A simple strategy for defining polynomial spline spaces over hierarchical T-meshes

M. Brovka[a], J.I. López[a], J.M. Escobar[a], R. Montenegro[a], J.M. Cascón[b]

[a] *University of Las Palmas de Gran Canaria, University Institute for Intelligent Systems and Numerical Applications in Engineering (SIANI), Spain.*
[b] *University of Salamanca, Department of Economics and Economic History, Faculty of Economics and Business, Spain.*

Abstract

We present a new strategy for constructing spline spaces over hierarchical T-meshes with quad- and octree subdivision scheme. The proposed technique includes some simple rules for inferring local knot vectors to define C^2-continuous cubic tensor product spline blending functions. Our conjecture is that these rules allow to obtain, for a given T-mesh, a set of linearly independent spline functions with the property that spaces spanned by nested T-meshes are also nested, and therefore, the functions can reproduce cubic polynomials. In order to span spaces with these properties applying the proposed rules, the T-mesh should fulfill the only requirement of being a *0-balanced* mesh. The straightforward implementation of the proposed strategy can make it an attractive tool for its use in geometric design and isogeometric analysis. In this paper we give a detailed description of our technique and illustrate some examples of its application in isogeometric analysis performing adaptive refinement for 2D and 3D problems.

Keywords: isogeometric analysis, multivariate splines, local refinement, T-mesh, nested spaces.

1. Introduction

The main drawback of using B-splines and NURBS for geometric design is the impossibility to perform local refinement. T-splines were introduced by

Email address: mbrovka@siani.es (M. Brovka)

Sederberg et al. [1] as an alternative to NURBS. Based on the idea of admitting meshes with T-junctions and inferring local knot vectors by traversing T-mesh edges, T-splines have provided a promising tool for geometric modelling that allows to perform local refinement without introducing a large number of superfluous control points. Later, in [2], T-splines were incorporated to the framework of isogeometric analysis. Isogeometric analysis (IGA) was introduced in 2005 by Hughes et al. in [3, 4]. It has arisen as an attempt to unify the fields of CAD and classical finite element methods. The main idea of IGA consists in using for analysis the same functions that are used in CAD representation of the geometry.

To use spline functions for numerical analysis and obtain a proper convergence behaviour, these functions must meet some requirements: linear independence, polynomial reproduction property, local supports and possibility to perform local adaptive refinement. This issue has been the object of numerous research works in recent years.

Analysis-suitable T-splines, proposed by Scott et al. in [5], are a class of T-splines defined over T-meshes that should meet certain topological restrictions formulated in terms of T-junction extensions. Basis functions defined over an extended analysis-suitable T-mesh are linearly independent [6] and form a partition of unity. The refinement algorithm allows to accomplish highly localized refinements and constructs nested T-spline spaces, but it presents an elevated implementation complexity and, as far we know, the generalization of the strategy to 3D cases is still an open question.

Another approach to the problem of local enrichment of the approximation space is the hierarchical refinement, originally introduced by Forsey and Bartels in [7] and later developed in [8]. Recently, hierarchical refinement technique in the context of isogeometric analysis was described in [9, 10, 11]. This approach is based on a simple and natural idea to construct multilevel spaces by replacing coarse level functions with finer basis functions. Starting from an initial uniform mesh, hierarchical refinement scheme leads to sequential construction of nested spline spaces with linearly independent basis functions. Simplicity of its implementation and straightforward generalization to 3D make it an attractive option for local refinement. However, a drawback of this strategy is the impossibility to define a spline space over a given arbitrary T-mesh as well as the presence of redundant basis functions and excessive support overlapping. An interesting theoretical approach to the latter problem was given in [12]. The truncation technique is applied to redefine the function supports and reduce their overlapping.

Other options for performing local refinement of spline spaces are C^1-continuous PHT-splines [13] or local refined splines (LR-splines) [14].

In the present paper we propose another possible alternative for the construction of spline functions that span spaces with nice properties. The technique we present here is designed for hierarchical T-meshes (multilevel meshes) with a quad- and octree subdivision scheme. This type of meshes can be efficiently implemented with tree data structures [15] which are frequently used in engineering. Due to the elevated complexity of all current strategies, the main goal we pursue here is the simplicity and low computational cost of the implementation, both in 2D and 3D. For that, we have to assume a restriction on the T-mesh. Namely, the T-mesh should fulfill the requirement of being a *0-balanced* mesh. A balanced mesh condition is usually imposed to have gradual transition from the coarse mesh to the finer zones. In addition, for our technique, this condition is an obligatory prerequisite that the T-mesh should fulfill. Assuming this reasonable restriction over the T-mesh, we can define easily cubic spline functions that span spaces with desirable properties: linear independence, C^2-continuous, cubic polynomial reproduction property, nestedness of spanned spaces and a straightforward implementation. The key of the strategy lies in some simple rules used for inferring local knot vectors for each blending function.

The paper is organized as follows. Some basic concepts about B-splines and T-meshes are given in Section 2. Section 3 includes the general scheme of our strategy and the description of its main stages. In Section 4 we explain in detail the key of our technique, that is, the rules used for inferring function supports in order to span spaces with desirable properties. In Section 5 the properties of the defined functions are given and the issue of support overlapping and sparsity of stiffness matrix is discussed. Computational examples of performing adaptive refinement for 2D and 3D Poisson problems are presented in Section 6. Conclusions are given in Section 7.

2. Basic concepts

2.1. B-spline basis functions

We start with a brief summary of the main concepts about B-splines.

A set of B-spline basis functions $B_{i,p}$ $(i = 1, 2, \ldots, n)$ of degree p, inferred from a non-decreasing sequence $\Xi = \{\xi_1, \xi_2, \ldots, \xi_{n+p+1}\}$, called knot vector, is defined by the Cox-de Boor recursion formula

$$B_{i,0}(\xi) = \begin{cases} 1 & \text{if} \quad \xi_i \le \xi < \xi_{i+1}, \\ 0 & \text{othewise.} \end{cases}$$

$$B_{i,p}(\xi) = \frac{\xi - \xi_i}{\xi_{i+p} - \xi_i} B_{i,p-1}(\xi) + \frac{\xi_{i+p+1} - \xi}{\xi_{i+p+1} - \xi_{i+1}} B_{i+1,p-1}(\xi).$$

A knot vector Ξ is called open knot vector if the first and the last knots are repeated $p+1$ times. At each knot of multiplicity m the basis functions are C^{p-m}.

A B-spline curve is defined as a linear combination of B-spline basis functions

$$\mathbf{S}(\xi) = \sum_{i \in I} \mathbf{P}_i \; B_{i,p}(\xi),$$

where coefficients $\mathbf{P}_i \in \mathbb{R}^s$ are called control points, typically $s = 2$ or 3.

Multivariate B-splines are defined as a tensor product of univariate B-spline functions

$$B_{\mathbf{i},p}(\boldsymbol{\xi}) = \prod_{k=1}^{d} B_{i_k,p}(\xi^k),$$

where $\boldsymbol{\xi} = (\xi^1, \ldots, \xi^d)$ and the multi-index $\mathbf{i} = (i_1, \ldots, i_d) \in I$. The multi-index set is defined by $I = \{1, 2, \ldots, n_1\} \times \cdots \times \{1, 2, \ldots, n_d\}$.

A B-spline surface (solid) is defined as a linear combination of bivariate (trivariate) B-spline functions

$$\mathbf{S}(\boldsymbol{\xi}) = \sum_{\mathbf{i} \in I} \mathbf{P}_\mathbf{i} \; B_{\mathbf{i},p}(\boldsymbol{\xi}),$$

where the control points $\mathbf{P}_{\boldsymbol{i}} \in \mathbb{R}^s$ ($s = 2$ or 3) form a control mesh.

For more details about B-splines see [16].

2.2. T-meshes and T-splines

In order to overcome the drawback of tensor product structure, which does not allow to perform local refinement, it is necessary to admit T-junctions in the mesh. The concept *T-junction* is similar to *hanging node* in the classical finite element method. An axes-aligned grid that allows T-junctions is called T-mesh. As was mentioned in the previous section, there are several strategies to define tensor product spline functions over T-meshes and one of these

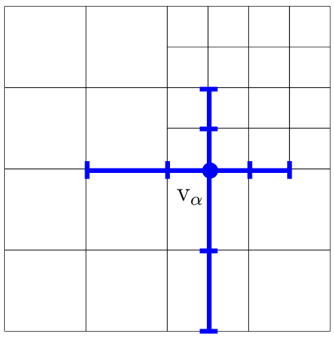


Figure 1: An example of T-mesh and inferring of local knot vectors for cubic T-spline functions by traversing T-mesh edges.

strategies is T-splines. The underlying idea of T-splines consists in defining blending functions by means of a set of local knot vectors instead of a global knot vector, as in the case of B-splines or NURBS. A local knot vector for each bivariate function B_α is inferred by traversing the T-mesh edges in both parametric directions starting from a vertex v_α of the mesh (the *anchor*), see Fig. 1. For a pair of local knot vectors $\Xi_\alpha^j = \left(\xi_1^j, \xi_2^j, \xi_3^j, \xi_4^j, \xi_5^j\right)$, $j = 1, 2$ the bicubic spline function B_α is defined as $B_\alpha(\xi^1, \xi^2) = B[\Xi_\alpha^1](\xi^1) B[\Xi_\alpha^2](\xi^2)$, where $B[\Xi_\alpha^j](\xi^j)$ is an univariate B-spline corresponding to the knot vector Ξ_α^j. In general, T-spline blending functions do not span a polynomial space. Some additional restrictions on the T-mesh configuration [5] should be satisfied in order to span a polynomial spline space. If these restrictions are not verified, the T-splines should be normalized in order to form a partition of unity. This leads to rational blending functions: $R_\alpha(\xi^1, \xi^2) = \dfrac{B_\alpha(\xi^1, \xi^2)}{\sum_{\beta \in A} B_\beta(\xi^1, \xi^2)}$, where A is the index set of the basis spanned by the T-mesh. These rational blending functions are capable of reproducing a constant function, but, in general, can not reproduce a polynomial of a higher order. A T-spline approximation is constructed as a linear combination of all blending functions: $S(\xi^1, \xi^2) = \sum_{\alpha \in A} P_\alpha \, R_\alpha(\xi^1, \xi^2)$.

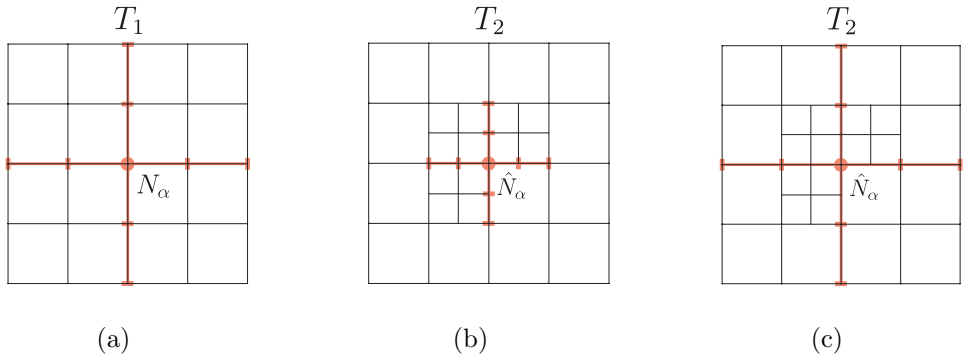


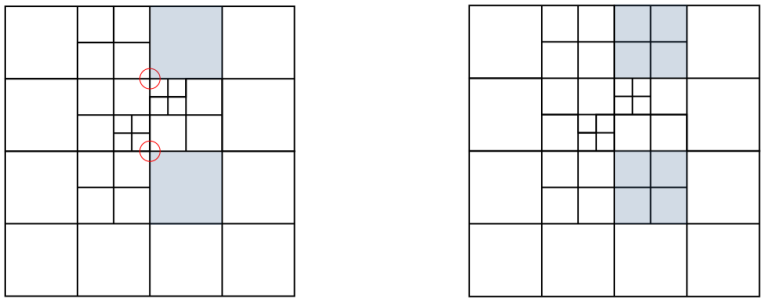
Figure 2: Motivation of the strategy. (a) Initial mesh and a basis function N_α. (b) Refined mesh, where the new T-spline space cannot reproduce the original function N_α. (c) We try to preserve the original basis function N_α in the new space if it cannot be recovered.

3. Strategy for the construction of polynomial spline spaces over hierarchical T-meshes.

In this section we describe our strategy to define tensor product spline functions over hierarchical T-meshes. The strategy we propose has some similarity with T-splines inasmuch as we define the blending functions from local knot vectors that are inferred by traversing the T-mesh edges. Some additional rules and requirements are imposed for the local knot vectors in order to obtain spline spaces with nice properties. The strategy is mainly motivated by the idea of preserving an original basis function if it cannot be reproduced with the basis of the new space after a refinement. For example, in the initial mesh of the Fig. 2(a), we define a basis function N_α associated to the vertex α. Then, we perform some cell refinements as shown in Fig. 2(b). It is easy to check that the new T-spline space obtained in Fig. 2(b) is not capable of reproducing the original basis function N_α. However, if we define the function $\hat{N}_\alpha$ as shown in Fig. 2(c), keeping in this case the initial basis unaltered in the new refined space, we guarantee the nestedness of spline spaces.

