Robust & Efficient Treatment of Industrial-Grade CAD Geometries for a Flat-Top Partition of Unity Method

Dissertation

zur

Erlangung des Doktorgrades (Dr. rer. nat.)

der

Mathematisch-Naturwissenschaftlichen Fakultät

der

Rheinischen Friedrich-Wilhelms-Universität Bonn

vorgelegt von

Albert Ziegenhagel

aus

Rastatt

Bonn 2022

Angefertigt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät
der Rheinischen Friedrich-Wilhelms-Universität Bonn

1. Gutachter: Prof. Dr. Marc Alexander Schweitzer

2. Gutachter: Prof. Dr. Jürgen Dölz

Tag der Promotion: 08. Dezember 2022

Erscheinungsjahr: 2022

Abstract

Even today, the treatment of industrial-grade geometries is a huge challenge in the field of numerical simulations. The geometries that are created by *computer aided design* (CAD) are often very complex and contain many flaws. Hence the discretization by mesh-based methods like the *finite element method* (FEM) is very time consuming and can take several months when human interaction is required. Therefore, a growing interest in so-called *meshfree* methods arose in the scientific community over the last few decades.
One such meshfree method is the *partition of unity method* (PUM), which is very promising because of its flexibility due to its very abstract formulations. But even though the PUM is meshfree in its core, the treatment of complex geometries is still lacking. In this thesis we develop methods to close that gap.
First we propose a post-processing step to the original cover construction algorithm employed in the PUM, that guarantees that stable approximation spaces can be constructed for arbitrary geometries in two and three space-dimensions. Then, we tackle the problem of efficient and robust integration in 2D, by proposing a *monotone decomposition* of the input geometry. By exploiting properties of the resulting decomposition, we can prove that all required intersection operations can be implemented reliably. By adding all *inflection points* of the domain's boundary when constructing local decompositions of the integration domains, we can prove that the resulting curved triangles always form a valid decomposition. In 3D, we propose to create a linear approximation of the input geometry. The linear representation allows all subsequent operations to be performed reliably and fast. Then, we develop a method to estimate the domain approximation error and relate that error to the approximation error of the PUM discretization. Refinement controlled by those error estimates then yields a method that can overall converge with optimal rates.
All methods proposed throughout that thesis are validated by numerical experiments. Thereby, we demonstrate the robustness on real-world industrial use cases. In 2D, we present results for a *shell problem* on the door of a car. In 3D, results for mechanical parts of the landing-gear of an Airbus A380 are presented.

Acknowledgements

I would like to take the opportunity to thank the people without whom I would not have been able to complete this research. First, I want to thank by supervisor Marc Alexander Schweitzer. He was the one who introduced the topic of numerical simulations to me. In the process, some of his own enthusiasm for the subject transferred over to me. Without his continued support, I probably would have never gotten this far. Thank you! I am also thankful to all the members of the "PUMA"-team at Fraunhofer SCAI and the Institute for Numerical Simulation at the University of Bonn. The inspiring discussions and welcoming atmosphere always made it a pleasant and fun environment to work.
Last but not least, I want to thank my family. My parents and my brother are the ones who supported me from the beginning, through school, through university and through everyday live. Without them, I would have never been able to be where I am now.

Contents

Introduction

Over the last decades, the prediction of physical behavior by numerical simulations carried out on computers has become more and more important in the industry. One of the main desires behind this development is to replace expensive real-world experiments by comparable simulations that can be performed both faster, as well as cheaper. Nowadays, the undisputable methods of choice across the industry to perform those simulations are the *finite difference method* (FDM), *finite volume method* (FVM) and *finite element method* (FEM). A common property of all these methods is that they are *mesh-based* methods. That means, they all require that the geometry of the object that is subject to the numerical simulation is discretized into a so-called *mesh* or *grid*. But the generation of those meshes is not an easy task and often needs human interaction. In [24], the time that is required to create an analysis suitable geometry and mesh for a FEM simulation is estimated to take up to 80% of the overall simulation time, where 60% are required to prepare the geometry and 20% for the mesh generation itself. In [60] an example from the automotive industry is given, stating that the process until an analysis suitable mesh for a vehicle is available can take up to four month. Due to the models of cars, planes, ships, etc. becoming increasingly complex over time, we expect those numbers to be even higher and not lower nowadays.

