresponding global control point (A).

		Element number (e)								
a	i	1	2	3	4	5	6	7	8	9
1	1	0	1	3	0	9	11	0	17	19
	2	0	2	4	0	10	12	0	18	20
2	1	1	3	5	9	11	13	17	19	21
	2	2	4	6	10	12	14	18	20	22
3	1	3	5	7	11	13	15	19	21	23
	2	4	6	9	12	14	16	20	22	24
4	1	0	9	11	0	17	19	0	25	27
	2	0	10	12	0	18	20	0	26	28
5	1	9	11	13	17	19	21	25	27	29
	2	10	12	14	18	20	22	26	28	30
6	1	11	13	15	19	21	23	27	29	31
	2	12	14	16	20	22	24	28	30	32
7	1	0	17	19	0	25	27	0	33	35
	2	0	18	20	0	26	28	0	34	36
8	1	17	19	21	25	27	29	33	35	37
	2	18	20	22	26	28	30	34	36	38
9	1	19	21	23	27	29	31	35	37	39
	2	20	22	24	28	30	32	36	38	40

$$P = \text{ID}(i, \text{IEN}(a, e))$$

Table 2: The LM array is the composition of the ID and IEN arrays. P is the global equation number, and i is the degree-of-freedom number.

5.3 Computing the Bézier mesh

Once the IEN array and element extraction operators have been computed, the control points for the Bézier elements can also be computed by localizing (25). In general we have

$$\mathbf{Q}^e = \left(\mathbf{W}^{b,e}\right)^{-1} \left(\mathbf{C}^e\right)^T \mathbf{W}^e \mathbf{P}^e \tag{79}$$

where $\mathbf{Q}^e$ are the Bézier control points, $\mathbf{P}^e$ the NURBS control points, $\mathbf{W}^{b,e}$ the diagonal matrix of Bézier weights, and $\mathbf{W}^e$ the diagonal matrix of NURBS weights corresponding to element e. In this example we have for each element

$$\begin{bmatrix} \mathbf{Q}_1^e \\ \mathbf{Q}_2^e \\ \mathbf{Q}_3^e \\ \mathbf{Q}_4^e \\ \mathbf{Q}_5^e \\ \mathbf{Q}_6^e \\ \mathbf{Q}_7^e \\ \mathbf{Q}_8^e \\ \mathbf{Q}_9^e \end{bmatrix} = \left(\mathbf{W}^{b,e}\right)^{-1} \left(\mathbf{C}^e\right)^T \mathbf{W}^e \begin{bmatrix} \mathbf{P}_1^e \\ \mathbf{P}_2^e \\ \mathbf{P}_3^e \\ \mathbf{P}_4^e \\ \mathbf{P}_5^e \\ \mathbf{P}_6^e \\ \mathbf{P}_7^e \\ \mathbf{P}_8^e \\ \mathbf{P}_9^e \end{bmatrix} \tag{80}$$

(see Appendix B for a complete list of the element control points). The resulting Bézier control elements are shown in Figure 10 with the NURBS control variables that contribute to the location of the element control variables indicated by the $\bigcirc$'s.

The collection of the Bézier elements is called the Bézier control mesh. We now have four "meshes" for the NURBS problem as shown in Figure 11: the parametric mesh, the control mesh, the Bézier control mesh, and the Bézier physical mesh. The parametric mesh has been used to facilitate the presentation of this example but it does not need to be constructed in practice. The knot vectors contain all the information that is needed to construct the extraction operators and the IEN array. The control mesh defines the geometry and, through the extraction operators, the Bézier control mesh. The Bézier control mesh is the closest of the four to a standard finite element mesh in that the global system is built by looping over its elements. However, comparing this mesh to the C^0 finite element mesh in Figure 8b we see that, even though both meshes have the same number of global basis functions, there are more elements in the Bézier mesh. The extraction operator constrains the extra degrees of freedom of the Bézier mesh to maintain the smoothness of the NURBS. The control mesh represents the continuous basis and the global system that is being solved. Lastly, the Bézier physical mesh is comprised of the domains of integration.