In our strategy, the process of spline space construction for a given T-mesh can be divided in the following three steps:

1. *Mesh pretreatment (0-balancing)*
2. *Inferring local knot vectors*
3. *Modification of local knot vectors*



(a) (b)

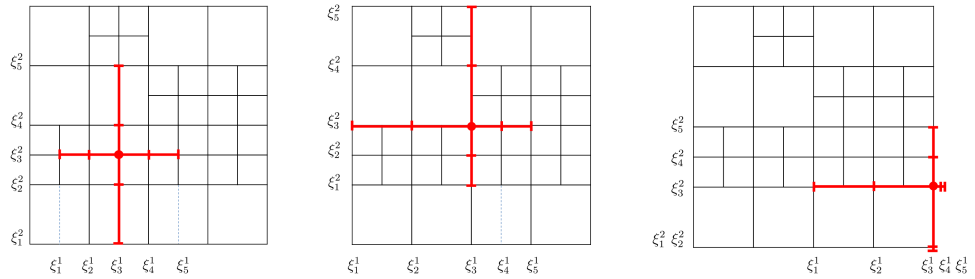
Figure 3: Example of *0-balancing* procedure. (a) A *0-unbalanced* quadtree mesh. Cells to be refined are shaded in blue. (b) Resulting *0-balanced* mesh.

Next, we give a description of each step of the process.

3.1. Mesh pretreatment. 0-balanced quadtree and octree T-meshes.

Due to its simplicity, quadtree and octree meshes are an attractive tool for performing adaptive refinement in IGA and geometric modelling. To guarantee a good quality of the approximation space constructed over a mesh, it is preferable to have a gradual transition from the coarse mesh to the finely refined zone. That is why it is common to work with balanced quadtree and octree meshes. The strategy we propose in this paper is designed exclusively for the *0-balanced* T-meshes. A mesh with tree structure is said to be *0-balanced* if for any k, no cell at level k shares a vertex (0-face) with a cell at level greater than $k+1$. In other words, a 0-balanced quadtree mesh implies that any cell has contact (through vertex, edge or face) only with cells that differ at most in one level of depth. An example of *0-balanced* procedure quadtree is shown in Fig. 3(b). To obtain a *0-balanced* quadtree, a standard balancing procedure is applied. Note that refinements performed during the *0-balancing* procedure do not propagate, see [17].

It should be highlighted that *0-balancing* the T-mesh is an essential prerequisite for the construction of spline spaces by means of our technique. In general, if the T-mesh is not *0-balanced*, our rules for inferring local knot vectors do not lead to polynomial spaces. Also, it is important to emphasize that, for our 2D (3D) T-meshes, a subdivision of any cell is performed by subdividing the cell in 4 (8) equal subcells so that, all cells of the same level



(a) No T-junction is skipped.

(b) Two T-junctions are skipped in each direction.

(c) Two T-junctions are skipped and the knots are repeated when the boundary is reached.

Figure 4: Inferring local knot vectors for a bivariate function by traversing the T-mesh edges.

have the same size and the edge size of a k-level cell is twice larger than the edge size of a $(k+1)$-level cell.

3.2. Inferring local knot vectors

Let us consider a T-mesh T of the squared parametric domain $\Omega = [0,1]^d$, $d = 2$ or 3. We call *regular node* the node of the mesh that is not a T-junction. We associate a blending function only to *regular nodes* of the mesh, as it is usual in classical finite element methods when working with hanging nodes. The *skeleton* of a d-dimensional mesh T is the geometric set of points composed of the union of all $(d-1)$-faces of the mesh and it is denoted by $\mathrm{skt}(T)$. That is, for a 2D space, the mesh *skeleton* is the union of all the edges of the mesh, and the *skeleton* of a 3D mesh is the union of all its faces.

To define our cubic tensor product spline blending functions over a given d-dimensional T-mesh, a local knot vector for d parametric directions should be assigned to each function N_α: $\Xi_\alpha^j = \left(\xi_1^j, \xi_2^j, \xi_3^j, \xi_4^j, \xi_5^j\right)$, $j = 1, ..., d$. Similarly to [1], these knot vectors are inferred by traversing the T-mesh *skeleton*. For simplicity, let us describe this procedure for a two-dimensional T-mesh. Starting from the central knot (ξ_3^1, ξ_3^2), i.e., the anchor of the function, we walk across the T-mesh until intersecting perpendicularly a mesh edge. According to our strategy, we should skip over the T-junctions where the missing edge is perpendicular to the direction of our marching, see Fig.

4. When the boundary of the parametric domain is reached while walking across the mesh, we repeat knots creating an open knot vector structure along the boundary, see Fig. 4(c). Note that all interior knots have multiplicity 1. Thus, we obtain for mesh T a set of blending functions $\{N_\alpha\}_{\alpha\in A_T}$, where A_T is the index set. Figure 5 illustrates an example of T-mesh in the parameter space and the anchors of all blending functions defined over this mesh. Any interior *regular node* has exactly one function associated to it and the boundary nodes have more than one function associated due to the open knot vector structure.

The process of inferring local knot vectors can be resumed as follows:

- *Blending functions are associated only to regular nodes of the mesh.*
- *Local knot vectors are inferred by walking across the mesh until intersecting the mesh skeleton. This intersection should not coincide with a T-junction perpendicular to the marching direction.*
- *Boundary knots are repeated to create an open knot vector structure along the boundary.*

Next, in order to span a spline space with good properties, some function supports should be modified. This issue is addressed in the next subsection.

3.3. Modification of local knot vectors

The key of our strategy lies in some simple rules used for the modification of the function supports that lead to the construction of a polynomial spline space over a given *0-balanced* T-mesh. In order to describe the idea, let us introduce some notation. For the local knot vectors $\Xi_\alpha^j = \left(\xi_1^j, \xi_2^j, \xi_3^j, \xi_4^j, \xi_5^j\right),\ j = 1, ..., d$ let us denote the length of each knot interval as $\Delta_i^j = \xi_{i+1}^j - \xi_i^j,\ j = 1, ..., d$ and $i = 1, ..., 4$.

The support of a d-variate blending function N_α is a d-dimensional rectangular *box*: $[\xi_1^1, \xi_5^1] \times \cdots \times [\xi_1^d, \xi_5^d]$. We are going to call *frame* of a function support the union of all $(d-2)$-faces of this *box* and we will denote it by $\mathrm{frm}(\mathrm{supp}\, N_\alpha)$. That is, for the rectangular support of a bivariate function, the *frame* is the union of the four vertices of this rectangle. For the cuboidal support of a trivariate function, its *frame* is composed of the union of the twelve edges of this cuboid.

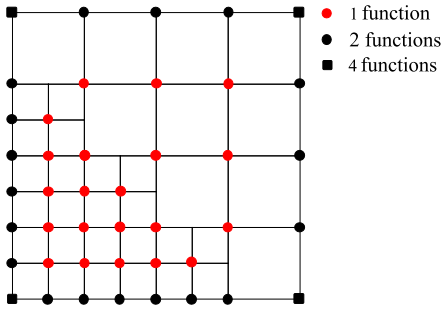


Figure 5: An example of T-mesh with its anchors. Red circles represent the interior nodes that have one blending function associated to them, black circles are the boundary nodes that have 2 blending functions and the black squares are the boundary nodes with 4 blending functions due to the open knot vector structure along the boundary.

Once the function supports are inferred, we modify them in such a way that, for each blending function N_α, its knot vectors Ξ^j_α $j = 1, ..., d$ verify the following simple conditions:

Condition 1: *Local knot vectors of the d-variate function* N_α *verify*[1]

$$\Delta^j_1 \geqslant \Delta^j_2 = \Delta^j_3 \leqslant \Delta^j_4, \quad j = 1, ..., d, \tag{1}$$

Condition 2: *The frame of the function support should be situated over the mesh skeleton*:

$$\mathrm{frm}(\mathrm{supp}\, N_\alpha) \in \mathrm{skt}(T). \tag{2}$$

Thus, the function supports that do not meet Conditions 1 or 2 should be modified. To perform this modification we extend the original support by changing some knot intervals until the resulting support satisfies both conditions. We are going to refer to these support modifications as *Extension rule* 1 and 2, respectevly.

In the next section we give a detailed description of this procedure for 2D and 3D cases.

[1]Except the cases involving repeated knots that are explained at the end of Section 4.1.

4. Support modification

Here, we present simple support Extension rules 1 and 2 to obtain local knot vectors that fulfill Conditions 1 and 2 formulated in the previous section. We proceed as follows. First, if after traversing the T-mesh *skeleton* the local knot vectors of a function do not satisfy Condition 1, we modify some of their knots in order to meet Condition 1. Then, the fulfillment of Condition 2 is checked and, if it is not satisfied, another appropriate modification of the support is carried out. As a result of these modifications we obtain a new extended support with local knot vectors which verify both conditions. These modifications are easily implemented taking into account the balanced tree structure of the mesh. Let see in detail this procedure.

To clarify the notation, in the rest of the paper we denote the parametric coordinates as (ξ, η, ζ) and it is related to the previous notation as $(\xi^1, \xi^2, \xi^3) = (\xi, \eta, \zeta)$. Consequently, $(\Xi^1, \Xi^2, \Xi^3) = (\Xi, \mathcal{H}, \mathcal{Z})$ and $(\Delta_i^1, \Delta_i^2, \Delta_i^3) = (\Delta_i^\xi, \Delta_i^\eta, \Delta_i^\zeta)$.

4.1. Support extension for 2D meshes

In order to facilitate the description and illustration of the strategy, some concepts and notation introduced in section 3 have to be particularized to the 2D case. The *skeleton* $\mathrm{skt}(T)$ of a two-dimensional mesh T is the union of all edges of the mesh. For a bivariate function let us denote the vertices of its rectangular support as $V_{1,1} = (\xi_1, \eta_1)$, $V_{5,1} = (\xi_5, \eta_1)$, $V_{5,5} = (\xi_5, \eta_5)$ and $V_{1,5} = (\xi_1, \eta_5)$. Then, the *frame* of a function support is the union of its four vertices, i.e., $\mathrm{frm}(\mathrm{supp}\, N_\alpha) = \{V_{n,m},\ n, m \in \{1, 5\}\}$. Figure 6 illustrates the introduced notation for a bivariate function support.

Formulation of Condition 1 for the local knot vectors Ξ and $\mathcal{H}$ is simple and does not need any clarification. Condition 2 adapted to 2D case is formulated as follows: *The four vertices of a function support should be situated over the mesh edges.*

Extension rule 1. If the local knot vector Ξ of a function does not satisfy Condition 1, we modify this vector by skipping over the minimal number of knots until $\Delta_1^\xi \geqslant \Delta_2^\xi = \Delta_3^\xi \leqslant \Delta_4^\xi$ is verified, and analogously, for $\mathcal{H}$. This modification is made independently for each parametric direction applying certain extension rules. Let see an example of support extension for a bivariate function. Leftmost function support shown in Fig. 7(a) does not meet Condition 1. For the knot vector Ξ we have $\Delta_3^\xi > \Delta_4^\xi$, so the knot interval Δ_4^ξ should be modified. Let us denote $h = \max(\Delta_2^\xi, \Delta_3^\xi) = \max(\Delta_2^\eta, \Delta_3^\eta)$.

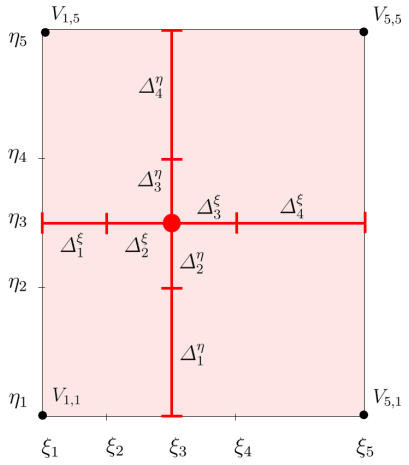


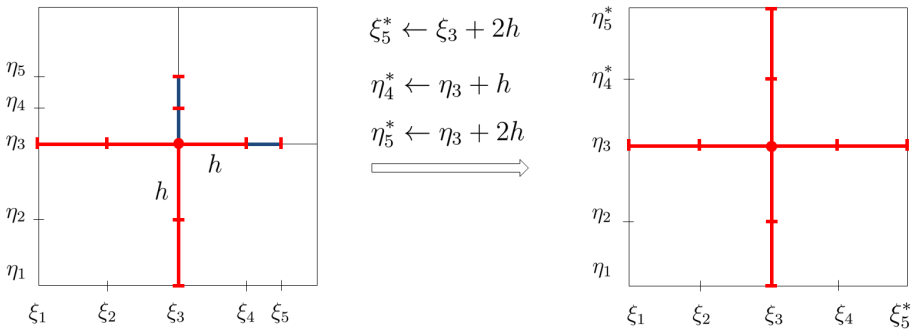
Figure 6: Support notation for 2D case.

Note that both maxima coincide due to the quadtree structure and the fact that the T-junctions are skipped. Then, the fifth knot ξ_5 is redefined as $\xi_5^* \leftarrow \xi_3 + 2h$. For the local knot vector $\mathcal{H}$ we have $\Delta_2^\eta > \Delta_3^\eta$, so the knots η_4 and η_5 should be modified as $\eta_4^* \leftarrow \eta_3 + h$, $\eta_5^* \leftarrow \eta_3 + 2h$.