A set of methods that try to overcome these costs are so-called *meshfree* (or *meshless*) methods. One of the earliest approaches that falls into that category is *smoothed particle hydrodynamics* (SPH) [45,84], which is a Lagrangian particle method. The *diffuse element method* (DEM) introduced in [87] was initially described as a generalization of the FEM by its authors, but it can also be considered to be based on moving-least squares (MLS) approximation spaces and hence closely related to SPH. The *element-free Galerkin method* (EFGM) introduced in [11] can be seen as an extension of the DEM, where some aspects that have been omitted in the DEM were reintroduced to increase its accuracy. Similarly, the *reproducing kernel particle methods* (RKPM) developed in [82, 83] can be seen as an extension of the original SPH, where correction functions were introduced to allow the exact reproduction of polynomial functions.

In [8, 9] the *partition of unity method* (PUM) has been introduced as a very general framework that uses the abstract mathematical concept of a partition of unity (PU) and an additional extrinsic basis to construct the involved approximation spaces. Here, the choice of the additional basis is basically arbitrary, which makes the method very powerful and especially well suited for problems where the solution is not necessarily very smooth. The *generalized finite element method* (GFEM) [127, 128] and the *extended finite element*

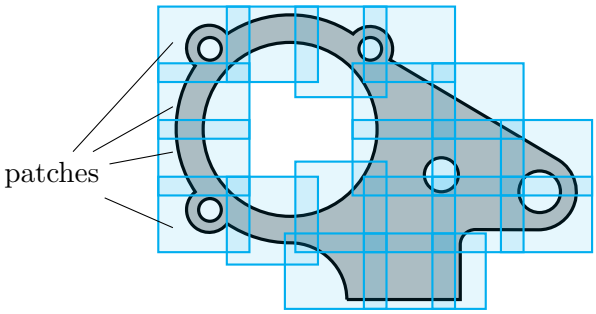

Figure 1: A set of patches (blue boxes) overlaps the domain of a mechanical part (gray) and forms a cover for the PUM.

method (XFEM) [12] are variants of the PUM where the classical linear shape functions of a finite element mesh are used to create the PU. Thus they are not necessarily entirely meshfree and can be seen as a combination of the FEM and the PUM. On the other hand, in [128] the employed mesh does not need to resolve the boundary of the domain, but simply overlap it, which makes the mesh generation domain independent and hence very simple. Both, the GFEM and the XFEM employ special discontinuous and singular basis functions to resolve local behavior imposed by cracks.

In [51] the *particle-PUM* has been introduced as a variant that is truly meshfree. Here, the PU functions are constructed by building a d-binary tree over an initial set of particles and then stretching the disjoint leaves of the tree in such a way, that an overlapping set of *patches* is retrieved, which forms the so-called *cover* (compare Figure 1). If the stretching is done according to carefully chosen stretch factors, the resulting PU functions fullfil the so-called *flat-top* property. While this flat-top property implies that the PU functions themselves can only reproduce constant functions, and hence additional smooth local basis functions need to be added everywhere to gain higher order approximation properties, they do on the other hand ensure that a globally stable basis can be created, regardless of the employed local basis functions. This variant of the PUM is the one considered in this thesis, although the initial set of particles is omitted in the construction of the cover. Therefore, we will simply refer to it as the PUM in the following.

We believe that the very abstract properties that are presupposed by each of the components involved in the construction of the PUM make it a very promising method to tackle the challenges of numerical simulations for all kinds of problems. In fact, it has been shown that the PUM framework allows the efficient treatment of many tasks that are typically encountered in meshfree (and mesh-based) methods. To enforce essential boundary conditions, an approach based on Nitsche's method [55] as well as a conforming approach [107] have been presented. To solve the resulting equation systems, an efficient multilevel solver has been developed [53] and it has been shown that efficient smoothers can be constructed, even when the local function spaces include polynomials of high degrees [50]. The parallelization of the method has been demonstrated by utilizing a domain decomposition approach and space filling curves in [54], which shows the scalability of the method to very large problems. Furthermore, it has been shown that it is possible to

always retrieve a stable basis in the PUM, regardless of the employed local basis functions, while keeping optimal approximation properties [105, 108]. This does especially allow the introduction of arbitrary enrichment functions that can be designed by using any *a priori* knowledge of the solution to the problem at hand. Besides from the classical case of analytical enrichments for non-smooth behavior at material interfaces or discontinuities and singularities at cracks, some additional examples where numerical enrichments have been used in the PUM are to solve the Schrödinger equation [3, 70] and in a stable global-local enrichment scheme [13]. Further topics are the treatment of non-linear materials [130], variational mass lumping for explicit dynamics [109], obstacle and contact problems [66] and the non-invasive coupling with classical FEM codes [111, 112].