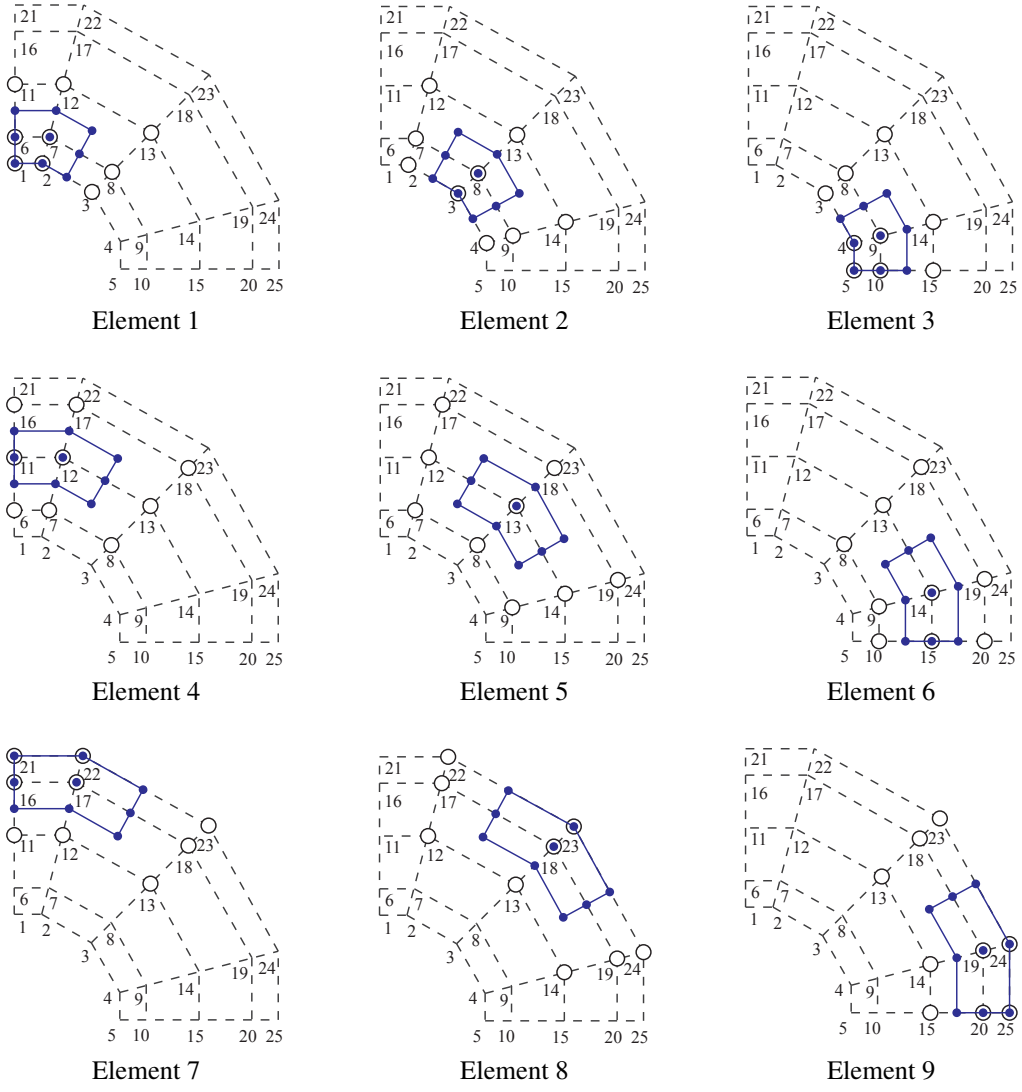


Figure 10: The extraction operators and IEN array can be used to construct the Bézier elements. For each element e the ◯'s indicate the global control points which influence the location of element control points, indicated by the •'s. Which global control points influence each element is determined by the IEN array, and the location of the element control points is computed with the element extraction operator $\mathbf{C}^e$.