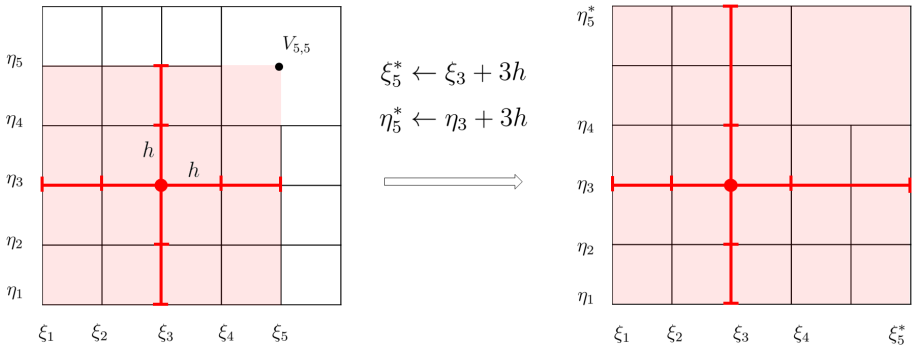
Extension rule 2. Once Condition 1 is satisfied, in order to fulfill Condition 2, we check whether the vertices of the function support are situated over the mesh edges. If not, we modify the knot vectors by skipping over a knot for both parametric directions and placing this vertex over the mesh edges. Figure 7(b) illustrates this procedure. The checking is performed independently for each of the four quadrants of the function support. Note that for our *0-balanced* quadtree we should make this checking only for some functions. For example, without loss of generality, the support vertex $V_{5,5} = (\xi_5, \eta_5)$ must be checked only if $\Delta_3^\xi = \Delta_4^\xi = \Delta_3^\eta = \Delta_4^\eta$. An example of a function support violating Condition 2 is illustrated in Fig. 7(b). The vertex $V_{5,5} = (\xi_5, \eta_5)$ of this support is not situated over a mesh edge, so the fifth knots for both parametric directions should be redefined as $\xi_5^* \leftarrow \xi_3 + 3h$, $\eta_5^* \leftarrow \eta_3 + 3h$, and thus, the new vertex $V_{5,5}$ is placed over the mesh edges.

The extension of any other function support is completely analogous to these two examples. In all possible cases, the extension of a function support implies to change one or two knot intervals by duplicating its size.

Detailed algorithms for Extension rule 1 and 2 used to modify a bivariate function support according to Conditions 1 and 2 are given in Algorithms 1



(a) Condition 1 is not satisfied because $\Delta_4^\xi < \Delta_3^\xi$ and $\Delta_3^\eta < \Delta_2^\eta$.



(b) Condition 2 is not satisfied because $V_{5,5} \notin skt(T)$.

Figure 7: Extension rules. (a) An example of support modification by Extension rule 1. (b) An example of support modification by Extension rule 2.

and 2.

Figure 8 shows some examples of support modification. Functions that satisfy both conditions and should not be modified are given in Fig. 8(a). Examples of support extension according to Condition 1 are shown in Fig. 8(b), (c), (d) and (e). And Fig. 8(f), (g) and (h) illustrate support extension according to Condition 2 or both.

Note that an exception for Condition 1 is a knot vector that contains a knot interval of length 0 due to the open knot vector structure along the boundary. In this case, a knot vector should fulfill the inequality (1) not taking into account the knot intervals of length 0. Consequently, an exception for the application of the extension rules is the case when the boundary of the parametric domain is reached traversing the T-mesh edges, see Fig. 8(c),

(d) and (e).

Remark 1. *It is important to highlight that the application of the extension rules always places the new knots over the mesh edges, i.e., the extension rules just skip over some knots, but do not create knots that are not induced by the T-mesh.*

Remark 2. *Note that in the case when a function support violates Condition 2, the extension of this support in order to place its vertex over the mesh skeleton can be made by modifying only one of the knot vectors instead of both. This option leads to the loss of symmetry for the spline space and to the non-uniqueness of the resulting function support. So, for simplicity, we extend the knot vectors in both parametric directions.*

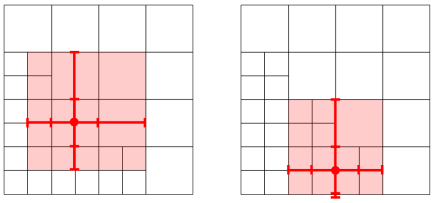
Algorithm 1: Extension rule 1.

Input: A knot vector $\Xi = (\xi_1, \xi_2, \xi_3, \xi_4, \xi_5)$, $\xi_i \in [0, 1]$.

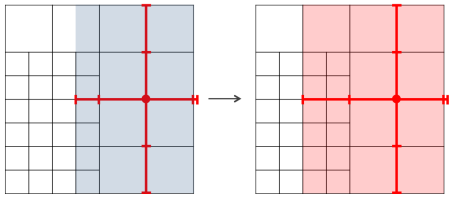
Function ***Modify1***(Ξ)

- $\Xi^* \leftarrow \Xi$
- $h = \max(\Delta_2, \Delta_3)$
- **if** $\Delta_2 < \Delta_3$ *and* $\xi_2 > 0$ **then**
  - $\xi_2^* \leftarrow \xi_3 - h$
  - $\xi_1^* \leftarrow \xi_2^*$
  - **if** $\xi_1^* > 0$ **then** $\xi_1^* \leftarrow \xi_3 - 2h$
- **if** $\Delta_1 < \Delta_2$ *and* $\xi_1 > 0$ **then**
  - $\xi_1^* \leftarrow \xi_3 - 2h$
- **if** $\Delta_2 > \Delta_3$ *and* $\xi_4 < 1$ **then**
  - $\xi_4^* \leftarrow \xi_3 + h$
  - $\xi_5^* \leftarrow \xi_4^*$
  - **if** $\xi_5^* < 1$ **then** $\xi_5^* \leftarrow \xi_3 + 2h$
- **if** $\Delta_4 < \Delta_3$ *and* $\xi_5 < 1$ **then**
  - $\xi_5^* \leftarrow \xi_3 + 2h$
- **return** Ξ^*

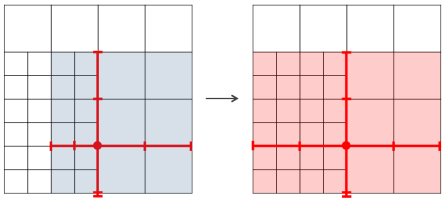
Output: A corrected knot vector Ξ^* that satisfies Condition 1.



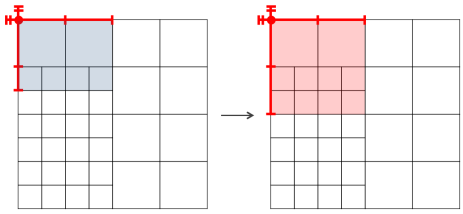
(a) Initial supports that satisfy both conditions and should not be modified.



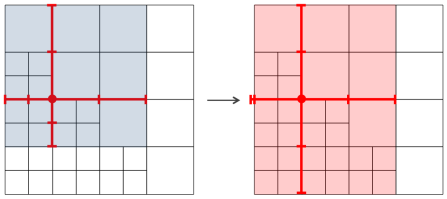
(b) Condition 1 is not satisfied because $\Delta_1^\xi < \Delta_2^\xi$, so $\xi_1^* \leftarrow \xi_3 - 2h$.



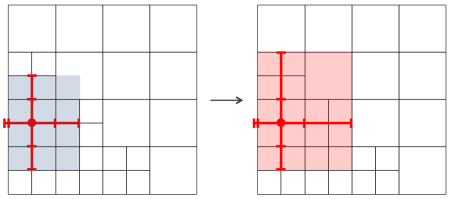
(c) Condition 1 is not satisfied because $\Delta_2^\xi < \Delta_3^\xi$, so $\xi_1^* \leftarrow \xi_3 - 2h$ and $\xi_2^* \leftarrow \xi_3 - h$.



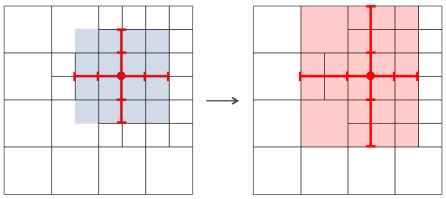
(d) Condition 1 is not satisfied because $\Delta_1^\eta < \Delta_2^\eta$, so $\eta_1^* \leftarrow \eta_3 - 2h$.



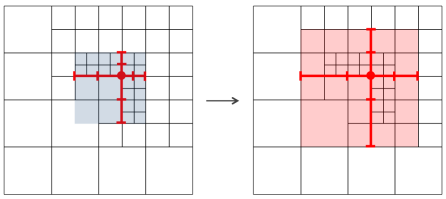
(e) Condition 1 is not satisfied because $\Delta_2^\xi < \Delta_3^\xi$ and $\Delta_2^\eta < \Delta_3^\eta$, so $\xi_2^* \leftarrow \xi_3 - h$, $\eta_2^* \leftarrow \eta_3 - h$ and $\eta_1^* \leftarrow \eta_3 - 2h$.



(f) Condition 2 is not satisfied because $V_{5,5} \notin skt(T)$, so $\xi_5^* \leftarrow \xi_3 + 3h$ and $\eta_5^* \leftarrow \eta_3 + 3h$.



(g) Condition 2 is not satisfied because $V_{1,1}$ and $V_{1,5} \notin skt(T)$, so $\xi_1^* \leftarrow \xi_3 - 3h$, $\eta_1^* \leftarrow \eta_3 - 3h$ and $\eta_5^* \leftarrow \eta_3 + 3h$.



(h) Conditions 1 and 2 are not satisfied because $\Delta_2^\xi > \Delta_3^\xi$, $\Delta_2^\eta > \Delta_3^\eta$ and $V_{1,1} \notin skt(T)$, so $\xi_4^* \leftarrow \xi_3 + h$, $\xi_5^* \leftarrow \xi_3 + 2h$, $\eta_4^* \leftarrow \eta_3 + h$, $\eta_5^* \leftarrow \eta_3 + 2h$, $\xi_1^* \leftarrow \xi_3 - 3h$, $\eta_1^* \leftarrow \eta_3 - 3h$.

Figure 8: Examples of function support modification with Extension rule 1 and 2. Initial support is marked in blue and extended support in red.

Algorithm 2: Extension rule 2 in 2D.

Input: A *0-balanced* mesh T and a pair of local knot vectors $S = \{\Xi, \mathcal{H}\}$.

Function ***Modify2***$(T,\ S)$
- $S^* \leftarrow S$
- $h = \max(\Delta_2^\xi, \Delta_3^\xi)$
- **for** $n \in \{1, 5\}$ **do**
  - **for** $m \in \{1, 5\}$ **do**
    - **if** $(\xi_n, \eta_m) \notin skt(T)$ **then**
      - $\xi_n^* \leftarrow\ \xi_3 + 3h\,\mathrm{sgn}(\xi_n - \xi_3)$
      - $\eta_m^* \leftarrow\ \eta_3 + 3h\,\mathrm{sgn}(\eta_m - \eta_3)$
- **return** S^*

Output: A modified support $S^* = \{\Xi^*, \mathcal{H}^*\}$ that satisfies Condition 2.

4.2. Support extension for 3D meshes

In this section we give a description and illustration of the proposed strategy for defining trivariate spline functions over *0-balanced* octree T-meshes.

The *skeleton* $\mathrm{skt}(T)$ of a three-dimensional mesh T is the union of all faces of the mesh. For a trivariate function let us denote the vertices of its support as $V_{n,m,k} = (\xi_n, \eta_m, \zeta_k)$ where $n, m, k \in \{1, 5\}$. And the edge of a support formed by the two vertices $V_{n,m,k}$ and $V_{p,q,r}$ is denoted as $E_{(n,m,k),(p,q,r)}$. Then, the *frame* $\mathrm{frm}(\mathrm{supp}\, N_\alpha)$ of a trivariate function support is the union of its twelve edges. Figure 9 illustrates the introduced notation for the support of a trivariate blending function.

The formulation of Condition 1 for the local knot vectors of a trivariate function is analogue to the 2D case. Condition 2 adapted to 3D meshes is stated as follows: *Edges of the cuboidal function support should be situated over the mesh faces.*

The implementation of the strategy for 3D is similar to the 2D case. To satisfy Condition 1, Extension rule 1 is applied to each of the three local knot vectors of a function analogously to the 2D case using Algorithm 1.

Extension rule 2. In order to fulfill Condition 2, we check whether the edges of a function support are situated over the mesh faces. If not, the two knot vectors perpendicular to this edge should be modified by skipping over a knot for both parametric directions and placing this edge over the mesh faces. This checking is performed independently for each of the eight quadrants of the function support and, in each quadrant, three edges should

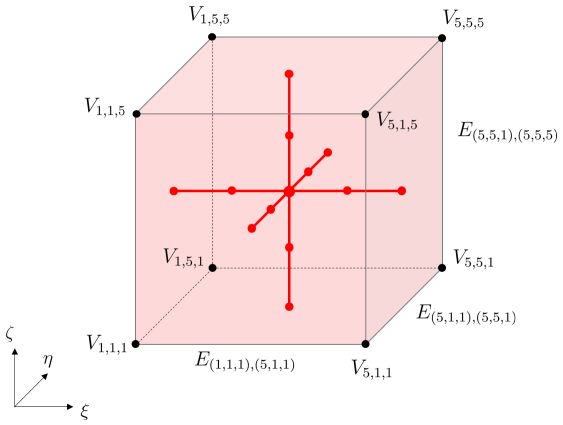


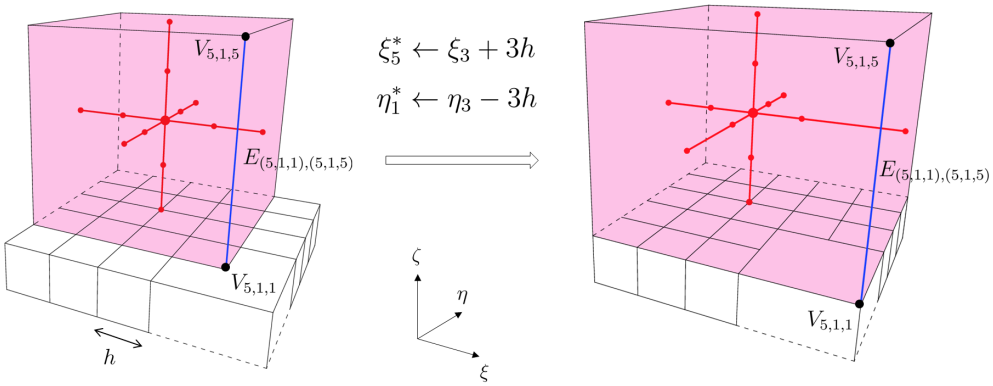
Figure 9: Support notation for 3D case.

be checked. Figure 10 illustrates the support extension procedure for the quadrant of the vertex $V_{5,1,1}$. Due to the octree structure only two cases can take place: (i) the quadrant contains one edge that is not situated over the mesh faces or (ii) the quadrant contains three edges and a vertex that are not situated over the mesh *skeleton*. Now we study each case.

