JOHANNES KEPLER
UNIVERSITÄT LINZ | JKU

TNF

Technisch-Naturwissenschaftliche
Fakultät

Theory and Algorithms for Truncated Hierarchical B–splines

DISSERTATION

zur Erlangung des akademischen Grades

Doktor der Naturwissenschaften

im Doktoratsstudium der

Naturwissenschaften

Eingereicht von:
Mag. Gábor Kiss

Angefertigt am:
Institut für Angewandte Geometrie

Beurteilung:
Prof. Dr. Bert Jüttler (Betreuung)
Prof. Dr. Stefanie Hahmann

Linz, Juni, 2015

Acknowledgments

This thesis summarizes my research conducted during my employment at the Doctoral Program "Computational Mathematics" (W1214, DK3) at the Johannes Kepler University in Linz. The research was additionally funded from the project EXAMPLE (EC GA nos. 324340) and the project "Geometry + Simulation" (FWF NFN S117).

First of all I would like to thank my supervisor, Prof. Bert Jüttler, for giving me the chance to work within the framework of the Doctoral Program, but also for his guidance and helpful advices during my studies.

I am also very grateful to Carlotta Giannelli for the countless hours she has invested in discussions during her stays in Linz and Munich. They have deepened my insight into several topics.

I also want to thank all past and present students at the Doctoral Program, especially my office mate Daniel Gerth, and all students at the Applied Geometry Institute for creating a very pleasant environment for discussion and work. The friendships which have developed during my time at the Program made this time not only productive, but also very enjoyable.

I would like to thank my host at Seoul National University, prof. Myung-Soo Kim, for creating such a productive work atmosphere during my stay. Furthermore, I want to express my gratitude to all employees of MTU Aero Engines, especially David Großmann and Johannes Barner, who helped with my work during my one year research stay in the framework of the EXAMPLE project. Their experience and advice were invaluable. Additional thanks goes to Florian Buchegger and David Gruber who made my stay in

Munich not only fruitful, but also very pleasant.

I also want to thank all patient readers and reviewers who helped to improve this thesis by giving valuable comments and suggestions.

Last but not least I want to thank my family and friends for constant support during my studies. Special thanks goes to my girlfriend Viktoria Vassova who encouraged me to join the DK programme and offered her help and support even in the most difficult times.

Thank you.

Kurzfassung

Aufgrund ihrer vielen wünschenswerten Eigenschaften nutzen Softwarebibliotheken im Computer–Aided Design (CAD) seit Jahrzehnten B–Splines und NURBS als Standard zur Darstellung von geometrischen Objekten mit Splines. Unglücklicherweise mussten Nutzer von CAD–Software jahrelang mit der Einschränkung leben, dass die Verallgemeinerung von B–Splines auf höhere Dimensionen, z.B. auf Flächen oder Volumen, auf Tensorprodukt–Techniken beruht, welche streng lokale Verfeinerungen ausschließen. Die Suche nach Alternativen zu Tensorprodukt–Splines, welche lokale Verfeinerungen unterstützen, hat in den letzten Jahren erheblich an Dynamik gewonnen, hauptsächlich durch die Einführung der Isogeometric Analysis [32]. Dort ist Adaptivität erforderlich, um lokale Verfeinerungen in numerischen Simulationen vorzunehmen. In dieser Arbeit konzentrieren wir uns auf eine dieser adaptiven Spline–Techniken; die so genannten *truncated hierarchical B–splines* (THB–Splines). Diese unterstützen streng lokale Verfeinerungen und bewahren gleichzeitig die Haupteigenschaften der Standard–B–Splines.

Nach einer kurzen theoretischen Einführung von hierarchischen Splines und THB–Splines präsentieren wir eine effiziente Implementation der grundlegenden Algorithmen, welche für die Manipulation von THB–Splines notwendig sind. Zur Speicherung der Gebietsstruktur der THB–Spline–Basis nutzen wir eine auf kD-trees beruhende Datenstruktur. Ergänzt wird dies durch eine weitere Datenstruktur, welche zur Speicherung der Informationen über die so genannten aktiven Basisfunktionen benötigt wird. Mittels dieser zwei Datenstrukturen erhalten wir eine effiziente Methode zur Konstruktion und Auswertung von THB–Splines.

Das Hauptaugenmerk unserer Arbeit ist die Anwendung der THB–Spline–Methode sowohl in Forschung als auch in der Praxis.

Zuerst präsentieren wir eine adaptive und automatische Methode zur Oberflächenrekonstruktion aus komplexen, der industriellen Anwendung entstammenden Daten. Am Beispiel kritischer Segmente von Turbinenblättern demonstrieren wir, dass das Fitten von Oberflächen mittels THB–Spline–Darstellungen zu einer deutlichen Verbesserung der erzielten Rekonstruktionen führt. Weiters zeigen wir, dass die lokale Auswertung von THB–Splines

in Bezug auf B–Spline–Patches mit kommerziellen Modellierkernen kombiniert werden kann, um mehrstufige Spline–Darstellungen in eine äquivalente CAD–Geometrie in Standard–B–Spline–Darstellung umzuformen.

Weiterhin untersuchen wir das Modellierungspotenzial der THB–Splines. Mittels einer einfachen Schnittstelle zwischen unserer THB–Spline Implementation und dem Axel Modellierer erzeugen wir mehrere einfache THB–Spline Geometrien durch interaktive Manipulation der Kontrollpunkte. Wir hoffen, dass dieses Kapitel einen Weg zur Einführung von THB–Splines in kommerzielle CAD–Software aufzeigt.

Abschließend untersuchen wir die Leistungsfähigkeit von THB–Splines in numerischen Simulationen, insbesondere den Einfluss der sog. "truncation" auf das Verhalten von Multigrid–Lösungsverfahren.

Abstract

For several decades the Computer–Aided Design (CAD) software libraries have been using B–splines and NURBS as their basic spline representation for geometric objects. This is due to the many desirable properties the B–splines possess. Unfortunately, for many years the users of CAD software had to deal with the fact that the generalization of the B–spline technology to higher dimensions, e.g. surfaces or volumes, is based on the tensor-product construction, which precludes strictly localized refinement. The investigation of alternatives to the tensor-product splines that support local refinement has gained significant momentum in the past few years, mainly due to the advent of Isogeometric Analysis (IgA) [32] where adaptivity is needed for performing local refinement in numerical simulations.

In our work we focus on one of these adaptive spline technologies, namely on the recently introduced *truncated hierarchical B–splines* (THB–splines). This technology does not only support strictly localized refinement, but at the same time preserves the main properties of the standard B–spline basis.

In the first part we present a short theoretical introduction to the topic of hierarchical and truncated hierarchical splines. Subsequently we introduce an efficient implementation of the fundamental algorithms needed for manipulation with THB–splines. To store the subdomain structure of the THB–spline basis we employ a kD-tree data structure. This is complemented by another data structure used for storing the information about the so called active basis function. Using these two structures we obtain an efficient technique for the construction and evaluation of THB–splines.

The main focus of our work is however on the application of the THB–spline technique in research and real world applications. Firstly, we present an adaptive and automatic surface reconstruction method used for the reconstruction of complex real world data. Using both, synthetic and real world point data we demonstrate that surface fitting schemes based on THB-spline representations lead to significant improvements of the reconstruction. Furthermore, we show that the local THB-spline evaluation in terms of B-spline patches can be properly combined with commercial geometric modeling kernels in order to convert

the multilevel spline representation into an equivalent standard B–spline CAD geometry. Secondly, we explore the modelling capabilities of the THB–spline technology. Using a simple interface between our THB–spline implementation and the Axel modeler we created several simple THB–spline geometries by interactive control grid manipulation. We hope this chapter also paves the way for introduction of THB–splines into commercial CAD software. Lastly, we have investigated the performance of THB–splines in numerical simulations, focusing on the influence of the truncation on the performance of multigrid solvers.

Contents

Chapter 1

Introduction

The Computer–Aided Design (CAD) software, standardly used by designers and engineers, provide graphical user interfaces that allow manipulation and shaping of geometric models using a set of functions which are implemented in the geometric modeling kernels, such as Parasolid^{TM} [41], C3D [6], or the open source Open Cascade Technology [40]. These kernels, and software use the B–spline technology and its non–uniform rational extension (NURBS) as their standard spline representation. The success of this representation lies in its fundamental properties, such as minimal support, non–negativity, partition of unity, and efficient algorithms for evaluation, refinement and degree elevation. The non–negativity and partition of unity properties imply that a B–spline is always completely contained in the convex hull of its control net, which resembles the shape of the underlying B–spline model. This means that the parametric spline representation can be shaped by the designer in an intuitive way simply by manipulating the control net of the given object. All these factors meet the requirements of CAD applications. Unfortunately, the higher-dimensional extension of the univariate B–splines is based on the tensor-product approach, and thus it precludes the possibility of strictly localized refinement.

Despite of an increasing interest to overcome this drawback from the side of the CAD and manufacturing communities, the localized mesh refinement remains a computationally expensive and non–trivial operation. This effort is more recently also supported by the

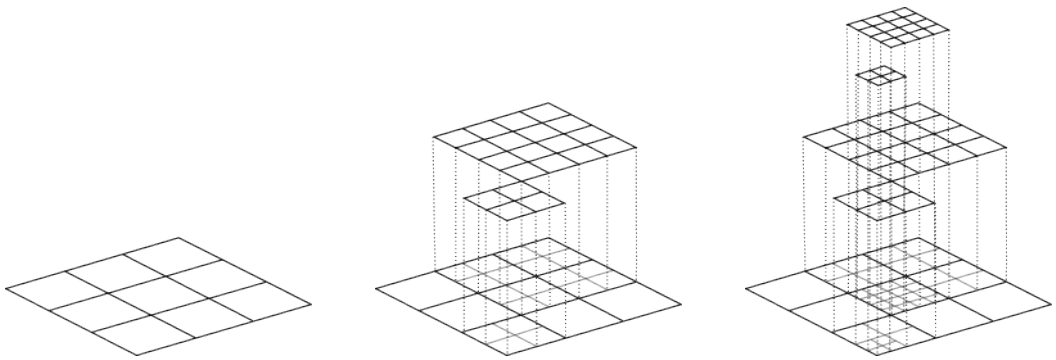


Figure 1.1: Hierarchical structure of domains used for construction of hierarchical B–splines.

developments in the field of numerical analysis, especially isogeometric analysis.

1.1 Related work

Over the past decades several different approaches were proposed to address the problem of local refinement by allowing so called T–junctions between the axis–aligned mesh segments. Among the most popular locally refinable spline technologies are:

1. **Hierarchical B–splines**:

   The original idea to perform modeling by manipulating the parametric representation at different levels of detail was introduced by Forsey and Bartels [17]. In order to localize the influence of the control points while editing detailed features, the refinement is performed on restricted patches of the surface using a sequence of *overlays* with nested knot vectors, as shown in Figure 1.1. Although this construction allowed local refinement of the surface, it did not address the issue of creating a basis for hierarchical splines. Several years later Kraft [35, 36] showed that the mesh refinement procedure can be complemented by a simple and automatic identification of basis functions from tensor-product B–splines of different refinement levels, see Figure 1.2. The selection process, which computes the hierarchical B–spline basis, is described in more detail in Section 2.2.2. Exploiting this underlying tensor-product

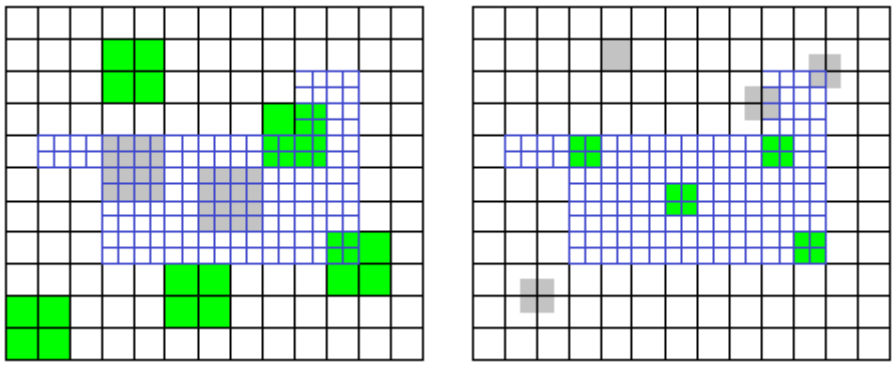


Figure 1.2: A hierarchical structure with 2 levels, level 0 black and level 1 blue. Examples of basis functions contributing to the bilinear HB–spline basis (highlighted by green) represented by their support for levels 0 (left) and level 1 (right). Some of the basis functions not contributing to the hierarchical basis are highlighted by grey.

structure at different refinement levels, the hierarchical B–splines (HB–splines) inherit several desirable properties, such as linear independence, non–negativity, local support, and completeness of the spline space, all of which are fundamental to define an effective spline representation.

The potential of HB–splines, especially in isogeometric analysis, has been recently demonstrated in [44, 53]. Additional results focusing on the partition of unity property, the approximation power, and the strong stability have been obtained by considering the truncated basis for the hierarchical B–spline space (*THB–splines*) in [20, 21, 33, 49]. Generalizations of the truncated basis to wider classes of spline spaces and an extension to cover arbitrary topologies have also been presented in [45, 57, 58].

2. **T–splines**:

   The more recently introduced T–spline technology (see [46, 47]) chooses a different approach to local refinement. Instead of using a sequence of nested tensor-product B–splines and collecting the basis functions from different levels, the definition of T–splines is using a so called T–mesh. The T–mesh is a rectangular grid which, in contrast to the tensor-product B–splines, allows so called T–junctions. That means a

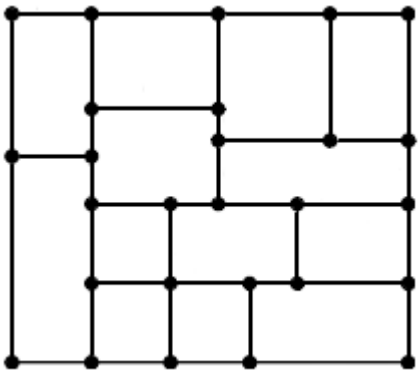


Figure 1.3: Example of a T–mesh with several T-junctions.

row (or column) of vertices of the T–mesh is allowed to terminate without traversing the mesh from one side to the other. A T–spline surface is computed by using blending functions which are a variation of the basis functions of standard B–splines. In contrast to the tensor-product case where all basis functions share the same knot vectors, in the case of the T–splines we have to determine the knots of every blending function separately.

Unfortunately, the blending functions are not necessarily linearly independent, and thus they do not always form a basis of a spline space. To address this issue we might impose additional restrictions on the T–mesh, as it was shown by Li in [38]. Another drawback of the T–spline technology, not completely solved yet, is its generalization to higher dimensions.

3. **Locally Refined B–splines**:

   One of the most recently developed locally refinable spline technology is the *Locally Refined B–splines (LR B–splines)* introduced in 2013 by Dokken, Lyche and Pettersen in [13]. The construction of the LR B–splines is based on the idea of the LR–mesh which is a special subset of *box partitions*. A box partition of a given domain Ω in $\mathbb{R}^d$ is a collection of d-dimensional boxes, such that the intersection of any two boxes is empty and the union of all boxes is Ω. For example see Figures 1.4 and 1.5.

   The LR B–splines are then defined as standard tensor-product B–splines that have a

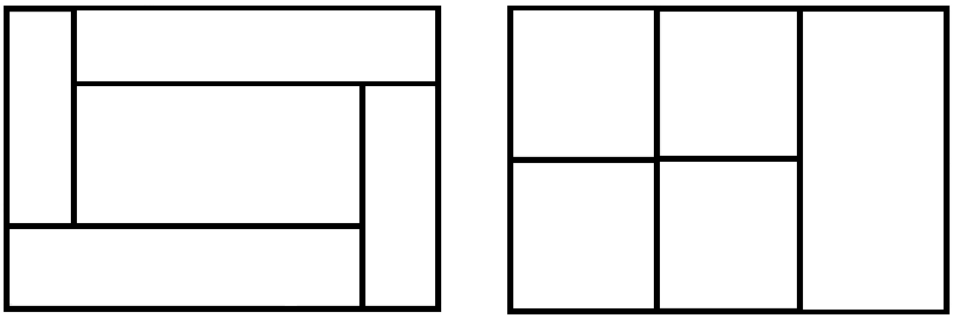


Figure 1.4: The box mesh on the left is not an LR–mesh. If the boundary of the rectangle is considered as the original, not refined mesh, we cannot create the given configuration by adding splits which always cut a box into 2 parts. The example on the right satisfies this criteria, and thus it is a LR–mesh.

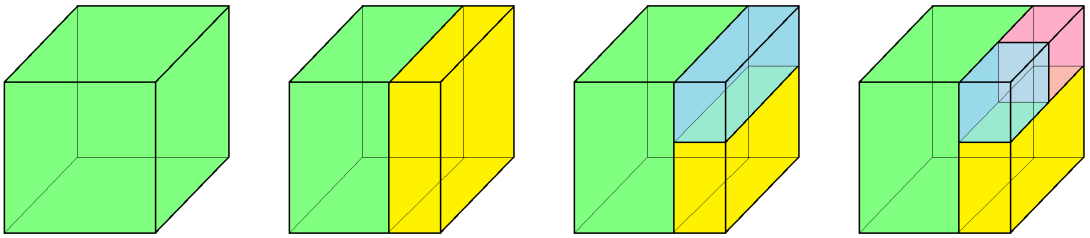


Figure 1.5: The construction of LR–meshes, and thus LR B–splines can be generalized any dimension. In the figure we can see a construction of 3-dimensional LR–mesh.

so called *minimal support* on a given LR–mesh. An example of a LR B–spline local refinement is shown in Figure 1.6.

In spite of the fact that the LR B–splines are not necessarily linearly independent directly by construction, there is a simple algorithm presented in [13] which removes the redundant basis functions without forcing any restrictions on the LR–mesh. Similarly, the partition of unity property is not necessarily preserved during the refinement process of the LR B–splines. Nevertheless, it may be recovered for example by rational scaling or introducing weights for the basis functions.

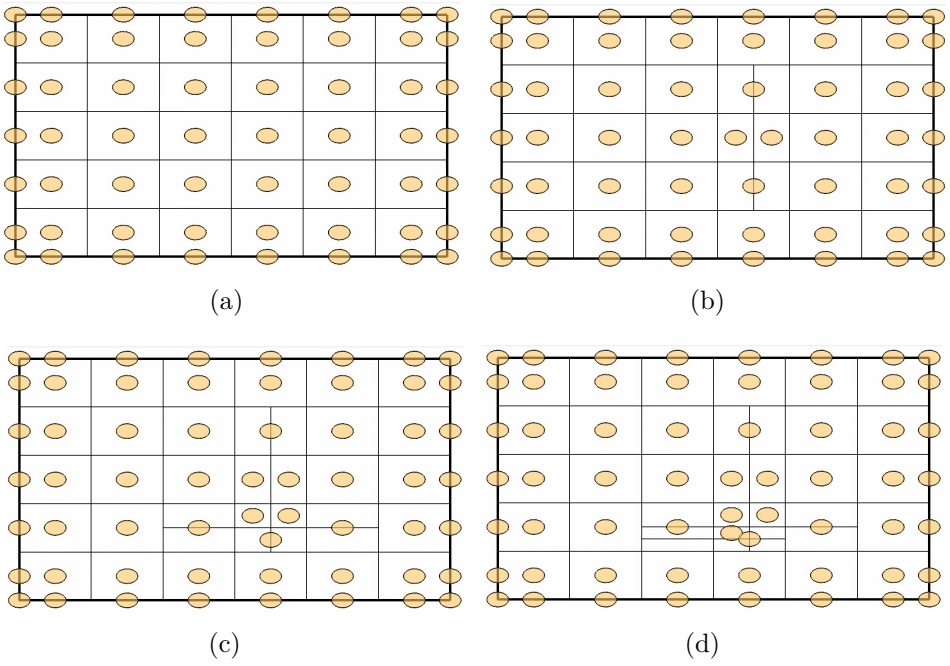


Figure 1.6: Construction of a LR–mesh by inserting cuts into a standard tensor-product mesh. The anchors of basis functions are depicted by the orange ellipses. The figures were created using the *LR B–spline Introduction* app for mobile devices [52].

All these technologies, and some additional ones – for example polynomial splines over T–meshes (PHT–splines) [10, 11], or hierarchical triangular splines [56] - are used in many areas of research, although the main motivation for their development in the recent years comes from the isogeometric analysis community.

Nevertheless we believe that many other research areas could benefit from the locally refineable spline technologies as well. These areas include – but are not restricted to – computer graphics and animation for example for simulation of thin materials [39, 37, 4], creation of free form sculptures [50], or computer vision for contour detection [9] and 3D model reconstruction from 2D images [48].

1.2 Outline of the thesis

In the presented work we focus on the hierarchical B–spline approach, more specifically on the *Truncated Hierarchical B–splines* (THB–splines). THB–splines offer several desired properties, for example linear independence of basis functions for every possible mesh configuration, partition of unity property obtained directly during the construction of the basis, i.e. without any additional scaling, or natural way of generalisation to any dimension. These properties are not always present for other spline representations which offers strictly localized refinement.

Our work consists from two big parts. The first part consists of Chapters 2 and 3, where we introduce the fundamentals necessary for the construction of the THB–spline basis and present an efficient way of its implementation. The second part, Chapters 4, 5 and 6, focuses on the application of THB–splines in different research areas.

In Chapter 2 we provide an introduction to the construction of hierarchical and truncated hierarchical B–splines and we discuss their main properties, advantages and drawbacks. Subsequently, in Chapter 3, we present an efficient implementation of suitable data structures, and algorithms for THB–splines in arbitrary dimension, based on our previous work shown in [33]. For representation of the d-dimensional domain structure we use a single kD-tree like data structure. This representation facilitates the frequently needed update of the domain structure, which may occur during an iterative refinement process, for example in surface reconstruction methods or design. The selection of active basis functions, as described in [20], is executed in terms of four queries. The information about these basis functions is then encoded in lists which provide an efficient access to this data during the evaluation of THB–splines.

Interpolation or approximation of measured data is required in many industrial applications, as for example instrument calibration, data analysis, or reverse engineering. An automatic surface fitting procedure of complex data (for example obtained from turbine blades of aero engines), with high precision is a non–trivial issue that should be properly address by several critical steps, see e.g. [54]. In Chapter 4 we present an adaptive and

automatic framework for surface reconstruction using THB–splines from measured point data. We use a simple least–squares approximation method combined with a smoothing term. The main goal of this chapter is to provide a comparison between the results obtained from surface reconstruction using standard tensor-product B–splines and THB–splines. As we can see in Sections 4.3.1 and 4.4, in all tested cases the THB–splines provide better results. Furthermore, due to the fact that B–splines are the standard in the CAD software, in Section 4.3.2 we provide a method to export THB–splines into standard CAD format. The results presented in this chapter were published in [34].

In Chapter 5 we present methods which allow manual construction and editing of THB–splines. The main difficulty posed by using THB–splines for design is to correctly compute the control point of the refined surface without changing its geometry. For this purpose we provide an algorithm in Section 5.1, which outputs a so called *transfer matrix*. This is subsequently used for computing the control points of the refined surface. Furthermore, we discuss the possibility of locally reducing the smoothness of THB–splines by increasing multiplicities of knots in higher levels of refinement. We present several simple examples created in the Axel modeling tool [1], which uses our THB–spline implementation from the G+Smo library [25].

In Chapter 6 we present a further application of the THB–spline transfer matrices. We employ them in the framework of isogeometric analysis to solve elliptic boundary value problems using multigrid solvers. In this chapter we analyze the influence of the truncation mechanism to the efficiency of these solvers by comparing the number of iterations necessary for acquiring a satisfactory solution of the boundary value problem using standard hierarchical splines and truncated hierarchical splines.

Finally we conclude our work by summarizing our results in Chapter 7.

Chapter 2

Truncated hierarchical B–splines

In many real world applications the standard tensor-product B–spline construction does not provide sufficient flexibility as an effective modelling option, as especially the multivariate tensor-product construction prevents the construction of adaptive spline representations that support local refinement. In this chapter we define the main concepts of truncated hierarchical B–splines (THB–splines) which provide the possibility of introducing different levels of resolution in an adaptive framework, while simultaneously preserving the main properties of standard B–splines.

2.1 B–spline

In many applications the use of curves consisting of only one polynomial segment (e.g. Bézier curves) is not sufficient, mainly due to the following drawbacks:

- High degree curves required to represent complex shapes are numerically unstable and inefficient to process.
- Local control of the curve is not possible. Every change in the control polygon (insert, delete or movement of a control point) influences the shape of the whole curve.

To address these problems one may use piecewise polynomial curves (splines) [42].

2.1.1 Univariate B–spline

In the following section we define the B–spline basis in the recursive way, as introduced by deBoor, Cox and Mansfield [7, 8], together with the construction of B–spline curves. We demonstrate the definitions on several examples.

Definition 2.1.1. Let $U = \{u_0, \dots, u_m\}, m \in \mathbb{N}, m \geq 2p$, where p is a given polynomial degree, be a non–decreasing sequence of real numbers, i.e. $u_i \leq u_{i+1}, i = 0, \dots, m-1$. Additionally, let $u_i < u_{i+p}$ for $i = 0, \dots, m-p$. We call u_i *knots*, the half open interval $[u_i, u_{i+1})$ the *knot span*, and the sequence U the *knot vector*.

Definition 2.1.2. Let us denote the ith B–spline basis function of degree p defined over a knot vector U by $\beta_{i,p}, i = 0, \dots, n$ where $n = m - p - 2$. We may evaluate $\beta_{i,p}$ at a given parameter value u by using the following recursive formula:

$$\beta_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i}\beta_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}}\beta_{i+1,p-1}(u)$$

where

$$\beta_{i,0}(u) = \begin{cases} 1 & \text{if } u_i \leq u < u_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

In case $u_{i+p} - u_i = 0$ or $u_{i+p+1} - u_{i+1} = 0$, we consider the corresponding coefficient to be 0. The B–spline basis $\mathcal{B}$ is the collection of all $\beta_{i,p}$ defined over the given knot vector.

Definition 2.1.3. A B–spline curve of degree p is defined by

$$C(u) = \sum_{i}^{n} \beta_{i,p}(u) P_i$$

where $P_i \in \mathbb{R}^d$ are the so called *control points*, the $\beta_{i,p}(u)$ are B–spline basis functions and $u \in [u_p, u_n]$.

Example 2.1.1. Figure 2.1 shows an example of a B–spline curve of degree $p = 2$ with 6 control points $P_0 = [0, 0], P_1 = [1, 2], P_2 = [2, 1], P_3 = [3, 1], P_4 = [5, -2], P_5 = [5, 4]$

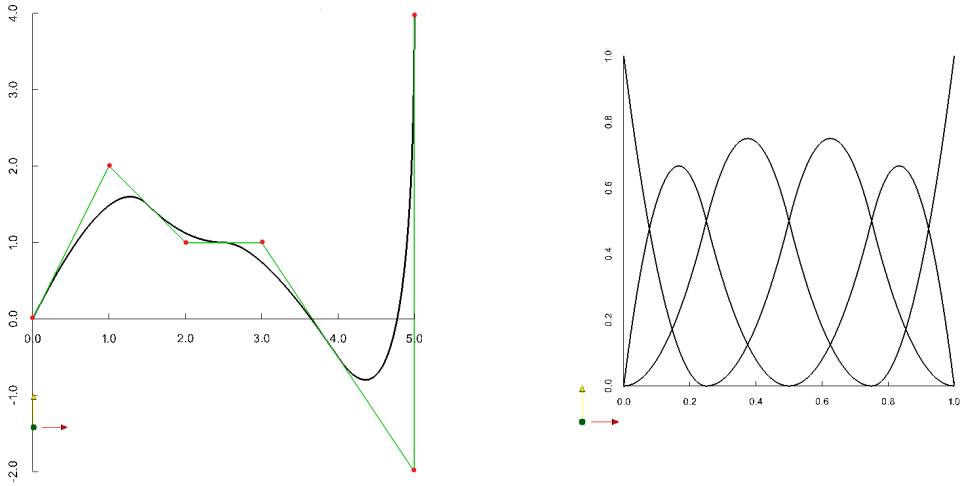


Figure 2.1: Example of a degree 2 curve (left) together with its control polygon(green) and the corresponding basis functions (right).

defined over the knot vector $U = \{0, 0, 0, 0.25, 0.5, 0.75, 1, 1, 1\}$. The polygon with vertices $P_i, i = 0, \ldots, 5$ is called the *control polygon* of the curve and it approximates the spline function.