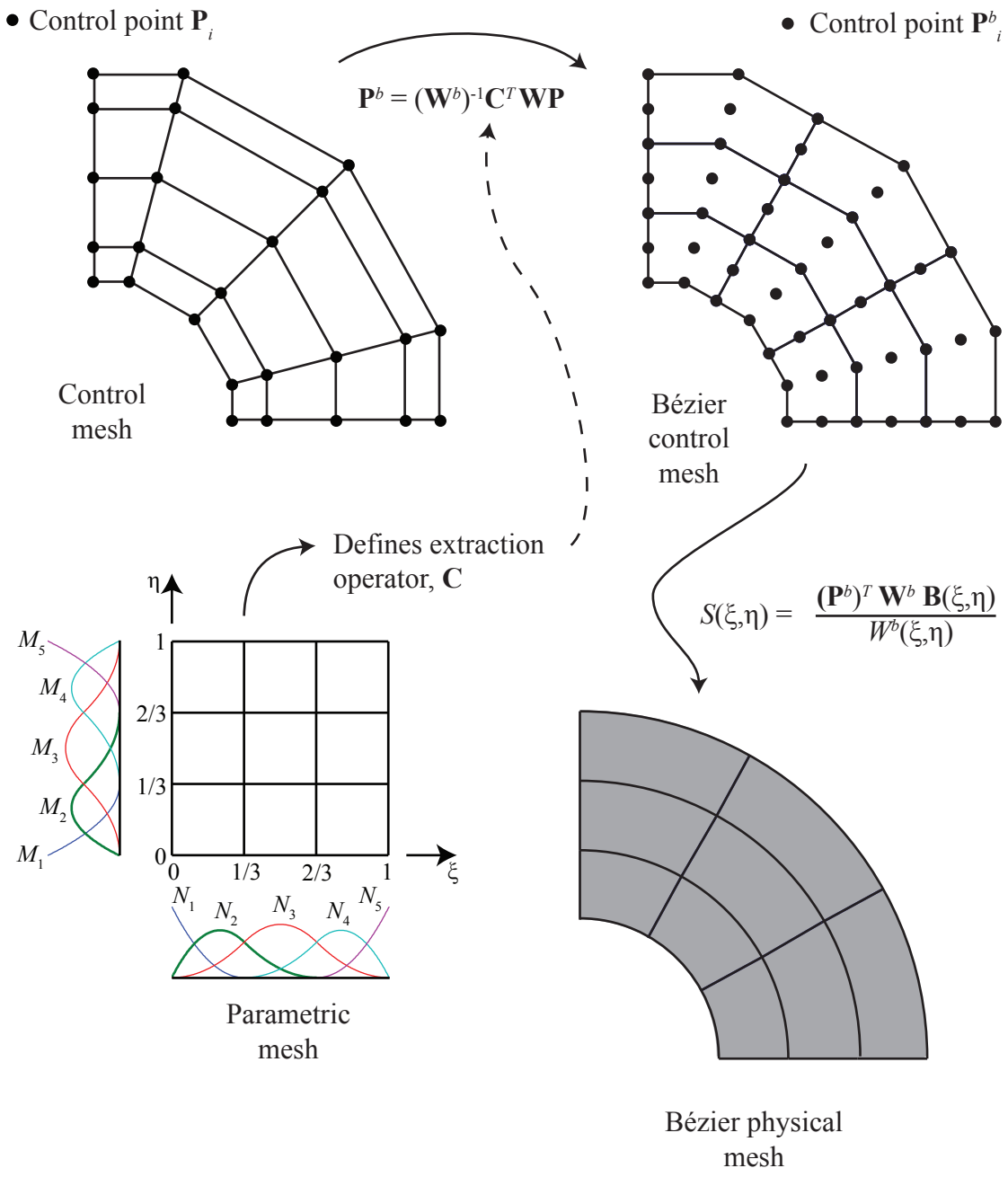


Figure 11: The "meshes" of the bivariate NURBS example. The parametric mesh is used to construct the extraction operator, $\mathbf{C}$, and the IEN array. The extraction operator and control mesh are used to define the Bézier control mesh. The assembly routines loop over the Bézier elements and perform integration over the element of the Bézier physical mesh.

6 Conclusions

We have introduced the Bézier extraction operator as a tool for integrating isogeometric analysis into existing finite element codes. For NURBS, we have shown how to compute the Bézier extraction operator by utilizing knot refinement. We then showed that the Bézier extraction operator provides an inherent element technology for computing with *any* basis that admits a Bézier representation. We noted that T-splines admit a Bézier representation, thus once the Bézier extraction operator has been incorporated into a code it will be possible to compute with T-splines without any modifications. In fact, we believe that the Bézier extraction operator provides the most natural approach to efficient computing with T-splines in finite element computer programs.

Beyond providing an element data structure, the Bézier extraction operator also provides a mechanism for localizing global basis information to an element. Benefits of this include representation of periodic boundary conditions and continuity between multiple NURBS patches without the need for additional data management arrays to maintain continuity. An important area for future research is investigating the use of Bézier elements and the Bézier extraction operator in mesh refinement. By providing a convenient element structure and localization mechanism, the Bézier extraction operator may accelerate the integration of isogeometric analysis into existing applications.

In future work we intend to generalize to T-splines eliminating the patch structure of NURBS. Multi-patch NURBS models have features in common with block-structured gridding utilized, for example, in finite differences, whereas T-splines with star points are unstructured.

Acknowledgement

M.J. Borden was supported by Sandia National Laboratories. Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000. J.A. Evans was partially supported by the Department of Energy Computational Science Graduate Fellowship, provided under grant number DE-FG02-97ER25308. T.J.R. Hughes and M.A. Scott were partially supported by the Office of Naval Research under Contract Number N00014-08-0992. T.J.R. Hughes was also partially supported by NSF grant 0700204, and M.A. Scott was also partially supported by the ICES CAM Graduate Fellowship.