But even though the method is meshfree in its roots, the simulation domain still needs to be considered. First, the construction of the cover of patches has some dependencies on the domain: The most obvious one is that the whole domain needs to be covered by those patches to make sure a solution can be approximated everywhere. On the other hand, we do not want to generate any patches that do not overlap the domain at all. Hence we need to robustly determine whether a patch overlaps the domain or not. Furthermore, it is not sufficient to ensure the flat-top property of the PU functions independent of the domain. The flat-top property of the PU for a patch is guaranteed by the existence of a region in the patch, that is not overlapped by any other patch. But if that flat-top region has an empty intersection with the domain, it is insufficient to guarantee that we can retrieve a globally stable basis. This deficiency has been reported in [110] and a possible solution has been proposed in [34, 35]. But the solution therein did not consider employing local function spaces with polynomials of higher degrees, nor the required properties of the cover construction to enable the construction of efficient multilevel solvers. Hence, a cover construction that guarantees a flat-top property under consideration of arbitrary domains is still an open question, and a new solution is proposed in this thesis.

The other step in which the simulation domain needs to be considered in the PUM is during the assembly of the stiffness matrix. Here, we need to compute integrals over intersection domains of patches and the complete simulation domain. This assembly step is usually the most expensive task of the overall simulation in many meshfree methods.[1] Hence, it is of high interest to use only efficient algorithms during that step. The requirement to evaluate integrals over domains, that result from intersecting rectangular or triangular objects with the simulation domain is not exclusive to the PUM. In fact, many other methods need to solve similar problems. E.g. in the GFEM mentioned before, where the finite element mesh that is used to construct the PU does not resolve the domain's boundary, one needs to evaluate such integrals. Similarly, a whole class of methods that falls under the category of *immersed* or embedded boundary methods is confronted with the same problem. Some representatives of those methods are the *fictitious domain methods* [73, 102], *unfitted finite element method* [10], *CutFEM* [16] and *finite cell methods* [31, 90]. The classical approach to tackle the integration problem in those methods is based on a space-tree decomposition where every cell that intersects the boundary of the domain is subdivided recursively into uniform subcells until a predefined depth is reached. On those final cells a known

[1] Under the assumption that an efficient solver for the linear equation system is available.

Gaussian quadrature rule for the subcells is applied, without further resolution of the domain's boundary. Hence, the integration is still subject to the discontinuity imposed by the boundary, and is thus not exact. In fact, the accuracy depends heavily on the depth of the space-tree decomposition and very deep refinements lead to a rapidly increasing number of quadrature points which has severe consequences for the overall efficiency of the method [32, 76]. On the other hand, a more efficient variant of the integration has been presented more recently in [76]. Here, an idea from the so-called *NURBS enhanced finite element method* (NEFEM) [118] has been picked up. In the NEFEM, the triangular elements that are located at the boundary are mapped to a curved triangle that than represents the boundary exactly. For the numerical integration, this map is used, such that the integration can be carried out on a reference triangle with known Gaussian quadrature rules. The ideas from the NEFEM are then combined with the space-tree subdivision, such that the recursive refinement of boundary cells is only performed until the subcells intersect the boundary of the simulation domain in a simple enough manner. Then, the resulting intersections can be decomposed into triangles easily, and a curved map is applied to create an exact quadrature rule for each of the final, curved triangular cells.

Unfortunately, those approaches turn out to be insufficient to be applied in the setting of the PUM. First we want to emphasis that due to the overlapping patches, we usually have more rectangular cells that intersect the boundary of the simulation domain and need to be integrated on, than in comparable methods as given above. Hence, it is even more important to generate as few quadrature points per such cell as possible. Furthermore, the construction of a multilevel solver in the PUM requires the assembly of the stiffness matrix for covers of different refinement degrees. Especially very coarse covers are of interest, since the coarsest possible level heavily influences the performance of the multilevel method. With an approach as given above, the integration on coarse covers will require many initial subdivisions of the space-tree since the geometry is very complex relative to the coarse cells. Hence, the assembly of these coarse covers can take a fair amount of time, which is undesirable. We will thus describe a new approach that creates curved, triangular integration cells right from the original cells, without the need to apply any recursive space-tree subdivisions at all.