2.1.2 Main properties of B–splines

The B–splines have several attributes which are one of the main reasons why they became de facto a standard in the computer–aided design (CAD) software. In the following list we highlight the most important of these properties:

1. **Local support**: $\{u : \beta_{i,p}(u) \neq 0\} = (u_i, u_{i+p+1})$

2. **Non–negativity**: $\beta_{i,p}(u) \geq 0$ for all $u \in [u_i, u_{i+p+1})$

3. **Partition of unity**: for all $u \in (u_p, u_n)$

$$\sum_{i=0}^{n} \beta_{i,p}(u) = 1$$

4. **Smoothness related to knot multiplicity**: The basis functions at knot u_i are C^{p-k} continuous where k is the multiplicity of the given knot.

5. **Local linear independence**: The B–splines having some support in any open set $\Omega' \subset \Omega$ are linearly independent.

6. **Convex hull property**: The B–spline curve is always in the convex hull of its control polygon.

7. **Piecewise linear curve approximation**: The control polygon is a piecewise linear approximation of the corresponding B–spline curve.

Because of these properties, the B–splines provide local control over the shape of the curve, i.e. the position of a control point influences the shape of only few polynomial segments of the whole curve. Additionally, the possibility to reduce the smoothness of the curve also allows to construct more sophisticated shapes, even with sharp features. Furthermore, as a consequence of the convex hull property and the fact the the control polygon approximates the shape of the curve B–spines provide a fairly intuitive control structure for designers.

2.1.3 Knot insertion algorithm

In this section we introduce one of the most important B–spline algorithms. Although it is not immediately apparent by applying the *knot insertion algorithm*, we can:

- evaluate points on B–spline curves,
- subdivide B–spline curves,
- extract Bézier curves,
- add control points (coefficients) - for example to increase the flexibility in shape control.

The knot insertion algorithm is performed as follows: Let C be a B–spline curve, as defined in Definition 2.1.3, over a knot vector $U = \{u_0, \ldots, u_m\}$, with control points P. We want to insert $\bar{u} \in [u_j, u_{j+1})$ into U to form a new knot sequence $\bar{U} = \{\bar{u}_0 = u_0, \ldots, \bar{u}_j = u_j, \bar{u}_{j+1} = \bar{u}, \bar{u}_{j+2} = u_{j+1}, \ldots, \bar{u}_{m+1} = u_m\}$, and to identify the new control points $\bar{P}$ of the curve such that

$$C(u) = \sum_{i=0}^{n} \beta_{i,p}(u) P = \sum_{i=0}^{n+1} \bar{\beta}_{i,p}(u) \bar{P}$$

where $\bar{\beta}_{i,p}$ are the B–spline basis functions defined over $\bar{U}$.

An efficient way to obtain these new coefficients is by applying the equation

$$\bar{P}_i = \alpha_i P_i + (1 - \alpha_i) P_{i-1} \tag{2.1}$$

where

$$\alpha_i = \begin{cases} 1 & \text{if } i \leq j - p, \\ \frac{\bar{u} - u_i}{u_{i+p} - u_i} & j - p + 1 \leq i \leq j, \\ 0 & i \geq j + 1, \end{cases} \tag{2.2}$$

as it was shown in [42].

In matrix representation we can write that $\bar{\boldsymbol{P}} = R\boldsymbol{P}$ where R is the $(n+1) \times n$ matrix

$$R = \begin{pmatrix} \alpha_0 & 0 & 0 & \cdots & \cdots & 0 \\ (1-\alpha_1) & \alpha_1 & 0 & \cdots & \cdots & 0 \\ 0 & (1-\alpha_2) & \alpha_2 & \cdots & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & (1-\alpha_{n-1}) & \alpha_{n-1} \\ 0 & 0 & 0 & 0 & \cdots & (1-\alpha_n) \end{pmatrix}.$$

By using the *transfer matrix* R one can define the relation between the two bases $\mathcal{B}$ and $\bar{\mathcal{B}}$ as follows $\mathcal{B} = R^T \bar{\mathcal{B}}$.

Example 2.1.2. Figure 2.2(a) shows the curve C from Example 2.1.1 after inserting the knot $\bar{u} = 0.1$ into the original knot vector $U = \{0, 0, 0, 0.25, 0.5, 0.75, 1, 1, 1\}$. We have computed the new control points $\bar{P}_0 = [0, 0], \bar{P}_1 = [0.4, 0.8], \bar{P}_2 = [1.2, 1.8], \bar{P}_3 = [2, 1], \bar{P}_4 = [3, 1], \bar{P}_5 = [5, -2], \bar{P}_6 = [5, 4]$ using Equation 2.1, where $j = 2$. Subsequently, we made a second knot insertion of the knot $\bar{u}$, thus increasing its multiplicity to 2. This additional knot insertions subdivides C into two B–spline curves - C_1 and C_2 with control points $\bar{P}_0 = [0, 0], \bar{P}_1 = [0.4, 0.8], \bar{P}_2 = [0.72, 1.2], \bar{P}_3 = [1.2, 1.8], \bar{P}_4 = [2, 1], \bar{P}_5 = [3, 1], \bar{P}_6 = [5, -2], \bar{P}_7 = [5, 4]$ where $\bar{P}_i, i = 0, 1, 2$ are the control points of C_1 and $\bar{P}_i, i = 2, \ldots, 7$ are the control points of C_2. Note that $\bar{P}_2 = C(\bar{u})$, i.e. we can evaluate points on the curve by inserting a given parameter value into the knot vector exactly p times.

2.1.4 Tensor-product construction

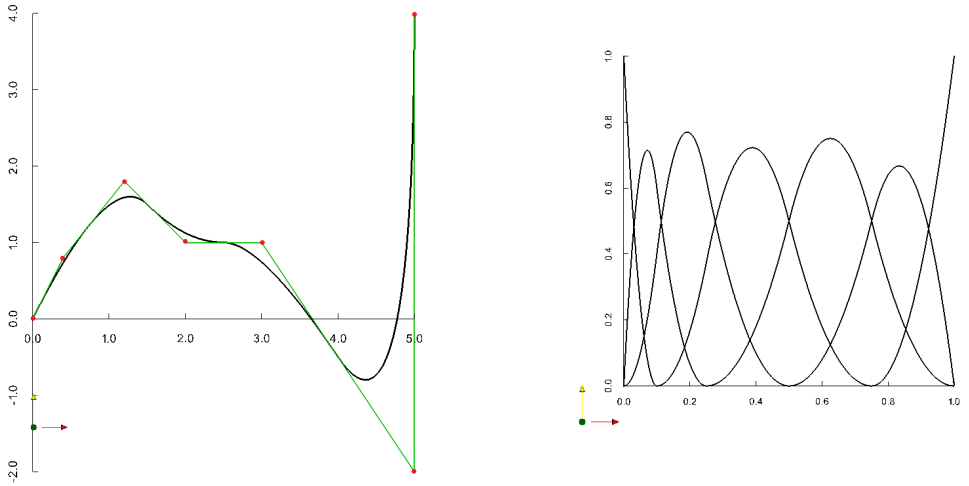
B–spline objects of higher dimensions, e.g. surfaces, volumes, etc. are constructed by taking the tensor-product of d univariate B–spline functions $\beta^1_{i^1,p^1}, \ldots, \beta^d_{i^d,p^d}$ defined over the corresponding knot vectors $U^1, \ldots, U^d$. More precisely, the multivariate B–spline function $S(u^1, \ldots, u^d)$ has the form

$$S(u^1, \ldots, u^d) = \sum_{i^1=0}^{n^1} \cdots \sum_{i^d=0}^{n^d} \beta^1_{i^1,p^1} \ldots \beta^d_{i^d,p^d} P_{i^1,\ldots,i^d}$$

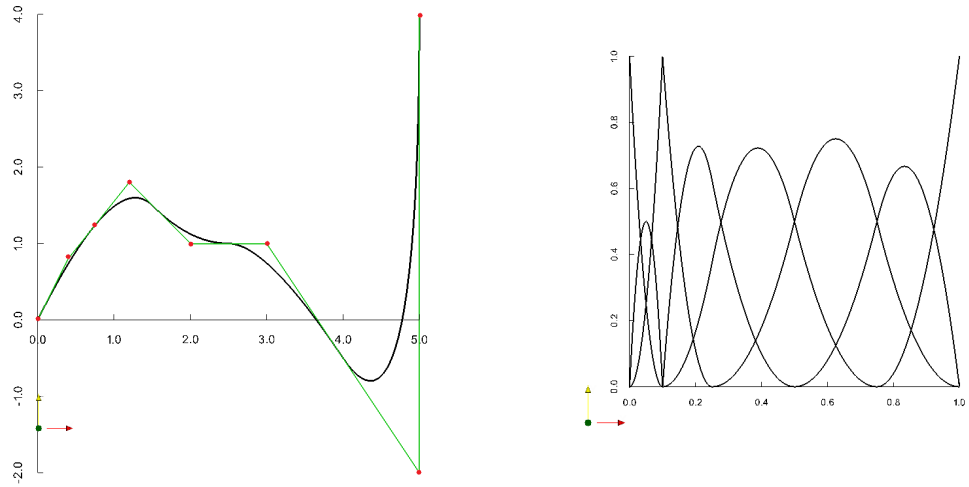
where $P_{i^1,\ldots,i^d}$ are the control points and $n^1, \ldots, n^d$ are the numbers of basis functions in the given direction. The resulting B–spline is defined over the Cartesian product of the d one dimensional domains

$$[u^1, \ldots, u^d] \in [u^1_{p^1}, u^1_{n^1+1}] \times \ldots \times [u^d_{p^d}, u^d_{n^d+1}].$$

The control points form the so called control mesh which is the higher dimensional generalization of the control polygon concept.



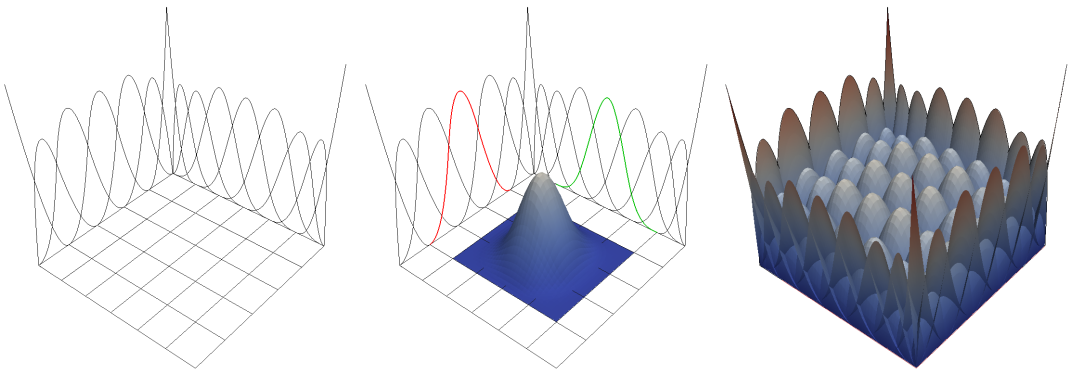
(a)



(b)

Figure 2.2: We performed two knot insertions using the knot $\bar{u} = 0.1$ on the curve presented in Figure 2.1. Every inserted knot introduces a new control point, see (a) and (b). Furthermore, if the multiplicity of a knot equals to the degree of the spline curve, the evaluated point lies on the curve and it subdivides the curve into two parts (b) as described in Example 2.1.2.

Example 2.1.3. Figure 2.3 shows a single basis function (b) of the two dimensional B–spline basis (c). It is obtained by the tensor-product construction from the highlighted basis



(a) (b) (c)

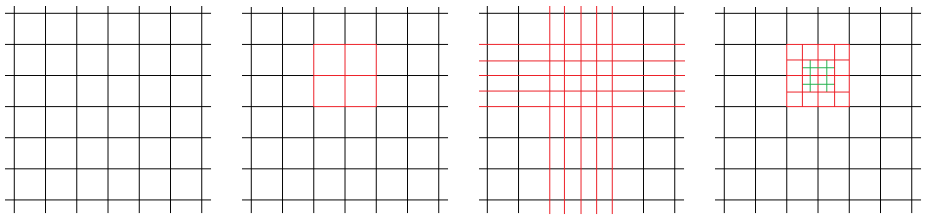
Figure 2.3: The two univariate B–spline bases (a) of degree 2 and 3 used for the construction of the tensor-product B–spline basis (c). Part (b) shows a selected basis function which was created by applying the tensor-product construction on the two univariate functions (highlighted by red and green). Part (c) depicts all tensor-product basis functions defined by the knot vectors mentioned in Example 2.1.3.

functions of the two univariate bases of degree 2 and 3 defined over the knot vectors $U^1 = \{0, 0, 0, 0.2, 0.4, 0.6, 0.8, 1, 1, 1\}$ and $U^2 = \{0, 0, 0, 0, 0.167, 0.334, 0.5, 0.667, 0.833, 1, 1, 1, 1\}$, respectively.

Naturally, we can use the knot insertion algorithm to obtain better control of the B–spline object. Similarly to the univariate case the control points of the refined B–spline can be computed as $\bar{P}_{i^1,\dots,i^d} = (\otimes R^j)P$ where $(\otimes R^j), j = 1, \dots, d$ is the tensor-product of the univariate transfer matrices.

2.2 (T)HB–splines

The simplicity of construction and the favourable properties of the B–splines not only made them a standard for computer–aided design libraries and modeling software, but also one of the main building blocks of isogeometric analysis [32]. Unfortunately, the tensor-product basis lacks the possibility of strictly localized refinement since, due to its construction, the



(a) initial grid (b) area of interest (c) knot insertion (d) hierarchical grid

Figure 2.4: Adaptive refinement of an initial tensor-product grid (a) with respect to a localized region (b) may be achieved by avoiding a propagation of the refinement due to the tensor-product structure (c) through a hierarchical approach (d).

refinement propagates through the whole mesh, as demonstrated in Figure 2.4(a–c). One possibility to overcome this drawback is to use the *hierarchical B–splines* (HB–splines), where a sequence of nested tensor-product B–spline bases and a sequence of nested domains are used to create strictly localized refinement as depicted in Figure 2.4(d). In this section we define the standard hierarchical B–spline basis, first introduced by Kraft in [35, 36], and discuss its main properties. As we will see in Section 2.2.2, HB–splines do not inherit the partition of unity property from the underlying set of tensor-product splines. To address this issue we introduce the *truncation mechanism*. By combining this technique with the HB–spline construction we obtain the so called *truncated hierarchical B–spline basis*, as described in Section 2.2.3.

2.2.1 The hierarchical setting

In the following sections we define an adaptive extension of the standard tensor-product B–spline construction based on a sequence of N nested multivariate B–spline spaces $\mathcal{V}^\ell, \ell = 0, \ldots, N-1$, such that

$$\mathcal{V}^0 \subset \mathcal{V}^1 \subset \ldots \subset \mathcal{V}^{N-1}.$$

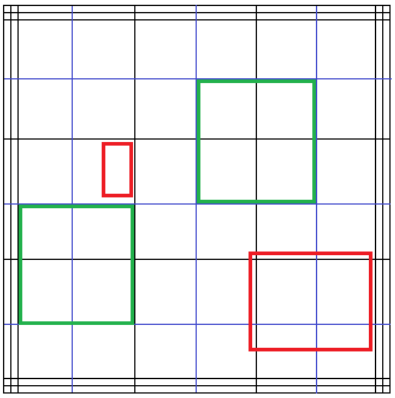


Figure 2.5: Domain with knot lines of level 0 and 1, shown in black and blue, and several examples of possible subdomains of level 1, depicted by green and red. The boxes highlighted by green satisfy our simplifying assumption, that the boundaries of Ω^ℓ have to be aligned with the knot lines of $\mathcal{B}^\ell$, whereas the red boxes does not follow this rule.

We assume that the coarsest spline space $\mathcal{V}^0$ is spanned by a normalized multivariate tensor-product B–spline basis $\mathcal{B}^0$. The finer bases $\mathcal{B}^\ell$ (spaces $\mathcal{V}^\ell$) are obtained by iteratively applying dyadic refinement, i.e. in every non–empty knot span $[u_i^j, u_{i+1}^j)$ we insert a knot $\bar{u}^j = u_i^j + (\frac{u_{i+1}^j - u_i^j}{2}), j = 1, \ldots, d$ by applying the knot insertion algorithm, as described in Section 2.1.3. This construction assures the nested nature of the spline spaces, as well as of the knot vectors.

Additionally, we consider a sequence of nested subdomains $\Omega = \{\Omega^\ell\}_{\ell=0,\ldots,N-1}$ such that

$$\Omega^\ell \subseteq \Omega^{\ell-1} \tag{2.3}$$

for $\ell = 1, \ldots, N-1$. For simplicity we assume that the boundary $\partial\Omega^\ell$ of these regions is aligned with the knot lines of the B–spline basis $\mathcal{B}^\ell$, as depicted in Figure 2.5. Each of these subdomain Ω^ℓ represents a region selected to be refined at level ℓ, and it is defined as a collection of d-dimensional cells created by the knot knot lines of $\mathcal{B}^\ell$, see Example 2.2.1.

Example 2.2.1. Figures 2.6 and 2.7 show three two-dimensional subdomain hierarchies:

- *rectangular* (refinement over rectangular–shaped regions),

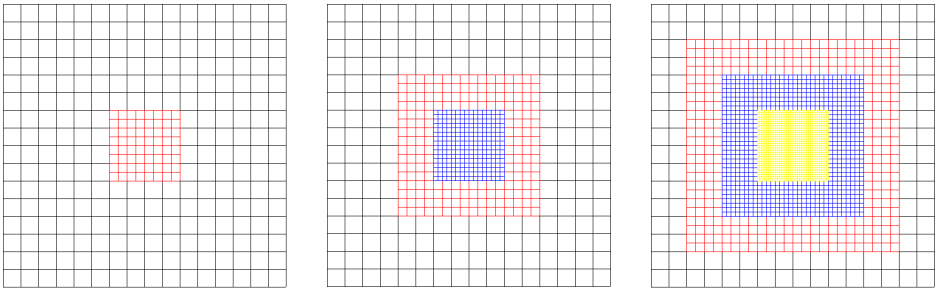


Figure 2.6: An example of hierarchical refinement over rectangular–shape regions where the central area of the mesh is always refined up to the maximum level of detail: two levels (left), three levels (middle) and four levels (right).

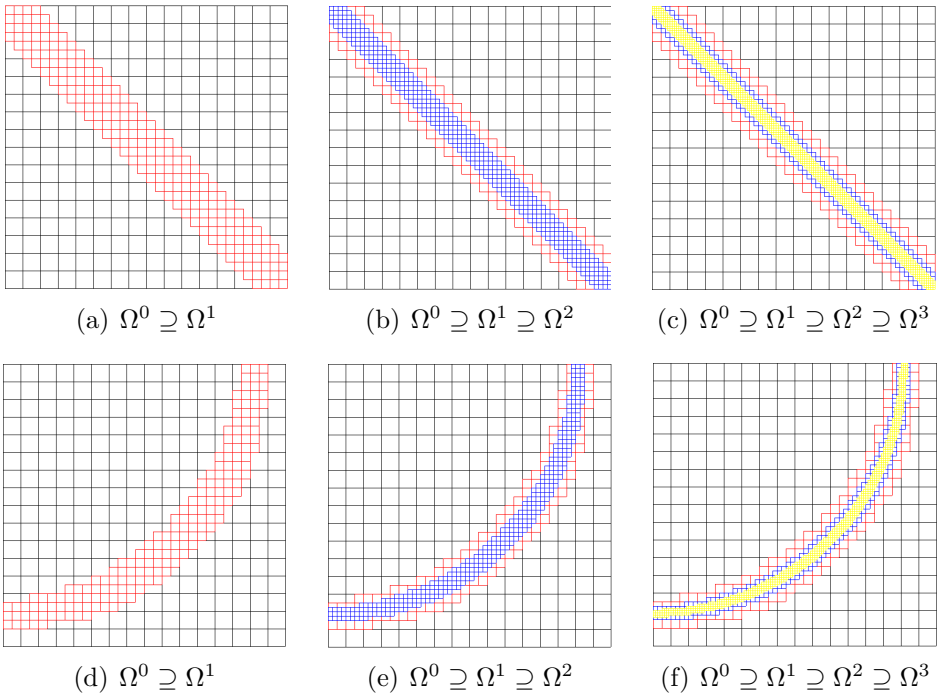
- *linear* (refinement along a diagonal layer),
- *curvilinear* (refinement along a curvilinear trajectory);

which will be used to demonstrate the performance of our algorithms and data structures described in Chapter 3.

By starting with an initial tensor-product configuration at level 0, the tensor-product grid associated with level $\ell+1$, defined by the knot lines of $\mathcal{B}^{\ell+1}$, is obtained by subdividing any cell of the previous level into four parts. Each subdomain Ω^{ℓ} is then defined as a collection of cells with respect to the grid of level ℓ, so that Equation 2.3 is satisfied. Figure 2.6 illustrates an example of hierarchical refinement over rectangular–shape regions where the central area of the mesh is always refined up to the maximum level of detail. In the other two mentioned subdomain hierarchies (see Figure 2.7) the area covered by the highest level subdomain is gradually decreasing.

2.2.2 Hierarchical B–spline basis

In Section 2.2.1 we described the necessary background machinery for the construction of the hierarchical B–splines introduced by Kraft in [35, 36] and later generalized in [20, 21]. In this part we define the HB–spline basis and we discuss some of its main properties.



(a) $\Omega^0 \supseteq \Omega^1$ (b) $\Omega^0 \supseteq \Omega^1 \supseteq \Omega^2$ (c) $\Omega^0 \supseteq \Omega^1 \supseteq \Omega^2 \supseteq \Omega^3$

(d) $\Omega^0 \supseteq \Omega^1$ (e) $\Omega^0 \supseteq \Omega^1 \supseteq \Omega^2$ (f) $\Omega^0 \supseteq \Omega^1 \supseteq \Omega^2 \supseteq \Omega^3$

Figure 2.7: Two nested sequences of subdomains for $d = 2$ — indicated as *linear* (top) and *curvilinear* (bottom). They satisfy relation (2.3) with respect to two (left), three (middle) and four (right) hierarchical levels.

Definition 2.2.1. The *hierarchical B–spline basis* $\mathcal{K}$ is recursively constructed as follows:

I Initialization:

$$K^0 = \{\beta \in \mathcal{B}^0 : \text{supp } \beta \neq 0\}$$

II Recursion:

$$K^{\ell+1} = K_A^{\ell+1} \cup K_B^{\ell+1}, \text{ for } \ell = 0, \ldots, n-2 \text{ where}$$

$$K_A^{\ell+1} = \{\beta \in K^\ell : \text{supp } \beta \not\subseteq \Omega^{\ell+1}\}$$

$$K_B^{\ell+1} = \{\beta \in \mathcal{B}^{\ell+1} : \text{ supp } \beta \subseteq \Omega^{\ell+1}\}$$

III Result:

$$\mathcal{K} = K^{N-1}$$

where $\operatorname{supp}\beta = \{\boldsymbol{x} : \beta(\boldsymbol{x}) \neq 0 \wedge \boldsymbol{x} \in \Omega^0\}$ and $\boldsymbol{x}$ is the vector of the euclidean coordinates of a point.

In the initialization step we select all basis functions from the B–spline basis $\mathcal{B}^0$ the support of which is completely contained in the domain Ω^0 as shown on Figure 2.8(b). The basis $K^{\ell+1}$ is constructed in the recursive step as a union of $K_A^{\ell+1}$ and $K_B^{\ell+1}$ where

- the part $K_A^{\ell+1}$ keeps all the basis functions from the previous iteration the support of which is not completely contained in $\Omega^{\ell+1}$ (see Figures 2.8(e) and (g)),
- $K_B^{\ell+1}$ contains all basis functions from $\mathcal{B}^{\ell+1}$ the support of which is completely contained in the subdomain $\Omega^{\ell+1}$ (see Figures 2.8(f) and (h)).

The resulting HB–spline basis $\mathcal{K}$ (Figure 2.8(i)) is obtained in the last iteration of the algorithm. The basis functions from $\mathcal{B}^l, l = 0, \ldots, N-1$ which are present in $\mathcal{K}$ are the so called *active* functions whereas the functions not considered in $\mathcal{K}$ are addressed as *passive* functions.

From the Definition 2.2.1 we can see that the HB–basis functions maintain most of the properties of the standard B–splines, like for example non–negativity and local support. Less obvious may be the linear independence of the basis functions.