A Element shape function subroutine

The element shape function subroutine is a fundamental component of any finite element code. For a given element, e, and a set of quadrature points in the parent element domain, the element shape function subroutine evaluates the local basis functions and any required derivatives at each quadrature point. The Jacobian determinate of the mapping from the parent domain to the physical domain must also be calculated in order to perform integration. In this appendix, we provide an example shape function subroutine for Bézier elements.

Recall from (3)-(5) that the Bernstein basis functions are defined over the interval $[0,1]$. To facilitate integration by quadrature we can redefine the basis over the interval $[-1,1]$ as

$$B_{a,p}(\xi) = \frac{1}{2}(1-\xi)B_{a,p-1}(\xi) + \frac{1}{2}(1+\xi)B_{a-1,p-1}(\xi) \tag{81}$$

where

$$B_{1,0}(\xi) \equiv 1 \tag{82}$$

and

$$B_{a,p}(\xi) \equiv 0 \quad \text{if } a < 1 \text{ or } a > p+1. \tag{83}$$

In this way, the map between the parametric domain and the parent element in Figure 5 becomes the identity map and the Jacobian determinant can be computed as

$$J = \left| \frac{\partial \mathbf{x}}{\partial \boldsymbol{\xi}} \right|. \tag{84}$$

We present two shape function routines: one for the general case where a full extraction operator has been computed for each element and one where the extraction operator is composed of univariate extraction operators. Each routine assumes the existence of several utility routines:

`Bernstein_basis_and_deriv` For a univariate Bernstein basis function of degree p, this functions will return two arrays containing the $p+1$ pre-computed basis function and derivative values at a give quadrature point.

`Bernstein_basis_and_derivs` For a trivariate Bernstein basis function of degree (p, q, r), this function returns an array containing the $(p+1) \times (q+1) \times (r+1)$ pre-computed basis function values and an array containing the $\big((p+1) \times (q+1) \times (r+1)\big) \times 3$ derivative values at a give quadrature point.

`inverse_Cramer` Uses Cramer's rule to compute the inverse of the matrix $\partial \mathbf{x}/\partial \boldsymbol{\xi}$.

`determinant` Computes J from $\partial \mathbf{x}/\partial \boldsymbol{\xi}$.

See [14] for details on the first two functions.

Algorithm 2 This Bézier element shape function routine is for the general case where an extraction operator has been computed for each element. The cost of generality in this case is an increase in the computational cost to compute the matrix products for (65) and (66).

Input The quadrature point, (ξ, η, ζ), the element number, e, the Bézier element control points, $\mathbf{Q}^e$, the corresponding weights, $\mathbf{W}^b$, stored as an array, the polynomial orders of the element, (p, q, r), the element extraction operator, $\mathbf{C}^e$, the IEN array, the weights for the smooth basis functions, $\mathbf{W}$, and the number of element shape functions, n_{en}.

Output An array of shape function values, R, an array of shape function derivative values, dR_dx, and the Jacobian determinate, J.

```
// Initialization:
ncpt = (p+1)×(q+1)×(r+1)

B(1:ncpt) = 0;
dB_dxi(1:ncpt, 1:3) = 0;

wb = 0;
dwb_dxi(1:3) = 0;

R(1:nen) = 0;
dR_dxi(1:nen, 1:3) = 0;
dR_dx(1:nen, 1:3) = 0;

dx_dxi(1:3, 1:3) = 0;
```

```
dxi_dx(1:3, 1:3) = 0;
J = 0;

// Get the pre-computed shape functions and derivatives for
// the parent domain.
call Bernstein_basis_and_derivs
    input: p, q, r, xi, eta, zeta
    output: B, dB_dxi

// Use the Bernstein basis to compute the weight functions.
for  a = 1 to ncpt  do
   wb = wb + B(a)×Wb(a);
   dwb_dxi(1) = dwb_dxi(1) + dB_dxi(a,1)×Wb(a);
   dwb_dxi(2) = dwb_dxi(2) + dB_dxi(a,2)×Wb(a);
   dwb_dxi(3) = dwb_dxi(3) + dB_dxi(a,3)×Wb(a);
end

// Use equation (65) and (66) to compute the element shape
// functions and derivatives w.r.t. the parent domain.
for  a = 1 to nen  do
   for  b = 1 to ncpt  do
      R(a) = R(a) + W(IEN(a,e))×C(e,a,b)×B(b)/wb;

      for  i = 1 to 3  do
         dR_dxi(a,i) = dR_dxi(a,i) ...
         ... + W(IEN(a,e))×C(e,a,b) ...
         ... ×(dB_dxi(b,i)/wb - dwb_dxi(i)×B(b)/(wb×wb));
      end
   end
end

// Compute the derivative of the mapping from the parent domain
// to the physical space.
for  a = 1 to ncpt  do
   for  i = 1 to 3  do
      for  j = 1 to 3  do
         dx_dxi(i,j) = dx_dxi(i,j) + Pb(a)×dB_dxi(a,j);
      end
   end
end

call inverse_Cramer
    input: dx_dxi
    output: dxi_dx

// Compute (67), the derivatives of the element shape
// functions w.r.t the physical coordinates.
for  a = 1 to nen  do
```

```
    for  i = 1 to 3  do
       for  j = 1 to 3  do
          dR_dx(a,i) = dR_dx(a,i) + dR_dxi(a,j)×dxi_dx(j,i);
       end
    end
end

call determinate
    input: dx_dxi
    output: J
```