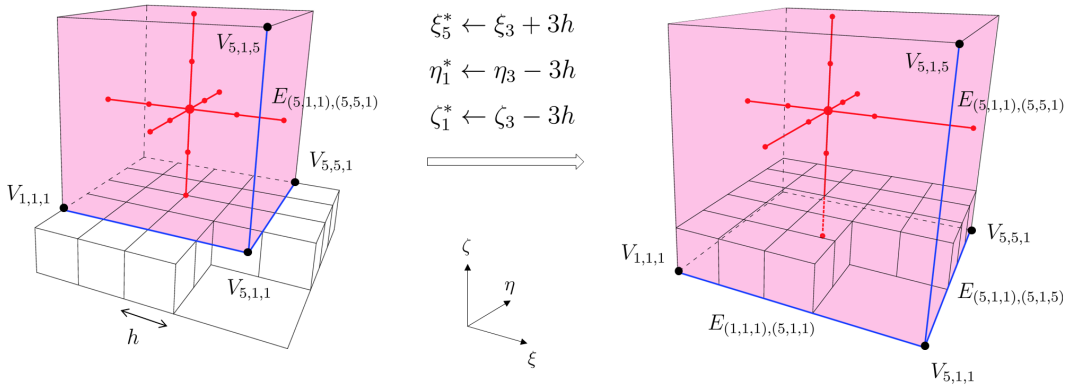
(i) If a quadrant contains one edge that does not fulfill Condition 2, then two knot vectors perpendicular to this edge are modified, see Fig. 10(a). For the function support shown in Fig. 10(a) left, the edge $E_{(5,1,1),(5,1,5)}$ is not situated over the mesh faces. Therefore, its two knot vectors Ξ and $\mathcal{H}$ perpendicular to this edge are modified in order to place the edge over the mesh faces, namely, knots ξ_5 and η_1 are redefined as $\xi_5^* \leftarrow \xi_3 + 3h$ and $\eta_1^* \leftarrow \eta_3 - 3h$, where $h = \max(\Delta_2^\xi, \Delta_3^\xi) = \max(\Delta_2^\eta, \Delta_3^\eta) = \max(\Delta_2^\zeta, \Delta_3^\zeta)$.

(ii) If a quadrant contains three edges that are not situated over the mesh faces, then the three knot vectors are modified by skipping over a knot for each of the three parametric directions, see Fig. 10(b). Vertex $V_{5,1,1}$ and the three edges connected to it are not situated over the mesh *skeleton*, so all the three knot vectors are modified to place the three edges over the mesh faces: $\xi_5^* \leftarrow \xi_3 + 3h, \quad \eta_1^* \leftarrow \eta_3 - 3h, \quad \zeta_1^* \leftarrow \zeta_3 - 3h.$

Algorithm 3 explains the Extension rule 2 used to modify a trivariate function support according to Condition 2.



(a) Only one edge violating Condition 2. Node $V_{5,1,1}$ is sited in the center of the face of size $2h$. The support is extended in two directions.



(b) Three edges violating Condition 2. Node $V_{5,1,1}$ is sited in the center of the cell of size $2h$. The support is extended in three directions.

Figure 10: Extension rule 2 for support modification of a trivariate function.

Algorithm 3: Extension rule 2 in 3D.

Input: A *0-balanced* T-mesh T and three local knot vectors $S = \{\Xi, \mathcal{H}, \mathcal{Z}\}$.

Function ***Modify2*** $(T,\ S)$

- $S^* \leftarrow S$
- $h = \max(\Delta_2^{\xi}, \Delta_3^{\xi})$
- $mov(i) := i + 4\,\mathrm{sgn}(3 - i)$
- **for** $n \in \{1, 5\}$ **do**
  - **for** $m \in \{1, 5\}$ **do**
    - **for** $k \in \{1, 5\}$ **do**
      - **if** $E_{(n,m,k),(mov(n),m,k)} \notin skt(T)$ **then**
        - $\eta_m^* \leftarrow \eta_3 + 3h\,\mathrm{sgn}(\eta_m - \eta_3)$
        - $\zeta_k^* \leftarrow \zeta_3 + 3h\,\mathrm{sgn}(\zeta_k - \zeta_3)$
      - **if** $E_{(n,m,k),(n,mov(m),k)} \notin skt(T)$ **then**
        - $\xi_n^* \leftarrow \xi_3 + 3h\,\mathrm{sgn}(\xi_n - \xi_3)$
        - $\zeta_k^* \leftarrow \zeta_3 + 3h\,\mathrm{sgn}(\zeta_k - \zeta_3)$
      - **if** $E_{(n,m,k),(n,m,mov(k))} \notin skt(T)$ **then**
        - $\xi_n^* \leftarrow \xi_3 + 3h\,\mathrm{sgn}(\xi_n - \xi_3)$
        - $\eta_m^* \leftarrow \eta_3 + 3h\,\mathrm{sgn}(\eta_m - \eta_3)$
- **return** S^*

Output: A modified support $S^* = \{\Xi^*, \mathcal{H}^*, \mathcal{Z}^*\}$ that satisfies Condition 2.

5. Properties of our spline functions, support overlapping, sparsity and condition number

Here, we summarize the properties and some characteristics of the spline spaces constructed by means of our method.

5.1. Properties

For any *0-balanced* mesh T, the set of blending functions defined according to our strategy spans the space $S_T = \text{span}\{N_\alpha : \alpha \in A_T\}$ with the following properties:

1. *Functions* $\{N_\alpha\}_{\alpha \in A_T}$ *are* C^2*-continuous.*
2. *Functions* $\{N_\alpha\}_{\alpha \in A_T}$ *are linearly independent.*
3. *Spaces spanned by nested T-meshes are also nested:* $T_1 \subset T_2 \;\Rightarrow\; S_{T_1} \subset S_{T_2}$.
4. *Polynomial reproduction property:* $\mathbb{P}_3(\Omega) \in S_T$.

A rigorous proof of these properties is currently under preparation and we plan to give it in a future work. However, for a better understanding of the strategy, some clues should be mentioned.

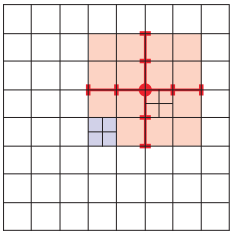
For the proof of linear independence of our blending functions it is essential to take into account that we do not repeat knots in the interior of the domain, and each regular node of the T-mesh has only one function assigned to it. Some reasoning from [18] can be used to proof the linear independence of such functions.

A brief outline of the proof of the property (3) is given in the appendix of the present paper.

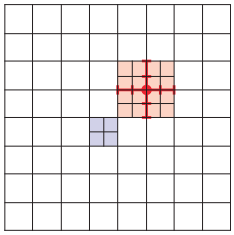
It is worth mentioning that we have carried out numerous numerical experiments and the claimed properties were verified in all of them.

5.2. Support overlapping

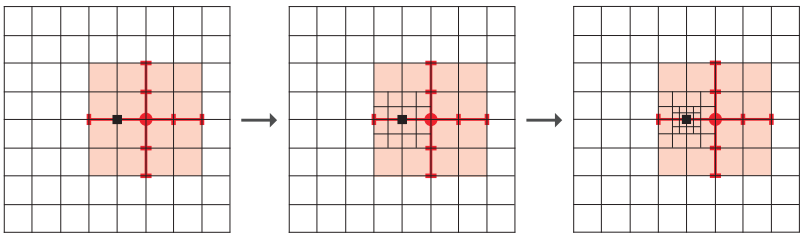
It should be pointed out that the proposed strategy can lead, in some cases, to increased overlapping of the function supports of different refinement levels, which can affect the conditioning and sparsity of the stiffness matrix. This effect can take place in some problems with a very sharp singularity when the area marked to refine by the error indicator at each iteration is smaller than the area occupied by the supports of the previous level. According to our strategy, a cell refinement always adds at least one new blending



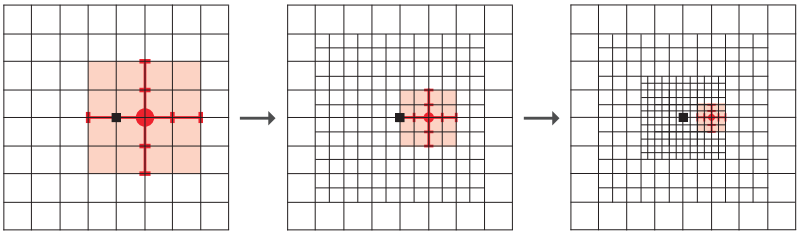
(a) A function support (red) that intersects the refined cell (shaded cell).



(b) New function support after applying additional refinements.



(c) Three refinement steps around a singular point (black square) that lead to an excessive overlapping of the function supports. Note that the red support does not change.



(d) The same three refinement steps applying additional refinements.

Figure 11: Additional refinements in order to avoid excessive overlapping of the function supports.

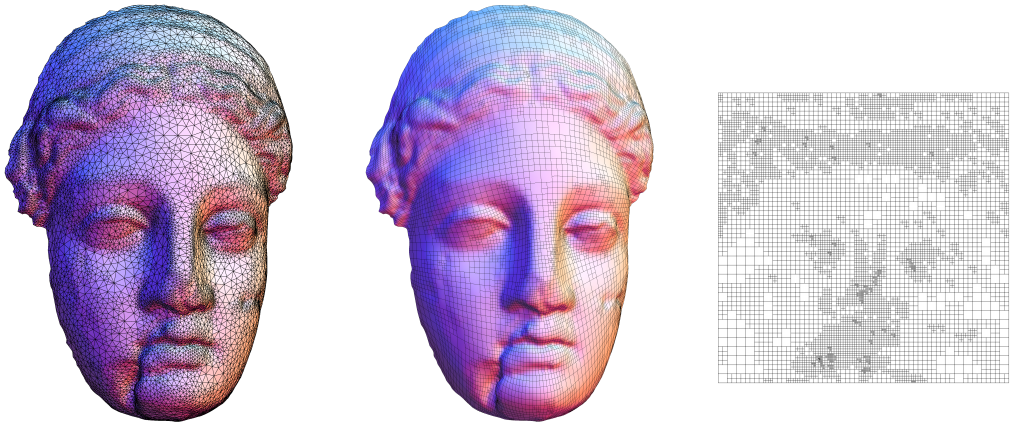
function, while a function of the coarse level is replaced by the finer ones only if all the cells that share their anchor are refined. To avoid a possible accumulation of the functions of different refinement levels, the following strategy can be adopted: if a function support intersects a cell marked to be refined, then the coarsest cells sharing the anchor of the function are refined, see Fig. 11(a) and (b).

Application of this strategy is illustrated in Fig. 11(c) and (d). In this example, let us suppose that the problem has a singularity at the center of the domain, and at each iteration, we refine only the four cells adjacent to the singularity point. Then, at each iteration there are functions whose support is not changed, see Fig. 11(c). However, if at each iteration we perform additional refinements, functions of the coarser level are replaced by the finer ones, see Fig. 11(d). This approach is similar to the one used in hierarchical refinement where all supports that intersect the marked cell are also refined in order to guarantee the replacing of the supports of the previous level by the finer ones at each refinement step. Application of this additional refinement, where needed, can reduce an excessive function overlapping and improve the stiffness matrix conditioning and sparsity. However, due to the unnecessary extension of the refined zone, the optimal rate of convergence can be lost. In practice, we did not observe a significant advantage of this approach, and taking into account its computational cost, we do not apply this strategy in the computational examples presented in this paper. Moreover, it was observed that for a large variety of problems, the excessive support overlapping is avoided naturally after various refinement steps due to the behaviour of the error indicator.

On the other hand, another possible solution for this problem is a more accurate and selective definition of the function supports in order to obtain a space with better support locality. The strategy we propose in this paper can be improved by including more sophisticated rules for inferring function supports, but that would probably lead to a more expensive implementation.

6. Computational examples

In this section we present computational examples of the application of our technique in geometric design and analysis for problems involving adaptive refinement. The proposed method is tested by performing surface interpolation and resolution of 2D and 3D Poisson problems using IGA.



(a) (b) (c)

Figure 12: Igea's face. Spline representation of the surface from its triangulation. (a) Input surface triangulation; (b) spline representation of the surface; (c) parametric T-mesh.