Lemma 2.2.1 ([53])**.** *The functions in $\mathcal{K}$ are linearly independent.*

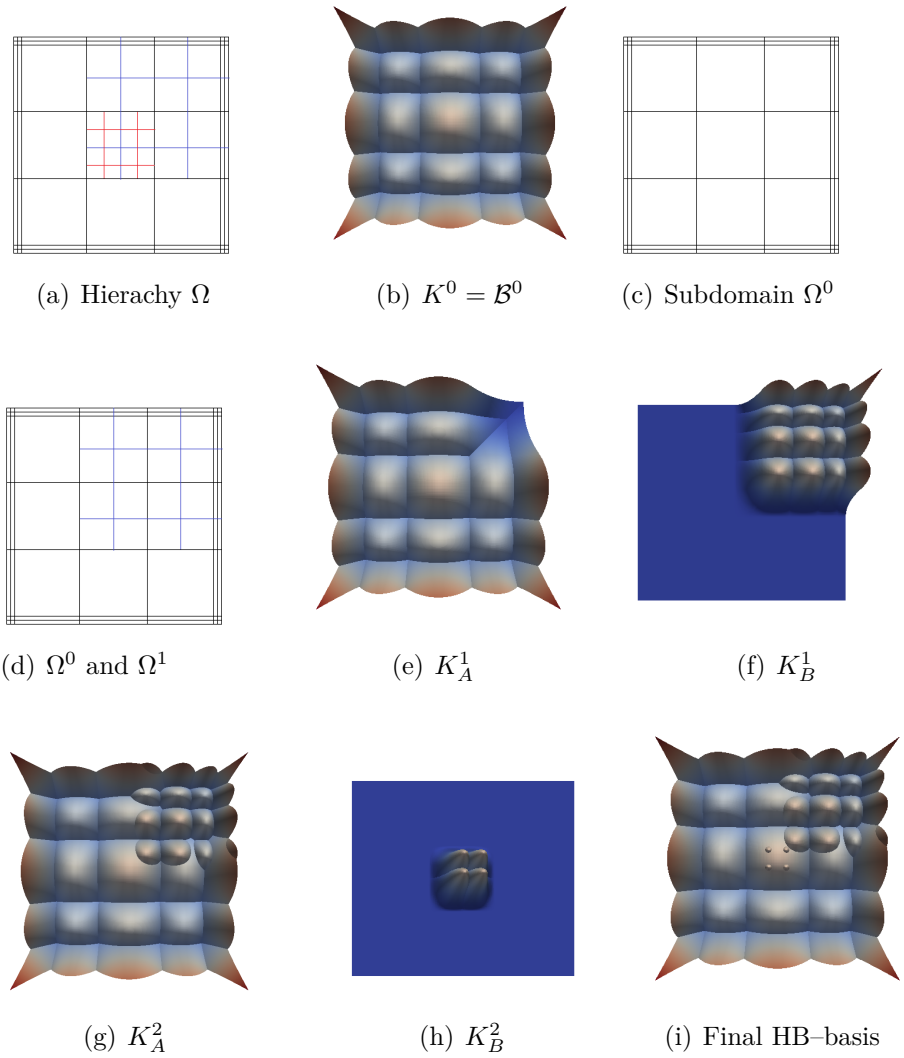
Proof. The sum

$$0 = \sum_{\beta \in \mathcal{K}} d_\beta \beta$$

can be re-arranged as

$$0 = \sum_{\beta \in \mathcal{K} \cap \mathcal{B}^0} d_\beta \beta + \sum_{\beta \in \mathcal{K} \cap \mathcal{B}^1} d_\beta \beta + \ldots + \sum_{\beta \in \mathcal{K} \cap \mathcal{B}^{N-1}} d_\beta \beta.$$

As we know, the basis functions in a B–spline basis are always linearly independent. Considering this fact we can easily see that functions $\beta \in \mathcal{K} \cap \mathcal{B}^0$ are linearly independent since they are a subset of $\mathcal{B}^0$. Taking into account that only these functions are non–zero



(a) Hierachy Ω (b) $K^0 = \mathcal{B}^0$ (c) Subdomain Ω^0

(d) Ω^0 and Ω^1 (e) K^1_A (f) K^1_B

(g) K^2_A (h) K^2_B (i) Final HB–basis

Figure 2.8: The hierarchical subdomain structure with three levels of refinement (a) is used for the construction of the hierarchical basis (i). During the initialization all basis functions from $\mathcal{B}^0$ (b), defined on Ω^0 (c), are selected. During the first iteration 4 basis functions, covered by Ω^1 (blue area on (d), are removed from K^0, creating K^1_A (e). These 4 basis functions are replaced by a set of basis functions from the refined basis, which are completely contained in Ω^1 (K^1_B) (f). In the second iteration no basis functions are removed (g). Four basis functions from the finest level (h) are added, creating $K^2 = \mathcal{K}$.

on $\Omega^0 \setminus \Omega^1$, and using their local independence, we can conclude that $d_\beta = 0$ for $\beta \in \mathcal{K} \cap \mathcal{B}^0$. Considering all next sums in the sequence, we may see that except the functions already

considered in previous sums, i.e. $\mathcal{K} \cap \mathcal{B}^0, \ldots, \mathcal{K} \cap \mathcal{B}^{\ell-1}$, only the functions in $\mathcal{K} \cap \mathcal{B}^\ell$ are non–zero on $\Omega^\ell \setminus \Omega^{\ell+1}$. Using a similar argument as for $\beta \in \mathcal{K} \cap \mathcal{B}^0$, we know that functions $\beta \in \mathcal{K} \cap \mathcal{B}^\ell$ are linearly independent. This implies that $d_\beta = 0$ for $\beta \in \mathcal{K} \cap \mathcal{B}^\ell$ with $\ell = 1, \ldots, N-1$. □

The main drawback of the HB–splines is the loss of the partition of unity and consequently of the convex hull property as well. As discussed in [53], these features can be restored by proper scaling of the basis functions with non–negative weights. These weights w_β can be computed using the fact that the constant function 1 belongs to the span of $\mathcal{K}$ (follows from Lemma 5.1.1), and thus we can write

$$1 = \sum_{\beta \in \mathcal{K}} w_\beta \beta.$$

Consequently, we can define a weighted basis function β_w for every basis function β in $\mathcal{K}$ such that $\beta_w = w_\beta \beta$. The normalized hierarchical B–spline $\mathcal{K}_w$, which forms a partition of unity, is then defined as follows:

$$\mathcal{K}_w = \left\{ \beta_w = w_\beta \beta : \beta \in \mathcal{K} \wedge 1 = \sum_{\beta \in \mathcal{K}} w_\beta \beta \right\}.$$

However, without considering additional restrictions on the size of the subdomains Ω^ℓ, we may encounter hierarchical configuration where some of the coefficients associated to the basis functions in higher refinement levels are zero, thus the contribution of the refined functions is canceled. To avoid the problem described above, we may use the *truncated hierarchical basis* (THB–spline basis), where the overlap of the supports of the basis functions is significantly reduced by the so called truncation process.

2.2.3 Truncated Hierarchical B–spline basis

In this section we introduce the *truncated hierarchical B–spline basis* which, compared to the HB–spline basis, does not only reduce the support of overlapping basis functions, but also

recovers the partition of unity property automatically during its construction. Furthermore, THB–splines possess several additional advantages compared to Kraft's hierarchical basis, as discussed in [21].

The truncation mechanism

The main idea behind this improved hierarchical construction came from the refinable nature of the B–splines. This refineble nature allows us to represent a B–spline function of level ℓ as a sum of $(p+1)^d$ (assuming the dyadic refinement as described in Section 2.2.1) properly scaled functions of level $\ell+1$. Thanks to the refinable nature, any function $\tau \in \mathcal{V}^\ell$ can be represented with respect to the refined basis $\mathcal{B}^{\ell+1}$ of $\mathcal{V}^{\ell+1}$, as follows:

$$\tau = \sum_{\beta \in \mathcal{B}^{\ell+1}} c_\beta^{\ell+1}\, \beta, \qquad c_\beta^{\ell+1} \in \mathbb{R}. \tag{2.4}$$

The coefficients $c_\beta^{\ell+1}$ can be easily computed using the knot insertion algorithm described in Section 2.1.3.

The *truncation* of a basis function τ from level ℓ with respect to $\mathcal{B}^{\ell+1}$ and $\Omega^{\ell+1}$ (see Figure 2.9) is then defined as

$$\operatorname{trunc}^{\ell+1}\tau = \sum_{\beta \in \mathcal{B}^{\ell+1}, \operatorname{supp}\beta \not\subseteq \Omega^{\ell+1}} c_\beta^{\ell+1}\, \beta. \tag{2.5}$$

By applying the truncation mechanism during every recursive step of the HB–spline basis construction, we obtain the THB–spline basis.

Definition 2.2.2 ([20])**.** The *truncated hierarchical B–spline basis* $\mathcal{T}$ is recursively constructed as follows:

I Initialization:

$$T^0 = K^0 = \{\beta \in \mathcal{B}^0 : \operatorname{supp}\ \beta \neq 0\}$$

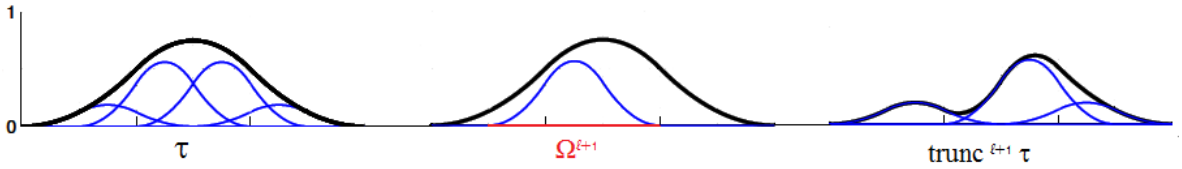


Figure 2.9: A B–spline basis function τ can be represented as a sum of $p+2$ basis functions from the refined basis (left) as already mentioned in Equation 2.4. If any of these finer functions is completely contained in a higher level subdomain, for example $\Omega^{\ell+1}$ (middle), we can remove its contribution from the representation of τ, as described in Equation 2.5, creating a truncated function $\text{trunc}^{\ell+1}\tau$ (right).

II Recursion:

$$T^{\ell+1} = T_A^{\ell+1} \cup T_B^{\ell+1}, \text{ for } \ell = 0, \ldots, n-2 \text{ where}$$

$$T_A^{\ell+1} = \{\text{trunc}^{\ell+1}\beta \in T^{\ell} : \text{supp } \beta \not\subseteq \Omega^{\ell+1}\}$$

$$T_B^{\ell+1} = \{\beta \in \mathcal{B}^{\ell+1} : \text{ supp } \beta \subseteq \Omega^{\ell+1}\}$$

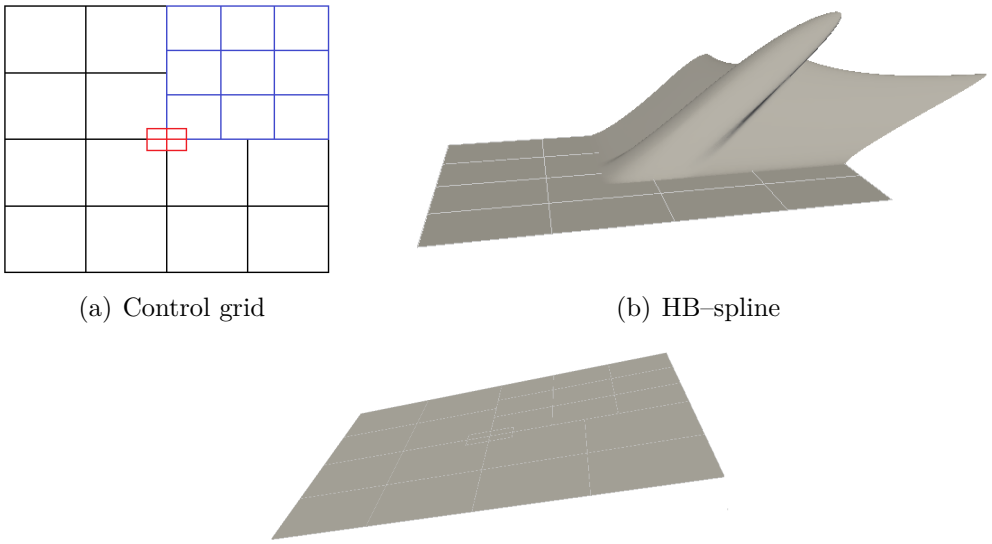
III Result:

$$\mathcal{T} = T^{N-1}$$

Similarly to the construction of the HB–spline basis, in the recursive step the $T^{\ell+1}$ is obtained as a union of $T_A^{\ell+1}$ and $T_B^{\ell+1}$ where

- $T_A^{\ell+1}$ keeps all the basis functions from the previous iteration, the support of which is not completely contained in $\Omega^{\ell+1}$, and applies the truncation mechanism on them (see Figures 2.11(e) and (g)),
- $T_B^{\ell+1}$ contains all basis functions from $\mathcal{B}^{\ell+1}$, the support of which is completely contained in the subdomain $\Omega^{\ell+1}$ (see Figures 2.11(f) and (h)).

The truncated hierarchical basis does not only maintains the main properties of B–splines such as non–negativity and linear independence, but compared to Kraft's hierarchical basis we also regain the partition of unity (convex hull property), see Figure 2.10. Additionally,



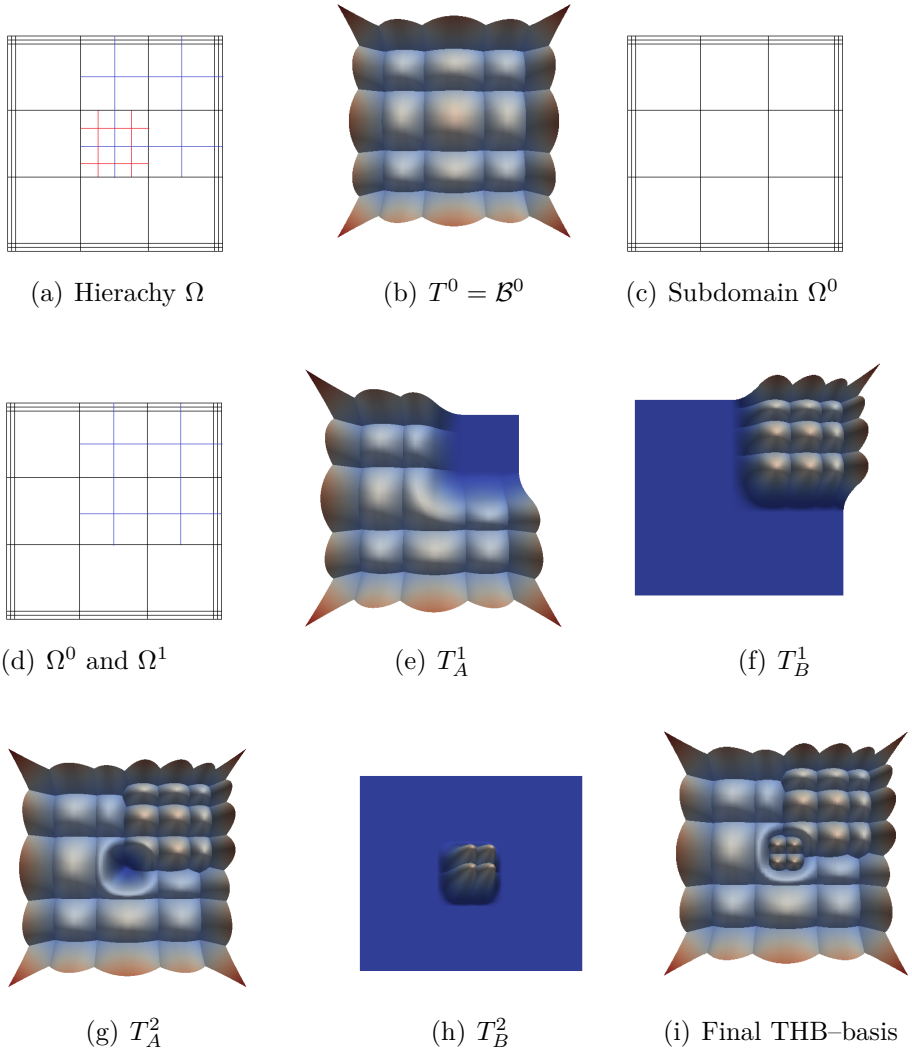
(a) Control grid

(b) HB–spline

(c) THB–spline

Figure 2.10: The HB–spline surface (b) and the THB–spline surface (c) have the same control grid (a) where all control points have the z coordinate set to 1. In the THB–spline case we obtain a flat surface, thanks to the partition of unity property of the underlying basis. In contrast, the HB–spline surface is distorted in the areas influenced by different refinement levels.

we obtain basis functions with smaller supports, what consequently improves the sparsity pattern of mass matrices used during numerical simulations [22] or of matrices in least–squares approximation methods [20].



(a) Hierachy Ω (b) $T^0 = \mathcal{B}^0$ (c) Subdomain Ω^0

(d) Ω^0 and Ω^1 (e) T^1_A (f) T^1_B

(g) T^2_A (h) T^2_B (i) Final THB–basis

Figure 2.11: The subdomain structure with three levels of refinement (a) is used for the construction of the truncated hierarchical basis (i). The initialization step, similarly to the case of HB–splines, select all basis functions from $\mathcal{B}^0$ (b), defined on Ω^0 (c). Again, we omit the 4 basis functions from T^0, covered by Ω^1 (blue area on (d), and apply the truncation on remaining basis functions, creating T^1_A (e). The omitted functions are replaced by basis functions from the refined basis, which are completely contained in Ω^1 (T^1_B) (f). In the second iteration no basis functions are removed. In (g) we apply the truncation on the basis functions in the T^1. Finally, four basis functions from the finest level (h) are added, creating $T^2 = \mathcal{T}$.

Chapter 3

Algorithms and data structures for THB–splines

The construction of a (truncated) hierarchical basis requires two main ingredients – a sequence of nested B–spline bases, and a sequence of nested subdomains. As described in Section 2.2.1, the sequence of nested bases can be constructed from an initial multivariate B–spline basis $\mathcal{B}^0$, defined by d knot vectors with $\kappa_i, i = 1, \ldots, d$ knot spans, and $\kappa_i + 1$ knots (possibly with higher multiplicity), by using dyadic refinement. The subdomain Ω^0 can be seen as a collection of $\kappa_1 2^{N-1} \times \ldots \times \kappa_d 2^{N-1}$ cells of the grid created by the knot lines of $\mathcal{B}^{N-1}$. Higher level subdomains are defined as a subset of cells in Ω^0, as we can see on Figures 2.6 and 2.7.

In the Section 3.1 we introduce an efficient way to represent the subdomain structure of hierarchical splines using a kD-tree data structure, and present the several important functions of the kD-tree. Afterwards, in Section 3.2, we describe a data structure which stores the information about the active and passive basis functions and provides fast access to it during the evaluation algorithm (see Section 3.3). Finally, we present several examples that show the time and memory consumption of the proposed algorithms in two-dimensional case.

3.1 Representation of the domain hierarchy

The main tasks of a data structure storing the subdomain hierarchy is to provide an efficient way to identify active and passive basis functions, as well as offering the possibility to dynamically change the structure of the nested subdomains. Over the years several different approaches have been developed to address this topic. For example in [17] a tree–like representation, where a given refinement level corresponds to a certain level of depth in the tree, was introduced. More recently an implementation of HB–splines using a tree data structure, the nodes of which represent the B–splines from different levels was presented in [3]. A further solution, presented in [44], consists of storing the data related to a knot span of a certain level, in particular the basis functions acting on it, in each node of the tree.

Our solution presented in this section, which uses a single kD-tree data structure storing all refinement levels, is a generalization of the concept presented in [33], into arbitrary dimension. In our previous work we considered a special case where $d = 2$ (THB–spline surfaces). This restriction of the dimension allowed us to use data structures specifically designed to work in the two-dimensional case, for example the quadtree data structure which was used to store the subdomain hierarchy Ω.

In this section we present our adapted kD-tree data structure which can store subdomain structures of arbitrary dimension. Additionally, we introduce the insertion algorithm which allows us to dynamically enlarge the subdomains Ω^ℓ, as well as to add new refinement levels. In the last part of this section we present the query functions which are used for an efficient identification of active basis functions of the hierarchical basis.

3.1.1 Structure of the kD-tree

The kD-tree is a binary tree data structure used for space partitioning. In our case we use it for creating a partition of the d-dimensional subdomain hierarchy. Every node of the kD-tree has the following structure:

```
struct kDnode{
  aabb  box;
  int   level;
  int   axis;
  int   position;
  *node a;
  *node b;
   };
```

kDnode	
aabb box	int level
int position	int axis
*kDnode a	* kDnode b

The axis–aligned bounding box `aabb box` is defined by the coordinates of two opposite corners (e.g. in the two-dimensional case by the lower left and the upper right corner), `level` defines the highest level in which the box is completely contained, and `a`, `b` are pointers to the two children of the node. The children are created by splitting the box stored in the given node into two parts, in the direction indicated by the value of the variable `axis` on the knot line which is defined by the `position`. An example of a kD-tree with a corresponding subdomain structure is presented in Figure 3.1.

3.1.2 Adding a box to a subdomain

As every subdomain Ω^ℓ can be defined as a collection of cells, we can see the enlargement (or insertion) of Ω^ℓ as insertion of an axis–aligned box into the current subdomain structure Ω.

Let B be an axis–aligned box defined by the coordinates of two opposite corners in a global index set I. The following recursive algorithm performs the insertion of B of a given level ℓ into a kD-tree:

Algorithm `INSERTBOX(box B, kDnode Q, int L)`

```
\\ box B is the box which will be inserted
\\ kDnode Q is the current node of the quadtree
\\ int L is the level to which we insert B
if B == Q.box then {
    Q.level = L
```

```
        visit all nodes in the subtree with root Q; if the level of a node is less than L,
            increase it to L }
    else {
        for child in {Q.a, Q.b} do {
            if child != null then {
                if B∩Q.box ≠ ∅ then INSERTBOX(B∩Q.box, child, L) }
            else {
                compute splitting value pos and axis axis
                Q.position = pos
                Q.axis = axis
                create the nodes Q.a and Q.b
                SETCHILD(Q.a, Q, 0)
                SETCHILD(Q.b, Q, 1)
                INSERTBOX(B∩Q.a.box, Q.a, L) }
                INSERTBOX(B∩Q.b.box, Q.b, L) }
                break()
            }
        }
    }
```

Algorithm `SETCHILD(kDnode C, kDnode Q, int side)`

```
\\ kDnode C is the current node of the kD-tree
\\ kDnode Q parent node to kDnode C
\\ int side defines which side of Q.box is used
compute box using Q.box, Q.axis, Q.position, side
C.box = box
C.level = Q.level
}
```

The value of `pos` is determined by one of such edges of the inserted box B that do not coincide with any of the edges of the `Q.box`.

Example 3.1.1. To explain the `INSERTBOX` algorithm, let us consider the subdomain hierarchy composed of three levels ($N = 3$). Two of them (level 0 and 1) are initially present

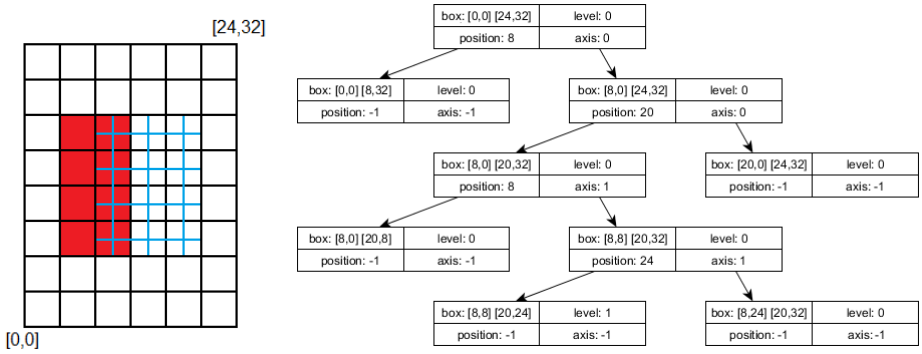


Figure 3.1: The initial subdomain structure (left) and the corresponding kD-tree (right) which stores the boxes related to levels 0 and 1 in the hierarchy. The box $B = [4, 8] \times [12, 24]$ (red) has to be inserted into the kD-tree at level 2.

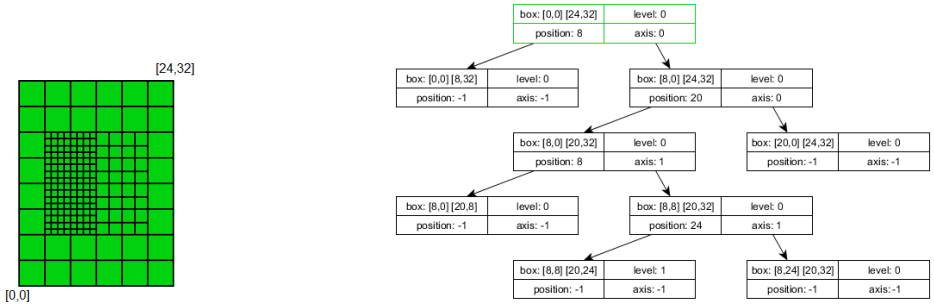
as shown on Figure 3.1 (left). The kD-tree structure related to the given Ω is shown on the right–hand side of Figure 3.1. The domain Ω^0 has $\kappa_1 = 6$ and $\kappa_2 = 8$ knot spans in x and y direction, respectively. The distance between two grid lines of Ω^0 is $2^{N-1} = 4$. The box $B = [4, 8] \times [12, 24]$ will be inserted into the hierarchy at level 2. The cells covered by B are depicted in red.

The execution of the algorithm is illustrated in Figure 3.2.

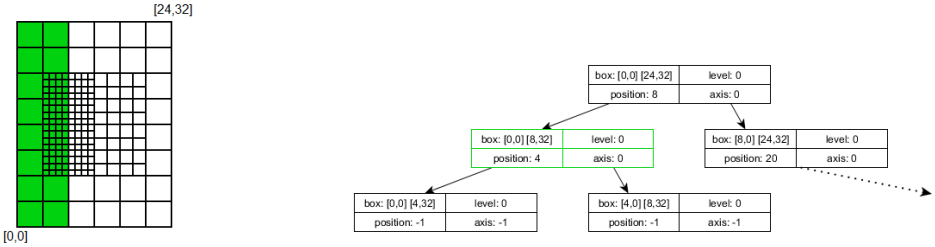
At each step, we highlight the current node `Q` and the corresponding box in the subdomain hierarchy (Figure 3.2, right and left column, respectively). The insertion starts at the root of the tree, where the box B is compared to the axis–aligned bounding box stored in the root. Since these two boxes are not the same, the level of the root remains unchanged.

Subsequently, we have to identify which boxes stored in the two children of the root overlap with B. In this case B is partly overlapped by both children of the root, and thus we have to split it into two smaller boxes $B_1 = [4, 8] \times [8, 24]$ and $B_2 = [8, 8] \times [12, 24]$.

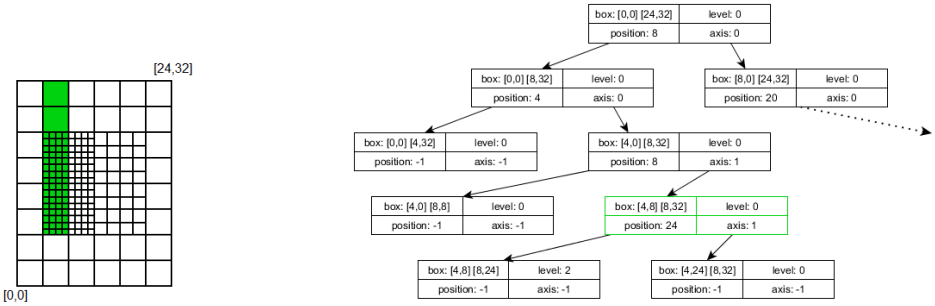
The recursive call of `INSERTBOX` is first applied to the left child of the root (Figure 3.2(b)). Similarly to the previous step, the box B_1 is not the same as the box in the considered node. Since this node has no children, we have to determine the position of the splitting line. To minimise the necessary number of splits, we use the extended edges of the inserted



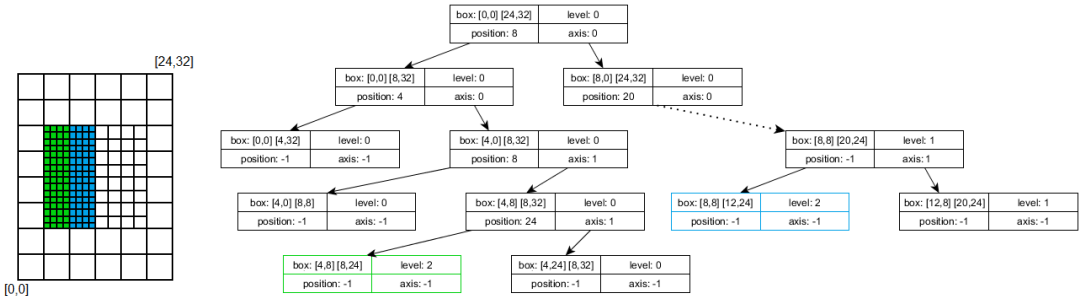
(a) First step



(b) Box considered after the first split



(c) Box considered after the third split



(d) Final kD-tree

Figure 3.2: Different steps performed by the `INSERTBOX` function to insert the box $B = [4, 8] \times [12, 24]$ of level 2 into the subdomain hierarchy in Figure 3.1 (as described in Example 3.1.1). In parts (b) and (c) we omit the right subtree of the structure due to space constrains.

box to determine their position. First, we use the edges where the two end points have the same x coordinate, i.e. lines parallel with the y axis, in our case the line $x = 4$. The `position` attribute of the current node is changed to 4 and the `axis` attribute indicating the direction of the split to 0. Consequently we create the two child nodes and recursively call the `INSERTBOX` on them.