Algorithm 3 This Bézier element shape function routine is specialized for tensor product element shape functions. This algorithm requires the univariate components of the element extraction operator. Performing the matrix products from (65) and (66) on the univariate B-spline components of the element shape functions decreases the cost of computation. Note also that the input for this algorithm requires the control points and weights associated with the smooth basis instead of those associated with the Bernstein basis of the Bézier element so that the control points and weights associated with the Bézier elements need not be computed.

Input: The quadrature point, (ξ, η, ζ), the control points for the smooth basis, $\mathbf{P}$, the control point weights for the smooth basis, $\mathbf{W}$, stored as an array, the element number, e, the univariate element extraction operators, $\mathbf{C}_i^e$, the IEN array, the polynomial orders of the element, (p, q, r), and the number of univariate element shape functions for each direction, n_{en}^i.

Output An array of shape function values, R, an array of shape function derivative values, dR_dx, and the Jacobian determinate, J.

```
// Initialization:
ncpt(1:3) = (p+1, q+1, r+1);

nentot = nen(1)+nen(2)+nen(3);

B(1, 1:p+1) = 0;
B(2, 1:q+1) = 0;
B(3, 1:r+1) = 0;

dB_dxi(1, 1:p+1) = 0;
dB_dxi(2, 1:q+1) = 0;
dB_dxi(3, 1:r+1) = 0;

N(1, 1:nen(1)) = 0;
N(2, 1:nen(2)) = 0;
N(3, 1:nen(3)) = 0;

dN_dxi(1, 1:nen(1)) = 0;
dN_dxi(2, 1:nen(1)) = 0;
dN_dxi(3, 1:nen(1)) = 0;
```

```
R(1:nentot) = 0;
dR_dxi(1:nentot, 1:3) = 0;
dR_dx(1:nentot, 1:3) = 0;

w = 0;
dw_dxi(1:3) = 0;

dx_dxi(1:3, 1:3) = 0;
dxi_dx(1:3, 1:3) = 0;
J = 0;

// Get the pre-computed univariate Bernstein basis functions and
// derivatives for the parent domain.
call Bernstein_basis_and_deriv
    input: p, xi
    output: B(1), dB_dxi(1)
call Bernstein_basis_and_deriv
    input: q, eta
    output: B(2), dB_dxi(2)
call Bernstein_basis_and_deriv
    input: r, zeta
    output: B(3), dB_dxi(3)

// Compute the univariate B-Spline functions and derivatives
// w.r.t. the parent domain.
for  i = 1 to 3  do
   for  a =1 to nen(i)  do
      for  b = 1 to ncpt(i)  do
         N(i,a) = N(i,a) + C(e,i,a,b)×B(i,b);

         dN_dxi(i,a) = dN_dxi(i,a) + C(e,i,a,b)×dB_dxi(i,b);
      end
   end
end

// Compute the numerators and denominator for the tensor product
// NURBS functions and derivatives.
a=0;
for  i = 1 to nen(3)  do
   for  j = 1 to nen(2)  do
      for  k = 1 to nen(1)  do
         a = a+1;
         R(a) = N(1,k)×N(2,j)×N(3,i)×W(IEN(a,e));
         w = w + R(a);

         dR_dxi(a,1) = dN_dxi(1,k)×N(2,j)×N(3,i)×W(IEN(a,e));
         dw_dxi(1) = dw_dxi(1) + dR_dxi(a,1);
         dR_dxi(a,2) = N(1,k)×dN_dxi(2,j)×N(3,i)×W(IEN(a,e));
```

```
        dw_dxi(2) = dw_dxi(2) + dR_dxi(a,2);
        dR_dxi(a,3) = N(1,k)×N(2,j)×dN_dxi(3,i)×W(IEN(a,e));
        dw_dxi(3) = dw_dxi(3) + dR_dxi(a,3);
      end
    end
end

// Divide by the denominators to complete the computation of the
// functions and derivatives w.r.t. the parent domain.
for  a = 1 to nentot  do
   R(a) = R(a) / w;

   for  i = 1 to 3  do
      dR_dxi(a,i) = (dR_dxi(a,i) - R(a)×dw_dxi(i)) / W;
   bf end
end

// Compute the derivative of the mapping from the parent domain
// to the physical space.
for  a = 1 to nen  do
   for  i = 1 to 3  do
      for  j = 1 to 3  do
         dx_dxi(i,j) = dx_dxi(i,j) + P(IEN(a,e))×dR_dxi(a,j);
      end
   end
end

call inverse_Cramer
    input: dx_dxi
    output: dxi_dx

// Compute (67), the derivatives of the element shape
// functions w.r.t the physical coordinates.
for  a = 1 to nen  do
   for  i = 1 to 3  do
      for  j = 1 to 3  do
         dR_dx(a,i) = dR_dx(a,i) + dR_dxi(a,j)×dxi_dx(j,i);
      end
   end
end

call determinate
    input: dx_dxi
    output: J
```

