NFN - Nationales Forschungsnetzwerk

Geometry + Simulation

http://www.gs.jku.at

A Versatile Strategy for the Implementation of Adaptive Splines

Andrea Bressan and Dominik Mokriš

G+S Report No. 50

November 2016

Der Wissenschaftsfonds.

JOHANNES KEPLER UNIVERSITY LINZ

A Versatile Strategy for the Implementation of Adaptive Splines

Andrea Bressan[1] and Dominik Mokriš[2]

[1] Department of Mathematics, University of Oslo, Norway
andbres@math.uio.no
[2] Institute for Applied Geometry, Johannes Kepler University Linz, Austria
dominik.mokris@jku.at

Abstract. This paper presents an implementation framework for spline spaces over T-meshes (and their d-dimensional analogs). The aim is to share code between the implementation of several spline spaces. This is achieved by reducing evaluation to a generalized Bézier extraction.
The approach was tested by implementing hierarchical B-splines, truncated hierarchical B-splines, decoupled hierarchical B-splines (a novel variation presented here), truncated B-splines for partially nested refinement and hierarchical LR-splines.

Keywords: Implementation, Bézier Extraction, THB-splines, LR-splines.

1 Introduction

A common method to represent shapes in Computer-Aided Design (CAD), Computer-Aided Engineering (CAE) and Computer-Aided Manufacturing (CAM) is to parametrize the desired geometry (or its boundary) with Non-Uniform Rational B-Splines (NURBS). B-splines have a global tensor-product structure, where each d-variate basis function is a product of d univariate basis functions. This means that changes in spatial resolution cannot be confined to a small region; they necessarily spread to a union of stripes of the domain (Fig. 1).

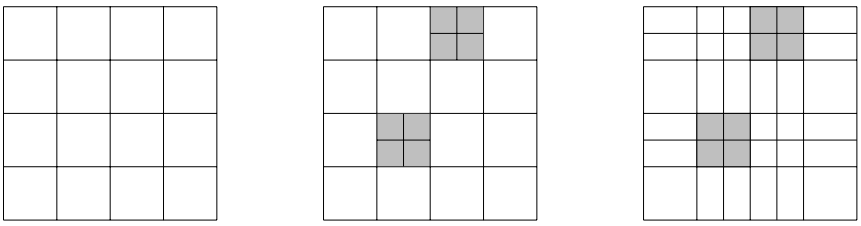


Fig. 1: Limit of the tensor construction. Left: the coarse grid; Middle: the desired refinement; Right: the coarsest tensor grid refined on the gray area.

Different constructions that allow for local refinement were proposed during the last two decades and gained support with the introduction of IsoGeometric Analysis (IGA) [22]. Indeed, IGA pushed the use of splines in numerical simulation where local refinement is a prerequisite of adaptive methods. The following list includes the best known constructions:

- Hierarchical B-splines (HB), introduced in [17]. This is a multiscale approach: each scale is associated to a different tensor-product B-spline space. Functions from each scale are *selected* depending on the locally required resolution and together they form the hierarchical B-spline basis. There are many variations of HB, among them: the Truncated Hierarchical B-splines (THB) [18], the Truncated Decoupled Hierarchical B-splines (TDHB) [29], the Truncated B-splines for partially nested refinement (TBPN) [39] and Decoupled Hierarchical B-splines (DHB) introduced here for the first time.
- T-splines (T), introduced in [35,34]. The central notion is the T-mesh: a planar graph with lengths. A B-spline corresponds to each vertex of the graph and its knot vectors depend on the length of the neighboring edges. These B-splines generate the space. Unfortunately they can be linearly dependent. *Analysis Suitable T-splines* (AST) avoid linear dependencies by restricting the class of allowed T-meshes [13]. AST spaces can be constructed in 2D [32] and also defined for 3D domains [31].
- Locally Refined splines (LR) were introduced in [15]. Their definition is given in terms of *minimally supported* B-splines contained in a space of piecewise polynomials. The generators are not always linearly independent. A bivariate construction that avoids linear dependencies are the hierarchical LR-splines (HLR) [7].

Several other spaces exist, among them [14,10,25,8]. On one hand, the mentioned spaces contain piecewise polynomials over box-shaped subdomains and allow for smooth functions. On the other hand, each construction was defined for a specific application and, as a consequence, described and analyzed with its own set of tools. Thus it is difficult to make a comparison involving more than a few spaces and having criteria that are not application-specific. A comparison of HB, THB and LR based on the conditioning of the mass matrix is presented in [23].

Our aim is to provide the description of a software framework that allows to implement different spline spaces efficiently. In this way we hope to facilitate both the comparison of different spline spaces and experimenting with alternative definitions.

The framework is presented in Section 2 without any reference to specific spline spaces. Section 3 discusses the space and time complexity of the proposed approach and presents possible optimizations. Section 4 describes how the framework can be applied to HB, THB, DHB, TBPN and HLR splines. These spaces were implemented. Their implementations are used in Section 5 to show how the different spaces behave in a few selected cases.

2 Implementation Method

The aim is to evaluate the generators of a spline space at arbitrary points of the domain $\Omega \subseteq \mathbb{R}^d$. Let $\Gamma = \{\gamma_1, \dots, \gamma_n\}$ be the generating set. We focus on the method `eval` that given a point $\mathbf{x} \in \Omega$ returns the row vector $\Gamma(\mathbf{x}) = \big(\gamma_1(\mathbf{x}), \dots, \gamma_n(\mathbf{x})\big)$. Note that the described interface of `eval` allows for the solution of interpolation problems and for the implementation of Galerkin methods based on numerical quadrature.

The spline spaces of interest have generating sets that are piecewise polynomials on a partition of Ω into axis-aligned boxes: the *elements*. This allows to represent their restriction on an element in the Bernstein basis of tensor-product polynomials. By doing so it is possible to repurpose Finite Element Method (FEM) codebases to IGA. This approach was proposed for NURBS in [4] under the name of Bézier extraction and extended to other spaces in [33,16].

The main idea of this paper is that if the above strategy is abstracted by replacing elements with more general subdomains and the Bernstein basis with an arbitrary local basis then the method is more versatile and it allows for the implementation of more spaces with less code.

2.1 Description

Assume that there exists a partition $\mathbf{D} = \{\mathcal{D}_1, \dots, \mathcal{D}_L\}$ of the domain Ω and a corresponding sets of local generators $\mathbf{B} = \{\mathscr{B}_1, \dots, \mathscr{B}_L\}$ such that the restriction of each $\gamma \in \Gamma$ to any $\mathcal{D}_i$ admits a representation in span $\mathscr{B}_i$. More precisely,

$$\forall \gamma \in \Gamma,\ \forall_{i=1}^{L},\ \forall \mathbf{x} \in \mathcal{D}_i : \quad \gamma(\mathbf{x}) = \sum_{\beta \in \mathscr{B}_i} m_{\beta,\gamma} \beta(\mathbf{x}) \ , \tag{1}$$

and thus

$$\Gamma(\mathbf{x}) = \mathscr{B}_i(\mathbf{x}) \mathbf{M}_i \ , \tag{2}$$

where $\mathscr{B}_i(\mathbf{x}) = \big(\beta(\mathbf{x})\big)_{\beta \in \mathscr{B}_i}$ is a row vector and $\mathbf{M}_i = \big(m_{\beta,\gamma}\big)_{\beta \in \mathscr{B}_i,\, \gamma \in \Gamma}$ is the matrix containing the coefficients from (1). The matrices $\mathbf{M}_i$ can be collected as blocks of the bigger matrix $\mathbf{M}$ as depicted in Fig. 2.

Provided a triplet of $\mathbf{D}$, $\mathbf{B}$ and $\mathbf{M}$, the evaluation of $\Gamma(\mathbf{x})$ can be performed using (2). By writing `eval` in terms of $\mathbf{D}$, $\mathbf{B}$ and $\mathbf{M}$, the space-specific code is reduced to the initialization of $\mathbf{D}$, $\mathbf{B}$ and $\mathbf{M}$. Note that Γ is uniquely determined by $\mathbf{D}$, $\mathbf{B}$ and $\mathbf{M}$, but different choices of $\mathbf{D}$, $\mathbf{B}$ and $\mathbf{M}$ are possible for the same Γ. This allows for different trade-offs as seen in Section 4.

The following simplified evaluation method suggests the interfaces and implementations of $\mathbf{D}$, $\mathbf{B}$ and $\mathbf{M}$.