The algorithm proceeds similarly in the next recursive call where the identified splitting line is $y = 8$, creating 2 new nodes with boxes $[4, 0] \times [8, 8]$ and $[4, 8] \times [8, 32]$ where the latter box contains the inserted box B_1.

As shown in Figure 3.2(c), by using $y = 24$ as the splitting line in the last step of the insertion of B_1 we obtain boxes $[4, 8] \times [8, 24]$ and $[4, 24] \times [8, 32]$. The first mentioned box coincides with B_1, therefore we set the level of the corresponding node to 2.

During the insertion of B_2 we proceed similarly. The resulting kD-tree structure is shown in Figure 3.2(d).

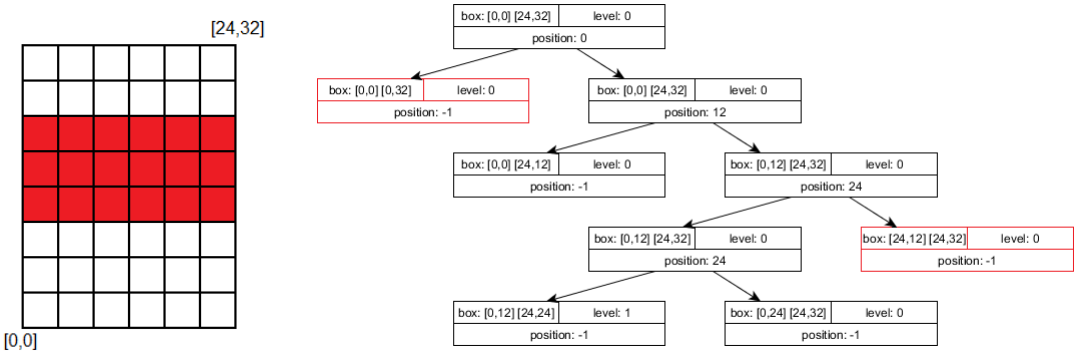
Clearly, the box to be inserted does not necessarily become a single node of the kD-tree, but may be stored in several nodes.

In contrast to the more standard construction of the kD-tree where the splitting directions are alternating, i.e. first split is in x direction than y,z,. . . ,x,y,z,. . . , in this case we first apply all splits in the x direction and them proceed to the next direction. On the one hand this forces us to store the direction of the split in the kD–tree structure, thus increasing the memory consumption, on the other hand, however, we can avoid degenerate splits which would occur in the standard case, as shown in Figure 3.3.

After each box insertion we perform a cleaning step, visiting all sub–trees recursively and deleting those where all nodes have the same level. This reduces the depth of the tree to a minimal value and optimises the performance of all algorithms that access the tree.

3.1.3 The kD-tree query functions

In order to analyse the position of a basis function with respect to different subdomains and to create the data structure for storing the information about the active/passive basis



(a) (b)

Figure 3.3: Inserting box $B = [0, 12] \times [24, 24]$, highlighted by red in (a), using the standard kD-tree construction approach, with alternating x,y z,... split directions, results into several degenerate splits, highlighted by red on (b), where one of the child stores a box degenerated into a line.

functions described in Section 3.2, we define four query functions on the kD-tree.

Given a box B defined as a collection of cells with respect to the tensor-product grid of level ℓ, the first query, `ISACTIVE`, returns true if

$$B \subseteq \Omega^\ell \quad \wedge \quad B \cap \Omega^i = \emptyset, \quad i > \ell. \tag{3.1}$$

Thus, if `ISACTIVE` returns true, then all the basis functions of level ℓ the support of which is completely contained in the box B are *active*, i.e. they are present in the hierarchical spline basis.

If the second query `ISPASSIVE` returns true, then all the basis functions of level ℓ the support of which is contained in the box B are *passive*, i.e. they are not present in the hierarchical spline basis. This is characterized by the following condition:

$$B \cap \Omega^\ell = \emptyset \quad \vee \quad B \subseteq \Omega^i, \quad \text{for some } i > \ell. \tag{3.2}$$

The third query `ISCONTAINED` returns the highest level ℓ with the property that Ω^ℓ

contains the box B.

The last query `HIGHESTLEVEL` returns the maximum value ℓ for which the following condition is satisfied:

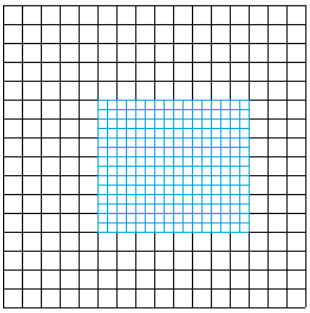
$$B \cap \Omega^{\ell} \neq \emptyset, \quad \ell = 0, \ldots, N-1. \tag{3.3}$$

All the four queries can easily be implemented with the help of the kD-tree structure described in Section 3.1.1. In particular, the structure of queries `ISACTIVE` and `ISPASSIVE` is similar – we visit the kD-tree until we find a node where the result of the query changes from to true to false. At that point, we can terminate the function and return false. On the other hand, queries `ISCONTAINED` and `HIGHESTLEVEL` require a complete visit of the kD-tree.

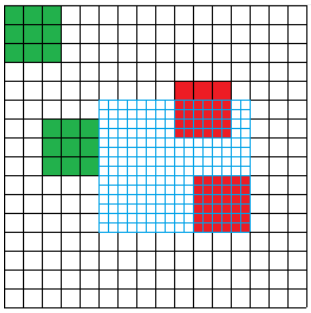
Example 3.1.2. Figure 3.4(b–d) shows the results of the four queries with respect to the subdomain hierarchy composed of two levels (level 0 and 1) shown in Figure 3.4(a) for four sampled boxes of level 0. Figures 3.4(b) and (c) display the results of `ISACTIVE` and `ISPASSIVE` for $\ell = 0$, respectively. The boxes highlighted in green correspond to a positive answer to the query, the boxes highlighted in red to a negative one. Finally, Figure 3.4(d) shows the results for the queries `ISCONTAINED` and `HIGHESTLEVEL`.

3.2 Data structures for active and passive functions

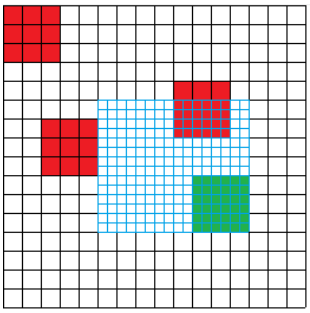
The most important information necessary for an efficient evaluation of the hierarchical basis is whether a given basis function is active or passive. To obtain this information purely by using the kD-tree query functions may be relatively slow, since in many cases we access the same basis functions several times during the computation, for example during the evaluation of THB–splines. To prevent repeated calls of queries for the same basis function, we store the information about the active/passive basis functions in a specialized data structure.



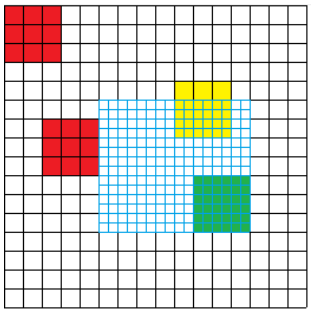
(a) level 0 and 1



(b) `ISACTIVE` for level 0



(c) `ISPASSIVE` for level 0



(d) results of `ISCONTAINED` and `HIGHESTLEVEL`

Figure 3.4: Results of the four query functions with respect to a subdomain hierarchy (a) with two levels. In case of `ISACTIVE` (b) and `ISPASSIVE` (c), the green/red boxes correspond to a positive/negative answer. `ISCONTAINED` (d) returns 1 for the green boxes and 0 for the red and yellow ones, whereas `HIGHESTLEVEL` returns 1 for the green and yellow boxes and 0 for the red ones.

3.2.1 Creating the list of active functions

At every level of the subdomain structure Ω we need to identify the tensor-product basis functions which contribute to the final hierarchical basis. The simplest and at the same time optimal way (with respect to the access time) is to store the basis functions of every level in a sorted list, organized by their tensor-product index in the given level, while assigning them values 1 or $0-1$ in case the given function is active, 0 if the function is passive. The sorted list for a given level ℓ is called a *list of active functions* X^ℓ. To optimize the creation of this structure, we considered two different approaches:

- the *one–by–one approach* where we determine the entries of the lists one by one by applying `ISCONTAINED` query to every single basis function,
- the *all–at–once approach* where we try to set as many values as possible in one single step. This requires a more sophisticated algorithm.

Using the all–at–once approach, we try to set many entries of the lists at the same time. In order to do this, the query functions are initially called for a box which covers the whole initial subdomain Ω^0. This box is the collection of all cells defined by the knot lines of $\mathcal{B}^0$. If we cannot decide if all the basis functions in the given box (represented by their supports in the parameter domain) are active or passive, we split the box into smaller parts and call the query functions for them separately. The `SETLIST` algorithm below creates a separate list of active functions for every level of the subdomain hierarchy.

Algorithm `SETLIST(qnode Q, seqlist X)`

`\\ kDnode Q` is the root of the kD-tree which stores the subdomain hierarchy

`\\ seqlist X` is the sequence of lists of active functions , i.e. `X[L]` is the list of active functions in level `L`

`for` all levels `L do {`

Create the index set `I` for all functions of level `L` acting on Ω^0. `I` is an axis–aligned box in an index space.

`SETBOX(I,X[L]) }`

The function `SETLIST` calls the algorithm `SETBOX`. When the answer active/passive cannot be given for the current call, the considered box is split into 2 disjoint axis–aligned bounding boxes using the split `axis` and `position` from the current kD-node. If the current kD-tree node is a leaf, we split `I` in half. The function `SETBOX` is then recursively applied to the 2 parts.

Algorithm `SETBOX(aabbis I, mat XL)`

`\\ aabbis I` is an axis-aligned box in an index space

`\\ list XL` is a list of active functions of level `L`

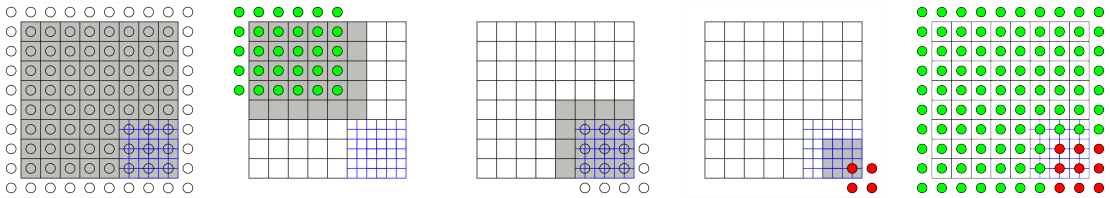


Figure 3.5: A subdomain hierarchy with two levels and some of the boxes `I` in index space (shown as circles) along with the associated bounding boxes `B` in parameter space (grey) considered by `SETBOX` when creating the list of active functions X^0 for this subdomain hierarchy (a–d). Active (green) and passive (red) functions of level 0 (e).

```
\\The level L is a global variable
Create the axis-aligned bounding box B covering all cells of level L which belong to
    the supports of functions with indices in I
if ISACTIVE(B, L) then {
    for all indices (i) in I do XL[i] = 1 }
elseif ISPASSIVE(B, L) then {
    for all indices (i) in I do XL[i]=0 }
elseif I is a single index (i) then {
    k =  ISCONTAINED(B, L)
    if k == L then XL[i]=1
    else XL[i]=0 }
else {
    split I into 2 disjoint axis-aligned bounding boxes I1 and I2
    Apply SETBOX to I1, I2 and XL }
```

Example 3.2.1. Figure 3.5 shows a subdomain hierarchy with two levels, consisting of Ω^0, shown in black, and a subdomain Ω^1 in the lower right corner, shown in blue. The four pictures (a–d) depict several index sets `I` (indicated by circles) and their associated boxes `B` (grey). These four index sets are examples of boxes considered by `SETBOX` when creating X^0 for biquadratic splines.

Initially, `SETBOX` considers the entire set of basis functions (a) and concludes that it has to be subdivided. The northwestern subset, obtained by a vertical split of the box stored in

the root of the kD-tree and an additional horizontal split, is shown in (b). Query `ISACTIVE` returns 1 for this box and therefore all the corresponding functions are active, no additional subdivision is needed. The southeastern subset S (c), has to be subdivided, since nor `ISACTIVE` nor `ISPASSIVE` queries can determine if all the considered functions are active or passive. Considering the southeastern subset of S (d) we can determine that all analyzed basis functions are passive, since query `ISPASSIVE` returns 1. The boxes not show in the example require several additional splits for correct classification of all considered basis functions, shown in (e).

3.2.2 Using sparse data structures

The representation of THB–splines in terms of lists of active functions, where an entry for every basis function is stored, allows a fast look-up during the evaluation process, as well as a simple update of the values when the underlying subdomain hierarchy changes. The shortcoming of this representation is the rather large memory consumption, especially in higher dimensions, which can exceed the available physical memory even for relatively small meshes and low numbers of levels. In fact, the memory consumption grows exponentially with the number of levels, since the number of basis functions in level $\ell + 1$ is approximately 2^d times bigger than in level ℓ (due to the dyadic refinement).

This problem can be solved by using a suitable data structure that explores the sparsity of the data. For this reason, we chose the to omit the entries which belong to the passive basis functions.

As detailed in Section 3.4, this simple trick significantly reduces the memory consumption of our approach, in many cases even by more than 90%, as detailed in Example 3.4.1. Besides, the cost paid for reduced memory requirements is only a small increase of the computational time (see Examples 3.4.2 and 3.4.3).

3.3 Evaluation algorithm for THB–splines

In addition to the list of active functions $\{X^\ell\}_{\ell=0}^{N-1}$, we consider another sequence of lists $\{C^\ell\}_{\ell=0}^{N-1}$ of the same size. Moreover, if $\chi_i^\ell = 0$, then $c_i^\ell = 0$, what means the coefficients of the passive functions are zero. This fact may be used to minimise the memory requirements of these data structures, as described in the previous section. These structures store the coefficients associated with the (active) basis functions in the representation of a spline function with respect to the truncated basis. The following simple algorithm performs the evaluation of a hierarchical spline function which is represented in terms of THB–splines.

The inputs of the evaluation algorithm are the sequences of lists $\{X^\ell\}_{\ell=0}^{N-1}$ and $\{C^\ell\}_{\ell=0}^{N-1}$, an array of length d containing the spline degrees of univariate B–splines in every direction, and the evaluation parameter. The output of `EVAL` is the value $f(\boldsymbol{u})$ of the given THB–spline.

The main idea of the algorithm lies in representing the B–spline basis functions acting on $\boldsymbol{u}$ (basis functions the support of which contains $\boldsymbol{u}$) from the coarse level as a sum of basis functions from the finest level (see Equation 2.5 and Definition 2.2.2). These fine basis functions can be then evaluated for example by the deBoor algorithm. This procedure of representation of coarse functions in terms of finer functions is done by using the knot insertion algorithm for dyadic refinement, followed by the adjustment of coefficients to obtain correct truncation.

Algorithm `EVAL(seqlist X, seqlist C, array D, vector U)`

\\ `seqlist X` is the sequence of lists of active functions, i.e. `X[L]` is the list of active functions in level `L`

\\ `seqlist C` is the sequence of coefficient lists associated with the spline function f, i.e. `C[L]` is the coefficient list of level `L`

\\ `array D` stores the spline degrees in all directions

\\ `vector U` stores the evaluation parameters

find `Lmin` as the minimum level of active basis functions acting on `U`

find `Lmax` as the maximum level of active basis functions acting on `U`

Identify the `(D[0]+1)`$\times\ldots\times$`(D[d-1]+1)` sub–set `M` of `C[Lmin]` which contains the coefficients of those B–splines of level `Lmin` that are non–zero at `U`

```
for L = Lmin+1 to Lmax do {
    Generate list S by applying one step of dyadic refinement to M
    Identify T, the sub–list of S, which contains the (D[0]+1)×...×(D[d-1]+1)
        coefficients of those B–splines of level L that are non–zero at U
    for each index i in T do {
        if X[L](i) == 1 then T(i) = C[L](i) }
    M = T }
return the value f obtained by applying de Boor's algorithm to M
```

In this algorithm, the sub–lists `M,S,` and `T` are at a certain level always accessed by global indices, that is indices with respect to the entire array of all tensor-product splines of that level. The following theorem clarifies the connection between the evaluation algorithm and the truncated hierarchical B–spline basis.

Theorem 3.3.1. *The value $f(\boldsymbol{u})$ computed by the* `EVAL` *algorithm is the value of a function represented in the THB–spline basis.*

This theorem can be proved by applying the algorithm to Kronecker–type coefficient data (where exactly one coefficient is non–zero and equals to 1).

The *cost* of the THB–spline evaluation algorithm `EVAL` is equal to $N - 1$ times the application of the B–spline subdivision rule, i.e. the knot insertion, plus the cost due to the standard de Boor's algorithm. Consequently it grows linearly with the number of levels and quadratically with the degree of the splines.

3.4 Results of the THB–spline implementation

Since the current implementation of THB–splines is a crucial part of the G+Smo library [25] and the code is highly optimised, we are not able to provide a comparison between the data structures discussed in Section 3.2.1.

Nevertheless, to highlight the importance of using optimized data structures, in this section we present the memory consumption and computational time results from our

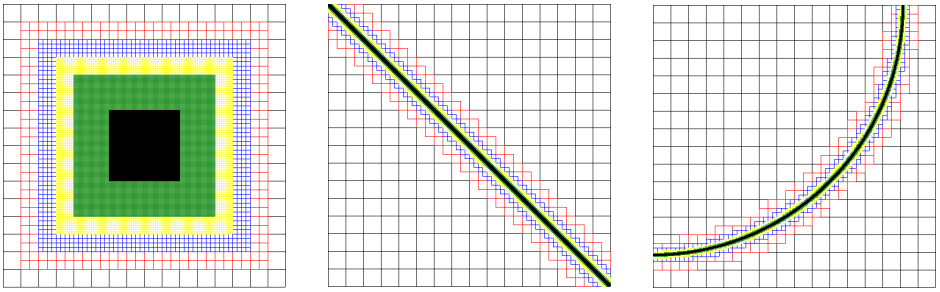


Figure 3.6: The three subdomain hierarchies considered in Example 3.4.1: rectangular (left), linear (middle) and curvilinear (right) refinement, all with six levels.

previous implementation, obtained for a special case of THB–spline surfaces discussed in our previous work [33]. This special case ($d = 2$) allowed us to use:

- quadtree data structure for the representation of the subdomain hierarchy
- matrices for storing the information about the active and passive basis functions (we call these matrices *characteristic matrices*).

We consider this data relevant, since in our experience the performance of our current implementation with kD-trees is comparable with the optimized implementation using quadtrees. This is supported by the fact that the insert and query functions on both structures have same asymptotic complexity, see [43].

Example 3.4.1. We compare the memory consumption of full characteristic matrices (not using sparse matrix representation) with the memory consumption of the matrices represented in the *compressed sparse column* structure [23] for the three subdomain hierarchies in Figure 3.6 (rectangular, linear, and curvilinear).

The experimental results in Figure 3.7 show that the memory needed by the sparse matrix data structure is considerably smaller than the one needed by the full matrix representation. Moreover, in case of the sparse matrix representation the memory consumption grows only *linearly* with the number of degrees of freedom (instead of exponentially with the number of levels). This is the optimal result, since a coefficient for each active basis function needs to be stored anyway.

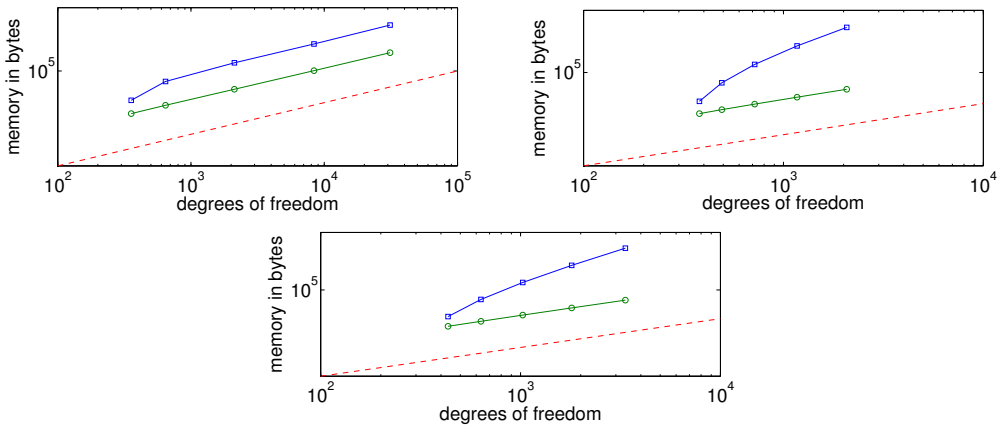


Figure 3.7: Memory needed to represent the characteristic matrices without (blue) and with (green) the use of sparse data structures for different number of degrees of freedom related to the square (top left), the circle (top right) and the line refinement (bottom). The dashed red line has slope 1 and indicates linear growth.

We also observe a difference between the results related to the rectangular–shaped refinement and related to the linear and curvilinear case. The reason is the different nature of the refinement procedures. In the linear and curvilinear case the refined area is reduced at each new level and the coarser levels do not change (see Figure 2.7). In the rectangular case the refined area of the highest level is constant and the size of the lower level subdomains increases (see Figure 2.6). Thus, in this case using the sparse data structure does not decrease the *order* of memory consumption, as the number of degrees of freedom grows exponentially with the number of levels.

The next example analyses the influence of using the sparse data structures to the time needed to evaluate the multilevel spline functions using the algorithm `EVAL`.

Example 3.4.2. Figure 3.8 visualizes the distribution of the computation times needed to evaluate the multilevel spline function at 1000 points with (blue bars in the plot) and without (red bars in the plot) the use of sparse data structures for the linear refinement shown in Figure 3.6. Two facts can be observed:

- the evaluation time does not depend significantly on the location of the point with

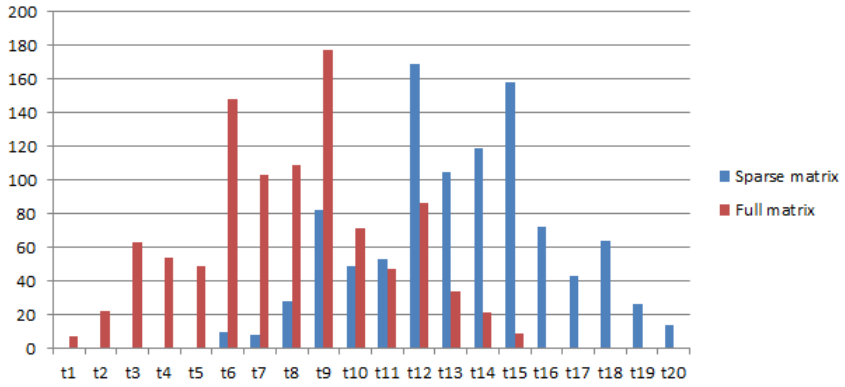


Figure 3.8: The labels t1,. . . ,t20 on the horizontal axis represent uniform time intervals between the minimal (0.153 ms) and the maximal (0.195 ms) time needed by the evaluation algorithm. The vertical axis indicates the number of points the evaluation time of which falls into these intervals.

respect to the subdomain hierarchy,

- using the sparse data structure increases the evaluation time only by a very small amount.

Note that the evaluation times in this example vary between 0.153 and 0.195 milliseconds.

Finally, we analyze the relation between evaluation time and the number of levels in the hierarchy.

Example 3.4.3. We consider the curvilinear refinement shown on the right–hand side of Figure 3.6. Figure 3.9 compares the evaluation times of the spline function with the curvilinear refinement for 10,000 parameters obtained by using either the full or the sparse matrix representation. We may note that the computational time grows *linearly* with the increasing level of refinement for both representations, with a small overhead on the side of the sparse data structure. The presented values do not include the time necessary for creating the corresponding data structures, only the evaluation algorithm `EVAL` is considered.

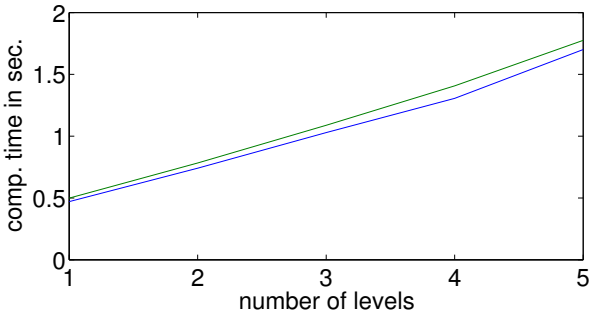


Figure 3.9: The computational time needed to evaluate the multilevel spline function at 10,000 points for curvilinear refinement with various numbers of levels with (green) and without (blue) using the sparse data structure.

Chapter 4

CAD model reconstruction

The idea to use hierarchical splines for reconstruction of point data is not novel. It has already been considered in [55] where triangular Bézier patches have been used to approximate measured data. Furthermore, it has been used in [18, 19, 24], where the standard hierarchical B–splines introduced by Kraft are considered. More recently a basic framework for approximation with THB–splines was also presented in [20]. This basic framework however does not provide all necessary elements for a successful surface reconstruction in real world applications, thus we expand it by adding a smoothing term which ensures that the system of equations solved in the least-squares approximation method is never singular. Furthermore, we investigate two different refinement strategies and their impact on the resulting surface.

The examples presented in Sections 4.3 and 4.4 show that the proposed hierarchical fitting scheme outperforms standard tensor-product B–spline approximations, not only with respect to a reduced number of degrees of freedom (control points), but also by the quality of the computed solution. In the presented work we focus only on surface reconstruction, nevertheless our method can be generalized to any dimension.

4.1 Least–squares approximation

Our goal is to create a surface which approximates a given set of measured data by computing a least-squares approximation

$$\mathbf{s}(u,v) = \sum_{\ell=0}^{N-1} \sum_{\mathbf{i}\in\mathcal{A}^\ell} \boldsymbol{c}_{\mathbf{i}}^{\ell}\, \tau_{\mathbf{i}}^{\ell}(u,v), \quad (u,v) \in [0,1]^2 \tag{4.1}$$

where $\tau_{\mathbf{i}}^{\ell}$ are the THB–spline basis functions, N is the number of levels in the subdomain hierarchy, and $\mathcal{A}^\ell$ is the index set of all active functions of level ℓ. The index set $\mathcal{A}^\ell$ corresponds to the basis functions stored in the list of active functions X^ℓ. We consider the given data

$$\boldsymbol{x} = (\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)}) = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m)^T \in \mathbb{R}^{m\times 3}$$

where $\boldsymbol{x}_i = (x_{i1}, x_{i2}, x_{i3})$ are the Cartesian coordinates of the m measured (or sampled) points with associated parameter values

$$(u_1, v_1), \ldots, (u_m, v_m) \in [0,1]^2.$$

The parameter values used in all our industrial examples are generated by the standard parametrisation methods described in [14, 16], while for the synthetic examples we simply applied the parameter values used for computing the sampled data.

A typical industrial data set which contains points on a fillet of a turbine blade is shown in Figure 4.1. As we can observe on the right–hand side of this figure, the obtained parameter distribution is highly non–uniform, mainly due to the shape of the fillet and the fact that our parameter domain is simply the unit square. As we will see later in Section 4.3.1, the non–uniformity makes it difficult to deal with this type of data when using standard tensor-product spline representations.