6.1. Spline representation of the surface from its triangulation

Here we construct a spline representation of a surface given by a triangulation. First, a global parameterization of the surface triangulation is obtained by means of the method proposed by M. Floater in [19]. As a result, we have one-to-one mapping from planar triangulation of the parametric domain (unit square) to the surface triangulation. Then, a quadtree T-mesh adapted to the planar triangulation is constructed. For this purpose we have chosen the following simple criterion. We start from a coarse T-mesh and refine it until each cell of the mesh contains no more than a certain number of points of the input triangulation (3 in our case). The spline approximation of the surface is built as a linear combination of our blending functions

$$S(\boldsymbol{\xi}) = \sum_{\alpha \in A_T} C_\alpha \, N_\alpha \left(\boldsymbol{\xi}\right).$$

The control points C_α are found by imposing the interpolation conditions

$$\boldsymbol{x}_\beta = \sum_{\alpha \in A_T} C_\alpha \, N_\alpha \left(\boldsymbol{\xi}_\beta\right), \qquad \forall \boldsymbol{\xi}_\beta, \ \beta \in A_T,$$

where $\boldsymbol{\xi}_\beta$ are interpolation points in the parametric domain and $\boldsymbol{x}_\beta$ are their images in the physical space determined by the triangular parameterization. As interpolation points we use the anchors of the functions, i.e., the *regular nodes* of the T-mesh, and some additional interpolation points associated to functions whose local knot vectors contain repeated knots on the boundary. These additional points are situated at the midpoint of the edges that have contact with the boundary and coincide approximately with the maximum point of the corresponding blending functions. See [20] for more details.

The resulting spline surface, parametric T-mesh and input surface triangulation are shown in Fig. 12.

6.2. Adaptive refinement for surface interpolation

In this example we interpolate the function

$$u(r) = r^{\frac{1}{2}}, \tag{3}$$

defined on the square domain $[0,1]^2$, being $r = \sqrt{(x-0.5)^2 + (y-0.5)^2}$.

The spline approximation of the surface (3) is built as a linear combination of our blending functions

$$u_h(\boldsymbol{\xi}) = \sum_{\alpha \in A_T} C_\alpha \, N_\alpha \left(\boldsymbol{\xi}\right).$$

And the control points C_α are found by imposing the interpolation conditions

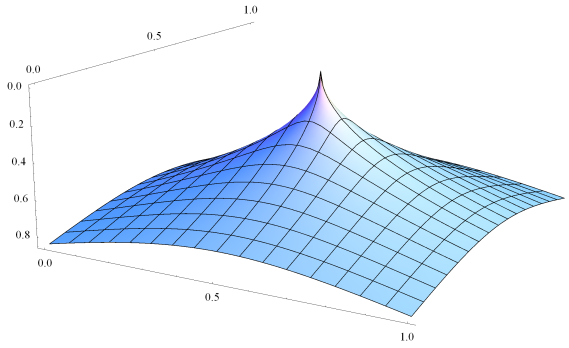
$$u(\boldsymbol{\xi}_\beta) = \sum_{\alpha \in A_T} C_\alpha \, N_\alpha \left(\boldsymbol{\xi}_\beta\right), \qquad \forall \boldsymbol{\xi}_\beta, \ \beta \in A_T,$$

Adaptive refinement is performed according to the indicator based on exact L^2 interpolation error:

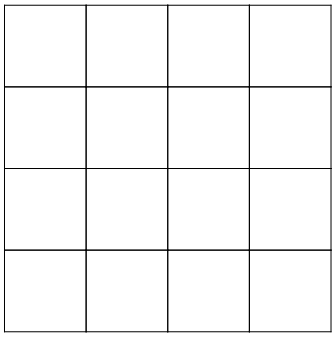
$$\eta(\Omega_e)^2 = \|u - u_h\|^2_{L^2,\Omega_e} = \int_{\Omega_e} (u - u_h)^2 \, \mathrm{d}\Omega.$$

A cell Ω_e is marked to be refined if $\eta(\Omega_e) > \gamma \max_i \{\eta(\Omega_i)\}$, being $\gamma \in [0,1]$.

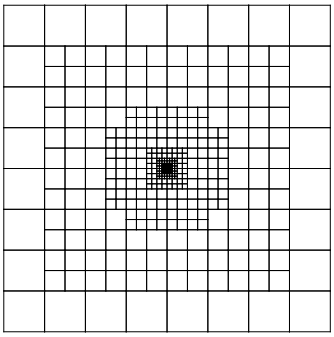
Some steps of adaptive refinement for the surface interpolation (3) are shown in Fig. 13, and Fig. 14 illustrates the convergence in L^2-norm and H^1-seminorm.



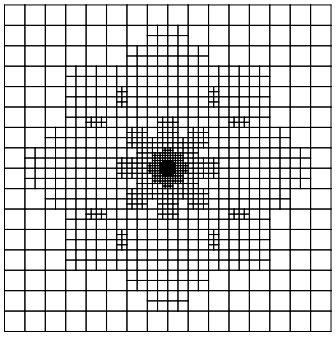
(a)



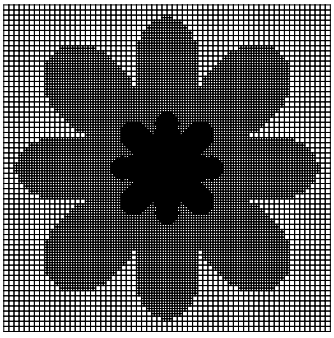
(b)



(c)



(d)



(e)

Figure 13: Adaptive refinement for surface interpolation (3). (a) Function to interpolate. (b) Initial mesh. (c), (d) and (e) Some steps of adaptive refinement.

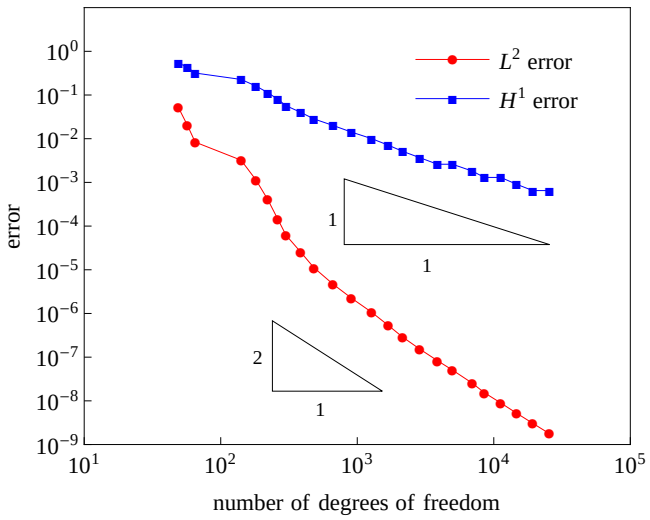


Figure 14: Convergence of the adaptive refinement for the interpolation example (3) in L^2-norm and H^1-seminorm, for $\gamma = 0.3$.

6.3. Poisson problem over a square domain

In this subsection we present an example of the resolution of a Poisson problem on a square domain $\Omega = [0, 1]^2$ using isogeometric analysis. Let us consider the problem

$$\begin{aligned} -\triangle u &= f \qquad \text{in } \Omega, \\ u &= g \qquad \text{on } \partial\Omega. \end{aligned} \tag{4}$$

Its variational formulation consists in finding $u \in V_g(\Omega)$ such that

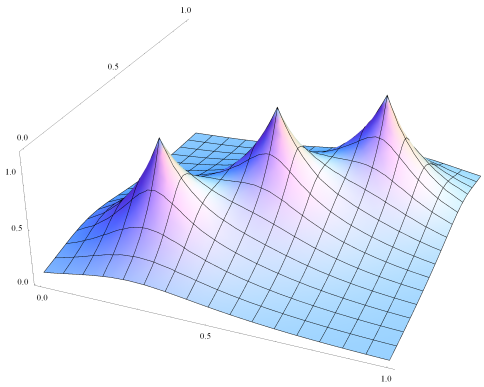
$$a(u, v) = (f, v) \quad \forall v \in V_0(\Omega),$$

where

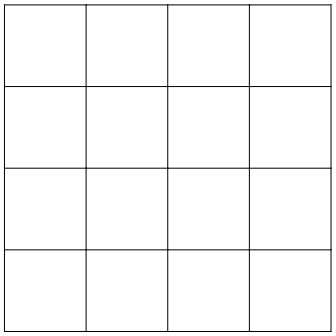
$$a(u, v) = \int_\Omega \nabla u \cdot \nabla v \, \mathrm{d}\Omega \quad \text{and} \quad (f, v) = \int_\Omega f \, v \, \mathrm{d}\Omega.$$

The test function space is

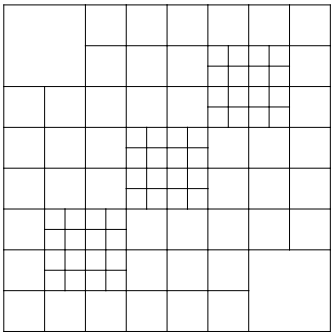
$$V_0(\Omega) = \{v \in H^1(\Omega) : v \,|_{\partial\Omega} = 0\},$$



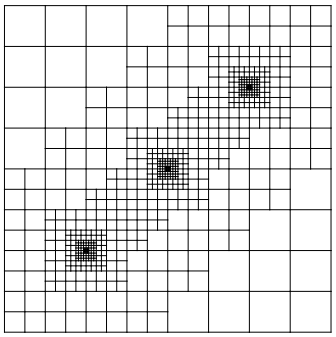
(a)



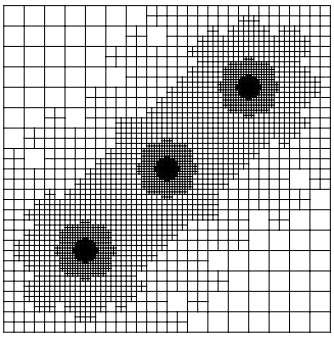
(b)



(c)

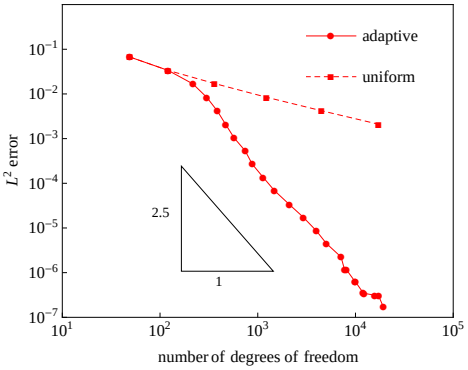


(d)

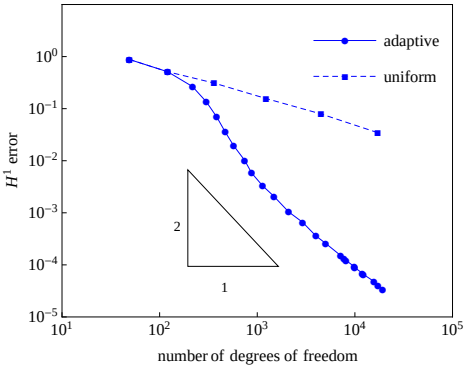


(e)

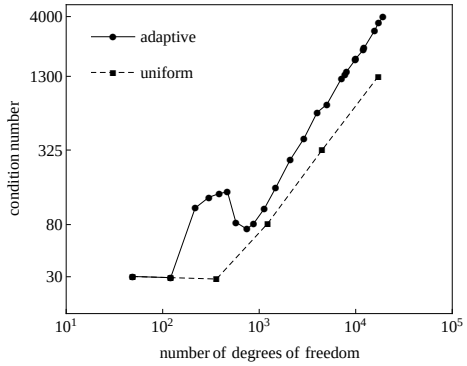
Figure 15: Resolution of the Poisson problem (4). (a) Numerical solution corresponding to the final refinement. (b) Initial mesh. (c), (d) and (e) Several steps of the adaptive process.



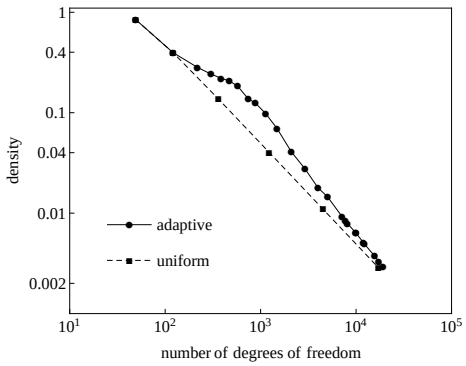
(a) L^2 error convergence.



(b) H^1 error convergence.



(c) Evolution of the condition number of the stiffness matrix.



(d) Evolution of the density of the stiffness matrix.

Figure 16: Convergence, condition number and density of the stiffness matrix for the Poisson problem (4), for $\gamma = 0.3$.

and the solution space is

$$V_g(\Omega) = \{v \in H^1(\Omega) : v \mid_{\partial\Omega} = g\}.$$

Let T be a T-mesh of Ω. Then we denote with $V_T(\Omega)$ the finite dimensional space spanned by the spline blending functions defined over T, and with $V_{g_T,T}(\Omega)$ the subspace of functions of $V_T(\Omega)$ that are equal to g_T at the boundary, where g_T is an interpolant of g.