Procedure: `evalSimple(x)`
  Input: point $\mathbf{x} \in \Omega$
  Output: $\Gamma(\mathbf{x}) = \big(\gamma_1(\mathbf{x}), \dots, \gamma_n(\mathbf{x})\big)$
  $i = \mathbf{D}$.`findSubdomain`$(\mathbf{x})$ `/* finds` i`:` $\mathbf{x} \in \mathcal{D}_i$ `*/`
  $\mathscr{B}_i(\mathbf{x}) = \mathscr{B}_i$.`eval`$(\mathbf{x})$
  $\Gamma(\mathbf{x}) = \mathscr{B}_i(\mathbf{x}) \mathbf{M}_i$

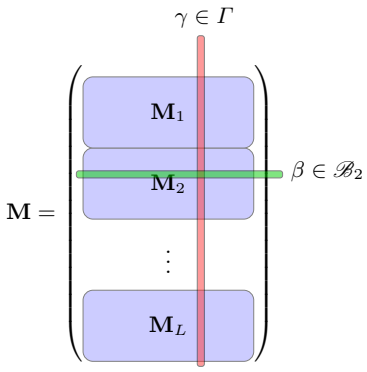


Fig. 2: Structure of the representation matrix. It has $\sum_{i=0}^{L} \#\mathscr{B}_i$ rows and $\#G$ columns.

Now we proceed by describing the implementations of **D**, **B** and **M**.

The partition **D** requires only the `findSubdomain` method. It can be efficiently implemented using a binary decision tree (more precisely a binary space partition, cf. [36,37]). For the spaces of interest it is possible to assume that the $\mathcal{D}_i$ are polytopes with axis-aligned faces. With this simplification, each fork in the tree corresponds to a spatial split along an axis-aligned affine hyperspace, i.e., to a comparison for a specific coordinate; each branch to taking the intersection with one of the corresponding half-spaces. Every leaf of the tree corresponds to the intersection of the taken half-spaces with Ω. Thus **D** can be represented by storing in each leaf the index of the subdomain containing the corresponding box. Fig. 3 depicts a partition and the corresponding tree.

It is worth noting that binary partition trees can be used also to implement refinement strategies. Indeed they can be seen as maps $\Omega \to \mathbb{N}$ and they allow for efficient implementation of binary operations (see the references above for union and the intersections). By constructing a tree that assigns to each point its refinement level it is easy to compute the coarsest common refinement of two meshes as it corresponds to pointwise-max operation. Similarly, the finest common submesh can be computed using a pointwise-min operation.

The collection of local generating sets **B** is simply a list of polymorphic objects implementing the `eval` interface. This allows for arbitrary local bases and thus, for example, Bernstein polynomials as in Bézier extraction, or B-splines as in all of our implementations, or enriched spaces of polynomials as in generalized B-splines [5].

Finally, **M** is a sparse matrix. However, the initialization of the matrix for a particular spline space usually requires most of the space-specific code.

The `eval` method extends the `evalSimple` method by computing simultaneously values and derivatives on more points contained in the same subdomain

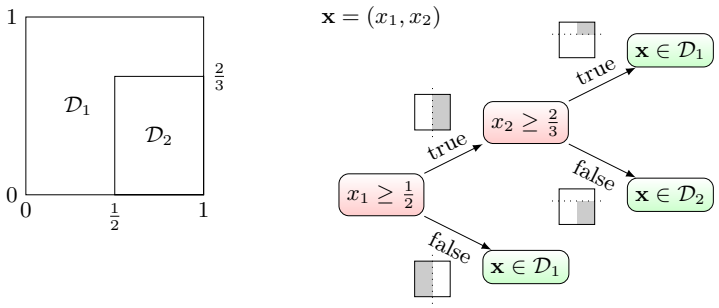


Fig. 3: A partition of Ω in $\mathcal{D}_1$ and $\mathcal{D}_2$ and a decision tree describing it. The darkened area in the domains depicted next to each branch highlight the region on which the branch is taken.

$\mathcal{D}_i$. Let $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_r\} \subset \mathcal{D}_i$ be the set of points. Redefine $\mathscr{B}_i(X)$ and $\Gamma(X)$ to be matrices whose columns correspond to a generating function (as before) and whose rows correspond to the requested data, e.g., to the combinations of requested derivatives and points:

$$\Gamma(X) = \begin{pmatrix} \gamma_1(\mathbf{x}_1) & \ldots & \gamma_n(\mathbf{x}_1) \\ \partial_1\gamma_1(\mathbf{x}_1) & \ldots & \partial_1\gamma_n(\mathbf{x}_1) \\ \vdots & & \vdots \\ \partial_d\gamma_1(\mathbf{x}_1) & \ldots & \partial_d\gamma_n(\mathbf{x}_1) \\ \gamma_1(\mathbf{x}_2) & \ldots & \gamma_n(\mathbf{x}_2) \\ \vdots & & \vdots \\ \partial_d\gamma_1(\mathbf{x}_r) & \ldots & \partial_d\gamma_n(\mathbf{x}_r) \end{pmatrix} . \tag{3}$$

The following is the `eval` procedure.

Procedure: `eval(`X`)`
Input: the set of points $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_r\} \in \Omega$
Assumptions: $\exists i : X \subset \mathcal{D}_i$
Output: $\Gamma(X)$
$i = \mathbf{D}$.`findSubdomain`$(\mathbf{x}_1)$
$\mathscr{B}_i(X) = \mathscr{B}_i$.`eval`$(X)$
$\Gamma(X) = \mathscr{B}_i(X)\mathbf{M}_i$

As mentioned, Bézier extraction is a special case of the proposed implementation. It is equivalent to our framework with the following choices: $\mathbf{D}$ is the partition of the domain into elements; $\mathscr{B}_i$ is the Bernstein basis remapped to the element $\mathcal{D}_i$ and $\mathbf{M}_i$ contains the expansion of the polynomial expression of the functions $\gamma \in \Gamma$ on the element. Consequently, the implementation of T-splines with Bézier extraction is feasible in this framework.

2.2 Subspaces and Functions

Consider a subspace of $\operatorname{span} \Gamma$ generated by $\Gamma' = \{\gamma'_1, \dots, \gamma'_k\}$. Then Γ' can be implemented by $\mathbf{D}, \mathbf{B}, \mathbf{M}'$ with

$$\mathbf{M}' = \mathbf{M}\mathbf{N}$$

where $\mathbf{N} = (n_{\gamma,\gamma'})_{\gamma \in \Gamma, \gamma' \in \Gamma'}$ contains in the i-th column the expansion of γ'_i in $\gamma_1, \dots, \gamma_n$, i.e., $\forall \mathbf{x} \in \Omega$, $\Gamma'(\mathbf{x}) = \Gamma(\mathbf{x})\mathbf{N}$.

As a consequence, `eval` is not only a suitable implementation of Γ, but also of the functions $f \in \operatorname{span} \Gamma$. Indeed they corresponds to $\mathbf{M}'$ having a single column and $\mathbf{N}$ being the column vector of the coefficients of f.

This method can also be used to construct a space satisfying Robin type boundary conditions or, as described in the next subsection, to construct multipatch spaces with the prescribed smoothness.

2.3 Multipatch Domains

The proposed framework can be extended in order to allow for multipatch domains. This can be achieved by adding an optional parameter, the patch index, to both `eval` and of `findSubdomain`. The `eval` procedure is then modified as follows.

Procedure: `eval(`X`,`p`)`
Input: the set of points $X = \{\mathbf{x}_1, \dots, \mathbf{x}_r\} \in \Omega$
Input (optional): the patch index p, default to 0
Assumptions: $\exists i : X \subset \mathcal{D}_i$
Output: $\Gamma(X)$
$i = \mathbf{D}.\texttt{findSubdomain}(\mathbf{x}_1, p)$
$\mathscr{B}_i(X) = \mathscr{B}_i.\texttt{eval}(X)$
$\Gamma(X) = \mathscr{B}_i(X)\mathbf{M}_i$

Combined with the ability to restrict to arbitrary subspaces this opens the way to the implementation of geometrically C^k functions on multipatch domains. A multipatch domain is the union of a finite number images of axis-aligned boxes by a geometry map. The adjective "geometrically" specifies that the smoothness is meant with respect of the multipatch domain and not with respect of the domains of the parametrizations.

This is an active research topic in IGA [26,9,12]. On one hand, higher smoothness allows for an easier treatment of high order equations and provides better approximation of the spectrum. On the other hand, enforcing smoothness can lead to locking phenomena and impair the approximation properties.

Provided that smoothness can be enforced, the proposed framework allows for an easy representation of the space of geometrically C^k functions starting from that of discontinuous functions. Indeed discontinuous functions on a multipatch domain correspond to a block diagonal representation matrix $\mathbf{M}$ with blocks corresponding to functions defined on each patch. The space of geometrically C^k functions is represented with $\mathbf{M}' = \mathbf{M}\mathbf{N}$ for a suitable $\mathbf{N}$. This is the strategy

used in [9] where, due to a different implementation of the patch spaces, the multiplication by $\mathbf{N}$ is done at a post-processing stage and thus incurs in an additional cost.