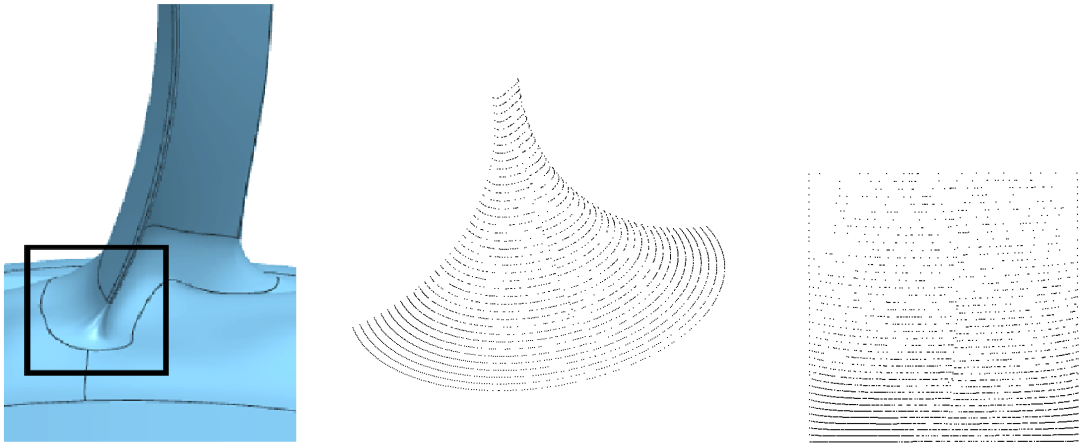


Figure 4.1: A turbine blade with the highlighted area of the fillet (left). The measurements of the fillets often produce non–uniformly sampled data sets where the upper part has significantly less measured data points than the bottom (middle). The corresponding distribution of parameter values is shown on the right.

We are looking for a vector of control points (coefficients)

$$\mathbf{c} = (\ldots, \boldsymbol{c}_{\mathbf{i}}^{\ell}, \ldots)^T \in \mathbb{R}^{n\times 3},$$

where n is the number of active basis functions from all levels, for the THB–spline representation (see Equation 4.1) that minimises the objective function

$$F(\mathbf{c}) = \sum_{k=1}^{m} ||\mathbf{s}(u_k, v_k) - \boldsymbol{x}_k||^2 + \lambda J(\mathbf{c}). \tag{4.2}$$

The term $J(\mathbf{c})$ is the smoothing (or regularization) term and λ is a positive weight that controls the influence of the smoothing term. Without the regularization term, the linear system obtained from F could easily become singular, e.g. when the number of local degrees of freedom in a certain region of the surface exceeds the number of available data points in that region. Also, as shown in Section 4.4.2, the higher values of λ increase the *fairness* of the approximating surface while they simultaneously increase the approximation error. For this reason we have to pay particular attention to the choice of the appropriate value of λ.

More precisely, as the smoothing term, we use the thin plate spline energy [2, 15]

$$J(\mathbf{c}) = \int\int_{[0,1]^2} (||\partial_{uu} s||^2 + 2\,||\partial_{uv} s||^2 + ||\partial_{vv} s||^2) \mathrm{d}u\mathrm{d}v \tag{4.3}$$

where $\partial_{uu} s, \partial_{uv} s$ and $\partial_{vv} s$ denote the partial derivatives of the THB spline surface $\boldsymbol{s}$.

The solution of the optimisation problem can be computed by solving the sparse linear system

$$(A^T A + \lambda E)\mathbf{c}^{(i)} = A^T \boldsymbol{x}^{(i)} \qquad (i = 1, 2, 3)$$

for every column $\mathbf{c}^{(i)}$ of the matrix of coefficients. The k-th row of matrix A contains the values of the THB–splines at (u_k, v_k), i.e. $A = (\tau_{\mathbf{i}}^{\ell}(u_k, v_k))_{k,(\mathbf{i},\ell)}$, and the matrix E is contributed by the regularization term. More details about the matrices A and E and their efficient assembly is provided in the next section.

4.1.1 Assembling the system

The matrix $A^T A$ has the elements

$$a_{(\mathbf{i},\ell),(\mathbf{i}',\ell')} = \sum_{k=1}^{m} \tau_{\mathbf{i}}^{\ell}(u_k, v_k)\, \tau_{\mathbf{i}'}^{\ell'}(u_k, v_k),$$
$$(\ell, \ell' = 0, \ldots, N-1;\ \mathbf{i} \in \mathcal{A}^{\ell};\ \mathbf{i}' \in \mathcal{A}^{\ell'})$$

where $\mathcal{A}^{\ell}$ and $\mathcal{A}^{\ell'}$ are the index sets of all active functions at level ℓ and ℓ', respectively. The matrix A is similar to a mass matrix in numerical simulations. In our case, the evaluation points are the parameter values of the given data, whereas in the case of numerical simulations, the evaluation points are the Gauss nodes of the mesh elements. For an efficient assembly of the matrix $A^T A$ we proceed as follows.

First, we identify all THB–splines that contribute a non–zero value at (u_k, v_k),

$$\bigcup_{\ell=0}^{N-1} \{(\mathbf{i}, \ell) \mid \mathbf{i} \in \mathcal{A}^{\ell} \text{ and } \tau_{\mathbf{i}}^{\ell}(u_k, v_k) \neq 0\} \tag{4.4}$$

for every $k = 0, \ldots, m$. The computation of this index set starts at level 0. Subsequently we increase the level until the indices of all $(p+1)^2$ B–splines of level $\ell + 1$ that do not vanish at this point belong to the set of passive basis functions of level $\ell + 1$. The support of the THB–spline basis functions can have a relatively complicated shape due to the modifications of the support caused by the truncation (see for example Figures 2.11 (e) and (g) in the central area). For this reason we simplify the computation of this index set by considering the support of the original B–splines $\beta_{\mathbf{i}}^{\ell}$, which is an axis-aligned box in the parameter domain. The obtained set, see Equation 4.4, is then a superset of the relevant indices.

Second, we evaluate all THB–splines with indices in this set at (u_k, v_k). Third, we create and add the contributions to the matrix elements $a_{(\mathbf{i},\ell),(\mathbf{i}',\ell')}$ for all relevant index pairs. We may use the same strategy for the efficient assembly of the right–hand side of the linear system, as well.

The matrix E, generated by the smoothing term, with the elements

$$e_{(\mathbf{i},\ell),(\mathbf{i}',\ell')} = \int\int_{[0,1]^2} (\partial_{uu}\tau_{\mathbf{i}}^{\ell}\,\partial_{uu}\tau_{\mathbf{i}'}^{\ell'} + 2\,\partial_{uv}\tau_{\mathbf{i}}^{\ell}\,\partial_{uv}\tau_{\mathbf{i}'}^{\ell'} + \partial_{vv}\tau_{\mathbf{i}}^{\ell}\,\partial_{vv}\tau_{\mathbf{i}'}^{\ell'})\mathrm{d}u\mathrm{d}v$$

resembles the stiffness matrix in numerical simulation. It is assembled by using the Gaussian quadrature on the polynomial pieces of the truncated hierarchical spline functions.

To acquire the polynomial pieces necessary for its construction, we consider all cells (i.e. the Cartesian products of the knot spans) of level 0. These cells are then repeatedly split by inserting the knots from higher refinement levels. This procedure, similarly to the construction of matrix A, terminates when we find such a cell of level ℓ that the indices of all B–splines of the next level, that do not vanish on this cell, belong to the set of passive basis functions of level $\ell + 1$. This cell then defines a polynomial piece of the hierarchical spline function. Simultaneously, we collect those indices of the THB–splines that do not vanish on this cell in a list. We evaluate all THB–splines and their derivatives at the Gauss nodes of the cell and add the contributions to the matrix elements $e_{(\mathbf{i},\ell),(\mathbf{i}',\ell')}$ for all relevant index pairs. For the evaluation of the derivatives of the THB–splines we use a modified version of the evaluation algorithm described in Section 3.3 where instead of simply applying deBoor's

algorithm we evaluate the derivatives of the B–splines.

4.2 Refinement strategies

After each step of regularized least–squares approximation we perform refinement in the critical regions of the surface, that is in the areas with the highest approximation error, thus providing additional degrees of freedom (control points) and a higher resolution for the next iteration. The procedure is initialized by choosing an initial B–spline basis $\mathcal{B}^0$ and it terminates if one of the following criteria is reached:

1. $90 - 95\%$ of data points are approximated with an error below $\sigma = 10^{-6}$ (a typical requirement in high-end applications, such as turbine blades),
2. we arrive at the maximal number of iterations.

We investigated two different refinement strategies.

- The *absolute threshold (AT)* approach: the points where the error exceeds a given fixed threshold σ are marked for refinement.
- The *relative threshold (RT)* approach: a certain percentage of points with the largest errors is marked for refinement.

Both approaches select a set of indices that identify the points $\boldsymbol{x}_k$, and the parameter values (u_k, v_k), where the approximation error is the highest. The areas of the THB–spline surface corresponding to these points are then refined. For each index k we first compute the current refinement level of the associated point:

$$\max\{\ell \mid (u_k, v_k) \in \Omega^\ell\}.$$

This operation can be performed by using the query functions described in Section 3.1.3. Subsequently, we identify the cell of level ℓ that contains (u_k, v_k). This cell and a certain number of neighbouring cells (determined by the value of an extension parameter, see

Figure 4.2) is then added to the subdomain $\Omega^{\ell+1}$, provided that $\ell+1 \leq N-1$. This is done by using the `INSERTBOX` algorithm described in Section 3.1.2. Note that the refinement strategies do not necessarily increase the number of refinement levels of the THB–spline. When the number of refinement levels is unchanged, the already existing subdomains are usually enlarged.

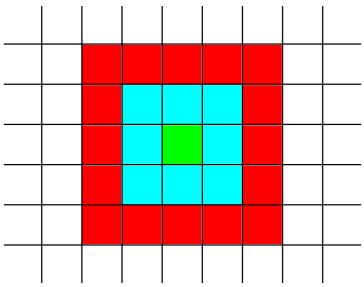


Figure 4.2: Extension of a selected cell (green) for extension parameters 1 (cyan) and 2 (red).

4.3 Incorporation of THB–splines into CAD systems

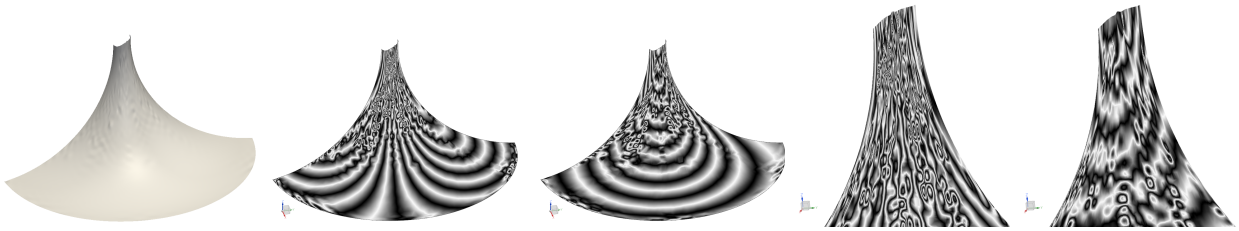
The process of converting a measured data set into a CAD object — often referred to as *geometry acquisition and reconstruction* — is a crucial part in certain industrial applications, for example, as in our case, in analyzing manufacturing tolerances with respect to their aerodynamic and structural mechanical impact. The introduced adaptive fitting framework with THB–splines improves the required number of degrees of freedom and the overall stability of the reconstruction process, while maintaining the accuracy of the standard fitting technique.

It also leads to a dramatic performance enhancement in related industrial applications. This is illustrated in Section 4.3.1 where the reconstruction process of a crucial part of an aircraft turbine blade is shown. Additionally, in Section 4.3.2 we present the export of THB–spline geometries into standard tensor-product B–splines, that provides a useful tool for integrating hierarchical spline representations into standard CAD software.

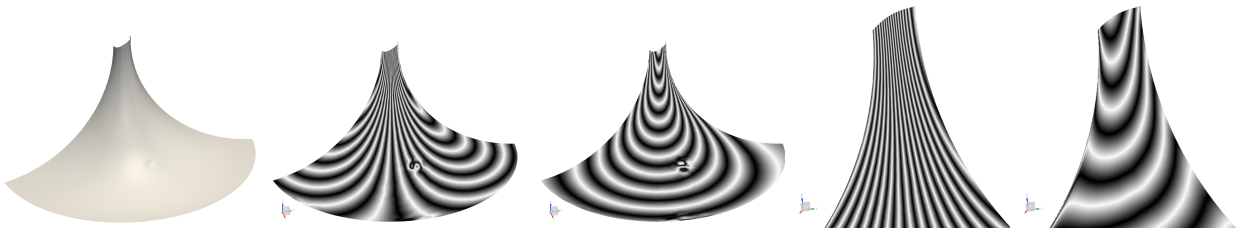
4.3.1 Adaptive geometry reconstruction

Industrial data sets are often generated by an optical measurement system producing a large amount of non–uniformly distributed points which describe the shape of the object. The non–uniform sampling and the strongly varying shape of the data cause several problems during the fitting procedure when using standard techniques. One of these problems is overfitting which causes oscillations of the resulting surface. This problem is well known and was already addressed in several publications, as for example in [12] or [31]. This limitation of the tensor-product structure can be eliminated by using the adaptive THB–splines fitting framework.

We investigated the reconstruction of the fillet part of a turbine blade as one of the challenging geometrical parts of an aero engine. Figure 4.1 shows the used point cloud which was parametrized by the technique described in [14]. The fully automatically reconstructed fillet geometry by using B–splines and THB–splines is shown in Figure 4.3. As the noisy reflection lines (see [27]) show, the tensor-product spline surface suffers from strong oscillations on its upper part, whereas the same region on the hierarchical spline surface is perfectly smooth. This distortion of the surface results from the fact that within the standard technique, no optimal number of degrees of freedom exists to avoid oscillations in the upper fillet part while generating an accurate fitting geometry on the lower fillet part. Note that the problem of oscillations cannot be solved with standard B–spline surfaces. Using higher values of the regularization parameter does not lead to an effective solution. Even if the amplitude and frequency of the oscillations may decrease while we increase the influence of the smoothing term, the unwanted oscillations will be present until the surface becomes planar, without obtaining the desired smooth curved surface. See Figure 4.4. For this reason, an adaptive scheme is needed to compute a flexible and accurate fitting by exploiting the possibility of identifying different levels of resolution. This demonstrates the superior behavior of the adaptive fitting framework with truncated hierarchical geometries in comparison with the use of standard tensor-product geometries.



(a) global refinement using B–splines and 1849 degrees of freedom



(b) local refinement using THB–splines and 475 degrees of freedom

Figure 4.3: Reconstruction of the fillet part with globally refined B–splines (top row) and with locally refined THB–splines (bottom row) using 1849 and 475 degrees of freedom, respectively. In both cases the regularization parameter and the error threshold are set to $\lambda = 10^{-9}$ and $\sigma = 10^{-6}$, respectively. The quality of the surfaces (leftmost plots) is visualized using reflection lines. The two rightmost plots in both rows show an enlarged view of the upper part of the fillet. Note that the small distortion of these lines in the central part of the fillet is caused by a measurement error in the provided data.

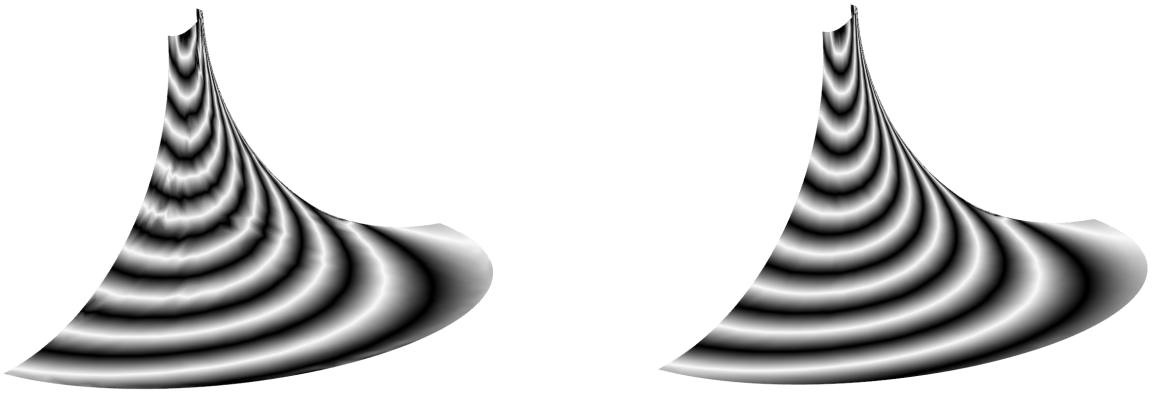


Figure 4.4: Reconstructed surface of the fillet using standard tensor-product B–splines and smoothing parameters $\lambda = 10^{-7}$ (left) and $\lambda = 10^{-5}$ (right). The oscillations in the upper part of the fillet are not completely removed by the increased smoothing term.

4.3.2 Conversion to tensor-product patches

Despite the advantages of THB–splines, the standard spline technology in current CAD libraries are tensor-product B–splines or, more generally, tensor-product NURBS. Therefore, a procedure for exporting THB–spline geometries into the standard format is needed.

The hierarchical construction of THB–splines offers a natural way to perform this operation by computing the coefficients of the tensor-product representation directly from the control points of the original THB–spline surface. To complete this conversion efficiently, we need to split the original geometry into several B–spline patches according to the different refinement levels of the THB–spline representation.

The algorithm `EXPORT` performs the conversion of a THB–spline geometry into several B–spline patches using the splitting techniques described below. It iterates through every level of the THB–spline and computes the so called rings which are defined as $\Omega^\ell \setminus \Omega^{\ell+1}$ where $\Omega^\ell, \ell = 0, \ldots, N$ are the subdomains of the THB–spline which we want to represent in B–spline form. Every ring can contain several connected components, see for example Figure 4.5(a) where the ring $\Omega^2 \setminus \Omega^3, \Omega^3 = \emptyset$ (corresponds to the red areas) has 2 connected components. These connected components are subsequently converted into boxes $b_0, \ldots, b_j$ using the splitting techniques described below. For every box $b_i, i = 0, \ldots, j$ obtained by

this procedure we apply the modified `EVAL` algorithm which takes into account all coarse basis functions acting on the given box and terminates at level ℓ. This modified evaluation is indicated as `EVAL*` in the algorithm below.

Algorithm `EXPORT(mat c, int N)`

`\\ mat c` are the THB–spline coefficients
`\\ int N` is the number of levels
`for l` from 0 to `N-1 do{`
create ring `r` $= \Omega^{\ell} \setminus \Omega^{\ell+1}$
create the list `R` of connected components of `r`
`for i` from 0 to `|R|-1 do{`
`boxes = SPLIT_TO_BOXES(R[i])`
`for j` from 0 to `|boxes|-1 do{`
Compute the coefficients of the restriction of the THB–spline surface to the box `boxes[j]` using `EVAL*` and export the obtained tensor-product B–spline surface.`}}}`

This algorithm relies on the `SPLIT_TO_BOXES` function that splits a connected component of one refinement level according to one of the following methodologies.

1. *Rectangular partition*: by using the B–spline representation of rectangular boxes completing the connected components `R[i]` (see Figure 4.5(b)).

2. *Smallest bounding box*: by using the B–spline representation of the smallest axis aligned bounding box covering the connected component `R[i]`. Its boundary curve is defined in the parameter domain of `R[i]` and is later used for trimming the tensor-product surface (see Figure 4.5(c) and (d)).

The rectangular partition approach may lead to a larger number of patches compared to the second approach, depending on the shape of the hierarchical domains. The number of additional degrees of freedom created by this approach is limited to the number of multiple copies of common control points between the adjacent patches. The second method leads to a small number of patches, equal to the number of connected components of rings of all

levels, however it increases the required memory storage as one needs to store additional "phantom" control points in the trimmed areas and the trimming curves themselves.

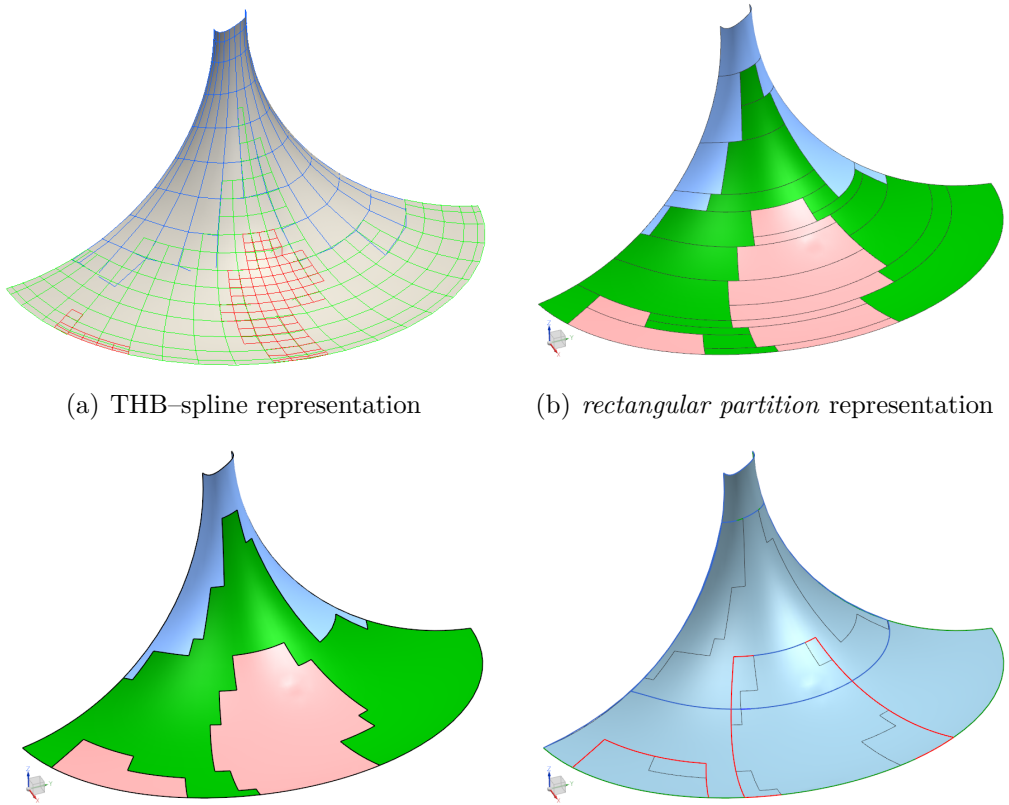
The domain structure of the THB–splines, stored in the kD-tree data structure described in Section 3.1, provides us naturally with a collection of boxes which necessary for the rectangular partition procedure. These boxes are stored in the leaves of the kD-tree. To optimize the number of patches, we have applied an additional step where adjacent areas of the same refinement level are joined together, creating larger rectangular boxes. The boxes stored in the kD-tree also allows us to compute the smallest rectangular box that covers the connected components of $\Omega^\ell \setminus \Omega^{\ell+1}$ for $\ell = 0, \ldots, N-1$, required for the smallest bounding box procedure.

Finally, we use the capabilities of a CAD system, in our case Parasolid$^{\text{TM}}$ by SIEMENS PLM Software [41], to bridge the gap between the THB–splines and the standard (commercial) applications. The geometric modeling kernel combines the geometric representations with a topological structure in order to handle trimmed entities and to build up large complex models based on several (connected) geometries. Thus, Parasolid can combine the generated B–spline patches into a single topological object referred to as a *sheet* without using any approximation. See Figure 4.5. This provides a straightforward and geometrically exact integration of THB–splines into standard industrial processes and applications.

In Figure 4.5 we present an example for the export of THB–splines into B–spline patches. The THB–spline surface representing the fillet (a) with 475 control points can be split either into 36 patches with 1788 control points (b), or into 4 trimmed patches with 1021 control points (c,d).

4.4 Numerical examples

Our tests were performed on several synthetic data sets created by uniform sampling of analytical functions, and on more challenging industrial data sets related to turbine blade parts (see Figures 4.6, 4.7 and Section 4.3). For all synthetic examples we used a sampling procedure which created a uniform grid with 10000 sampled points over the parameter



(a) THB–spline representation

(b) *rectangular partition* representation

(c) *smallest bounding box* representation

(d) bounding boxes and trimming curves for patches in (c)

Figure 4.5: Reconstructed fillet part geometry: the approximated THB–spline surface is defined by three refinement levels with a corresponding control grid of 475 control points (a). An optimal *rectangular partition* split into standard B–spline surfaces requires 36 patches with 1788 control points in total (b). The *smallest bounding box* split generates four B–spline patches with 1021 control points in total (c). The corresponding patch layout is shown in (d). The coloring of patches corresponds to different refinement levels. The size of the original data set is 3280 measured points.

domain. All tests were initialized with a THB–spline basis of size 8×8 and bi-degree (3,3), resulting in curvature continuous surfaces, as it is standard in demanding engineering applications.

The THB–splines and the adaptive fitting algorithm were implemented in `C++`. The presented tests were executed on PC (Intel XEON E31240 3.30 GHz, 16 GB RAM, 64 bit)

running SUSE Linux Enterprise Desktop 11. The errors presented in the following examples are computed as the distance between the data point $\boldsymbol{x}$ with parameters (u, v) and the corresponding point on the THB–spline surface $\boldsymbol{s}(u, v)$, i.e.

$$e = \sqrt{(x_1 - s_1(u,v))^2 + (x_2 - s_2(u,v))^2 + (x_3 - s_3(u,v))^2}$$

4.4.1 Comparison of the refinement strategies

The convergence speed and the resulting domain structure of the THB–spline fitting procedure strongly depends on the refinement strategy considered in the hierarchical approximation framework. In Section 4.2 we proposed two different approaches to identify the regions with higher errors: the absolute (AT) and the relative threshold (RT).

In Table 4.1 we compare the number of iterations and degrees of freedom (control points) of the absolute (AT) and the relative threshold (RT) refinement strategies for the *3 peak*

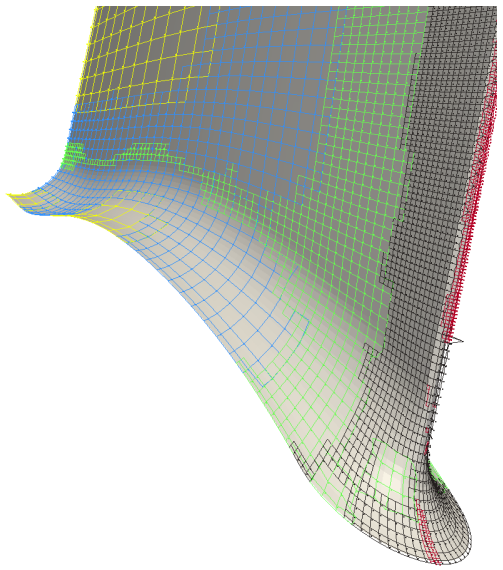


Figure 4.6: Turbine blade fillet represented by truncated hierarchical B–splines. The different meshes represent control points at different levels of the THB–spline hierarchy. The size of the original data set is 38240 points.