The isogeometric approximation consists in finding $u_h \in V_{g_T,T}(\Omega)$ such that

$$a(u_h, v_h) = (f, v_h) \quad \forall v_h \in V_{0,T}(\Omega).$$

The problem (4) is set up in such a way that its analytical solution is a function with singularities taken from [12] and given by

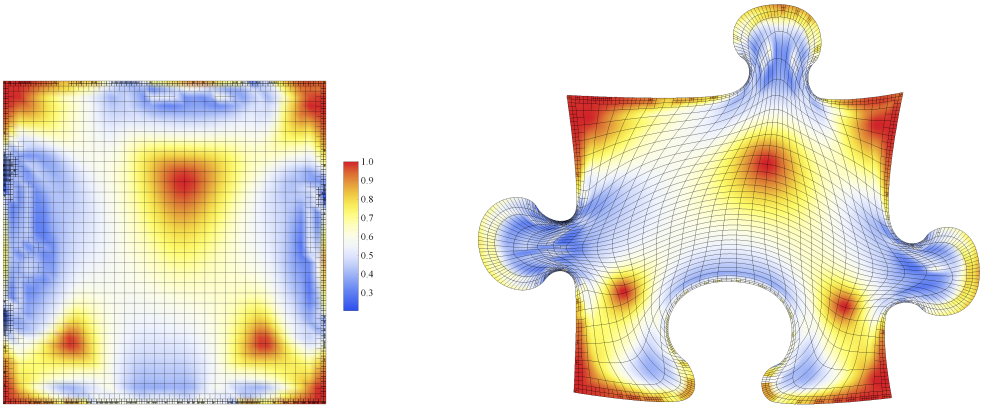
$$\begin{aligned} u(x,y) = &\exp\left(-7\sqrt{(x-0.5)^2+(y-0.5)^2}\right) + \\ &+\exp\left(-7\sqrt{(x-0.25)^2+(y-0.25)^2}\right) + \\ &+\exp\left(-7\sqrt{(x-0.75)^2+(y-0.75)^2}\right). \end{aligned} \tag{5}$$

We perform an adaptive refinement based on an a posteriori error indicator to improve the quality of the numerical solution. We have chosen a simple residual-type error estimator given by

$$\eta(\Omega_e)^2 = \|h\,(f + \Delta u_h)\,\|^2_{0,\Omega_e} = \int_{\Omega_e} h^2\,(f + \Delta u_h)^2\ \mathrm{d}\Omega,$$

where h is the diameter of the cell Ω_e. The estimator is jump free through the cell interfaces because of the smoothness of the isogeometric approximation. A cell Ω_e is marked to be refined if $\eta(\Omega_e) > \gamma\ \max_i\{\eta(\Omega_i)\}$, being $\gamma \in [0,1]$.

During the assembly process we accomplish the numerical integration on each element (cell) of the mesh, therefore it should be taken into account that for exact integration results (when possible) the numerical integration should be performed on each Bezier element, that is, on the subregion of the cell where the blending functions are pure polynomials. For our balanced quadtree meshes it implies to subdivide some cells in 2 or 4 subelements. However, taking into account the smoothness of the functions across the



(a) (b)

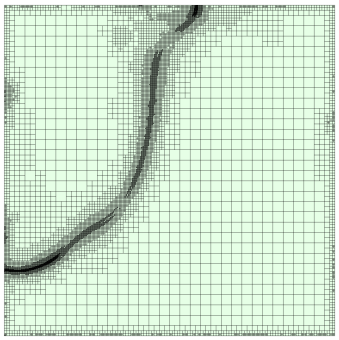
Figure 17: Parameterization of the computational domain for the problem of the section 6.4. (a) Colormap of the mean ratio Jacobian represented in the parametric domain. (b) Colormap of the mean ratio Jacobian represented in the physical domain.

subelements boundary, more efficient quadrature rules can be used. For example, in [21] authors propose a numerical procedure to compute weights and points of quadrature rules on macro elements. These rules are exact for blending functions and efficient in the sense that requires less evaluations than classical Gauss rules on each element.

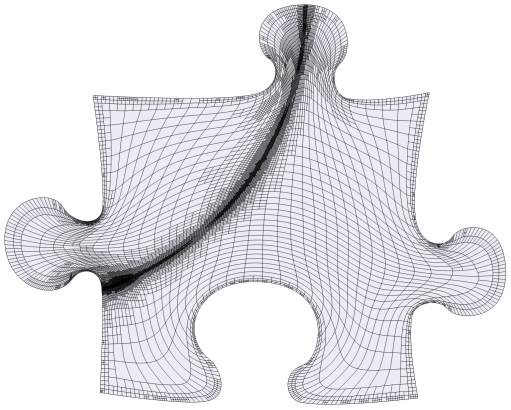
Some steps of adaptive refinement for the problem (4) and the numerical solution corresponding to the final refinement iteration are shown in Fig. 15. The convergence behaviour of the adaptive refinement in L^2-norm and H^1-seminorm is shown in Fig. 16(a) and (b). The evolution of the density of the stiffness matrix A, its condition number and the comparison of these values for an uniform refinement are represented in Fig. 16(c) and (d), respectively. The density of the matrix A is the fraction of non-zero elements and the condition number is defined as $\kappa(A) =\| A \|_2 \| A^{-1} \|_2$.

6.4. Poisson problem on a complex domain

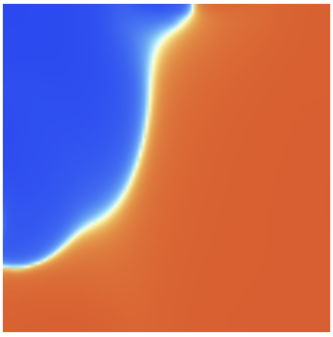
In the next example, we present the result of the resolution of a Poisson problem on a complex domain. We denote by $\widehat{\Omega} = [0, 1]^2$ the parametric domain and by Ω the physical or computational domain. Now the spline blending functions are defined over a mesh T of the parametric domain. Given Ω, we build a one to one parametric transformation $\mathbf{S}$, that maps $\widehat{\Omega}$



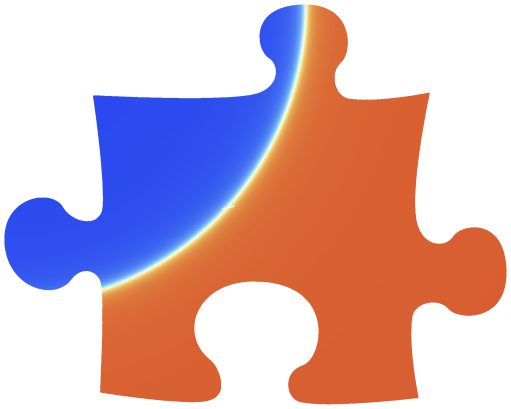
(a) Final refinement in the parametric domain.



(b) Final refinement in the physical domain.



(c) Numerical solution in the parametric domain.



(d) Numerical solution in the physical domain.

Figure 18: Results of the adaptive refinement for the Poisson problem of the section 6.4.

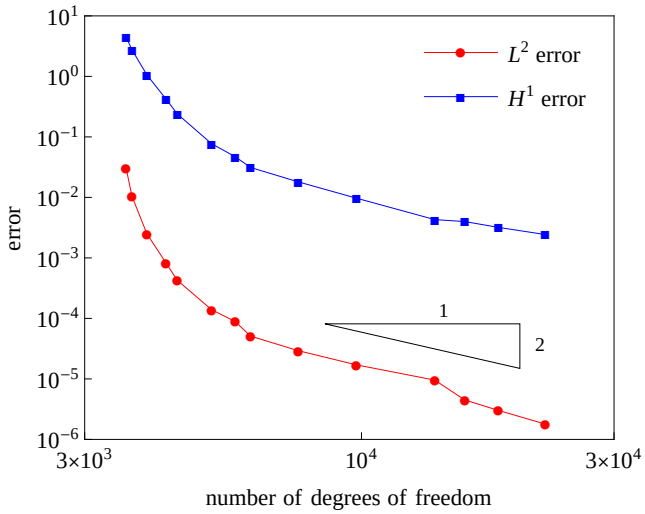


Figure 19: Convergence of L^2-norm and H^1-seminorm error for the Poisson problem of the section 6.4, for $\gamma = 0.3$.

to Ω, i.e $\mathbf{S} : \widehat{\Omega} \to \Omega$, using the algorithm described in our previous work [20]. This technique, based on a T-mesh untangling and optimization procedure, allows us to obtain a good quality parameterization suitable for application of IGA. The procedure is an extension of the ideas presented in our works [22, 23, 24, 25]. Another method for parameterization of computational domain, based on pillowing technique, was given in [26, 27].

The mean ratio Jacobian is used to evaluate the quality of the parameterization in the sense of its orthogonality and uniformity. Figures 17(a) and (b) show the colormap of the parameterization quality for our parametric and physical domains. It should be mentioned that for all numerical examples the parameterization of the computational domain is performed using the same blending functions that are used for the solution approximation, so isoparametric concepts holds during isogeometric analysis. That is, if $\mathbb{V}_T(\widehat{\Omega})$ is the finite dimensional space spanned by the spline functions associated to a parametric mesh T, then the parametric mapping $\mathbf{S}$ is constructed as a linear combination of functions from $\mathbb{V}_T(\widehat{\Omega})$ and the discrete approximation space $\mathbb{V}_T(\Omega)$ in physical domain is defined as follows:

$$\mathbb{V}_T(\Omega) = \left\{ v \in H^1(\Omega) \,:\, v = \widehat{v} \circ \mathbf{S}^{-1}, \text{ for all } \widehat{v} \in \mathbb{V}_T(\widehat{\Omega}) \right\}.$$

Now we consider a Poisson problem with Dirichlet boundary condition on Ω, which exact solution is a function with steep wave front given by

$$u(r) = \arctan(\alpha(r - r_0)),$$

where $r = \sqrt{(x - x_c)^2 + (y - y_c)^2}$, the parameter α determines the steepness of the wave front and r_0 is its location. In this example $\alpha = 200$ and $r_0 = 0.6$. The center of the wave front $(x_c, y_c) = (0, 0)$ is situated outside our computational domain, so the function is smooth in Ω. The numerical solution of the problem and the mesh corresponding to the final refinement iteration is shown in Fig. 18. As expected, the error estimator has marked for refinement the zone of the wave front. The evolution of the exact error in L^2-norm and H^1-seminorm are shown in Fig. 19.

6.5. Poisson problem on 3D domain

The next computational example is the resolution of a 3D Poisson problem using IGA. The computational domain is a spline approximation of a sphere portion, see Fig. 20. The initial uniform mesh was composed by $4 \times 4 \times 4$ cells. The approximation is constructed using our spline blending function. The Poisson problem with Dirichlet boundary condition is set up so that the analytical solution of the problem is

$$u(r) = \sin\left(\frac{1}{\alpha + r}\right),$$

where $r = \sqrt{x^2 + y^2}$ and parameter $\alpha = 1/10\pi$. This is a smooth function with an oscillation near the origin. Results of the final refinement iteration are shown in Fig. 20. The convergence behaviour of the adaptive refinement is illustrated in Fig. 21.

7. Conclusions and future research

In this paper we have proposed a strategy for defining C^2-continuous cubic tensor product spline functions over quadtree and octree T-meshes. We only demand these T-meshes to be *0-balanced* and this requirement can be easily satisfied by using a standard balancing procedure. The proposed strategy includes simple instructions used for inferring local knot vectors to define blending functions. We conjecture that the resulting spline spaces have nice properties: linear independence and the characteristic that spaces spanned

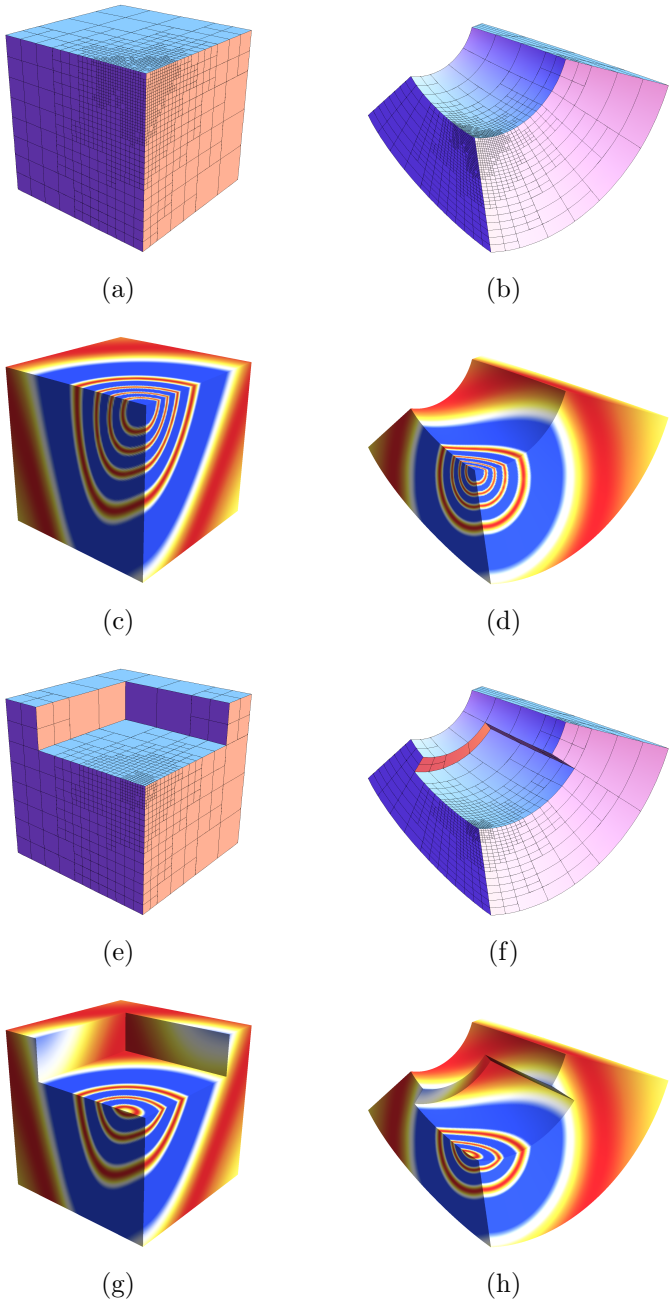


Figure 20: Mesh and numerical solution in the sixth adaptive refinement step for the 3D Poisson problem of the section 6.5. (a) Parametric mesh. (b) Physical mesh. (c) Numerical solution in the parametric domain. (d) Numerical solution in the physical domain. (e) A section of the parametric mesh. (f) A section of the physical mesh. (g) Numerical solution in a section of the parametric domain. (h) Numerical solution in a section of the physical domain.