3 Complexity

Delegating the evaluation to a local basis and computing the linear combination incurs in an additional computational cost. Moreover, storing the coefficients of $\mathbf{M}$ can require a substantial amount of memory.

3.1 Space complexity

Most of the memory required is used by $\mathbf{M}$. The tested implementation uses a row-compressed format: only the nonzero coefficients are stored in lexicographic order of their indices. The column position of the nonzero entries is stored in a second vector. The row position is deduced by storing a pointer to the first nonzero of each row. This means that the total required memory is proportional to the sum of the number of rows plus the number of nonzero entries. The number of rows of $\mathbf{M}$ equals

$$\sum_{i=1}^{L} \#\mathscr{B}_i \ .$$

Consequently there is a memory cost associated to functions of the local bases even if they are not used in any $\mathcal{D}_i$ to represent Γ.

The number of nonzero coefficients in $\mathbf{M}$ depends on the complexity of the mesh and on the shape of the generators. The number of nonzero coefficients in the column corresponding to $\gamma \in \Gamma$ is

$$\sum_{i:\gamma(\mathcal{D}_i)\neq\{0\}} \#\{\beta : m_{\beta,\gamma} \neq 0\} \ .$$

Thus it is minimized if γ is supported in a single $\mathcal{D}_i$ and if it equals a local basis $b \in \mathscr{B}_i$. In contrast, the generators γ whose supports intersect many subdomains or whose shape requires many coefficients to be represented in a domain require more memory.

3.2 Time complexity

The time cost of the `eval` procedure can be described as

$$\mathfrak{C}(\texttt{eval}) = \mathfrak{C}(\texttt{findSubdomain}) + \mathfrak{C}(\mathscr{B}_i.\texttt{eval}) + \mathfrak{C}(matrix) \ , \tag{4}$$

where $\mathfrak{C}(matrix)$ is the cost of computing the matrix-matrix product $\mathscr{B}_i(X)\mathbf{M}_i$. Now we analyze each of the terms.

The cost of `findSubdomain` depends on the tree structure and on the complexity of the mesh. For a balanced tree this would be proportional to $\log_2 \ell$,

where ℓ is the number of leaves in $\mathbf{D}$. However, a balanced tree is not necessarily optimal, as the tree should take the usage pattern into account. For instance, if we assume a uniform sampling of the domain then the optimal tree will have leaves of depth inversely proportional to the measure of the corresponding region. Our test implementation tries to reduce the size of the tree by avoiding unnecessary splits. With this simple optimization $\mathfrak{C}(\texttt{findSubdomain})$ was insignificant in our profiling tests. Moreover, in Galerkin applications the elements can be iterated according to the containing subdomains, avoiding the tree traversal.

The complexity of $\mathscr{B}_i.\texttt{eval}$ depends on the specific local basis used. If the local basis has optimal complexity with respect of the output size, then it is proportional to $w\#X\#\mathscr{B}_i$, where X is the set of points and w is the amount of data per point and function to evaluate.

The cost $\mathfrak{C}(matrix)$ of the matrix-matrix product $\mathscr{B}_i(X)\mathbf{M}_i$ is the dominating cost of the evaluation. Since the dimensions of $\mathscr{B}_i(X)$ and $\mathbf{M}_i$ are $(w\#X, \#\mathscr{B}_i)$ and $(\#\mathscr{B}_i, \#\Gamma)$ respectively, the complexity is

$$\mathfrak{C}(matrix) \cong w\#X\#\mathscr{B}_i\#\Gamma \ .$$

Comparing this to the output size $w\#X\#\Gamma$ shows that the method is rather expensive if $\#\mathscr{B}_i$ is big. The next sections show how this cost can be reduced.

3.3 Local Basis and Compression

If the functions in Γ and $\mathscr{B}_i$ have small supports, then the number of nonzero columns in $\mathscr{B}_i(X)$ and in $\Gamma(X)$ is small compared to $\#\mathscr{B}_i$ and $\#\Gamma$, respectively. This suggests the use of a compressed format for $\mathscr{B}_i(X)$ and $\Gamma(X)$, where only the nonzero values and their positions are stored. This is standard in FEM and other numerical methods and also the default in our implementation. Let a be the list of the indices of the nonzero columns of $\mathscr{B}_i(X)$. Then $\mathscr{B}_i(X)$ is implemented by the pair $(a, \widetilde{\mathscr{B}}_i)$, where $\widetilde{\mathscr{B}}_i$ is the matrix containing the columns of $\mathscr{B}_i(X)$ with the indices in a. For instance if $\mathscr{B}_i(X)$ has 5 columns and $a = (3, 5)$ then $\widetilde{\mathscr{B}}_i$ is a 2 column matrix containing the third and the fifth column of $\mathscr{B}_i(X)$. The set A of the nonzero columns of $\Gamma(X)$ and their content $\widetilde{\Gamma}$ is obtained by a sparse matrix-matrix product. A function $\gamma \in \Gamma$ (or $\beta \in \mathscr{B}_i$) is called *active* on X if the corresponding column in $\Gamma(X)$ (or $\mathscr{B}_i(X)$) is nonzero.

By using the compressed format $\mathfrak{C}(\mathscr{B}_i.\texttt{eval})$ is reduced to

$$\mathfrak{C}(\mathscr{B}_i.\texttt{eval}) \cong w\#X\#a$$

and $\mathfrak{C}(matrix)$ decreases to

$$\mathfrak{C}(matrix) \cong w\#X\#a\#A \ . \tag{5}$$

Comparing this with the output size $w\#X\#A$ shows that the algorithm has optimal complexity provided there is an upper bound on $\#a$. For instance, if the local bases are Bernstein polynomials or polynomial splines of degree p and the points are contained in one element, then $\#a = (p+1)^d$ and the time cost of `eval` is quasi-optimal for a given degree (h-refinement) but behaves badly as the degree increases (p-refinement).

3.4 Tensor factorization

The tensor-product structure allows to reduce d-variate computations to computations on univariate objects. In our case it allows to replace the computation of the linear computation of d-variate functions with d-linear combinations of univariate functions. This is advantageous because the cost of the matrix-matrix product is roughly proportional to the product of the three involved dimensions. This optimization reduces one of the dimensions to its d-th root. Furthermore, it can be combined with other optimizations based on the tensor structure such as the *sum-factorization*[2,3].

The spline spaces of interest do not have a global tensor-product structure, but this is not necessary, as it is sufficient that each $\gamma \in \mathit{\Gamma}$ and each $\beta \in \mathscr{B}_i$ can be factored into products of univariate functions:

$$\gamma(\mathbf{x}) = \prod_{c=1}^{d} \gamma^{(c)}(x_c), \qquad \beta(\mathbf{x}) = \prod_{c=1}^{d} \beta^{(c)}(x_c) \ . \tag{6}$$

Here the notation $\square^{(c)}$ means the *factor* of $\square$ corresponding to the c-th coordinate. By analogy the same notation will be used for tensors $\square = \bigotimes_{c=1}^{d} \square^{(c)}$ and Cartesian grids of points $\square = \bigtimes_{c=1}^{d} \square^{(c)}$. This should not be confused with the *components* of vectors and tensors that are denoted by subscripts as in $\mathbf{v} = (v_1, v_2)$.

The requirement (6) is satisfied by HB and HLR splines, but not by THB, DHB and TBPN. Thus we decided not to implement this optimization and the following is only a theoretical analysis.

To describe the optimization it is necessary to factor each object into univariate components:

$$\mathscr{B}_i^{(c)} = \{\beta^{(c)} : \beta \in \mathscr{B}_i\} \ ; \tag{7}$$

$$\mathit{\Gamma}^{(c)} = \{\gamma^{(c)} : \gamma \in \mathit{\Gamma}\} \ ; \tag{8}$$

$$X^{(c)} = \{x_c : \mathbf{x} = (x_1, \ldots, x_d) \in X\} \ . \tag{9}$$

Necessarily $\mathit{\Gamma}^{(c)} \subseteq \operatorname{span} \mathscr{B}_i^{(c)}$, which means that there exists a matrix $\mathbf{M}_i^{(c)}$ such that for all $\mathbf{x} \in \mathcal{D}_i$

$$\mathit{\Gamma}^{(c)}(\mathbf{x}) = \mathscr{B}_i^{(c)}(\mathbf{x}) \mathbf{M}_i^{(c)} \ . \tag{10}$$

Let S be the set of the multiindices that define $\mathit{\Gamma}$ as a subset of $\bigotimes_{c=1}^{d} \mathit{\Gamma}^{(c)}$:

$$\mathit{\Gamma} = \left\{ \prod_{c=1}^{d} \gamma_{\mathbf{s}_c}^{(c)} : (\mathbf{s}_1, \ldots, \mathbf{s}_d) \in S,\, \gamma_{\mathbf{s}_c}^{(c)} \in \mathit{\Gamma}^{(c)} \right\} \subset \bigotimes_{c=1}^{d} \mathit{\Gamma}^{(c)}. \tag{11}$$

For simplicity it is assumed that $\mathscr{B}_i = \bigotimes_{c=1}^{d} \mathscr{B}_i^{(c)}$ and $X = \bigtimes_{c=1}^{d} X^{(c)}$, but a proper subset (similarly as for $\mathit{\Gamma}$) can be considered at the expense of a more involved notation.