B Control points for the NURBS example in Section 5

Table 3 lists the control point coordinates for the NURBS geometry in Section 5. Once the element extraction operators have been computed for this geometry, we can use (80) to compute the Bézier element control points. The Bézier element control points are listed in Table 5.

Control point	x	y	w
1	0.0	1.0	1.0
2	0.2612	1.0	0.9024
3	0.7346	0.7346	0.8373
4	1.0	0.2612	0.9024
5	1.0	0.0	1.0
6	0.0	1.25	1.0
7	0.3265	1.25	0.9024
8	0.9182	0.9182	0.8373
9	1.25	0.3265	0.9024
10	1.25	0.0	1.0
11	0.0	1.75	1.0
12	0.4571	1.75	0.9024
13	1.2856	1.2856	0.8373
14	1.75	0.4571	0.9024
15	1.75	0.0	1.0
16	0.0	2.25	1.0
17	0.5877	2.25	0.9024
18	1.6528	1.6528	0.8373
19	2.25	0.5877	0.9024
20	2.25	0.0	1.0
21	0.0	2.5	1.0
22	0.6530	2.5	0.9024
23	1.8365	1.8365	0.8373
24	2.5	0.6530	0.9024
25	2.5	0.0	1.0

Table 3: The control point ($\mathbf{P}$) coordinates (x, y) and weights (w) for the control mesh in Figure 7b.

a	x	y	w
1	0.0	1.0	1.0
2	0.2612	1.0	0.9024
3	0.7346	0.7346	0.8373
4	0.0	1.25	1.0
5	0.3265	1.25	0.9024
6	0.9182	0.9182	0.8373
7	0.0	1.75	1.0
8	0.4571	1.75	0.9024
9	1.2856	1.2856	0.8373

$e = 1$

a	x	y	w
1	0.2612	1.0	0.9024
2	0.7346	0.7346	0.8373
3	1.0	0.2612	0.9024
4	0.3265	1.25	0.9024
5	0.9182	0.9182	0.8373
6	1.25	0.3265	0.9024
7	0.4571	1.75	0.9024
8	1.2856	1.2856	0.8373
9	1.75	0.4571	0.9024

$e = 2$

a	x	y	w
1	0.7346	0.7346	0.8373
2	1.0	0.2612	0.9024
3	1.0	0.0	1.0
4	0.9182	0.9182	0.8373
5	1.25	0.3265	0.9024
6	1.25	0.0	1.0
7	1.2856	1.2856	0.8373
8	1.5	0.3918	0.9024
9	1.5	0.0	1.0

$e = 3$

a	x	y	w
1	0.0	1.25	1.0
2	0.3265	1.25	0.9024
3	0.9182	0.9182	0.8373
4	0.0	1.75	1.0
5	0.4571	1.75	0.9024
6	1.2856	1.2856	0.8373
7	0.0	2.25	1.0
8	0.5877	2.25	0.9024
9	1.6528	1.6528	0.8373

$e = 4$

a	x	y	w
1	0.3265	1.25	0.9024
2	0.9182	0.9182	0.8373
3	1.25	0.3265	0.9024
4	0.4571	1.75	0.9024
5	1.2856	1.2856	0.8373
6	1.75	0.4571	0.9024
7	0.5877	2.25	0.9024
8	1.6528	1.6528	0.8373
9	2.25	0.58771	0.9024

$e = 5$

a	x	y	w
1	0.9182	0.9182	0.8373
2	1.25	0.3265	0.9024
3	1.25	0.0	1.0
4	1.2856	1.2856	0.8373
5	1.75	0.4571	0.9024
6	1.75	0.0	1.0
7	1.6528	1.6528	0.8373
8	2.25	0.5877	0.9024
9	2.25	0.0	1.0