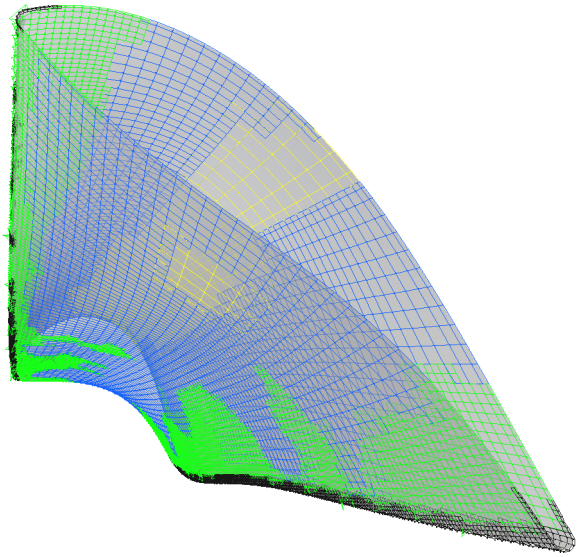


Figure 4.7: Adaptive airfoil model based on THB–spline representations. Note the different resolutions of the hierarchical meshes. The data set used for the reconstruction contains 31254 points.

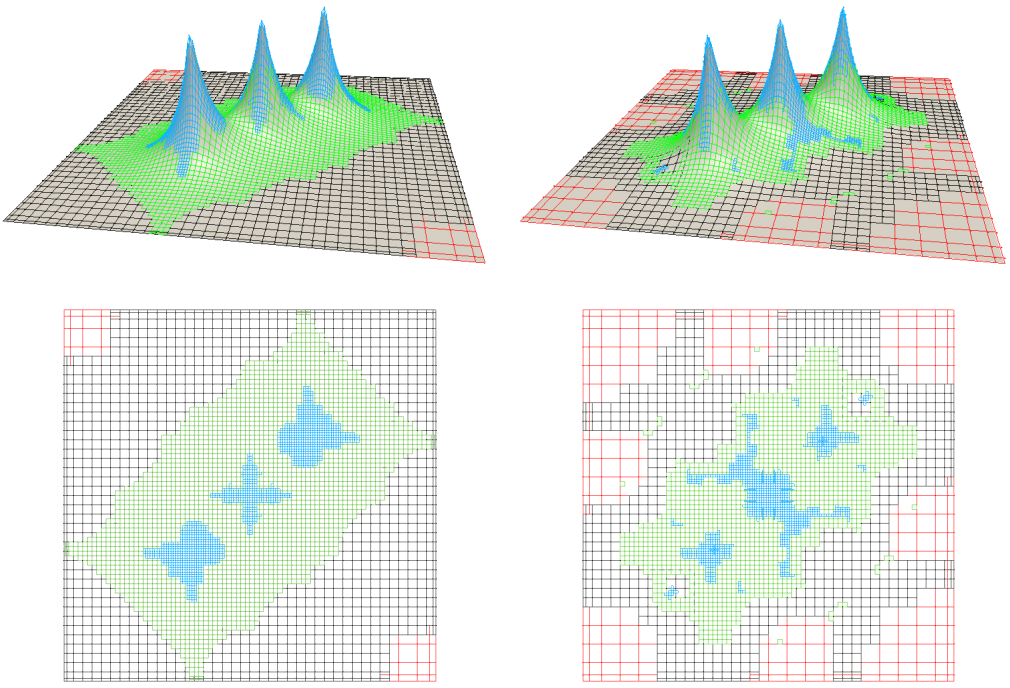
data set[1] (already considered in [20]), see Figure 4.8. The sampled data points are contained in an axis–aligned box of size $2 \times 2 \times 0.65$. The refinement procedure was terminated when at least 99% of the errors were below the threshold $\sigma = 10^{-6}$ (thereby even exceeding the standard industrial precision requirements mentioned in the Section 4.2), or when the number of iterations M reached 10. By setting the regularization parameter to $\lambda = 10^{-9}$, we obtain the number of iterations and degrees of freedom reported in Table 4.1. We may observe that, in general, even if both refinement strategies can achieve the same level of accuracy, the AT strategy requires less iterations. Furthermore, the choice of the optimal refinement percentage in case of the RT strategy is not easy and may depend both on the shape and the distribution of the data in the data set.

In addition, as shown in Figure 4.8, the RT strategy tends to create a more fractal–like domain structure. This fact has a negative influence on the complexity of the final surface

[1]The data is computed by uniform sampling points of the function $f(x,y) = 1.5(\sqrt{(10x-3)^2+(10y-3)^2})^{-1} + 1.5(\sqrt{(10x+3)^2+(10y+3)^2})^{-1} + 1.5(\sqrt{(10x)^2+(10y)^2})^{-1}$ for $(x,y) \in [-1,1]^2$.

strategy	# iterations	degrees of freedom	maximum error	percentage
AT	5	5,637	2.55e-06	99.94
RT (1 %)	10	1,981	5.15e-05	32.16
RT (5 %)	8	6,434	2.24e-06	99.94
RT (10 %)	6	5,053	2.25e-06	99.94
RT (20 %)	5	5,569	2.55e-06	99.94

Table 4.1: Number of iterations and degrees of freedom associated with the absolute (AT) and relative (RT) threshold strategies for the 3 peak data set. The last column specifies the percentage of the number of points that satisfy the error threshold $\sigma = 10^{-6}$. Note that the number of iterations (refinement steps) is not necessarily equal to the number of refinement levels, as mentioned in Section 4.2



(a) absolute threshold (b) relative threshold

Figure 4.8: THB–spline approximation with the corresponding control mesh for the 3 peak data set. The absolute threshold approach (a) generates a less fractal–like subdomain structure in comparison to the relative threshold strategy (RT 10%) (b). The top views of the control grids are also shown.

and the required number of tensor-product B–spline patches needed to perform an exact export of the THB–splines into a standard CAD format, as described in Section 4.3.2. Also,

the RT strategy does not lead to a substantially smaller number of control points.

The RT strategy has some advantages if one tries to find the optimal THB–spline surface for a given number of levels. In this situation, one would use a rather small tolerance. The RT strategy concentrates on the refinement in the regions where the largest errors occur, unlike the AT refinement, which would quickly produce a large number of control points.

However, in all practical test cases with a given error threshold, AT performed better than the RT strategy. Therefore in the remaining examples we will only consider the absolute threshold refinement strategy.

4.4.2 Influence of the regularization term

The regularization parameter assures that the solved system of equations is not singular even for high refinement levels. At the same time it smooths the resulting surface in case of noisy input data. Naturally, in case of data sets with sharp features, the smoothing effect is in contradiction to the required accuracy of the approximation. For these reasons the choice of λ is essential for generating accurate results and at the same time for minimising the number of required iterations.

The dependency of the approximation error and of the condition number of the matrix on the regularization parameter λ is discussed in Table 4.2. Our tests show that in the two analyzed[2] cases the regularization parameter should be smaller then 10^{-8} to reach the required accuracy (more than 99% data points with an error below the threshold $\sigma = 10^{-6}$). For larger values of λ the smoothing effect prevents the solution from capturing the fine details of the original surface, even for high refinement levels. In general, the choice of $\lambda = 10^{-9}$ provided a satisfactory behavior for all tested data sets, including the industrial examples shown in Section 4.3.

[2]In addition to the 3 peak data set previously introduced, we consider the *Rvachev data set* computed by uniform sampling of the function $f(x,y) = \frac{(x+y)}{2} + \sqrt{\left(\frac{x-y}{2}\right)^2}$ for $[x,y] \in [0,1]^2$.

(a) 3 peak data set

λ	degrees of freedom	%	condition number	objective function	maximum error
10^{-7}	7,077	98.80	2.70e+09	2.70e-03	2.22e-04
10^{-8}	5,587	99.58	7.48e+10	3.90e-04	2.54e-05
10^{-9}	5,629	99.94	7.20e+11	1.26e-04	2.55e-06
10^{-10}	5,629	100	6.27e+12	9.93e-05	2.55e-07

(b) Rvachev data set

λ	degrees of freedom	%	condition number	objective function	maximum error
10^{-7}	8,501	95.00	2.41e+09	2.39e-02	1.17e-04
10^{-8}	8,959	97.02	8.97e+09	2.57e-03	1.24e-05
10^{-9}	8,833	99.02	2.07e+10	2.59e-04	1.25e-06
10^{-10}	8,759	100	1.94e+11	2.59e-05	1.25e-07

Table 4.2: The number of degrees of freedom, the condition number and the value of the objective function (see Equation 4.2) obtained with different values of the regularization parameters λ for the 3 peak (a) and the Rvachev (b) data set shown in Figure 4.9. The third column specifies the percentage of points that satisfy the error threshold $\sigma = 10^{-6}$.

4.4.3 Impact of the extension parameter

The choice of the extension parameter, which is used for determining the size of the refined area (see Section 4.2), may influence the accuracy of the approximation, as well as the size of the corresponding THB–spline representation, i.e. the number of resulting degrees of freedom. By enlarging this parameter, the number of newly introduced degrees of freedom increases, thereby providing more flexibility in the neighbourhood of the area with high error.

Table 4.3 compares the number of degrees of freedom and the approximation errors for different values of the extension parameter, again for the 3 peak data set. As expected, the optimal value of this parameter is $\lceil \frac{p}{2} \rceil$, where p is the degree of the considered spline. This corresponds to the smallest possible extension that is required to add at least one new basis function to the THB–spline basis at the next refinement level. For larger values of the extension parameter, we may observe that an increase in the number of coefficients does not always correspond to significant improvements in the accuracy of the approximation.

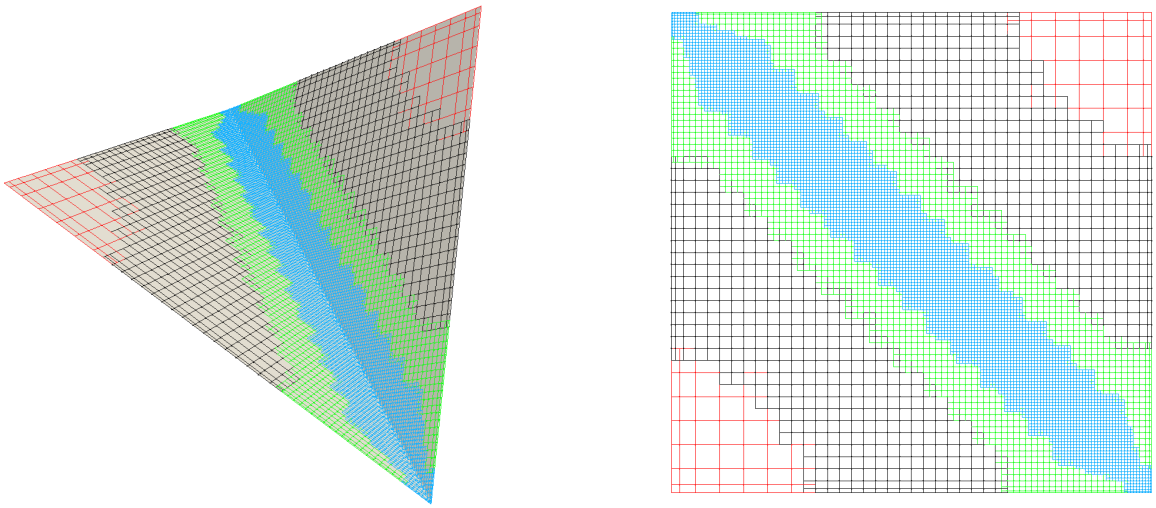


Figure 4.9: Approximation of the Rvachev function with THB–splines (left) with the regularization parameter $\lambda = 10^{-10}$. The top view of the corresponding control mesh is also shown (right).

extension	# iterations	degrees of freedom	%	objective function	maximum error
2	5	5,629	99.94	1.26e-04	2.55e-06
3	5	6,170	99.94	9.96e-05	2.55e-06
4	5	6,841	99.94	7.08e-05	2.55e-06
5	5	7,396	99.94	5.93e-05	2.55e-06

Table 4.3: Number of iterations, degrees of freedom and related value for the objective function with respect to different extension parameters obtained by sampling 10^4 data points from the 3 peak data set. The regularization parameter is set to $\lambda = 10^{-9}$, the required error threshold to $\sigma = 10^{-6}$.

4.4.4 Global vs. local refinement

The accurate modeling and reconstruction of surfaces with sharp features and small details causes many difficulties for the current CAD software based on tensor-product B–splines. Furthermore, the enormous size of the resulting geometries has a negative influence on all post–processing steps, as well. In contrast to the standard technology, the local refinement of THB–splines provides an effective tool for constructing the detailed structures, while simultaneously minimising the size of the resulting geometry.

Table 4.4 and Figure 4.10 compare the number of degrees of freedom for different number of iterations of the fitting procedure for the Rvachev data set. The growth of the size of the standard B–spline basis is exponential. Thus, the global refinement method reaches its limits after a few refinement steps and the solving of the underlying system of equations becomes too expensive in terms of computational time and memory. In contrast to this, the growth of the THB–spline basis remains fairly moderate.

	degrees of freedom		objective function		maximum error	
#	local	global	local	global	local	global
1	169	169	8.69e+00	8.69e+00	1.28e-02	1.28e-02
2	529	529	2.53e+00	2.53+00	6.36e-03	6.36e-03
3	1,729	1,849	7.97e-01	7.97e-01	2.97e-03	2.97e-03
4	4,147	6,889	2.42e-01	2.42e-01	1.02e-03	1.02e-03
5	8,841	26,569	2.59e-04	2.59e-04	1.26e-06	1.26e-06
6	9,233	104,329	2.51e-04	2.36e-04	1.19e-06	1.15e-06
7	9,625	413,449	2.44e-04	2.25e-04	1.16e-06	1.10e-06
8	10,017	n/a	2.43e-04	n/a	1.15e-06	n/a

Table 4.4: The size of the THB–spline basis remains moderate even for high refinement levels (first column), while it grows much faster in case of the tensor-product B–splines (Rvachev data set).

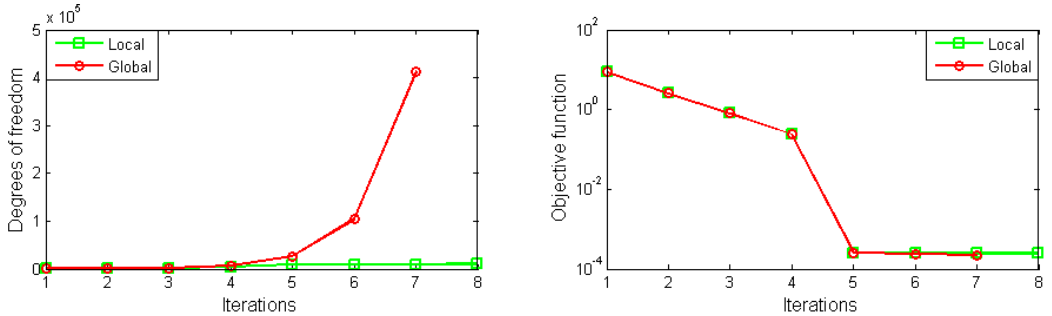


Figure 4.10: While the size of the globally refined tensor-product basis grows exponentially and the size of the locally refined basis remains moderate (left), the difference between the resulting values of the objective function in neglectable (right).

4.4.5 Computing times

Each iteration of the presented adaptive fitting procedure consists of three steps:

1. the refinement of the THB–spline basis, i.e. the adaptation and maintenance of the sets of active functions,

2. the assembly of the linear system,

3. the solution of the linear system[3].

In our current experimental implementation the total computing time is dominated by the first two steps. To give an idea, in order to generate the THB–spline approximation of the fillet data set with 38,260 points (Figure 4.6) where the number of degrees of freedom varies between 169 for 1 level and 20,553 for 7 levels after several iterations, the first two steps need a few seconds up to a few minutes, while the solver time is always less than a second.

We are currently exploring various possibilities to speed up these computations. For example, the B–spline representations of THB–splines is precomputed in order to speed up their evaluation. More precisely, each THB–spline is represented by its B–spline coefficients at the coarsest possible level. This precomputation is performed by slightly modifying the THB–spline evaluation algorithm. More details about the computational times of the current implementation, which was additionally improved and optimized by Špeh, are available in [22]. Obviously, a natural way to further optimize the performance for data sets of high complexity relies on parallel computing techniques.

[3]We used the biconjugate gradient stabilized method solver (BiCGSTAB) from the Eigen library, eigen.tuxfamily.org

Chapter 5

Hierarchical construction and editing of surfaces

As we have already mentioned the tensor-product B–spline technology is the basic spline representation in most computer–aided geometric design software and libraries. Nevertheless this representation has a significant drawback when it comes to design of objects with small details. To create these small features, we have to refine the surface (or volume) to obtain the necessary resolution in the given areas. However, in case of tensor-product splines the refinement spreads over the control grid and thus create a big amount of unnecessary control points. To solve this problem several approaches were proposed, for example HB–splines [17, 35, 36], T–splines [46, 47] or LR B–splines [13].

These approaches provide the possibility of a strictly localized refinement, nevertheless suffer from other drawbacks, such as missing convex hull property or difficult generalization to dimensions higher than 2.

In this chapter we illustrate some of the modeling capabilities of THB–splines. First, we introduce the `TRANSFER_T` algorithm, using which we can determine the control points of the refined THB–spline object without changing its geometry. Furthermore, we comment on the possibility of creating surfaces with locally reduced smoothness and present several examples.

5.1 Refinement of THB–splines

As mentioned already in Sections 2.1.3 and 2.1.4 we can easily compute the control points of a curve, surface or any higher dimensional B–spline object after a refinement step, by multiplying a transfer matrix R with the matrix storing the control points of the original B–spline. In this section we present an algorithm which performs the computation of the matrix R for THB–splines. It proceeds iteratively, determining the refinement coefficient of every basis function.

5.1.1 Computation of the transfer matrix

Let us consider subdomain sequences Ω and $\bar{\Omega}$ such that $\Omega^\ell \subseteq \bar{\Omega}^\ell$ for every $\ell = 0, \ldots, N$ with the corresponding THB–spline bases $\mathcal{T}$ and $\bar{\mathcal{T}}$. Considering the nested nature of the subdomain structures Ω and $\bar{\Omega}$ we know the following:

Lemma 5.1.1 ([53]). *Consider the two sequence of nested domains*

$$\Omega^0 \supseteq \Omega^1 \supseteq \ldots \supseteq \Omega^N = \emptyset \quad \textit{and} \quad \bar{\Omega}^0 \supseteq \bar{\Omega}^1 \supseteq \ldots \supseteq \bar{\Omega}^N = \emptyset$$

together with the corresponding hierarchical bases $\mathcal{T}$ and $\bar{\mathcal{T}}$. If $\Omega^\ell \subseteq \bar{\Omega}^\ell$ for every $\ell = 0, \ldots, N$

$$\operatorname{span} \mathcal{T} \subseteq \operatorname{span} \bar{\mathcal{T}}.$$

Thus for every $f \in span\mathcal{T}$ we obtain the following result

$$f(\boldsymbol{x}) = \sum_{i \in I} c_i \tau_i(\boldsymbol{x}) = \sum_{\bar{i} \in \bar{I}} \bar{c}_i \bar{\tau}_i(\boldsymbol{x}) \quad \tau \in \mathcal{T}, \bar{\tau} \in \bar{\mathcal{T}}$$

where I and $\bar{I}$ are the index sets for for bases $\mathcal{T}$ and $\bar{\mathcal{T}}$.

Similarly to the B–spline case (see Section 2.1.3) we can compute the control points $\bar{P} = (\bar{c}_i)_{i \in \bar{I}}$ corresponding to $\bar{\mathcal{T}}$ using the transfer matrix R, i.e. $\bar{P} = RP$, where $P = (c_i)_{i \in I}$ are the control pints associated with $\mathcal{T}$. Unfortunately, compared to the knot insertion

algorithm, it is significantly more difficult to compute the matrix R in case of THB–splines. We have to consider not only the refinement but also the possible truncation of functions at different levels as well. Using the list of active functions, described in Section 3.2, for $\mathcal{T}$ and $\bar{\mathcal{T}}$ as well as a sequence storing the transfer matrices between underlying bases $\mathcal{B}^\ell$ and $\mathcal{B}^{\ell+1}$ for $\ell = 0, \ldots, N-1$, the algorithm `TRANSFER_T` determines the matrix R for $\mathcal{T}$ and $\bar{\mathcal{T}}$. The number of rows of the output matrix R is equal to the number of basis functions in $\bar{\mathcal{T}}$ and number of columns to the number of basis functions in $\mathcal{T}$. In the following algorithm we denote by $m^\ell, \ell = 0, \ldots, N$ the number of basis functions of the B–spline $\mathcal{B}^\ell$.

Algorithm `TRANSFER_T(seqlist OLD, seqlist NEW, seqmat BTRANS)`

```
\\ seqlist OLD sequence of lists of active functions for T
\\ seqlist NEW sequence of lists of active functions for T̄
\\ seqmat BTRANS transfer matrices between B^ℓ and B^{ℓ+1} for ℓ = 0,...,N-1
create null matrix R
for all levels L do{
    for i from 0 to end of OLD[L] do{
        create vector M of size m^L
        M[OLD[L][i]] = 1
        for j  from L to N-1 do{
            if j>L{
              for k  from from 0 to end of OLD[j]do{
                   M[OLD[j][k]] = 0 }}
            for k  from 0 to end of NEW[j]do{
              set the corresponding value of R to M[NEW[j][k]]}
            M = BTRANS[L] M}}}
return R}
```

This algorithm iterates through all basis functions of $\mathcal{T}$ (represented by the `seqlist OLD`) and determines their representation in the refined basis $\hat{\mathcal{T}}$.

For completeness we provide the the algorithm `TRANSFER_K` which computes the transfer matrices for two nested HB–splines. This algorithm will be used for the computations in Chapter 6.

Algorithm `TRANSFER_K(seqlist OLD, seqlist NEW, seqmat BTRANS)`

`\\ seqlist OLD` sequence of lists of active functions for $\mathcal{T}$

`\\ seqlist NEW` sequence of lists of active functions for $\bar{\mathcal{T}}$

`\\ seqmat BTRANS` transfer matrices between $\mathcal{B}^\ell$ and $\mathcal{B}^{\ell+1}$ for $\ell = 0, \ldots, N-1$

create null matrix `R`

`for` all levels `L do{`

`for i` from 0 to end of `OLD[L] do{`

create vector M of size m^L

M[`OLD[L][i]`] $= 1$

`for j` from `L` to $N-1$ `do{`

`for k` from 0 to end of `NEW[j]do{`

set the corresponding value of `R` to M[`NEW[j][k]`]

M[`NEW[j][k]`] $= 0$`}`

$M =$ `BTRANS[L]` M`}}}`

return R`}`

5.1.2 Local change of THB–spline smoothness

As already mentioned in Section 2.1.2 the smoothness of a B–spline is closely related to its knot vector. Since the construction of the THB–spline basis is based on the standard B–spline technology, i.e. every THB–spline can be represented as a sum of standard B–splines (see Equation 2.5), we observe the same connection between the multiplicity of knots and the smoothness.

Recall that our work focused on a simple dyadic refinement of the B–spline bases $\mathcal{B}^\ell$ where we inserted exactly one knot to every not empty knot span (although theoretically more general refinement strategies are possible). In fact the kD-tree data structure described in Section 3.1 is constructed in a way which only supports dyadic refinement, since the coordinates of inserted and stored boxes are computed with respect to dyadically refined mesh. However, for reducing the smoothness of a THB–spline, we exploit the fact that increasing the multiplicity of a knot u at any refinement level ℓ does not require any change of the kD–tree structure. However, to maintain the nested nature of the underlying knot

vectors we have to increase the multiplicity of the inserted knot at every level higher than ℓ and adjust all corresponding lists of active functions.

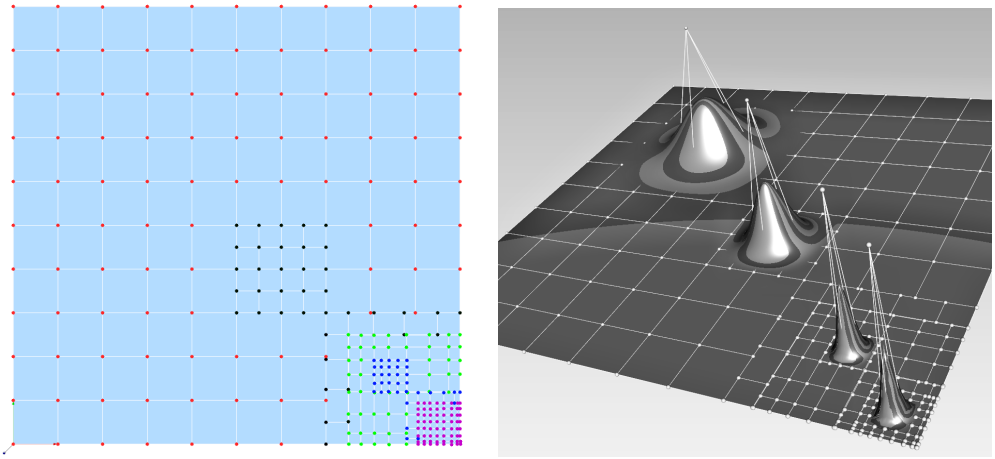
To compute the control points of the THB–spline after the knot insertion we can use an improved implementation of the `TRANSFER_T` algorithm presented in the previous section which also considers the possible function refinements inside of a given level, which occurs due to the knot insertion. We would like to point out that the reduced smoothness in level ℓ introduced by a multiple knot u is present in every connected component of $\Omega^k, k > \ell$ which overlaps with the knot line given by u. This is due to the fact that all connected components of Ω^k stare one underlying B–spline basis $\mathcal{B}^k$.

5.2 Examples

In this section we present several THB–spline geometries which are constructed and refined interactively. All presented examples are created in the Axel [1] modeling tool together with our THB–spline implementation from the G+Smo library [25]. Despite the fact, that Axel only offers the most basic modeling options, such as modeling using the control points of a surface, it provides a fairly simple interface for the integration THB–splines.

Example 5.2.1. We construct a truncated hierarchical surface, where the underlying B–spline basis $\mathcal{B}^0$ is of degree 3 in both direction, defined over the knot vectors U and V where $U = V = \{0, 0, 0, 0, 0.125, 0.25, 0.375, 0.5, 0.625, 0.75, 0.875, 1, 1, 1, 1\}$. The B–spline bases $\mathcal{B}^\ell, \ell > 0$ are constructed as described in Section 2.2.1. As we can see in Figure 5.1 the control points connected to refined areas influence smaller and smaller parts of the surface as the refinement level grows. Furthermore, we can observe that the refinement is strictly localized and does not propagate over the mesh. Additionally, thanks to the non–negativity and partition of unity properties of the THB–spline basis, the created surface is still in the convex hull of its control net and the control points have clear geometric meaning.

Example 5.2.2. We now present a more advanced example of local multilevel editing in the bivariate case. Our initial object is a wine glass in B-spline form shown on the



(a) Control grid (b) THB–spline surface

Figure 5.1: The control grid shows the different levels of refinement where the colors red, black, green, dark blue and purple correspond to different levels of refinement (from 0 to 5 respectively). The area of influence of control points from levels 0, 1, 3, 5, from the upper left corner of the surface to the lower right corner decreases. This allows to create complex shapes with relatively small number of control points in specialized CAD software.

left side of Figure 5.2. We consider three additional refinement steps. The locally refined subdomain structures and the corresponding THB–spline geometries with their multilevel control nets are shown in Figure 5.2 (center and right). As we can see the adaptive nature of THB-spline refinement allows to add additional control points only in the upper part of the model, what makes it possible for us to modify the bowl part of the model without any prolongation of the change to the stem or foot area of the glass. These newly created control points can be interactively moved to change the shape of the bowl of the glass and introduce additional local features as shown in Figure 5.3. The standard tensor-product B–spline representation would have required a global refinement of the mesh, what would introduce a huge amount of degrees of freedom (control points) which are not necessary for creating the desired shape. The presented wine glass example is already shown in [22].