The tensor structure propagates to the set of active functions. Here a contains the multiindices of the functions of $\mathscr{B}_i$ corresponding to nonzero columns of $\mathscr{B}_i$. Similarly A contains the subset of the multiindices in S that correspond to nonzero columns in Γ. Analogously to the other symbols, $a^{(c)}$ and $A^{(c)}$ denote the collection of the entries relative to the c-th coordinate in a and A respectively.

Each derivative of $\gamma \in \Gamma$ is a product of derivatives of the $\gamma^{(c)}$. In the following cost computations $w^{(c)}$ denotes the number of derivatives of $\gamma^{(c)}$ that are required in order to compute all the w requested partial derivatives of γ.

The procedure `compose` assembles $\square$ out of its factors $\square^{(c)}$ and a list of the necessary products P as in the description of S above. If P is omitted, it is assumed that $\square = \bigotimes_{c=1}^{d} \square^{(c)}$ and thus that P contains all the Cartesian multiindices.

Procedure: `compose`$(\square^{(1)}, \ldots, \square^{(d)}, X, P)$
  Input: the tensor components $\square^{(c)}$
  Input: the list of required products P
  Input: the list of points X
  Output: $\Gamma(X)$
  foreach $\mathbf{p} \in P$ **do**
    add column to $\square$
    foreach $\mathbf{x} = (x_1, \ldots, x_d) \in X$ **do**
      `/* write row block of the derivatives of` $\square_{\mathbf{p}}$ `at x */`
      $\square_{\mathbf{p}} = \prod_{c=1}^{d} \square^{(c)}_{\mathbf{p}_c}(x_c)$ `/* value */`
      ... `/* derivatives */`
    end
  end

The complexity $\mathfrak{C}$(`compose`) is proportional to $d\#w\#P\#X$, because for each of the $w\#P\#X$ output data there are d products required. The `compose` procedure allows the rewriting of `eval` by splitting $\mathscr{B}_i$`.eval` in two independent steps: the evaluation of its tensor components and the composition.

Procedure: `eval`(X)
  Input: the points X
  Assumptions: $\exists i : X \subset \mathcal{D}_i$
  Output: $\Gamma(X)$
  $i = \mathbf{D}$`.findSubdomain`$(\mathbf{x}_1)$
  for $c = 1, \ldots, d$ **do**
    $\mathscr{B}^{(c)} = \mathscr{B}_i^{(c)}$`.eval`$(X^{(c)})$ `/* local evaluation */`
  end
  $\mathscr{B}_i(X) =$ `compose`$(\mathscr{B}^{(1)}, \ldots, \mathscr{B}^{(d)}, X)$ `/* composition */`
  $\Gamma(X) = \mathscr{B}_i(X)\mathbf{M}_i$ `/* linear combination */`

Table 1: Comparison of the cost for the standard and optimized evaluation for spaces with tensor structure.

	standard	optimized
linear combination	$w\#X\#A\#a$	$\sum_{c=1}^{d} w^{(c)}\#X^{(c)}\#A^{(c)}\#a^{(c)}$
composition	$dw\#X\#a$	$dw\#X\#A$

Furthermore, tensor composition and linear combination can be swapped, giving the following optimized version of `eval`:

Procedure: `eval(`X`)`
  Input: the points X
  Assumptions: $\exists i : X \subset \mathcal{D}_i$
  Output: $\Gamma(X)$
  $i = \mathbf{D}$.`findSubdomain`$(\mathbf{x}_1)$
  for $c = 1, \ldots, d$ **do**
    $\mathscr{B}^{(c)} = \mathscr{B}_i^{(c)}$.`eval`$(X^{(c)})$ `/* local evaluation */`
    $\Gamma^{(c)} = \mathscr{B}^{(c)}\mathbf{M}_i^{(c)}$ `/* linear combination */`
  end
  $A = S \cap \bigtimes_{c=1}^{d} A^{(c)}$ `/* actives */`
  $\Gamma(X) =$ `compose`$(\Gamma^{(1)}, \ldots, \Gamma^{(d)}, X, A)$ `/* composition */`

The cost of each step in the two different versions is reported in Table 1. The cost of the optimized version is proportional to the output size with a factor that is independent of the mesh (h-refinement) and of the degree (p-refinement) provided that there exists a σ such that for $c = 1, \ldots, d$

$$\#a^{(c)}\#A^{(c)} \leq \sigma\#A \ . \tag{12}$$

This is a reasonable assumption for the spline spaces of interest. Indeed, for polynomial splines of degree p with points contained in a single polynomial element $\#a^{(c)} = p + 1$ and $\#A \geq (p + 1)^d$. This means that $\#A^{(c)}$ should be bounded by $\sigma(p + 1)^{d-1}$. This is the case for σ-admissible HB meshes [11], where $\#A^{(c)} \leq \sigma(p + 1)$, and for the HLR basis described in [7], for which $\#A^{(c)} \leq 2(p + 1)$.

4 Spline Spaces

We tested the proposed strategy by implementing several spline spaces in the G+SMO object oriented library [24]. We have implemented HB, THB, TBPN, DHB and HLR splines.

The interested reader can compare with other implementations that are either available or described in the literature. (T)HB are implemented in the G+SMO open-source library [20]. The code, as of 2014, is described in [27]. Another implementation of (T)HB tailored for IGA research is described in [38]. The source code of bivariate LR-splines is available as a part of the goTools library [21], but no technical description is available.

The presented framework focuses on versatility and not on performance. Nevertheless, we believe the the following two notes are worth mentioning. The choice of the binary partition tree for $\mathbf{D}$ was dictated by performance considerations. The other strategies that we tried provided abysmal performances for (T)HB compared to the implementation in G+SMO, particularly for initialization. We tested the correctness of our (T)HB implementation by comparing with the results provided by G+SMO in selected 2D examples. For those examples the new implementation was both faster and used less memory compared to G+SMO.

4.1 (Truncated) Hierarchical B-Splines

The hierarchical B-spline basis [28] is defined from a sequence of tensor-product B-spline bases $\Psi_1, \dots, \Psi_L$ such that

$$i < j \Rightarrow \operatorname{span} \Psi_i \subset \operatorname{span} \Psi_j \tag{13}$$

and from a decreasing sequence of nested closed domains

$$\Omega = \mathcal{Q}_1 \supseteq \cdots \supseteq \mathcal{Q}_L \supseteq \mathcal{Q}_{L+1} = \emptyset \ .$$

The hierarchical basis is constructed by selecting some functions from each basis through the Kraft procedure:

$$\mathcal{H} = \bigcup_{i=1}^{L} \{\psi \in \Psi_i \ : \ \operatorname{support} \psi \subseteq \mathcal{Q}_i \text{ and } \operatorname{support} \psi \cap (\mathcal{Q}_i \setminus \mathcal{Q}_{i+1}) \neq \emptyset\} \ . \tag{14}$$

As usually, the support is restricted to Ω, i.e., $\operatorname{support} f = \{\mathbf{x} \in \Omega \ : \ f(\mathbf{x}) \neq 0\}$.

The truncated variant of the hierarchical spline basis $\mathcal{H} = \{\gamma_1, \dots, \gamma_n\}$ is the THB basis $\mathcal{H}' = \{\gamma'_1, \dots, \gamma'_n\}$, cf. [18]. Each basis function γ'_i is obtained from γ_i by repeated *truncation*. This means that each $\gamma \in \Psi_i \cap \mathcal{H}$ is represented as a linear combination of functions from Ψ_{i+1}. Then the coefficients of the functions in $\Psi_{i+1} \cap \mathcal{H}$ (i.e., of the selected functions from level $i+1$) are set to zero. The resulting function is then represented as a linear combination of functions in Ψ_{i+2}, the coefficients of the functions in $\Psi_{i+2} \cap \mathcal{H}$ are set to zero and so on, up to $i = L$. The resulting linear combination of Ψ_L is g'. This procedure improves the locality of the resulting basis and guarantees that the basis forms a convex partition of unity and preserves the coefficients of the corresponding tensor-product basis [19]. The drawback is that it breaks the tensor structure and thus it disallows the optimization described in Section 3.4.