$e = 6$

a	x	y	w
1	0.0	1.75	1.0
2	0.4571	1.75	0.9024
3	1.2856	1.2856	0.8373
4	0.0	2.25	1.0
5	0.5877	2.25	0.9024
6	1.6528	1.6528	0.8373
7	0.0	2.5	1.0
8	0.6530	2.5	0.9024
9	1.8365	1.8365	0.8373

$e = 7$

a	x	y	w
1	0.4571	1.75	0.9024
2	1.2856	1.2856	0.8373
3	1.74	0.4571	0.9024
4	0.5877	2.25	0.9024
5	1.6528	1.6528	0.8373
6	2.25	0.5877	0.9024
7	0.6530	2.5	0.9024
8	1.8365	1.8365	0.8373
9	2.5	0.6530	0.9024

$e = 8$

a	x	y	w
1	1.2856	1.2856	0.8373
2	1.75	0.4571	0.9024
3	1.75	0.0	1.0
4	1.6528	1.6528	0.8373
5	2.25	0.5877	0.9024
6	2.25	0.0	1.0
7	1.8365	1.8365	0.8373
8	2.5	0.6530	0.9024
9	2.5	0.0	1.0

$e = 9$

Table 4: The local NURBS control points ($\mathbf{P}^e$) and weights (w) for the elements shown in Figure 10.

a	x	y	w
1	0.0	1.0	1.0
2	0.2612	1.0	0.9024
3	0.4890	0.8723	0.8698
4	0.0	1.25	1.0
5	0.3265	1.25	0.9024
6	0.6113	1.0903	0.8698
7	0.0	1.5	1.0
8	0.3918	1.5	0.9024
9	0.7336	1.3084	0.8698

$e = 1$

a	x	y	w
1	0.4890	0.8723	0.8698
2	0.7346	0.7346	0.8373
3	0.8723	0.4890	0.8698
4	0.6113	1.0903	0.8698
5	0.9182	0.9182	0.8373
6	1.0903	0.6113	0.8698
7	0.7336	1.3084	0.8698
8	1.1019	1.1019	0.8373
9	1.3084	0.7336	0.8698

$e = 2$

a	x	y	w
1	0.8723	0.4890	0.8698
2	1.0	0.2612	0.9024
3	1.0	0.0	1.0
4	1.0903	0.6113	0.8698
5	1.25	0.3265	0.9024
6	1.25	0.0	1.0
7	1.3084	0.7336	0.8698
8	1.5	0.3918	0.9024
9	1.5	0.0	1.0

$e = 3$

a	x	y	w
1	0.0	1.5	1.0
2	0.3918	1.5	0.9024
3	0.7336	1.3084	0.8698
4	0.0	1.75	1.0
5	0.4571	1.75	0.9024
6	0.8558	1.5265	0.8698
7	0.0	2.0	1.0
8	0.5224	2.0	0.9024
9	0.9781	1.7445	0.8698

$e = 4$

a	x	y	w
1	0.7336	1.3084	0.8698
2	1.1019	1.1019	0.8373
3	1.3084	0.7336	0.8698
4	0.8558	1.5265	0.8698
5	1.2855	1.2855	0.8373
6	1.5265	0.8558	0.8698
7	0.9781	1.7445	0.8698
8	1.4692	1.4692	0.8373
9	1.7445	0.9781	0.8698

$e = 5$

a	x	y	w
1	1.3084	0.7336	0.8698
2	1.5	0.3918	0.9024
3	1.5	0.0	1.0
4	1.5265	0.8558	0.8698
5	1.75	0.4571	0.9024
6	1.75	0.0	1.0
7	1.7445	0.9781	0.8698
8	2.0	0.5224	0.9024
9	2.0	0.0	1.0

$e = 6$

a	x	y	w
1	0.0	2.0	1.0
2	0.5224	2.0	0.9024
3	0.9781	1.7445	0.8698
4	0.0	2.25	1.0
5	0.5877	2.25	0.9024
6	1.1003	1.9626	0.8698
7	0.0	2.5	1.0
8	0.6530	2.5	0.9024
9	1.2226	2.1807	0.8698

$e = 7$

a	x	y	w
1	0.9781	1.7445	0.8698
2	1.4692	1.4692	0.8373
3	1.7445	0.9781	0.8698
4	1.1003	1.9626	0.8698
5	1.6528	1.6528	0.8373
6	1.9626	1.1003	0.8698
7	1.2226	2.1807	0.8698
8	1.8365	1.8365	0.8373
9	2.1807	1.2226	0.8698

$e = 8$

a	x	y	w
1	1.7445	0.9781	0.8698
2	2.0	0.5224	0.9024
3	2.0	0.0	1.0
4	1.9626	1.1003	0.8698
5	2.25	0.5877	0.9024
6	2.25	0.0	1.0
7	2.1807	1.2226	0.8698
8	2.5	0.6530	0.9024
9	2.5	0.0	1.0

$e = 9$

Table 5: Bézier element control points ($\mathbf{Q}^e$) and weights (w) for the elements shown in Figure 10.