Example 5.2.3. In this example we would like to demonstrate the possibility of creating surfaces with locally reduced smoothness. Our initial THB–spline basis is constructed by

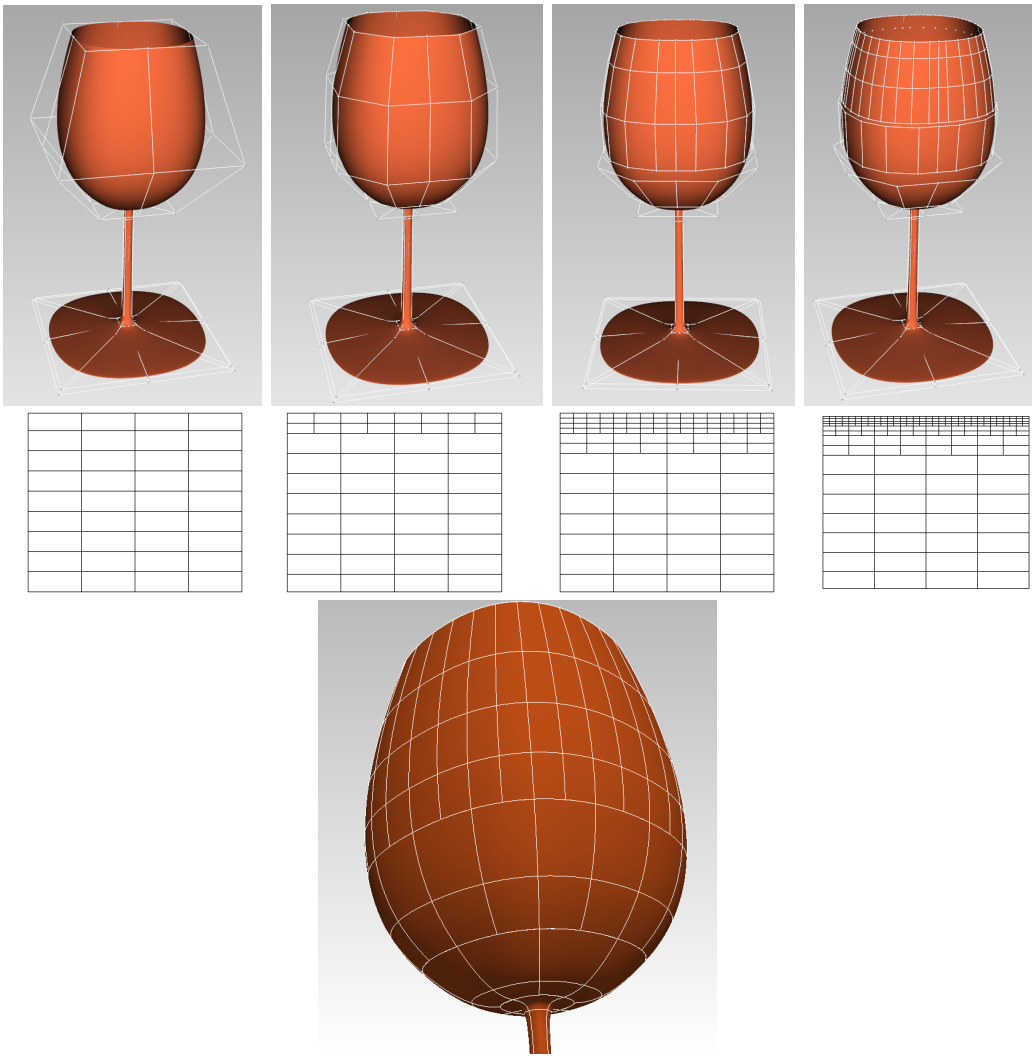


Figure 5.2: Wine glass model considered in Example 5.2.2. The original B–spline geometry (top left) is shown together with three refined THB-spline representations (top). The hierarchical subdomain configurations for the different stages of the refinement are shown (center). Finally, detail of the knot lines along the THB–spline surface showing the T-junctions at the interface between two subsequent hierarchical levels is shown (bottom).

using $\mathcal{B}^0$ from Example 5.2.1. However, instead of simply refining the surface we insert the knot $u = 0.5$ twice into the knot vector U at level 1, while the knot vector V in level 1 remains unchanged. Using this configuration, the areas of the surface without any refinement remain C^2 continuous, since they are connected with the B–spline $\mathcal{B}^0$ where no

multiple knots are present, whereas the refined areas (to level 1 or higher) around the knot line $u = 0.5$ are C^0 continuous due to the increased knot multiplicity. This can be easily observed on Figure 5.4, where the reflection lines on the peak not influenced by the level 1 refinement remained smooth, whereas they are discontinuous on the peak in the lower, refined, part of the surface (see Figure 5.4 (c)).

Example 5.2.4. We present a more complicated object with locally reduced smoothness in Figure 5.5. The original surface is a bicubic B–spline with single knots, see Figure 5.5 (a). We increase the multiplicity of the knot $u = 0.5$ in vertical direction to 3 in level 1 and higher. Subsequently we refine the right half of the surface and the area corresponding to the nose. Until the computed control point does not change their location the surface remains C^2 continuous as we can see in the mouth and chin a parts of the face model. On the upper part of the surface we made small perturbation of the control points around the edge between the levels 0 and 1. This result into a visible "break" in the surface which shows that the surface is in deed C^0 continuous in the area influenced by the knot $u = 0.5$, see Figure5.5 (b). As last step we increased the refinement level around the tip of the nose to level 2 so we can more clearly see the "blending" area where the C^0 part of the surface (the nose part) is joined with the C^2 smooth part of the face model.

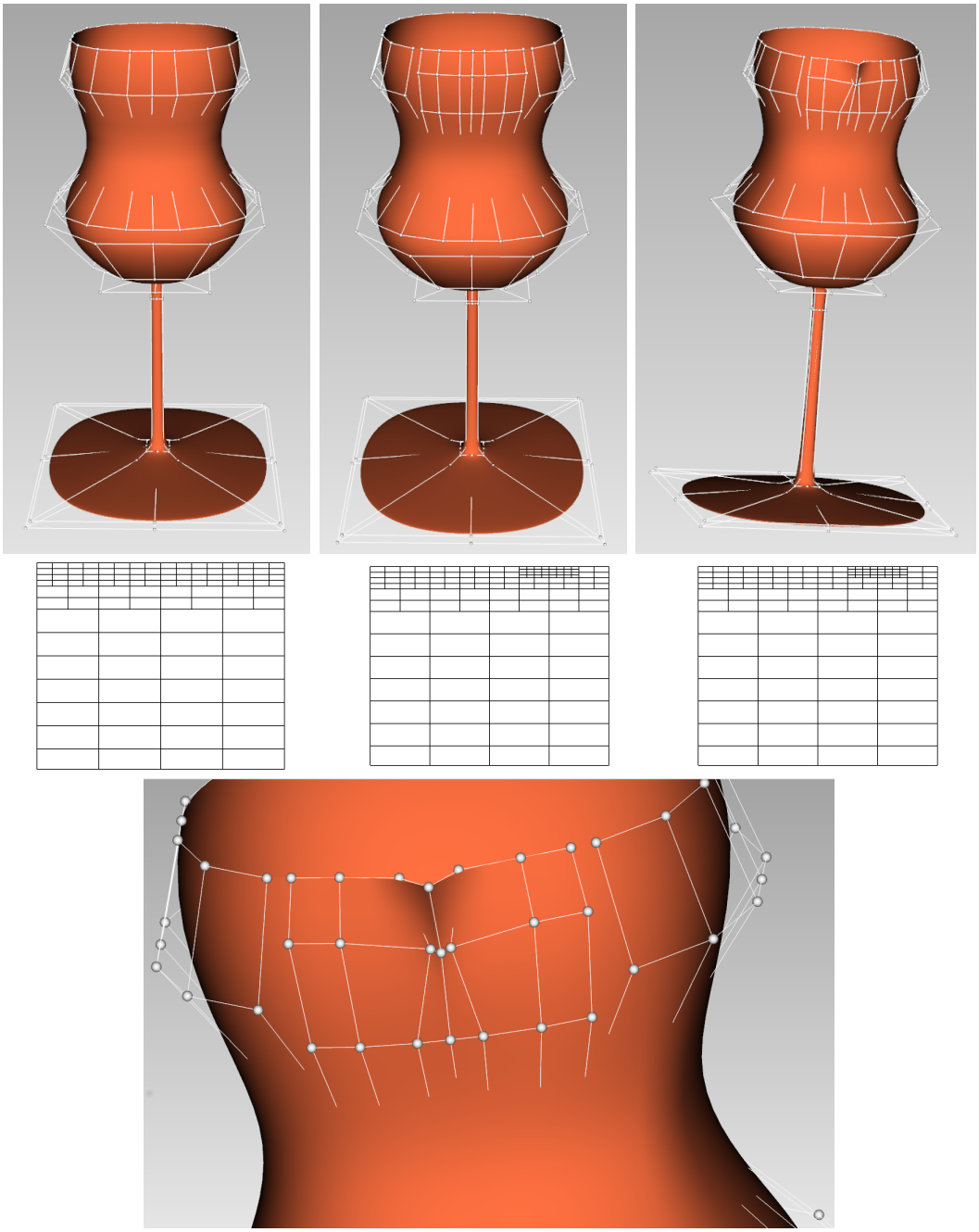
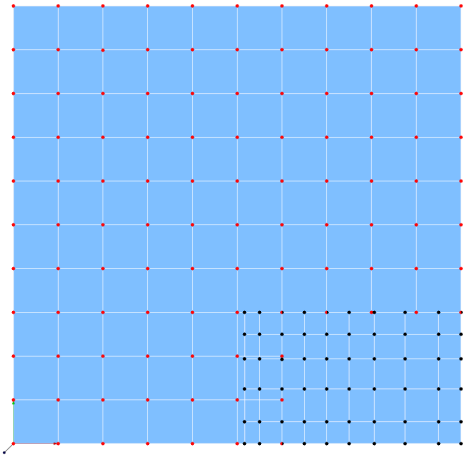
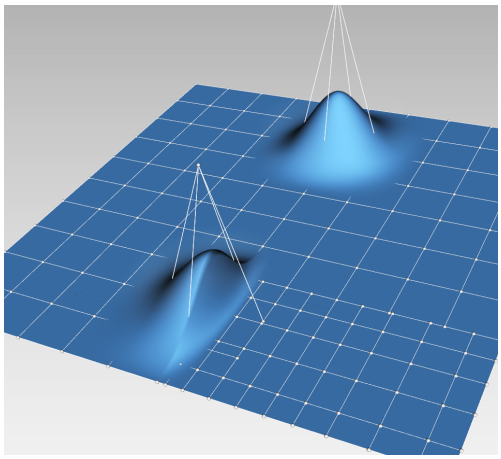


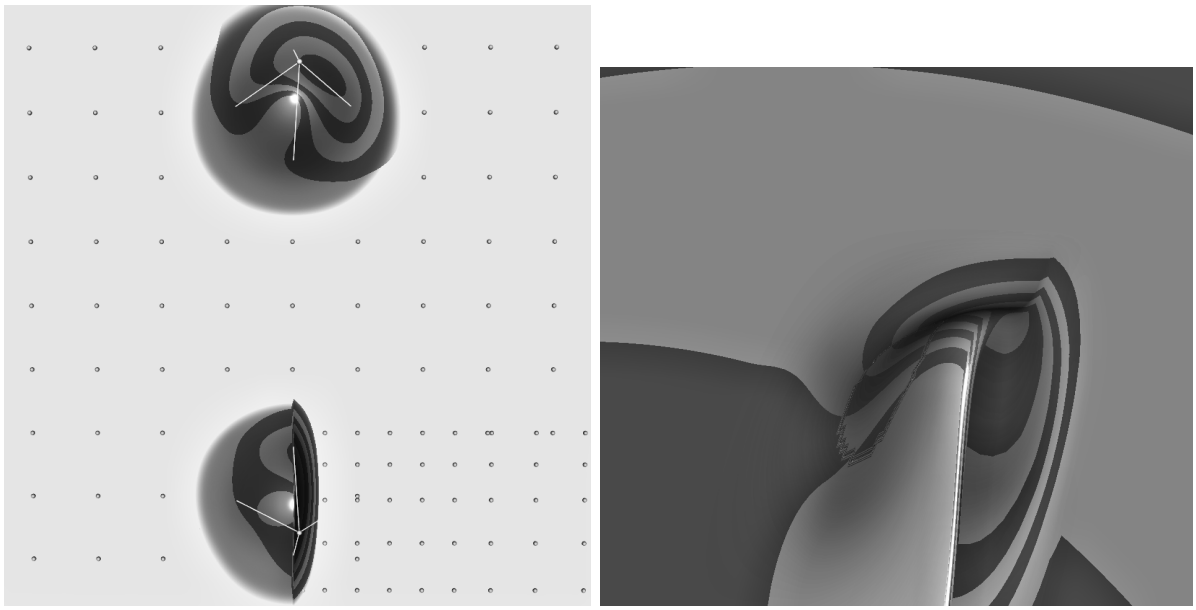
Figure 5.3: Local editing of the wine glass considered in Example 5.2.2. Starting with THB–spline object obtained by 2 refinement steps, introduced in Figure 5.2 (third picture in the top row), we change the shape of the glass bowl by moving several control points towards the center of the bowl. This results in the shape shown (top left). Additionally, we refine a small area on the top of the bowl (top middle) where we create a small "cusp" (top right and bottom). The subdomain configurations at the different stages are also shown (center).



(a) Control grid

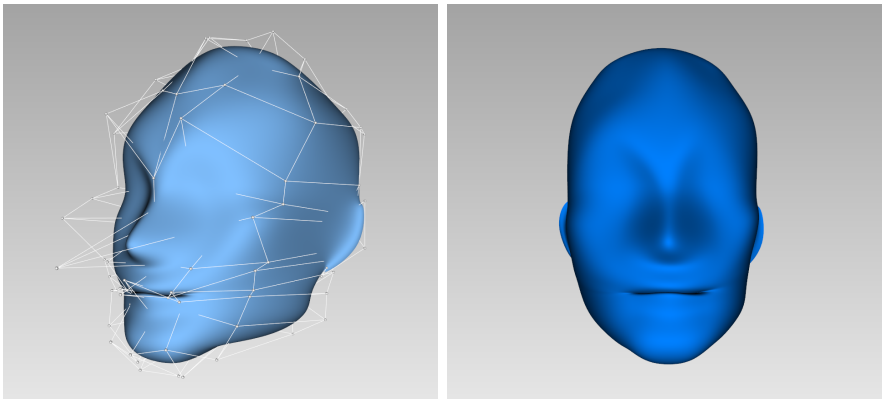


(b) THB–spline surface

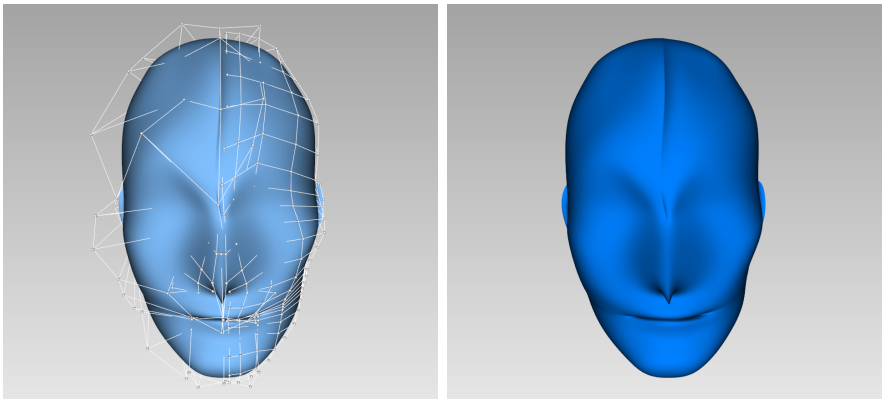


(c) Deail of the area with reduced smoothness

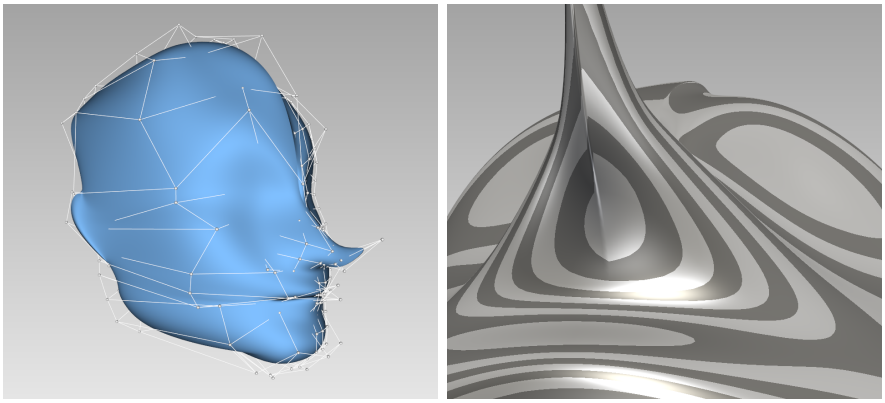
Figure 5.4: The control net of the refined surface with a multiple knot $u = 0.5$ in level 1. As we can see the surface in the areas not influenced by the refinement (upper part of surface (b)) are still smooth as we can see it on the reflection lines on the left–hand side of (c) whereas the part of the surface influenced by the level 1 refinement has only C^0 continuity as we can see on the detail on the right–hand side of (c).



(a) B–spline face



(b) THB–spline face with reduced smoothness



(c) Additional refinement and detail of the surface on the boundary of C^2 and C^0 areas

Figure 5.5: In the original C^2 continuous B–spline surface (a), we increased the multiplicity of the knot $u = 0.5$ to 3 in level 1, as described in Example 5.2.4. After refining the surface and changing the position of several control points on the upper part of the surface, the C^0 continuous area is clearly visible (b). In figure (c) we made an additional refinement of the nose area of the surface and we show the detail of the surface where the C^2 and C^0 parts meet.

Chapter 6

Multigrid solvers for isogeometric analysis with THB–splines

Isogeometric analysis (IgA) is a recently developed technology in numerical analysis that offers the possibility of integrating methods for analysis and Computer–Aided Design (CAD) into a single, unified process. The standard approach in numerical analysis, the finite element analysis (FEA), uses polygonal meshes to describe the geometry of an object, while the standard technology used by computer–aided design community are B–splines, NURBS or subdivision surfaces. The data exchange between the FEA and CAD communities requires re-parametrisation of the CAD geometry, which is an expensive process that may take more than 80% of the total time required for analysis, and in most cases it introduces discretization errors. The main idea of isogeometric analysis is to use the same description of geometry, in this case the CAD representation, for both, design and analysis. The first results regarding this approach were published by Hughes, Cottrell and Bazilevs in [32], where B–splines (NURBS) are used as basis functions for finite element analysis. In the following years the IgA community grew significantly providing many results on different topics of FEA. In this chapter we use (T)HB–splines as a discretization basis to solve elliptic partial differential equations using multigrid solvers in the IgA framework. Our goal is to analyze the impact of the truncation on the efficiency of the multigrid solvers. The results

presented in this chapter are based on our collaboration with C. Hofreiter [28].

6.1 Multigrid methods for adaptively refined isogeometric discretizations

In this section we introduce the necessary background machinery used in the examples in Section 6.2.

6.1.1 Nested sequences of hierarchical B-spline spaces

Let us assume that for each $k \in \mathbb{N}$, we have a finite sequence of nested subdomains

$$\Omega_k = (\Omega_k^\ell)_{\ell \in \mathbb{N}}.$$

Additionally, we make an assumption that

$$\Omega_k^\ell \subseteq \Omega_{k+1}^\ell \qquad \forall k, \ell \in \mathbb{N}.$$

Schematically, we have the relations

$$\begin{array}{lccccccc} \Omega = & \Omega_k^0 & \supseteq \ldots \supseteq & \Omega_k^\ell & \supseteq & \Omega_k^{\ell+1} & \supseteq \ldots \supseteq \emptyset \\ & \| & & \cap| & & \cap| & & \\ \Omega = & \Omega_{k+1}^0 & \supseteq \ldots \supseteq & \Omega_{k+1}^\ell & \supseteq & \Omega_{k+1}^{\ell+1} & \supseteq \ldots \supseteq \emptyset. \end{array}$$

That means the subdomain structure Ω_{k+1} is obtained by refining Ω_k.

Due to Lemma 5.1.1, this yields

$$\operatorname{span}\{\mathcal{T}_0\} \subseteq \operatorname{span}\{\mathcal{T}_1\} \subseteq \ldots$$

where $\mathcal{T}_i$ denotes the truncated hierarchical basis over Ω_i. A similar conclusion holds for the standard hierarchical basis $\mathcal{K}_i$ defined over Ω_i.

6.1.2 Construction by adaptive refinement

In practice, we use adaptive refinement based on an a posteriori error estimator to construct the sequence of nested subdomains Ω_k. We start with a coarse tensor-product B–spline discretization defined over the parameter domain. This means that the initial hierarchy Ω_0 contains only a single domain, namely, the entire parameter domain. Let $\mathcal{V}_0$ be the hierarchical spline space defined over Ω_0, which is in this case a standard tensor-product spline space, and let u_0 be a discrete solution of a discretized boundary value problem over this coarse space.

Based on this solution, we compute an a posteriori error estimator (see [22]) and refine the parts of the domain where the error is largest. This results in a new hierarchy

$$\Omega_1 = (\Omega, \Omega_1^1),$$

which induces a refined hierarchical spline space $\mathcal{V}_1 \supset \mathcal{V}_0$. Now we solve the discretized boundary value problem in $\mathcal{V}_1$ and obtain a new discrete solution u_1. Subsequently, we repeat the process of a posteriori error estimation and adaptive refinement. At every iteration we introduce at most one new level to the hierarchy, so that after the k-th step we obtain a hierarchy

$$\Omega_k = (\Omega, \Omega_k^1, \ldots, \Omega_k^k)$$

and associated hierarchical spline space $\mathcal{V}_k$. We terminate this iterative process once we are satisfied with the accuracy of the obtained solution u_k.

6.1.3 Application of multigrid

The nested sequence of discretization spaces $\mathcal{V}_0 \subset \mathcal{V}_1 \subset \ldots \subset \mathcal{V}_k$, constructed during the iterative refinement process in the previous section, allows us to apply a multigrid framework to the solution of the discretized problem in $\mathcal{V}_k$. The key idea of the multigrid approach is the projection of a large problem defined over a fine grid into a much smaller problem defined over a very coarse grid. This procedure is called restriction. Consequently

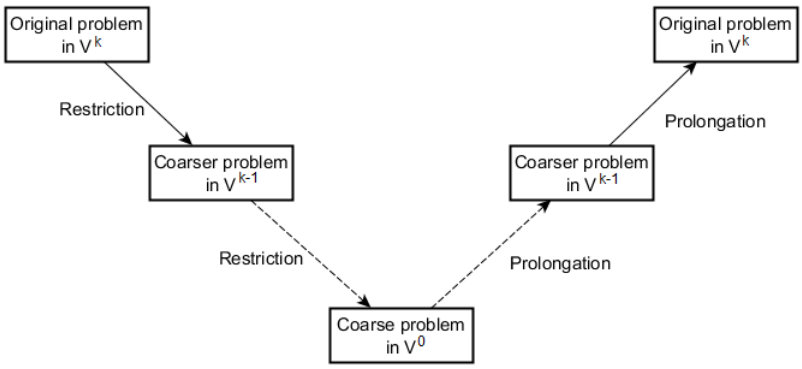


Figure 6.1: A discretized problem in $\mathcal{V}_k$ is restricted, possibly in several steps to a coarse problem in $\mathcal{V}_0$, where we solve the problem using a direct solver. Consequently we project this solution back to $\mathcal{V}_k$.

we compute a discrete solution of the restricted problem problem which is finally projected back onto the original grid, so called prolongation. We call these three steps a V-cycle, see Figure 6.1[1]. For the computation of the transfer matrices R_j which describe the embedding of $\mathcal{V}_j$ in $\mathcal{V}_{j+1}$, we use the procedure described in Section 5.1.1. More specifically the matrices computed by the algorithms `TRANSFER_T` and `TRANSFER_K` are the prolongation matres. The restriction matrix is obtained by inverting the output of these algorithms.

In the multigrid framework it is also required to use a smoothing step after every restriction and prolongation. In the present work, we use a simple Gauss-Seidel smoother, i.e. the smoothing matrix S_k on level k is chosen as the lower triangular part of the stiffness matrix on the same level. It is understood by now, that in the isogeometric setting, this simple smoothing strategy does not always result in optimal multigrid solvers, especially in case of higher spline degrees and space dimensions [30]. However, the design of optimal smoothers for isogeometric analysis is still a field of active research, and we therefore stick with the simple choice of the Gauss-Seidel smoother.

[1]For details about the multigrid framework see [5, 26, 51].

6.2 Numerical experiments

In this section, we present experimental results for three model problems. Example 6.2.1 uses a synthetically generated sequence of meshes and a smooth solution, while Examples 6.2.2 and 6.2.3 use adaptive refinement by *a posteriori* error estimation for problems with non–smooth solutions.

In all examples, we compare the number of V-cycle iterations required for obtaining a satisfactory solution of a discretized boundary value problem using THB–splines and HB–splines. Throughout we use V-cycle iteration with one pre- and one post-smoothing Gauss-Seidel step on each level. The stopping criterion in all cases is the reduction of the Euclidean norm of the initial residual by a factor of 10^{-8}.

Example 6.2.1. On the square $\Omega = (-1, 1)^2$, we construct a hierarchy of THB–spline spaces by adaptively fitting the function

$$v(x, y) = \exp(52\sqrt{(10x - 2)^2 + (10y - 3)^2})^{-1}$$

starting from a coarse tensor-product spline space using the fitting procedure described in Chapter 4. An example of the mesh resulting after several fitting steps is shown in Figure 6.2.

We then solve the Poisson equation

$$-\Delta u = f \quad \text{in } \Omega, \qquad u|_{\partial\Omega} = g,$$

where the right–hand side and Dirichlet data are chosen according to the smooth solution

$$g(x, y) = \sin(\pi x) \sin(\pi y).$$

The subdomain structure generated by the fitting process and the corresponding (T)HB–spline spaces are used as the grids in our multigrid method. The iteration numbers required to reduce the Euclidean norm of the initial residual by a factor of 10^{-8} are show in Table 6.1.

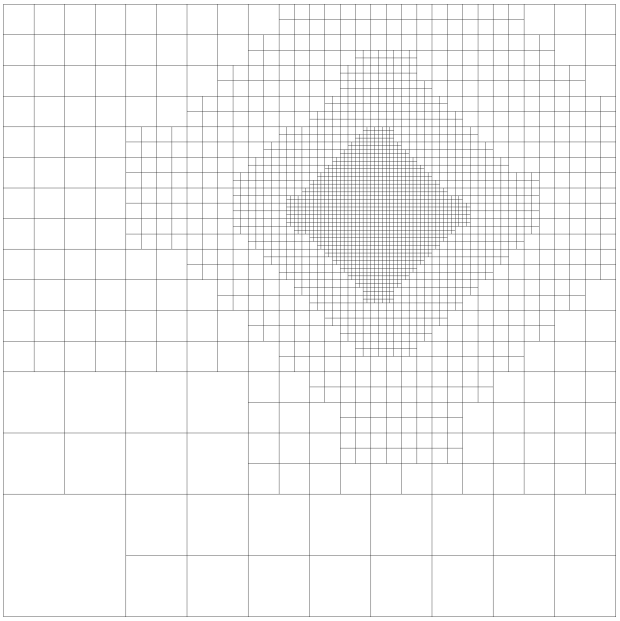


Figure 6.2: The domain structure obtained after five steps of adaptive fitting with biquadratic THB–splines used in Example 6.2.1.