We describe two possible implementations that correspond to different trade-offs between code complexity and memory requirements. Both are limited to

bases $\Psi_1, \ldots, \Psi_L$ of the same degree (i.e., only h-refinement is allowed) and domains $\mathcal{Q}_i$ that are unions of elements of $\operatorname{span} \Psi_i$. This restriction compares well with the other implementations available. For instance, the implementation in G+SMO requires Ψ_{i+1} to be the dyadic refinement of Ψ_i, which makes it impossible to experiment with other refinements.

Implementation 1 The simplest implementation defines $\mathbf{D}$ by:

$$\mathcal{D}_i = \mathcal{Q}_i \setminus \mathcal{Q}_{i+1}, \qquad \mathcal{D}_L = \mathcal{Q}_L$$

and $\mathbf{B}$ by

$$\mathscr{B}_i = \Psi_i.$$

The representation matrix $\mathbf{M}$ is built iteratively while discovering the functions selected by the Kraft procedure. Starting from level $i = m$ up to level 0, the functions of Ψ_i that are active on $\mathcal{D}_i$ are collected using the implementation of the tensor-product space. For each function the Kraft conditions (14) are tested. If they are satisfied a new column is added to $\mathbf{M}$, otherwise the function is discarded. The coefficients in the added column are computed using the standard knot insertion algorithm. Proceeding from the finest level to the coarsest allows to perform truncation in the same pass by discarding the coefficients corresponding to the selected functions of finer levels. Starting from the coarsest level does not allow to perform truncation in the same pass because the list of the selected functions from finer levels is not available yet.

Implementation 2 The choices above are the simplest, but they can cause a very high memory consumption. According to Subsection 3.1 the memory usage depends on the total number of rows in $\mathbf{M}$. For dyadic refinement of the Ψ_i, the number of rows grows as 2^{dL+d}, where d is the domain dimension and L is the number of levels. Since each row requires a memory pointer, this means that an empty $\mathbf{M}$ for a 3D example with 10 levels exceeds 10 gigabytes in size.

The problem can be solved using slightly more complex code. The main idea is to remove the rows containing only zeros from $\mathbf{M}$. This is the case for the rows corresponding to the functions $\psi \in \Psi_i$ with $\psi(\mathcal{D}_i) = \{0\}$. Let $\widetilde{\mathbf{D}}$ be the tree representing the partition $\{\mathcal{Q}_i \setminus \mathcal{Q}_{i+1}\}_{i=1,\ldots,L-1}$. Then $\mathbf{D}$ is defined as

$$\mathbf{D} = \{\text{boxes corresponding to the leaves of } \widetilde{\mathbf{D}}\} = \{\mathcal{D}_i\}.$$

The set $\mathbf{B}$ is defined by setting $\mathscr{B}_i$ to the smallest tensor-product basis containing the functions of Ψ_j active on $\mathcal{D}_i$ for j such that $\mathcal{D}_i \subset \mathcal{Q}_j \setminus \mathcal{Q}_{j+1}$. In this way the implementation bases $\mathscr{B}_i$ are subsets of the definition bases Ψ_j containing the functions with non-zero coefficients. The construction of $\mathbf{M}$ is done as in the previous implementation, while taking into account the index change. This solution is not available in our code for (T)HB, but the required machinery was implemented for DHB.

4.2 Truncated B-Splines for Partially Nested Refinement

This is a generalization of (T)HB-splines and was proposed in [39]. It allows for independent refinement in different parts of the domain (see Fig. 4) and can help for multipatch geometries as shown in Example 2.

The requirement (13) is dropped and the sequence of nested domains is replaced by a partition of Ω in *patches* $\mathcal{Q}_1, \dots, \mathcal{Q}_L$. The construction requires the following compatibility condition: if $\mathcal{D}_i$ and $\mathcal{D}_j$ share a $(d-1)$-dimensional interface $I_{i,j} = \partial\mathcal{D}_i \cap \partial\mathcal{D}_j$, then

$$\operatorname{span}\Psi_i \subset \operatorname{span}\Psi_j \quad \text{or} \quad \operatorname{span}\Psi_j \supset \operatorname{span}\Psi_i \ .$$

This means that $\{\operatorname{span}\Psi_i\}$ is not totally ordered anymore, only *partially ordered*. In particular if the boundaries are disjoint or their intersection is not $(d-1)$-dimensional, the spaces $\operatorname{span}\Psi_i$ and $\operatorname{span}\Psi_j$ do not have to be comparable by $\subset$. Note that the construction requires "sufficient separation" of the patches associated to two incomparable spaces. The details can be found in [39].

Fig. 4: Left: TBPN-splines allow to refine the subdomains Ω_a and Ω_b independently. Right: THB-splines requires nested knot vectors for any pair of subdomains.

Basis functions are again a subset of $\bigcup_{i=1}^{L}\Psi_i$ and are selected using a modifications of Kraft's procedure based on *slave functions*. A function $\psi \in \Psi_i$ is called a *slave* if it is active on an $(n-1)$-dimensional interface $I_{i,j}$ with $\operatorname{span}\Psi_j \subset \operatorname{span}\Psi_i$. The set of slaves of level i is

$$\mathcal{S}_i = \{\psi \in \Psi_i \ : \ \exists j : \psi(I_{i,j}) \neq \{0\}, \operatorname{span}\Psi_j \subset \operatorname{span}\Psi_i, \dim I_{i,j} = n-1\} \ .$$

Slave functions are functions whose coefficients are determined by the coefficients of the functions of coarser bases on nearby patches together with the smoothness conditions.

The selected functions are defined by

$$\mathcal{M} = \bigcup_{i=1}^{L} \mathcal{M}_i \ , \tag{15}$$

where $\mathcal{M}_i$ contains the *master functions* of level i, i.e., the functions of Ψ_i that are active on $\mathcal{Q}_i$ and that a are not slaves:

$$\mathcal{M}_i = \{\psi \in \Psi_i : \psi(\mathcal{Q}_i) \neq \{0\},\ \psi \notin \mathcal{S}_i\} \ . \tag{16}$$

Truncation is defined in the same way as in the case of THB-splines. The resulting basis is called *truncated B-splines for partially nested refinement* (TBPN). The set $\mathcal{M}$ forms a non-negative partition of unity, it is a basis and, similarly to THB, it preserves the coefficients of polynomial representation. Moreover, if (13) holds, then TBPN reduces to THB with the same bases and appropriate subdomains. We refer to [39] for details.

Implementation We implemented only the truncated version of the construction. The partition $\mathbf{D}$ can be defined as

$$\mathcal{D}_i = \mathcal{Q}_i$$

and $\mathbf{B}$ by

$$\mathscr{B}_i = \Psi_i.$$

The matrix $\mathbf{M}$ is built iteratively while discovering the functions selected by the modified Kraft procedure. Again the first step is to find the functions in Ψ_i that are active on $\mathcal{D}_i$ using the implementation of tensor product space. For each function the modified Kraft conditions (16) are tested. If they are satisfied, we add a new column to $\mathbf{M}$, otherwise the function is discarded. Analogously to THB it is possible to compute truncation in the same pass. Let ψ be in $\mathcal{M}_i$ and γ be its truncated version. The coefficients $m_{\beta,\gamma}$ are computed using a recursive algorithm. For all j such that $\operatorname{span}\Psi_i \subset \operatorname{span}\Psi_j$ and $\dim(I_{i,j} \cap \operatorname{support}\psi) = n-1$ the expansion of ψ with respect of Ψ_j is computed by knot insertion. Then for each functions in $\mathcal{S}_j$ with a non-zero coefficient the procedure is repeated giving the coefficients of slaves of finer levels. It is possible that the same $\beta \in \mathscr{B}_k$ appears during different recursions while computing the same column. In this case care must be taken in order to appropriately sum or discard the different contributions.

The implementation described has the same problem of the first implementation of (T)HB: unreasonable memory consumption for the 3D case. This can be solved using the same strategy described for (T)HB.

4.3 Decoupled Hierarchical B-Splines

Contrarily to tensor-product B-splines, (T)HB do not always span the full space of piecewise polynomials on their mesh [30]. This observation was the starting point of the development of TDHB [29]. There *decoupling* is used in conjunction with truncation in to relax the assumptions required to span the full piecewise polynomial space. We implemented a modification of TDHB, which we call simply *decoupled hierarchical B-splines* (DHB). The novelty is that truncation is abandoned in favour of recursive decoupling.