References

[1] I. Akkerman, Y. Bazilevs, V.M. Calo, T.J.R. Hughes, and S. Hulshoff. The role of continuity in residual-based variational multiscale modeling of turbulence. *Computational Mechanics*, 41:371–378, 2008.

[2] F. Auricchio, L. B. da Veiga, C. Lovadina, and A. Reali. The importance of the exact satisfaction of the incompressibility constraint in nonlinear elasticity: mixed fems versus NURBS-based approximations. *Computer Methods in Applied Mechanics and Engineering*, 199:314–323, 2010.

[3] F. Auricchio, L.B. da Veiga, A. Buffa, C. Lovadina, A. Reali, and G. Sangalli. A fully "locking-free" isogeometric approach for plane linear elasticity problems: A stream function formulation. *Computer Methods in Applied Mechanics and Engineering*, 197:160–172, 2007.

[4] Y. Bazilevs, V.M. Calo, J.A. Cottrell, T.J.R. Hughes, A. Reali, and G. Scovazzi. Variational multiscale residual-based turbulence modeling for large eddy simulation of incompressible flows. *Computer Methods in Applied Mechanics and Engineering*, 197:173–201, 2007.

[5] J.A. Cottrell, T.J.R Hughes, and Y. Bazilevs. *Isogeometric analysis: Toward integration of CAD and FEA*. Wiley, 2009.

[6] J.A. Cottrell, T.J.R. Hughes, and A. Reali. Studies of refinement and continuity in isogeometric structural analysis. *Computer Methods in Applied Mechanics and Engineering*, 196:4160–4183, 2007.

[7] J.A. Cottrell, A. Reali, Y. Bazilevs, and T.J.R. Hughes. Isogeometric analysis of structural vibrations. *Computer Methods in Applied Mechanics and Engineering*, 195:5257–5296, 2006.

[8] T. Elguedj, Y. Bazilevs, V.M. Calo, and T.J.R. Hughes. $\bar{B}$ and $\bar{F}$ projection methods for nearly incompressible linear and non-linear elasticity and plasticity using higher-order NURBS elements. *Computer Methods in Applied Mechanics and Engineering*, 197:2732–2762, 2008.

[9] H. Gomez, V.M. Calo, Y. Bazilevs, and T.J.R. Hughes. Isogeometric analysis of the Cahn-Hilliard phase-field model. *Computer Methods in Applied Mechanics and Engineering*, 197:4333–4352, 2008.

[10] T.J.R. Hughes. *The Finite Element Method: Linear Static and Dynamic Finite Element Analysis*. Dover Publications, Mineola, NY, 2000.

[11] T.J.R. Hughes, J.A. Cottrell, and Y. Bazilevs. Isogeometric analysis: CAD, finite elements, NURBS, exact geometry, and mesh refinement. *Computer Methods in Applied Mechanics and Engineering*, 194:4135–4195, 2005.

[12] T.J.R Hughes, A Reali, and G. Sangalli. Efficient quadrature for NURBS-based isogeometric analysis. *Computer Methods in Applied Mechanics and Engineering*, 199:301–313, 2010.

[13] S. Lipton, J.A. Evans, Y. Bazilevs, T. Elguedj, and T.J.R. Hughes. Robustness of isogeometric structural discretizations under severe mesh distortion. *Computer Methods in Applied Mechanics and Engineering*, 199:357–373, 2010.

[14] L. Piegl and W. Tiller. *The NURBS Book (Monographs in Visual Communication), 2nd ed.* Springer-Verlag, New York, 1997.

[15] D. F. Rogers. *An Introduction to NURBS With Historical Perspective*. Academic Press, San Diego, CA, 2001.

[16] T.W. Sederberg, J. Zheng, A. Bakenov, and A. Nasri. T-splines and T-NURCCSs. *ACM Transactions on Graphics*, 22 (3):477–484, 2003.

[17] W.A. Wall, M.A. Frenzel, and C. Cyron. Isogeometric structural shape optimization. *Computer Methods in Applied Mechanics and Engineering*, 197:2976–2988, 2008.