	$p = 2$			$p = 3$			$p = 4$		
		THB	HB		THB	HB		THB	HB
ℓ	dofs	iter	iter	dofs	iter	iter	dofs	iter	iter
0	49			64			81		
1	109	11	24	169	42	42	196	165	165
2	365	15	47	475	88	548	528	873	13264
3	829	12	31	1174	75	434	1272	629	12802
4	1487	13	29	2266	81	412	2509	433	10701
5	2317	12	29	3580	75	408	4136	415	9075

Table 6.1: Comparison of iteration numbers for Example 6.2.1.

Example 6.2.2. We solve the Poisson equation

$$-\Delta u = f \quad \text{in } \Omega, \qquad u|_{\partial\Omega} = 0$$

with the source function

$$f(x,y) = \begin{cases} 1, & (x-0.25)^2 + (y-1.6)^2 < 0.2^2, \\ 0, & \text{otherwise.} \end{cases}$$

The domain Ω approximates a quarter annulus in the first quadrant with inner and outer radius 1 and 2, respectively, by means of quadratic tensor-product B–splines.

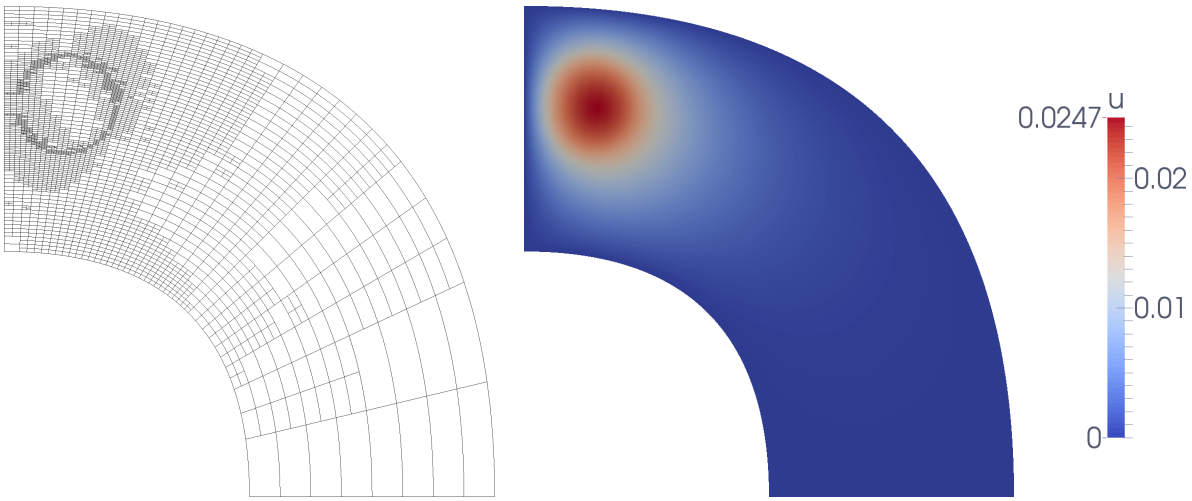


Figure 6.3: *Left:* Subdomain from Example 6.2.2 with quadratic THB-splines after six adaptive refinement steps (6249 dofs). *Right:* Solution field.

Starting from a coarse tensor-product B–spline basis ($\ell = 0$), we construct a hierarchy of THB–spline spaces by means of adaptive refinement based on an *a posteriori* error estimator. An example of a THB–spline basis arising from this adaptive refinement process, as well as a plot of the obtained solution, are shown in Figure 6.3.

We then set up a multigrid method as described above which always has $\ell = 0$ as its coarse grid and solves the problem on some given level ℓ by V-cycle iteration. The comparison of

	$p=2$			$p=3$			$p=4$		
		THB	HB		THB	HB		THB	HB
ℓ	dofs	iter	iter	dofs	iter	iter	dofs	iter	iter
0	100			121			144		
1	182	15	34	195	68	238	226	183	3656
2	343	18	41	353	75	382	381	397	6112
3	701	24	57	668	131	1176	673	478	6113
4	1355	26	64	1326	198	734	1216	2012	25559
5	2822	23	98	2635	144	987	2312	1436	34310

Table 6.2: Iteration numbers for Example 6.2.2.

iteration numbers obtained by HB– and THB–splines is shown in Table 6.2.

Example 6.2.3. We solve the classical benchmark problem for a singularity arising from a reentrant corner on the L-shape domain

$$-\Delta u = 0 \quad \text{in } \Omega, \qquad \text{where } \Omega = (-1,1)^2 \setminus [0,1]^2.$$

We choose pure Dirichlet boundary conditions according to the exact solution, given in polar coordinates,

$$g(r,\varphi) = r^{2/3}\sin((2\varphi-\pi)/3),$$

where the argument is assumed to have the range $\varphi \in [0, 2\pi)$. The geometry is represented by piecewise linear splines.

We then perform adaptive refinement using THB–splines as described in Example 6.2.2. An example of a THB-spline mesh arising from adaptive refinement is shown in Figure 6.4. We present the resulting iteration numbers for this example in Table 6.3.

All our experiments show that using THB–splines significantly reduces the number of V–cycle iterations necessary for reduction of the initial residual by factor 10^{-8} in comparison to standard HB–splines. The difference in iteration numbers increased for higher spline degrees. We assume that the smaller iteration numbers in the THB–spline case result from the reduced size of the support of basis functions. We would like point out that for high

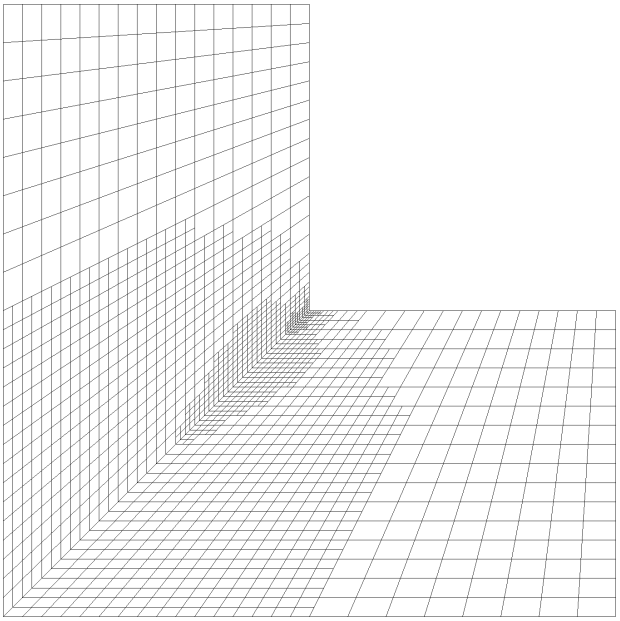


Figure 6.4: L-shape domain with cubic THB–splines after four adaptive refinement steps (2379 dofs)

	$p=2$			$p=3$			$p=4$		
		THB	HB		THB	HB		THB	HB
ℓ	dofs	iter	iter	dofs	iter	iter	dofs	iter	iter
0	630			703			780		
1	810	18	33	909	71	297	982	392	3487
2	1163	16	35	1252	68	317	1332	370	3300
3	1631	18	33	1721	84	359	1783	557	4266
4	2338	17	41	2379	73	395	2458	446	5873
5	3327	18	32	3414	87	382	3514	615	6804

Table 6.3: Iteration numbers for Example 6.2.3.

spline degrees, the iteration numbers are not satisfactory even when using THB–splines. This can be caused by the fact that the used Gauss-Seidel smoother does not perform well for high spline degrees [29]. As mentioned already before the research of optimal and robust smoothers for isogeometric multigrid methods is ongoing.

Chapter 7

Conclusion

The presented work addressed the topic of truncated hierarchical B–splines. It is split into two main parts.

The first part focuses on the theoretical background and definition of the truncated hierarchical B–spline basis. This is supplemented by a presentation of efficient data structures and algorithms which are useful for the implementation of THB–splines.

As we have seen in Chapter 2, the construction of the THB–spline basis requires additional effort during its construction, compared to the standard HB–spline basis, what is caused by the truncation of the basis functions. However, this additional effort is not that significant and it is compensated by the improved properties of the basis. These include for example a reduced overlap between the basis functions of different levels, reduced support of basis functions, and the regained partition of unity.

For an efficient representation of the nested subdomain hierarchy Ω, which is a basic building block of the THB–spline representation, we used a kD–tree structure. This structure is not only a compact representation of Ω, but also provides functions which enable additional refinement and fast identification of active functions.

The presented work focused only on dyadic refinement. In some applications it may be beneficial to use different refinement strategies, or even use different refinement strategy in

every refinement level. These generalized approaches would require a more flexible data structure for storing the subdomain structure of a THB–spline, and could be an interesting topic for further research.

In this thesis we have proposed two different ways to store information about the basis functions from different levels contributing to the final basis. The first approach stores information about all B–spline basis functions from all levels, both active and passive, thus providing fast access to the data. The second one only stores information about the active functions. As our examples show, storing only the information about active functions significantly reduces the memory consumption of our method, in many cases by more than 90 %. This is compensated by a small decrease in the speed of the evaluation algorithm, usually around 10-15 %. This is due to the slower access to the data in the data structure which explores the sparsity of the stored data.

Our current implementation of the THB–spline basis is available as a module in the G+Smo library [25], which is an open source, object-oriented C++ library focused on isogeometric analysis. The library exploits object polymorphism and inheritance techniques to provide a variety of different discretization bases, for example B–spline, Bernstein, NURBS bases, hierarchical and truncated hierarchical B-spline bases of arbitrary polynomial order, etc. The implementation of bases and geometries is dimension-independent. That means the curves, surfaces, volumes, and other higher dimensional objects are instances of code templated with respect to the dimension of the parameter domain.

The second part of the thesis concentrates on the application of the THB–spline technology in various fields, more specifically on surface reconstruction from measured data, interactive modeling, and isogeometric analysis. Among other results we described a simple regularized least-squares approximation method for CAD model reconstruction, together with two different refinement strategies – the absolute and relative threshold strategy. As we have seen, both can provide results that satisfy the strict industrial requirements on precision. The absolute threshold strategy is usually more efficient in terms of required iterations, the relative threshold strategy is nevertheless more preferable in cases when the maximum

allowed level of refinement is small.

In addition to the simple academic examples, we have tested our reconstruction method on challenging real world data sets and compared its results to the results obtained using standard tensor-product B–splines. We observed a significant improvement in the reconstruction, not only in terms of degrees of freedom, but also in the quality of the surface. While the results obtained from B–spline reconstruction suffer from strong oscillations, the THB–spline surface reconstructed the data precisely. This behaviour is most notable in case of data sets with a strongly varying data density.

Further, we presented two options for exporting THB–splines into a standard CAD format. The THB–spline surface is in both cases divided into several B–spline patches which are exported to a format compatible with the standard CAD software (in our case Parasolid$^{\mathrm{TM}}$). These B–spline patches represent the original THB–spline geometry exactly, no error is introduced during the export process.

In our work we did not search for a box partition with minimal number of boxes in the case of the *rectangular partition* splitting method. Nevertheless, it could be interesting for industrial applications where the number of resulting patches may influence the speed of the post–processing steps.

Another topic for future research could be the generalisation of our export algorithm to higher dimensions, again with special focus on partition of the THB–spline subdomains into a small number of patches.

A further topic addressed in our work is the possibility to manipulate THB–spline objects by moving or adding control points, and refining the object interactively. On the one hand this is important for creating simple test cases necessary for additional research. On the other hand a suitable modeling tool would make the THB–spline technology more popular in other areas, for example design or engineering.

We have also presented an algorithm that performs the computation of transfer matrices between two nested THB–splines. We have realised a connection between our THB–spline implementation in G+Smo library and the Axel modeler developed by INRIA [1]. Axel provides only a basic set of modeling tools which may be sufficient for creating simple objects,

as we have presented in Section 5.2, but they are not sufficient for creating complicated models needed in real life applications. Nevertheless, the presented locally refined objects, and the objects with locally reduced smoothness show that the THB–splines can be indeed a valuable modeling tool. We believe this chapter will motivate further improvements and research in this area.

The last field where we employed THB–splines is isogeometric analysis. Our goal was to analyse the influence of the truncation mechanism on the performance of multigrid solvers. Therefore we used both HB– and THB–splines as a discretization basis in the multigrid solver framework for isogeometric analysis. We have presented three numerical examples where we compared the efficiency of both bases in terms of the number of V–cycle iterations. We observed that in all tests the performance of the THB–splines is superior to the HB–splines. In most cases the multigrid solver using THB–splines as discretization basis required significantly fewer V–cycle iterations than with HB–splines to achieve the same improvement of accuracy. Nevertheless, we would like to point out that the iteration numbers for THB–splines are still considerably larger than for the standard B–splines, especially for high degrees. As already mentioned, the smoothers used in standard multigrid techniques are not optimal in the case of isogeometric analysis. There is an ongoing research on efficient smoothers for standard B–splines that could later lead to investigation of efficient smoothers for hierarchical splines as well.

Bibliography

[1] Axel. Axel - algebraic geometric modeling. Inria inventeurs du monde numérique, 2015. www.axel.inria.fr.

[2] F. L. Bookstein. Principal warps: thin-plate splines and the decomposition of deformations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(6): 567–585, 1989. doi: 10.1109/34.24792.

[3] P.B. Bornemann and F. Cirak. A subdivision-based implementation of the hierarchical B-spline finite element method. *Computer Methods in Applied Mechanics and Engineering*, 253:584 – 598, 2013. doi: 10.1016/j.cma.2012.06.023.

[4] R. Bridson, S. Marino, and R. Fedkiw. Simulation of clothing with folds and wrinkles. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '03, pages 28–36. Eurographics Association, 2003. ISBN 1-58113-659-5.

[5] W. L. Briggs, V. E. Henson, and S. F. McCormick. *A Multigrid Tutorial*. Society for Industrial and Applied Mathematics, 2000. ISBN 9780898714623.

[6] C3D kernel. C3D kernel. C3D Labs, 2015. URL `http://c3dlabs.com/en/`.

[7] M. G. Cox. The Numerical Evaluation of B-Splines. *IMA Journal of Applied Mathematics*, 10:134–149, 1972. doi: 10.1093/imamat/10.2.134.

[8] C. de Boor. On calculating with B-splines. *Journal of Approximation Theory*, 6(1):50 – 62, 1972. doi: 10.1016/0021-9045(72)90080-9.

[9] R. Delgado-Gonzalo and M. Unser. Spline-based framework for interactive segmentation in biomedical imaging. *IRBM-Ingénierie et Recherche Biomédicale / BioMedical Engineering and Research*, 34(3):235–243, 2013.

[10] J. Deng, F. Chen, and Y. Feng. Dimensions of spline spaces over T–meshes. *Journal of Computational and Applied Mathematics*, 194:267–283, 2006.

[11] J. Deng, F. Chen, X. Li, C. Hu, W. Tong, Z. Yang, and Y. Feng. Polynomial splines over hierarchical T-meshes. *Graphical Models*, pages 76–86, 2008.

[12] P. Dierckx. *Curve and Surface Fitting with Splines*. Oxford University Press, Inc., 1993. ISBN 0-19-853441-8.

[13] T. Dokken, T. Lyche, and K. F. Pettersen. Polynomial splines over locally refined box–partitions. *Computer Aided Geometric Design*, 30:331–356, 2013.

[14] M. S. Floater. Parametrization and smooth approximation of surface triangulations. *Computer Aided Geometric Design*, 14(3):231–250, 1997.

[15] M. S. Floater. How to approximate scattered data by least squares. SINTEF report STF42 A98013, Department of Mathematics, 1998.

[16] M.S. Floater and K. Hormann. Surface parameterization: a tutorial and survey. In N. A. Dodgson, M. S. Floater, and M. A. Sabin, editors, *Advances in Multiresolution for Geometric Modelling*, Mathematics and Visualization, pages 157–186. Springer, 2005.

[17] D. R. Forsey and R. H. Bartels. Hierarchical B-spline refinement. *Computer Graphics*, 22:205–212, 1988.

[18] D. R. Forsey and R. H. Bartels. Surface fitting with hierarchical splines. *ACM Transactions on Graphics*, 14:134–161, 1995.

[19] D. R. Forsey and D. Wong. Multiresolution surface reconstruction for hierarchical B-splines. In *Graphics Interface*, pages 57–64, 1998.

[20] C. Giannelli, B. Jüttler, and H. Speleers. THB–splines: the truncated basis for hierarchical splines. *Computer Aided Geometric Design*, 29:485–498, 2012.

[21] C. Giannelli, B. Jüttler, and H. Speleers. Strongly stable bases for adaptively refined multilevel spline spaces. *Advances in Computational Mathematics*, 40(2):459–490, 2014. doi: 10.1007/s10444-013-9315-2.

[22] C. Giannelli, B. Jüttler, S. K. Kleiss, A. Mantzaflaris, B. Simeon, and J. Špeh. THB-splines: An effective mathematical technology for adaptive refinement in geometric design and isogeometric analysis. in preparation, 2015.

[23] J. R. Gilbert, C. Moler, and R. Schreiber. Sparse matrices in Matlab: Design and implementation. *SIAM J. Matrix Anal. Appl.*, 13(1):333–356, January 1992. doi: 10.1137/0613024.

[24] G. Greiner and K. Hormann. Interpolating and approximating scattered 3D-data with hierarchical tensor product B-splines. In A. Le Méhauté, C. Rabut, and L. L. Schumaker, editors, *Surface Fitting and Multiresolution Methods*, pages 163–172. Vanderbilt University Press, 1997.

[25] G+Smo. Geometry + simulation modules. Online, 2015. URL `http://www.gs.jku.at/gismo`.

[26] W. Hackbusch. *Multi-Grid Methods and Applications*, volume 4 of *Springer Series in Computational Mathematics*. Springer-Verlag, 2003.

[27] H. Hagen, S. Hahmann, T. Schreiber, Y. Nakajima, B. Wordenweber, and P. Hollemann-Grundstedt. Surface interrogation algorithms. *IEEE Comput. Graph. Appl.*, 12(5): 53–60, 1992. doi: 10.1109/38.156013.

[28] C. Hofreiter, B. Jüttler, G. Kiss, and W. Zulehner. Multigrid methods for isogeometric analysis with THB-splines. in preparation, 2015.

[29] C. Hofreither and W. Zulehner. Mass smoothers in geometric multigrid for isogeometric analysis. NuMa Report 2014-11, Institute of Computational Mathematics, Johannes Kepler University Linz, Austria, 2014. Accepted for publication in the proceedings of the 8th International Conference Curves and Surfaces.

[30] C. Hofreither and W. Zulehner. Spectral analysis of geometric multigrid methods for isogeometric analysis. NuMa Report 2014-10, Institute of Computational Mathematics, Johannes Kepler University Linz, Austria, 2014.

[31] K. Höllig and J. Hörner. *Approximation and Modeling with B-Splines*. SIAM, 2013. ISBN 1611972949, 9781611972948.

[32] T.J.R. Hughes, J.A. Cottrell, and Y. Bazilevs. Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement. *Computer Methods in Applied Mechanics and Engineering*, 194:4135 – 4195, 2005. doi: 10.1016/j.cma.2004.10.008.

[33] G. Kiss, C. Giannelli, and B. Jüttler. Algorithms and data structures for truncated hierarchical B-splines. In M. Floater et al., editors, *Mathematical Methods for Curves and Surfaces*, volume 8177 of *Lecture Notes in Computer Science*, pages 304–323. Springer, 2014.

[34] G. Kiss, C. Giannelli, U. Zore, B. Jüttler, D. Großmann, and J. Barner. Adaptive cad model (re-)construction with thb-splines. *Graph. Models*, 76(5):273–288, 2014. doi: 10.1016/j.gmod.2014.03.017.

[35] R. Kraft. Adaptive and linearly independent multilevel B–splines. In A. Le Mehaute, C. Rabut, and L. L. Schumaker, editors, *Surface Fitting and Multiresolution Methods*, pages 209–218. Vanderbilt University Press, 1997.

[36] R. Kraft. *Adaptive und linear unabhängige Multilevel B–Splines und ihre Anwendungen*. PhD thesis, Universität Stuttgart, 1998.

[37] P. Li and P. G. Kry. Multi-layer skin simulation with adaptive constraints. In *Proceedings of the Seventh International Conference on Motion in Games*, MIG '14, pages 171–176. ACM, 2014. ISBN 978-1-4503-2623-0. doi: 10.1145/2668084.2668089.

[38] X. Li, J. Zheng, T. W. Sederberg, Thomas J.R. Hughes, and Michael A. Scott. On linear independence of T-spline blending functions. *Computer Aided Geometric Design*, 29(1):63 – 76, 2012. doi: 10.1016/j.cagd.2011.08.005.

[39] R. Narain, T. Pfaff, and J. F. O'Brien. Folding and crumpling adaptive sheets. *ACM Transactions on Graphics*, 32(4):51:1–8, 2013. Proceedings of ACM SIGGRAPH 2013, Anaheim.

[40] Open CASCADE. Open CASCADE kernel. Online, 2015. URL `http://www.opencascade.org/`.

[41] Parasolid. Parasolid 3D geometric modeling engine. Siemens PLM Software Inc., 2013. URL `www.plm.automation.siemens.com/en_us/products/open/parasolid`.

[42] L. Piegl and W. Tiller. *The NURBS Book (2nd Ed.)*. Springer-Verlag New York, Inc., 1997. ISBN 3-540-61545-8.

[43] H. Samet. *Foundations of Multidimensional and Metric Data Structures (The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling)*. Morgan Kaufmann Publishers Inc., 2005. ISBN 0123694469.

[44] D. Schillinger, L. Dedé, M. A. Scott, J. A. Evans, M. J. Borden, E. Rank, and 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. *Computer Methods in Applied Mechanics and Engineering*, 249:116 – 150, 2012.

[45] M. A. Scott, D. C. Thomas, and E. J. Evans. Isogeometric spline forests. *Computer Methods in Applied Mechanics and Engineering*, 269:222–264, 2014.

[46] T. W. Sederberg, J. Zheng, A. Bakenov, and A. Nasri. T-splines and T-NURCCS. *ACM Transactions on Graphics*, 22:477–484, 2003.

[47] T. W. Sederberg, D. L. Cardon, G. T. Finnigan, N. S. North, J. Zheng, and T. Lyche. T-spline simplification and local refinement. *ACM Transactions on Graphics*, 23:276 – 283, 2004.

[48] C. Shu and G. Roth. Free-form surface reconstruction from multiple images. In *3-D Digital Imaging and Modeling, 2003. 3DIM 2003. Proceedings.*, pages 163–174, 2003. doi: 10.1109/IM.2003.1240246.

[49] H. Speleers and C. Manni. Effortless quasi-interpolation in hierarchical spaces. *Numerische Mathematik*, pages 1–30, 2015. doi: 10.1007/s00211-015-0711-z. Online.

[50] L. Stanculescu, R. Chaine, M.-P. Cani, and K. Singh. Sculpting multi-dimensional nested structures. *Computers and Graphics*, 37(6):753–763, 2013. doi: 10.1016/j.cag.2013.05.010. Special Issue: Shape Modeling International (SMI) Conference 2013.

[51] U. Trottenberg, C.W. Oosterlee, and A. Schüller. *Multigrid*. Academic Press, 2001. ISBN 9780127010700.

[52] VikingScientist. LR B-spline introduction. Online, 2012. URL `play.google.com/store/apps/details?id=com.vikingscientist.lr.introduction&hl=cs`.

[53] A.-V. Vuong, C. Giannelli, B. Jüttler, and B. Simeon. A hierarchical approach to adaptive local refinement in isogeometric analysis. *Computer Methods in Applied Mechanics and Engineering*, 200(49-52):3554–3567, 2011. doi: 10.1016/j.cma.2011.09.004.

[54] V. Weiss, L. Andor, G. Renner, and T. Varády. Advanced surface fitting techniques. *Computer Aided Geometric Design*, 19:19–42, 2002.

[55] A. Yvart, S. Hahmann, and G.-P. Bonneau. Smooth adaptive fitting of 3D models using hierarchical triangular splines. *Shape Modeling and Applications, International Conference on*, 0:13–22, 2005. doi: 10.1109/SMI.2005.43.

[56] A. Yvart, S. Hahmann, and G.-P. Bonneau. Hierarchical triangular splines. *ACM Trans. Graph.*, 24(4):1374–1391, 2005. doi: 10.1145/1095878.1095885.

[57] U. Zore and B. Jüttler. Adaptively refined multilevel spline spaces from generating systems. *Computer Aided Geometric Design*, 31(7-8):545 – 566, 2014. doi: 10.1016/j.cagd.2014.04.003.

[58] U. Zore, B. Jüttler, and J. Kosinka. On the linear independence of (truncated) hierarchical subdivision splines. Technical Report 9, NFN Geometry + Simulation, 2014. Available at www.gs.jku.at.

Curriculum Vitae

Personal Details

Full Name	Gábor Kiss
Title	Magister
Date of birth	May 6, 1987
Place of birth	Nové Zámky, Slovakia
Nationality	Slovak

Education

2011 – Present	Johannes Kepler Universität Linz Institute of Applied Geometry Doctoral Program "Computational Mathematics"
2009 – 2011	Faculty of Mathematics, Physics and Computer Science, Comenius University, Bratislava Field of study: Computer Graphics and Geometry Master degree
2006 – 2009	Faculty of Mathematics, Physics and Computer Science Comenius University, Bratislava Field of study: Applied Informatics Bachelor degree

Conferences and workshops attended

Aug. 25 – 31, 2014	Workshop on Approximation Theory, CAGD, Numerical Analysis and Symbolic Computation
Jun. 29 – Jul., 1 2014	Geometric Modeling and Processing
Apr. 7 – 10, 2014	IsoGeometric Analysis and Applications 2014
Jun. 28 – Jul. 3, 2012	Eighth International Conference on Mathematical Methods for Curves and Surfaces

Jun. 18 – 23, 2012	Summer school: IsoGeometric Analysis: a New Paradigm in the Numerical Approximation of PDEs
Jan. 2011	Visual Computing Trends 2011
May 2010	Spring Conference on Computer Graphics 2010

Scientific stays

Feb. 2013 – Jan. 2014	MTU Aero Engines München, Germany
Aug. 20 – Sep. 26	Seoul National University, South Korea. Visiting the group of Prof. Myung-Soo Kim

Publications

G. Kiss, C. Giannelli, and B. Jüttler. Algorithms and data structures for truncated hierarchical B-splines. In M. Floater et al., editors, Mathematical Methods for Curves and Surfaces, volume 8177 Lecture Notes in Computer Science, pages 304–323. Springer, 2014.

G. Kiss et al. Adaptive CAD model (re-)construction with THB-splines. Graphical Models, 76:273 – 288, 2014.

Projects

Feb. 2013 – Jan. 2014	**EXAMPLE** - Exact and Adaptive Modeling and Simulation of the Air Passage of Aircraft Engines
Jun. 2012 – Present	**G+Smo** – Geometry + Simulation modules (C++ library)

Sworn Declaration

I hereby declare under oath that the submitted doctoral dissertation has been written solely by me without any outside assistance, information other than provided sources or aids have not been used and those used have been fully documented. The dissertation here present is identical to the electronically transmitted text document.