First we introduce decoupling in a slightly more general version compared to [29]. Let f be a function in $\operatorname{span}\Psi$, let $c_{f,\psi}$ be the coefficients of its expansion with respect of Ψ

$$f = \sum_{\psi\in\Psi} c_{f,\psi}\psi$$

and let $O \subseteq \Omega$ be a domain. The *decoupling graph* $\Gamma(f,\Psi,O)$ is the graph whose vertices are

$$\Gamma_V(f,\Psi,O) = \{\psi \in \Psi : c_{f,\psi} \neq 0\} \tag{17}$$

and the edges are

$$\Gamma_E(f,\Psi,O) = \{(\psi,\psi') : \operatorname{support}\psi \cap \operatorname{support}\psi' \cap \overline{O} \neq \emptyset\} \ . \tag{18}$$

The *decoupling operator* $D_{\Psi,O}$ is a relation that associates to function $f \in \operatorname{span}\Psi$ one or more *decoupled functions* in $\operatorname{span}\Psi$:

$$D_{\Psi,O}(f) = \left\{ \sum_{\psi\in K} c_{f,\psi}\psi \ : \ K \text{ is a connected component of } \Gamma(f,\Psi,O) \right\} \ .$$

Let $\Psi_1,\dots,\Psi_L$ and $\mathcal{Q}_1 \supseteq \cdots \supseteq \mathcal{Q}_L$ be as in (T)HB. The *decoupled basis* Δ is defined by first recursively decoupling and then applying the Kraft selection mechanism. Let $\Delta_L = \Psi_L$ and

$$\Delta_i = \bigcup_{\psi\in\Psi_i} D_{\Delta_{i+1},\mathcal{Q}_i\setminus\mathcal{Q}_{i+1}}(\psi) \ . \tag{19}$$

Then according to Kraft's method:

$$\Delta = \bigcup_{i=1}^{L} \{f \in \Delta_i : \operatorname{support} f \subseteq \mathcal{Q}_i,\ f(\mathcal{Q}_i \setminus \mathcal{Q}_{i+i}) \neq \{0\}\} \ . \tag{20}$$

If for every $i = 1,\dots,L-1$ the support of each function $\psi \in \Psi_{i+1}$ intersects $\mathcal{Q}_i$ in a connected set, then the proposed decoupled hierarchical basis is the same as the TDHB basis. Consequently, by [29, Theorem 13] it is algebraically complete, i.e., it generates the full spline space on the underlying hierarchical mesh.

Implementation We proceeded as in the second implementation of (T)HB. For this space it is necessary to have bases $\mathscr{B}_i$ that are not active on the whole domain in order to discern between different outcomes of decoupling coming from the same function. Let $\widetilde{\mathbf{D}}$ the tree describing the subdomains $\mathcal{Q}_i \setminus \mathcal{Q}_{i+1}$. Then

$$\mathbf{D} = \{\text{boxes corresponding to the leaves of } \widetilde{\mathbf{D}}\} = \{\mathcal{D}_i\}$$

and $\mathbf{B}$ is defined by setting $\mathscr{B}_i$ to the smallest tensor-product basis containing the functions of Ψ_j active on $\mathcal{D}_i$ for j such that $\mathcal{D}_i \subseteq \mathcal{Q}_j \setminus \mathcal{Q}_{j+1}$.

The construction of $\mathbf{M}$ follows the definition of the space. First each Δ_i is constructed: for each function in Δ_i we store its expansion with respect of Δ_{i+1} and its originating function in Ψ_i. Then the Kraft selection mechanism is employed and for each selected function we insert a column in $\mathbf{M}$. Computing the coefficient $m_{\beta,\gamma}$ for $\gamma \in \Delta_i$ and $\beta \in \Psi_j$ $(j > i)$ is performed by first composing the precomputed change of bases from Δ_i to Δ_j and then storing the obtained coefficients according to the subdomain and the originating function.

4.4 Hierarchical Locally Refined Splines

HLR-splines are a special case of LR-splines. The definition of the LR-splines starts from a pair $\mathcal{M} = (\mathscr{K}, \mu)$, where $\mathscr{K}$ is partition of Ω into boxes (Cartesian products of intervals) and μ assigns to each interface between two boxes a nonnegative integer. The *spline space* $\mathbb{S}_p(\mathcal{M})$ associated to $\mathcal{M}$ is the space of functions whose restrictions to boxes in $\mathscr{K}$ are polynomials of degree p in each variable and such that their smoothness across the interface α is at least $p - \mu(\alpha)$.

A B-spline β is *nested* in a B-spline β' relatively to $\mathbb{S}_p(\mathcal{M})$, written $\beta \prec \beta'$, if there exists a sequence of B-splines $\beta = \beta_1, \ldots, \beta_n = \beta'$ such that each $\beta_i \in \mathbb{S}_p(\mathcal{M})$ and such that each β_{i+1} is obtained from β_i by knot insertion.

The *LR-spline collection* is the set of minimal elements for the ordering $\prec$ that are comparable with at least one Bernstein polynomial on Ω.

The obtained generators do not necessarily span the whole space of piecewise polynomials satisfying the smoothness conditions, nor they are always linearly independent [15,7]. Many properties of the generators are linked together, in particular local linear independence and being a partition of unity are equivalent [6].

HLR are a class of LR-splines enjoying both local linear independence and the partition of unity property. This is obtained by mimicking the HB approach and constructing the mesh from the grids of a sequence of tensor-product B-spline spaces $V_1 \subset \ldots \subset V_L$ with $V_i = \operatorname{span} \Psi_i$ and a corresponding sequence of subdomains $\mathcal{Q}_1 \supseteq \ldots \supseteq \mathcal{Q}_L$. The partition $\mathscr{K}$ is

$$\mathscr{K} = \bigcup_{i=1}^{L} \{\eta \text{ element of the partition corresponding to } V_i : \eta \subseteq \mathcal{Q}_i \setminus \mathcal{Q}_{i+1}\} \quad (21)$$

and μ describes the smoothness of the space V_i on $\mathcal{Q}_i \setminus \mathcal{Q}_{i+1}$. With this construction and assuming that

- V_i is obtained by refining a single tensor-component of V_{i-1}, that is by h-refinement in a single direction;
- the borders of the $\mathcal{Q}_i$ are sufficiently separated,

the generators form a partition of unity and they are locally linearly independent [7].

Implementation In our implementation $\mathbf{D}$ is defined by:

$$\mathcal{D}_i = \mathcal{Q}_i \setminus \mathcal{Q}_{i+1}, \qquad \mathcal{D}_L = \mathcal{Q}_L$$

and $\mathbf{B}$ by

$$\mathscr{B}_i = \Psi_i.$$

For each $\psi \in \Psi_i$ that is active on $\mathcal{Q}_i$ we check if it is a minimal support B-spline. If so we add a column to $\mathbf{M}$ and we compute the coefficients using knot insertion as for (T)HB. If not, we refine the function by inserting recursively the knots that are compatible with the mesh. For each of the refined functions that is active on $\mathcal{Q}_i$ a column is added to $\mathbf{M}$ (care must be taken to avoid duplicated columns).

5 Examples

This section contains some selected examples that can be useful to grasp the similarities and the differences between the implemented spline spaces. The basis functions have been plotted with Paraview [1] using the data produced with the implementations described in the previous section.

Example 1. We consider bivariate hierarchical splines of bi-degree $(4, 4)$ on a mesh shown in Fig. 5. The function with the support indicated by the red tiling is selected in the hierarchical basis (Fig. 6 left), truncated in the truncated hierarchical basis (Fig. 6 right) and decoupled into four different functions (that are selected) in the decoupled hierarchical basis (Fig. 7).

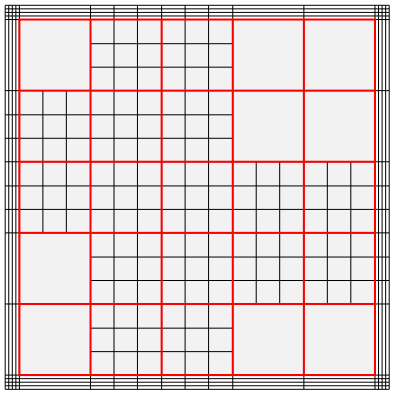


Fig. 5: Hierarchical mesh and a support of a function from Example 1.

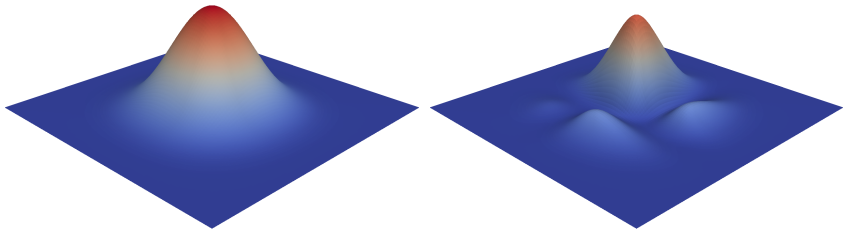


Fig. 6: Function with the support in Fig. 5 as selected into the hierarchical basis (left) and truncated in the truncated basis (right).

Example 2. The design process often involves several patches. In order to achieve continuity between the patches without losing accuracy, it is necessary that the restrictions of the two spaces are compatible on the interface. That means that one has to be a subspace of the other.