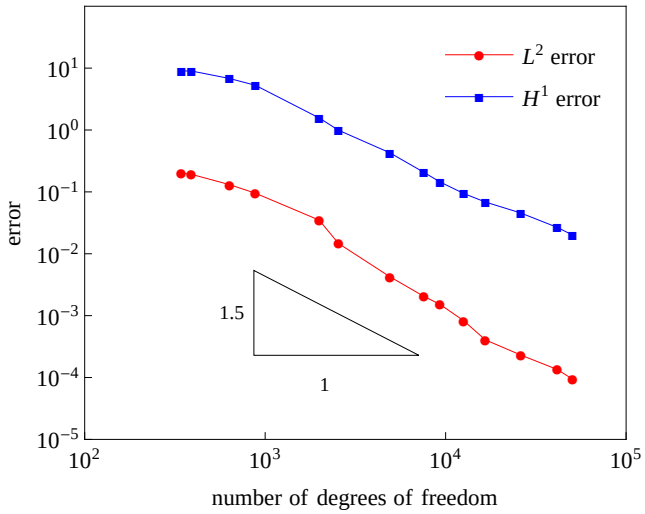


Figure 21: Convergence of L^2-norm and H^1-seminorm error for the Poisson problem of the section 6.5, for $\gamma = 0.3$.

by nested T-meshes are also nested. The above mentioned properties were verified in numerous numerical experiments and we plan to provide a rigorous proof in our next work.

The implementation of our technique is straightforward taking into account the balance mesh restriction. Examples of adaptive refinement using IGA for 2D and 3D Poisson problems have been presented. In all of them, optimal rates of convergence have been obtained. We believe that the simplicity of our technique can make it an attractive tool for its application in IGA and geometric design.

In future works we also plan to improve the strategy obtaining a better locality of the function supports.

Acknowledgements

This work has been supported by the Spanish Government, "Secretaría de Estado de Universidades e Investigación," "Ministerio de Economía y Competitividad,", "Programa de FPU del Ministerio de Educación, Cultura y Deporte", "Programa de FPI propio de la ULPGC" and FEDER, grant contracts: CGL2011-29396-C03-01 and CGL2011-29396-C03-03; "Junta Castilla León," grant contract: SA266A12-2. It has also been supported

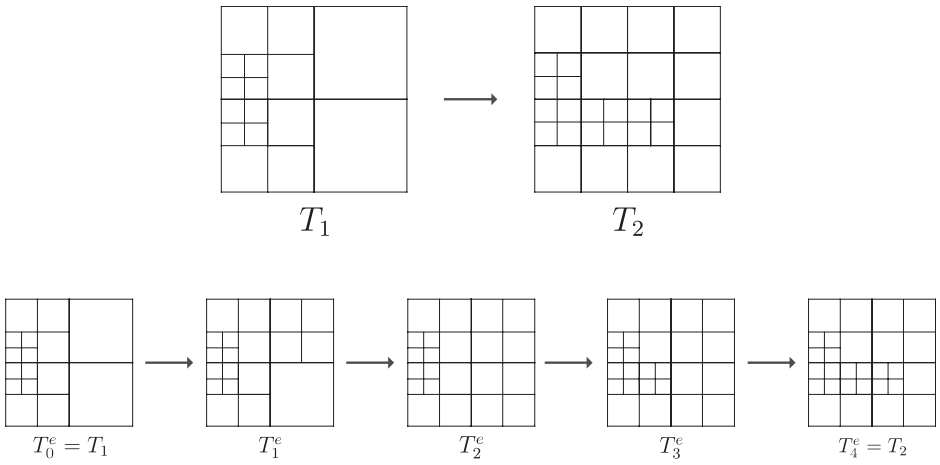


Figure 22: Elemental refinements. A possible sequence of 0-balanced meshes $\{T_i^e\}$, where T_i^e is obtained from T_{i-1}^e by refining only one cell.

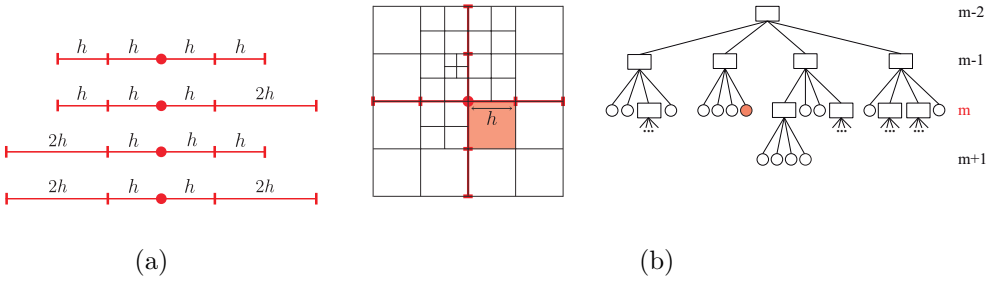
by CONACYT-SENER ("Fondo Sectorial CONACYT SENER HIDROCARBUROS," grant contract: 163723).

Appendix

Here we include an outline of the proof (2D case) of the nestedness of the spline spaces constructed via our strategy.

It is worth noting that the rules we propose and the 0-balanced restriction for the mesh lead to a very few number of all possible function supports that can be defined over a T-mesh. In order to proof the nestedness of the spaces for any two nested T-meshes $T_1 \subset T_2$, we should verify that any blending function of S_{T_1} can be represented as linear combination of blending functions from S_{T_2}. To this end, we have to perform this verification for all types of function supports and for any possible mesh configuration. Applying some reasoning, it is easy to see that this verification can be reduced to a few number of basic cases. Firstly, to simplify the process, we assume that for any two 0-balanced meshes T_1 and T_2 such that $T_1 \subset T_2$ there exists a sequence of meshes $\{T_i^e\}_{i=1,n}$ such that

(i) $T_1 = T_0^e \subset T_1^e \subset T_2^e \subset \cdots \subset T_{n-1}^e \subset T_n^e = T_2$,
(ii) any mesh T_i^e is obtained from mesh T_{i-1}^e by refining only one cell,



(a) (b)

Figure 23: (a) All possible configurations for a knot vector (without repeated knots). (b) On the left, a function support of level m defined over a 0-balanced T-mesh. The level of this function is determined by the cell marked in red. On the right, the quadtree representation of the T-mesh. The tree node represented in red corresponds to the marked cell of the mesh.

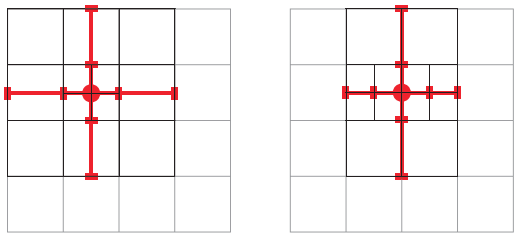
(iii) all the meshes $\{T_i^e\}_{i=1,n}$ are 0-balanced.

A pair of 0-balanced meshes T_2^e and T_1^e is called *elemental refinement* if T_2^e is obtained from T_1^e by refining only one cell, see Fig. 22. Since it is sufficient to verify the nestedness of spaces for any elemental refinement, from now on we assume that $T_1 \subset T_2$ is an elemental refinement. We have to verify that, after refining a cell, the blending functions of the new space S_{T_2} are able to represent any blending function of the previous space S_{T_1}. The proof can be divided in two steps. First, we analyze some basic cases corresponding to the simplest mesh configuration, then, more general mesh configuration should be considered applying some recursivity idea to proof the nestedness of spaces for arbitrary elemental refinement.

To make the necessary studies more graphic, first, we classify all function supports that can take place according to our strategy. Then, we study for each type all possible situations when a cell refinement affects this type function support.

Classification of function supports

Let us consider any knot vector $\Xi = (\xi_1, \xi_2, \xi_3, \xi_4, \xi_5)$ and let $\Delta_i^\xi = \xi_{i+1} - \xi_i$, $i = 1, ..., 4.$, so that, a set $\Delta^\xi = (\Delta_\mathbf{1}^\xi, \Delta_\mathbf{2}^\xi, \Delta_\mathbf{3}^\xi, \Delta_\mathbf{4}^\xi)$ corresponds to each knot vector Ξ. Due to the extension rule 1 and the balanced mesh condition we have the following possible configurations for the set Δ^ξ: (h, h, h, h), $(h, h, h, 2h)$, $(2h, h, h, h)$ and $(2h, h, h, 2h)$, see Fig. 23(a). All possible bivariate function supports are obtained by combining all configurations for Ξ



(a) (b)

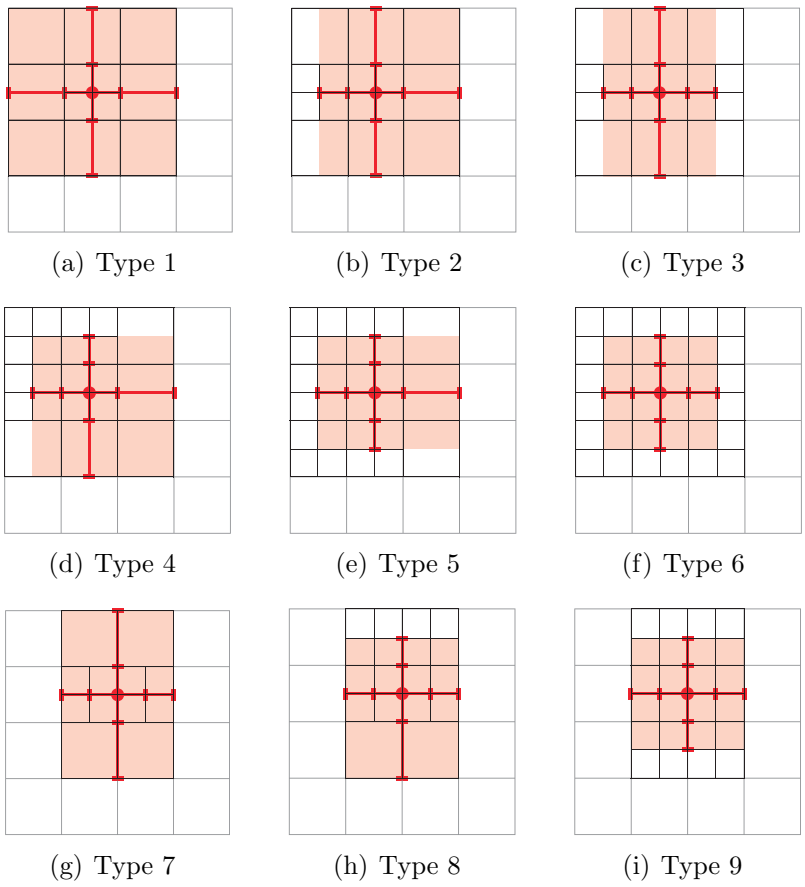
Figure 24: Two possible scenarios for a function support according to the underlying local mesh. (a) The anchor of the m-level function coincides with the center of a $(m-1)$-level cell of the quadtree. (b) The rest of the functions.

and $\mathcal{H}$, i.e., combining two sets Δ^ξ and Δ^η. Note that since we ignore the hanging nodes while inferring two knot vectors, the value of h is the same for Δ^ξ and Δ^η, i.e., $h = \Delta^\xi_2 = \Delta^\xi_3 = \Delta^\eta_2 = \Delta^\eta_3$ and the value of h determines the level of a function. A blending function is said to be of level m if the largest cell contacting its anchor is of level m, see Fig. 23(b).

Also, we can distinguish different function supports according to the underlying local mesh that leads to this support.Basically, it depends on the position of the anchor of the function, see Fig. 24. Scenario (a) the anchor of the m-level function coincides with the center of a $(m-1)$-level cell of the quadtree, scenario (b) the rest of the functions, i.e., the anchor of the m-level function does not correspond to any center of a $(m-1)$-level cell of the quadtree.

Finally, taking into account all the above considerations about values of Δ^ξ and Δ^η and the underlying local mesh, we can classify all function supports in nine types that are illustrated in Fig. 25. First six types correspond to scenario (a) and the other three types correspond to scenario (b). Note that type 6 and type 9 function supports can be considered as the same type, although they come from different scenarios. Function supports with repeated knots across the domain boundary can be included in the nine types shown in Fig. 25.

Next, we have to see how an elemental refinement can affect a function support of each type. As it was said before, the proof of the nestedness can be reduced to the proof of nestedness for some basic cases.



(a) Type 1 (b) Type 2 (c) Type 3

(d) Type 4 (e) Type 5 (f) Type 6

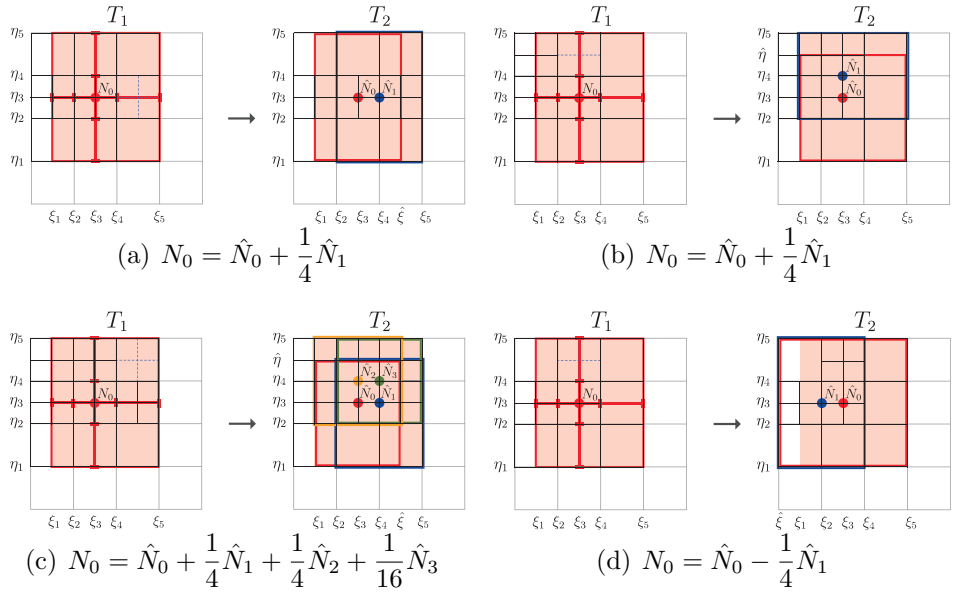
(g) Type 7 (h) Type 8 (i) Type 9

Figure 25: Function support classification according to the underlying local mesh and configurations for Δ^{ξ} and Δ^{η}.