Sometimes a new patch must be introduced to bridge between two given patches that should not be modified. Thus the restriction of the space of the bridge patch to each boundary must be a superspace of the restrictions of the other space. If the two given patches have different knot vectors, THB-splines would lead to significant refinement. On the other hand, the TBPN space can achieve interface compatibility without adding unnecessary degrees of freedom.

We have constructed cubic basis on the mesh depicted in Fig. 8 and observed that the THB basis has 72 degrees of freedom, whereas the TBPN basis only has 60.

Example 3. We compare cubic HB, THB, DHB and HLR on a mesh shown in Fig. 9. For each of these spaces, we plot all the basis functions in Fig. 10. Note that the number of basis function in the middle of the patch is higher for HLR and DHB. In particular, HB and THB basis have 49 elements each; HLR and DHB have 53 and are complete, as the meshes fulfill the assumptions from [7] and [29].

6 Conclusions

The effectiveness of the proposed framework is demonstrated by the implementation of various spline spaces that share the same evaluation code. The space-specific code is reduced to the initialization of the required data structures as demonstrated by the implementations of HB, THB, TBPN, DHB and HLR. Moreover, the proposed approach grants the following advantages:

1. code reduction both by sharing evaluation between different spaces and between spaces and functions;

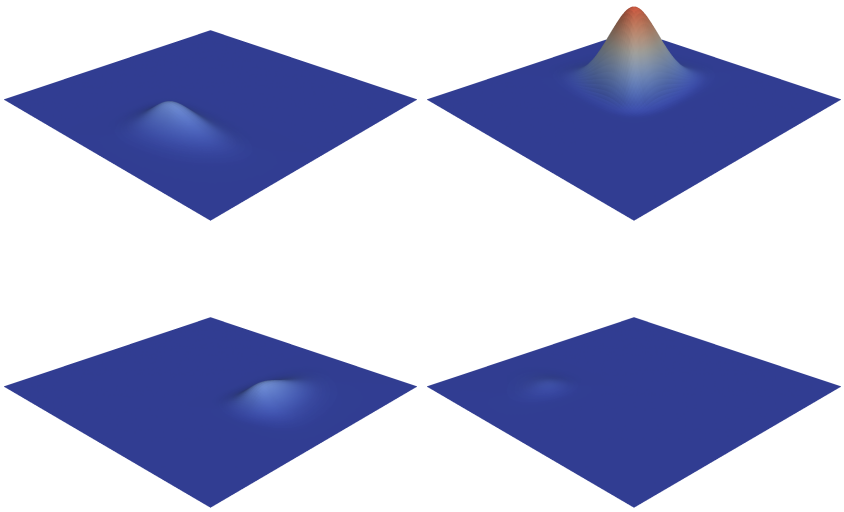


Fig. 7: Decoupled functions stemming from the B-spline with the support indicated in Fig. 5.

2. arbitrary local bases that, in principle, open the way to experimentation with hierarchical constructions based on generalized splines [5], or to the use of ad-hoc functions near a priori known singularities;
3. transparent handling of multipatch domains.

Acknowledgments

The authors have been supported by the Austrian Science Fund (FWF, NFN S117 "Geometry + Simulation") and by the Seventh Framework Programme of the EU (project EXAMPLE, GA No. 324340). This support is gratefully acknowledged.

References

1. Ahrens, J., Geveci, B., Law, C., Hansen, C.D., Johnson, C.: ParaView: An end-user tool for large-data visualization (2005)
2. Ainsworth, M., Andriamaro, G., Davydov, O.: Bernstein-Bézier finite elements of arbitrary order and optimal assembly procedures. SIAM J. Sci. Comput. 33(6), 3087–3109 (2011), http://dx.doi.org/10.1137/11082539X
3. Antolin, P., Buffa, A., Calabrò, F., Martinelli, M., Sangalli, G.: Efficient matrix computation for tensor-product isogeometric analysis: the use of sum factorization.

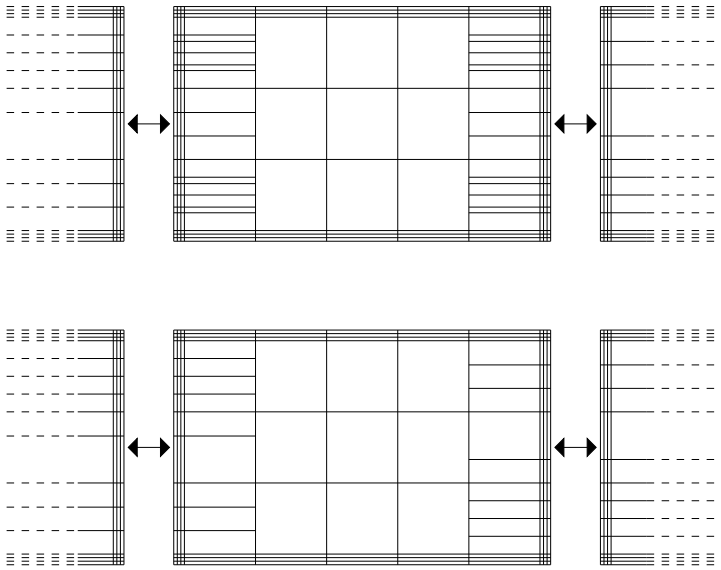


Fig. 8: Meshes from Example 2. Top: THB mesh; bottom: TBPN mesh.

Comput. Methods Appl. Mech. Engrg. 285, 817–828 (2015), http://dx.doi.org/10.1016/j.cma.2014.12.013
4. Borden, M.J., Scott, M.A., Evans, J.A., Hughes, T.J.R.: Isogeometric finite element data structures based on Bézier extraction of NURBS. Internat. J. Numer. Methods Engrg. 87(1-5), 15–47 (2011), http://dx.doi.org/10.1002/nme.2968
5. Bracco, C., Lyche, T., Manni, C., Roman, F., Speleers, H.: Generalized spline spaces over T-meshes: dimension formula and locally refined generalized B-splines. Appl. Math. Comput. 272(part 1), 187–198 (2016)
6. Bressan, A.: Some properties of LR-splines. Comput. Aided Geom. Design 30(8), 778–794 (2013), http://dx.doi.org/10.1016/j.cagd.2013.06.004
7. Bressan, A., Jüttler, B.: A hierarchical construction of LR meshes in 2D. Comput. Aided Geom. Design 37, 9–24 (2015), http://dx.doi.org/10.1016/j.cagd.2015.06.002
8. Brovka, M., López, J., Escobar, J., Montenegro, R., Cascón, J.: A simple strategy for defining polynomial spline spaces over hierarchical T-meshes. Computer-Aided Design 72, 140–156 (2016)
9. Buchegger, F., Jüttler, B., Mantzaflaris, A.: Adaptively refined multi-patch B-splines with enhanced smoothness. Appl. Math. Comput. 272(part 1), 159–172 (2016), http://dx.doi.org/10.1016/j.amc.2015.06.055
10. Buffa, A., Garau, E.M.: New refinable spaces and local approximation estimates for hierarchical splines. arXiv preprint arXiv:1507.06534 (2015)
11. Buffa, A., Giannelli, C.: Adaptive isogeometric methods with hierarchical splines: error estimator and convergence. Math. Models Methods Appl. Sci. 26(1), 1–25 (2016), http://dx.doi.org/10.1142/S0218202516500019
12. Collin, A., Sangalli, G., Takacs, T.: Approximation properties of multi-patch C^1 isogeometric spaces. arXiv preprint arXiv:1509.07619 (2015)

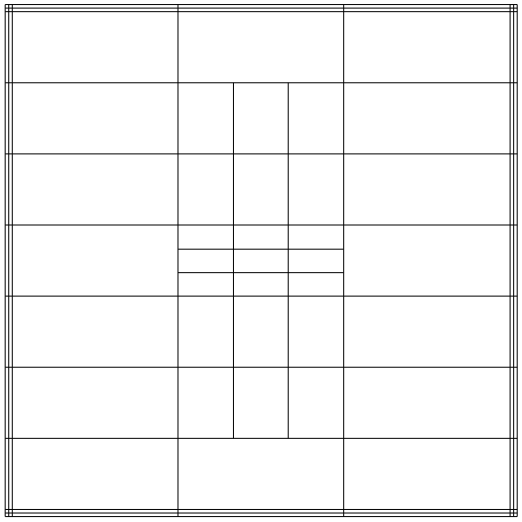
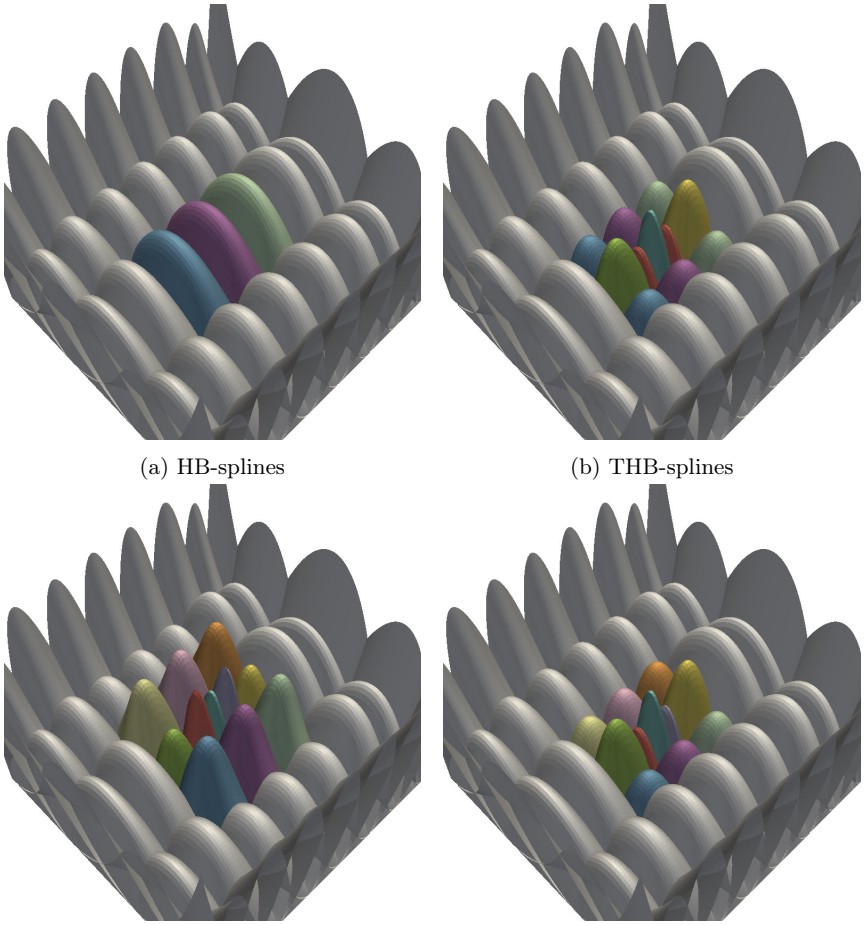


Fig. 9: Mesh from Example 3.

13. Da Veiga, L.B., Buffa, A., Sangalli, G., Vázquez, R.: Analysis-suitable T-splines of arbitrary degree: definition, linear independence and approximation properties. Math. Models Methods Appl. Sci. 23(11), 1979–2003 (2013)
14. Deng, J., Chen, F., Li, X., Hu, C., Tong, W., Yang, Z., Feng, Y.: Polynomial splines over hierarchical T-meshes. Graphical models 70(4), 76–86 (2008)
15. Dokken, T., Lyche, T., Pettersen, K.F.: Polynomial splines over locally refined box-partitions. Comput. Aided Geom. Design 30(3), 331–356 (2013)
16. Evans, E.J., Scott, M.A., Li, X., Thomas, D.C.: Hierarchical T-splines: analysis-suitability, Bézier extraction, and application as an adaptive basis for isogeometric analysis. Comput. Methods Appl. Mech. Engrg. 284, 1–20 (2015), http://dx.doi.org/10.1016/j.cma.2014.05.019
17. Forsey, D.R., Bartels, R.H.: Hierarchical B-spline refinement. SIGGRAPH Comput. Graph. 22(4), 205–212 (Jun 1988)
18. Giannelli, C., Jüttler, B., Speleers, H.: THB-splines: the truncated basis for hierarchical splines. Comput. Aided Geom. Design 29(7), 485–498 (2012), http://dx.doi.org/10.1016/j.cagd.2012.03.025
19. Giannelli, C., Jüttler, B., Speleers, H.: Strongly stable bases for adaptively refined multilevel spline spaces. Adv. Comput. Math. 40(2), 459–490 (2014), http://dx.doi.org/10.1007/s10444-013-9315-2
20. Geometry + simulation modules (g+smo). http://www.gs.jku.at/gismo (2016), open Source C++ library for Isogeometric Analysis
21. GoTools. https://github.com/SINTEF-Geometry/GoTools (2016), collection of C++ libraries connected to geometry
22. Hughes, T.J.R., Cottrell, J.A., Bazilevs, Y.: Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement. Comput. Methods Appl. Mech. Engrg. 194(39-41), 4135–4195 (2005), http://dx.doi.org/10.1016/j.cma.2004.10.008



(a) HB-splines (b) THB-splines

(c) HLR-splines (d) DHB-splines

Fig. 10: Details of the bases from Example 3. Only the basis functions that differ have been marked in colour. Note that not all the HB basis functions are visible: three are hidden in the central area.

23. Johannessen, K.A., Remonato, F., Kvamsdal, T.: On the similarities and differences between classical hierarchical, truncated hierarchical and LR B-splines. Comput. Methods Appl. Mech. Engrg. 291, 64–101 (2015), http://dx.doi.org/10.1016/j.cma.2015.02.031
24. Jüttler, B., Langer, U., Mantzaflaris, A., Moore, S.E., Zulehner, W.: Geometry + simulation modules: Implementing isogeometric analysis. PAMM 14(1), 961–962 (2014)
25. Kang, H., Xu, J., Chen, F., Deng, J.: A new basis for PHT-splines. Graphical Models 82, 149–159 (2015)

26. Kapl, M., Vitrih, V., Jüttler, B., Birner, K.: Isogeometric analysis with geometrically continuous functions on two-patch geometries. Comput. Math. Appl. 70(7), 1518–1538 (2015), http://dx.doi.org/10.1016/j.camwa.2015.04.004
27. Kiss, G., Giannelli, C., Jüttler, B.: Algorithms and data structures for truncated hierarchical B-splines. In: Mathematical methods for curves and surfaces, Lecture Notes in Comput. Sci., vol. 8177, pp. 304–323. Springer, Heidelberg (2014), http://dx.doi.org/10.1007/978-3-642-54382-1_18
28. Kraft, R.: Adaptive and linearly independent multilevel B–splines. In: Le Méhauté, A., Rabut, C., Schumaker, L.L. (eds.) Surface Fitting and Multiresolution Methods. pp. 209–218. Vanderbilt University Press, Nashville (1997)
29. Mokriš, D., Jüttler, B.: TDHB-splines: the truncated decoupled basis of hierarchical tensor-product splines. Comput. Aided Geom. Design 31(7-8), 531–544 (2014), http://dx.doi.org/10.1016/j.cagd.2014.05.004
30. Mokriš, D., Jüttler, B., Giannelli, C.: On the completeness of hierarchical tensor-product B-splines. J. Comput. Appl. Math. 271, 53–70 (2014), http://dx.doi.org/10.1016/j.cam.2014.04.001
31. Morgenstern, P.: 3D analysis-suitable T-splines: definition, linear independence and m-graded local refinement. arXiv preprint arXiv:1505.05392 (2015)
32. Morgenstern, P., Peterseim, D.: Analysis-suitable adaptive T-mesh refinement with linear complexity. Comput. Aided Geom. Design 34, 50–66 (2015), http://dx.doi.org/10.1016/j.cagd.2015.02.003
33. Scott, M.A., Borden, M.J., Verhoosel, C.V., Sederberg, T.W., Hughes, T.J.R.: Isogeometric finite element data structures based on Bézier extraction of T-splines. Internat. J. Numer. Methods Engrg. 88(2), 126–156 (2011), http://dx.doi.org/10.1002/nme.3167
34. Sederberg, T.W., Cardon, D.L., Finnigan, G.T., North, N.S., Zheng, J., Lyche, T.: T-spline simplification and local refinement. ACM Trans. Graph. 23(3), 276–283 (Aug 2004), http://doi.acm.org/10.1145/1015706.1015715
35. Sederberg, T.W., Zheng, J., Bakenov, A., Nasri, A.: T-splines and T-NURCCs. ACM Trans. Graph. 22(3), 477–484 (Jul 2003), http://doi.acm.org/10.1145/882262.882295
36. Thibault, W.C., Naylor, B.F.: Set operations on polyhedra using binary space partitioning trees. In: Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques. pp. 153–162. SIGGRAPH '87, ACM, New York, NY, USA (1987), http://doi.acm.org/10.1145/37401.37421
37. Toth, C.D., O'Rourke, J., Goodman, J.E.: Handbook of discrete and computational geometry. CRC press (2004)
38. Vázquez, R., Garau, E.: Algorithms for the implementation of adaptive isogeometric methods using hierarchical splines. Tech. Rep. 16-08, IMATI-CNR, Pavia (July 2016)
39. Zore, U.: Constructions and Properties of Adaptively Refined Multilevel Spline Spaces. Dissertation, Johannes Kepler University Linz (2016), http://epub.jku.at/obvulihs/download/pdf/1273941?originalFilename=true