Function recuperation for the basic cases

For each support type, we have to analyze all possible elemental refinements that affect this function and verify that the blending functions of the finer mesh are able to reproduce the function under consideration. Let see in detail an example of this study.

Here we illustrate type 2 function support. This type of function support has the largest number of cases to study. Namely, we should study four different situations that take place after a cell refinement. In these situations the original type 2 function N_0 associated to the anchor v_0 is not a blending function of the new space S_{T_2}. The new function $\hat{N}_0$ associated to v_0 can be



(a) $N_0 = \hat{N}_0 + \frac{1}{4}\hat{N}_1$ (b) $N_0 = \hat{N}_0 + \frac{1}{4}\hat{N}_1$

(c) $N_0 = \hat{N}_0 + \frac{1}{4}\hat{N}_1 + \frac{1}{4}\hat{N}_2 + \frac{1}{16}\hat{N}_3$ (d) $N_0 = \hat{N}_0 - \frac{1}{4}\hat{N}_1$

Figure 26: Basic cases to study for type 2 function. Support of the function N_0 defined over T_1 is represented by pink shadowed area in the leftmost and the rightmost images. Blue discontinuous lines in the leftmost figure mark the refinement of T_1 that leads to a new mesh T_2. Supports and the anchors of the blending from S_{T_2} necessary to recuperate the function N_0 are marked by different colors in the rightmost image. (a) Type 2 support turns into a type 3 support after a cell refinement. (b) Type 2 support turns into a type 4 support. (c) Type 2 support turns into a type 5 support after a cell refinement. (d) Type 2 support turns into a type 5 support.

type 3, 4, 5 or 1, as shown below.

Type 2 to type 3. In this case, type 2 function N_0 of level m can be recuperated by two new functions of the same level: type 3 function $\hat{N}_0$ and type 7 function $\hat{N}_1$. Only one of local knot vectors undergoes a knot insertion. Without loss of generality, suppose that a new knot $\hat{\xi}$ is inserted between the fourth and the fifth knot of Ξ, see Fig. 26(a). Then, using the knot insertion formula, it is easy to see that

$$N_0(\xi,\eta) = B[\Xi]B[\mathcal{H}] = B[\Xi_1]B[\mathcal{H}] + \frac{1}{4}B[\Xi_2]B[\mathcal{H}] = \hat{N}_0 + \frac{1}{4}\hat{N}_1,$$

where $\Xi_1 = (\xi_1, \xi_2, \xi_3, \xi_4, \hat{\xi})$ and $\Xi_2 = (\xi_2, \xi_3, \xi_4, \hat{\xi}, \xi_5)$.

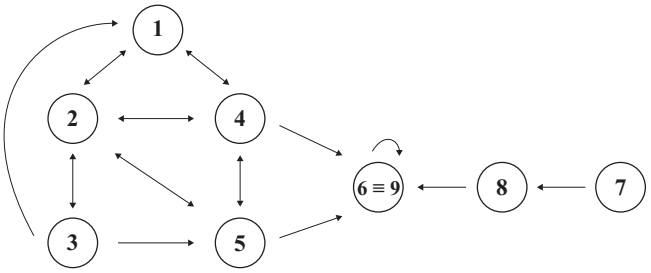


Figure 27: Graph of relations between different function types. An arrow coming out of a circle with type number and pointing to another type means that this type support can turn into the second one after a cell refinement.

Type 2 to type 4. Function N_0 is split into two new functions of the same level: type 4 function $\hat{N}_0$ and type 8 function $\hat{N}_1$. Local knot vector $\mathcal{H}$ undergoes a knot insertion. A new knot $\hat{\eta}$ is inserted in its local knot vector $\mathcal{H} = (\eta_1, \eta_2, \eta_3, \eta_4, \eta_5)$ between the fourth and the fifth knot, see Fig. 26(b). Then

$$N_0(\xi, \eta) = B[\Xi]B[\mathcal{H}] = B[\Xi]B[\mathcal{H}_1] + \frac{1}{4}B[\Xi]B[\mathcal{H}_2] = \hat{N}_0 + \frac{1}{4}\hat{N}_1,$$

where $\mathcal{H}_1 = (\eta_1, \eta_2, \eta_3, \eta_4, \hat{\eta})$ and $\mathcal{H}_2 = (\eta_2, \eta_3, \eta_4, \hat{\eta}, \eta_5)$.
Type 2 to type 5. For this case both local knot vectors undergo a knot insertion. New knot $\hat{\xi}$ is inserted between the fourth and the fifth knot of Ξ and knot $\hat{\eta}$ is inserted between the fourth and the fifth knot of $\mathcal{H}$, see Fig. 26(c). Then function N_0 is split into four functions of the same level: type 5 function $\hat{N}_0$, type 8 function $\hat{N}_1$, type 9 function $\hat{N}_2$ and type 9 (or 6) function $\hat{N}_3$.

$$\begin{aligned} N_0(\xi, \eta) &= B[\Xi]B[\mathcal{H}] = (B[\Xi_1] + \frac{1}{4}B[\Xi_2])(B[\mathcal{H}_1] + \frac{1}{4}B[\mathcal{H}_2]) = \\ &= B[\Xi_1]B[\mathcal{H}_1] + \frac{1}{4}B[\Xi_2]B[\mathcal{H}_1] + \frac{1}{4}B[\Xi_1]B[\mathcal{H}_2] + \frac{1}{16}B[\Xi_2]B[\mathcal{H}_2] = \\ &= \hat{N}_0 + \frac{1}{4}\hat{N}_1 + \frac{1}{4}\hat{N}_2 + \frac{1}{16}\hat{N}_3. \end{aligned}$$

where $\mathcal{H}_1 = (\eta_1, \eta_2, \eta_3, \eta_4, \hat{\eta})$, $\mathcal{H}_2 = (\eta_2, \eta_3, \eta_4, \hat{\eta}, \eta_5)$, $\Xi_1 = (\xi_1, \xi_2, \xi_3, \xi_4, \hat{\xi})$ and $\Xi_2 = (\xi_2, \xi_3, \xi_4, \hat{\xi}, \xi_5)$.

Type 2 to type 1. In this case, due to extension rule 2, the support of a function becomes larger after a cell refinement, see Fig. 26(d). Using knot insertion formula, it is easy to see that $\hat{N}_0 = N_0 + \frac{1}{4}\hat{N}_1$. So, the original function N_0 can be recuperated by two new functions: type 1 function $\hat{N}_0$ and type 7 function $\hat{N}_1$ as follows

$$N_0(\xi,\eta) = B[\Xi]B[\mathcal{H}] = B[\Xi_2](\xi)B[\mathcal{H}] - \frac{1}{4}B[\Xi_1](\xi)B[\mathcal{H}] = \hat{N}_0 - \frac{1}{4}\hat{N}_1.$$

where $\Xi_1 = (\hat{\xi}, \xi_1, \xi_2, \xi_3, \xi_4)$ and $\Xi_2 = (\hat{\xi}, \xi_2, \xi_3, \xi_4, \xi_5)$.

For the rest of the function supports the necessary verifications are carried out in an analogous way. The graph given in Fig. 27 reflects the relation between different function types and the number of situations to study for each type. More specifically, an arrow coming out of a circle with type X and pointing to another type Y means that a type X function can turn into the type Y new function associated to the same anchor after a cell refinement. For example, there are four arrows coming out of type 2 function, in line with the four studied cases.

[1] T. W. Sederberg, J. Zheng, A. Bakenov, A. Nasri, T-splines and T-NURCCSs, ACM Trans. Graph. 22 (2003) 477–484.

[2] Y. Bazilevs, V. M. Calo, J. A. Cottrell, J. A. Evans, T. J. R. Hughes, S. Lipton, M. A. Scott, T. W. Sederberg, Isogeometric analysis using T-splines, Comput. Meth. Appl. Mech. Eng. 199 (2010) 229–263.

[3] T. J. R. Hughes, J. A. Cottrell, Y. Bazilevs, Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement, Comput. Meth. Appl. Mech. Eng. 194 (2005) 4135–4195.

[4] J. A. Cottrell, T. J. R. Hughes, Y. Bazilevs, Isogeometric analysis: Toward integration of CAD and FEA, John Wiley & Sons, Chichester, 2009.

[5] M. A. Scott, X. Li, T. W. Sederberg, T. J. R. Hughes, Local refinement of analysis-suitable T-splines, Comput. Meth. Appl. Mech. Eng. 213-216 (2012) 206–222.

[6] X. Li, J. Zheng, T. W. Sederberg, T. J. R. Hughes, M. A. Scott, On linear independence of T-spline blending functions, Comput. Aid. Geom. Design 29 (2012) 63–76.

[7] D. R. Forsey, R. H. Bartels, Hierarchical B-spline refinement, Computer Graphics 22(4) (1988) 205–212.

[8] R. Kraft, Surface Fitting and Multiresolution Methods, Vol. 2, Vanderbilt University Press, 1997, Ch. Adaptive and linearly independent multilevel B-splines, pp. 209–218.

[9] A. V. Vuong, C. Giannelli, B. Jüttler, B. Simeon, A hierarchical approach to adaptive local refinement in isogeometric analysis, Comput. Meth. Appl. Mech. Eng. 200 (2011) 3554–3567.

[10] D. Schillinger, L. Debé, M. Scott, J. A. Evans, M. J. Borden, E. Rank, T. J. R. Hughes, An isogeometric design-through-analysis methodology based on adaptive hierarchical refinement of nurbs, immersed boundary methods, and T-spline CAD surfaces, Comput. Meth. Appl. Mech. Eng. 249–252 (2012) 116–150.

[11] P. B. Bornemann, F. Cirak, A subdivision-based implementation of the hierarchical b-spline finite element method, Comput. Meth. Appl. Mech. Eng. 253 (2013) 584–598.

[12] C. Giannelli, B. Jüttler, H. Speleers, THB-splines: The truncated basis for hierarchical splines, Computer Aided Geometric Design 29 (2012) 485–498.

[13] J. Deng, F. Chen, X. Li, C. Hu, W. Tong, Z. Yang, Y. Feng, Polynomial splines over hierarchical T-meshes, Graphical Models 70 (2008) 76–86.

[14] T. Dokken, T. Lyche, K. F. Pettersen, Polynomial splines over locally refined box-partitions, Comput. Aid. Geom. Design 30(3) (2013) 331–356.

[15] H. Samet, Foundations of Multidimensional and Metric Data Structures, Morgan Kaufmann Publishers, Burlington, Massachusetts, 2006.

[16] L. Piegl, W. Tiller, The NURBS book, Springer, New York, 1997.

[17] D. Moore, The cost of balancing generalized quadtrees, in: Proceedings of the Third ACM Symposium on Solid Modeling and Applications, SMA '95, ACM, New York, NY, USA, 1995, pp. 305–312. doi:10.1145/218013.218078.

[18] A. Wang, G. Zhao, Y. Li, Linear independence of the blending functions of T-splines without multiple knots, Expert Systems with Applications 41 (2014) 3634–3639.

[19] M. S. Floater, Parametrization and smooth approximation of surface triangulations, Comput. Aid. Geom. Design 14 (1997) 231–250.

[20] M. Brovka, J. I. López, J. M. Escobar, J. M. Cascón, R. Montenegro, A new method for T-spline parameterization of complex 2D geometries, Engineering with Computers 30 (4) (2014) 457–473.

[21] T. J. R. Hughes, A. Reali, G. Sangalli, Efficient quadrature for NURBS-based isogeometric analysis, Comput. Meth. Appl. Mech. Eng. 199, 5-8 (2010) 301–313.

[22] J. M. Escobar, J. M. Cascón, E. Rodríguez, R. Montenegro, A new approach to solid modeling with trivariate T-splines based on mesh optimization, Comput. Meth. Appl. Mech. Eng. 200 (2011) 3210–3222.

[23] J. M. Escobar, R. Montenegro, E. Rodríguez, J. M. Cascón, The meccano method for isogeometric solid modeling and applications, Engineering with Computers 30 (3) (2014) 331–343.

[24] R. Montenegro, J. M. Cascón, J. M. Escobar, E. Rodríguez, G. Montero, An automatic strategy for adaptive tetrahedral mesh generation, Appl. Num. Math. 59 (2009) 2203–2217.

[25] J. M. Cascón, E. Rodríguez, J. M. Escobar, R. Montenegro, Comparison of the meccano method with standard mesh generation techniques, Engineering with Computers 31 (1) (2015) 161–174.

[26] Y. Zhang, W. Wang, T. J. R. Hughes, Solid T-spline construction from boundary representations for genus-zero geometry, Comput. Meth. Appl. Mech. Eng. 249-252 (2012) 185–197.

[27] W. Wang, Y. Zhang, L. Liu, T. J. R. Hughes, Trivariate solid T-spline construction from boundary triangulations with arbitrary genus topology, Computer-Aided Design 45 (2013) 351–360.