Another difficulty that often gets little attention in scientific research papers is the vast complexity and imperfections of actual industrial geometries. In [4], a representative of Siemens PLM, states that the models that are commonly used in scientific research[2] are "hopelessly unrealistic" and not good representatives of real-world data. Common errors that are encountered in industrial geometries described by the ISO *standard for the exchange of product model data* (STEP) are listed in [138]. In [137] a case study has been conducted that categorized the most common flaws in such geometries. Such flaws are usually introduced during the construction of these geometries via *computer aided design* (CAD) which represents one of the main sources for geometries used in the industry. Hence, we are convinced that even seemingly simple operations like the test whether a point is within a CAD geometry or whether a box intersects the boundary of the CAD

[2] He mentions the famous *Stanford Bunny* [131] and *Happy Buddha* [25] from the *Stanford 3D Scanning Repository* (`http://graphics.stanford.edu/data/3Dscanrep/`). *Accessed: 2022-10-17.*

geometry, require thorough understanding of the underlying mathematical equations, such that these problems can be solved robustly. To this end, we tackle the problem from the bottom up and show how methods can be developed that aim to answer these fundamental questions reliably for our use cases.
In the remainder of this thesis, we deal with the problems mentioned above. It is organized as follows:

- We start by giving a summary of the basic theory of the partition of unity method and its fundamental approximation properties. This is followed by short summaries of topics related to the PUM that are relevant to this thesis.

- Then, we propose a new post-processing step for the cover construction that resolves instabilities that can arise when considering complex geometries. The numerical examples at the end of this chapter show that the post-processing step is in fact capable of constructing stable function spaces on complex two- and three-dimensional geometries and that our method does achieve optimal convergence rates. Furthermore, we show that we are still capable of creating an efficient multilevel solver, after the different cover levels have been adjusted by the proposed post-processing step.

- The description of the PUM and the cover post-processing is followed by a short summary of geometries described by the STEP standard. We introduce the basic topological and geometrical representations, the implications of those descriptions and the mathematical formulations used therein. We give a short explanation why we can usually not expect real-world geometries to be "flawless" in the sense that we have to deal with gaps, self-intersections and other defects in these geometries.

- We then describe the problems we need to solve for an efficient and robust integration in the PUM on two-dimensional geometries. We first propose to implement a *monotone decomposition* of the curves that describe the input geometry and show that such a decomposition can be computed efficiently for all supported input geometries. We then show how this monotone decomposition can be used to guarantee the robust computation of all intersection operations that are required to be performed during the assembly step of our PUM. Next we show how to decompose the resulting domains into curved triangles. We propose to insert all *inflection points* into the point sets when creating the local triangulations. We can prove that the insertion of those points allows us to develop efficient and robust tests to detect invalid curved triangles and show how to resolve these cases based on the previously developed test. Finally, we describe how a mapping based on a *transfinite interpolation* can be used to create quadrature points for each curved triangle, which completes the description of how to deal with two-dimensional geometries.

- We end the handling of two-dimensional geometries by presenting results of numerical examples. First, we show that we can still achieve optimal convergence rates for curved domains and demonstrate the benefits of an exact boundary descriptions. This is followed by an extensive analysis of the performance characteristics of the

presented approach. Finally, we present a highly complex industrial use case where we solve a *shell problem* on a real-world CAD geometry of the door of a car using the proposed PUM.

- For three-dimensional solid geometries, we then start by motivating why an exact treatment of curved domains is a far more difficult problem than it was in two-dimensions. Hence, we propose to use an approximate representation for such geometries. After developing a simple approach to construct such approximations, we continue to describe how those approximations allow us to compute the required intersection operations and decompositions into integration cells, both, robust and efficiently. Finally, we develop a strategy to estimate the errors of the geometry approximation and put it into relation to the approximation error of the PUM discretization, to yield a method that can overall converge with optimal rates.

- The handling of three-dimensional geometries is completed by numerical examples that first show that we are in fact capable of achieving close to optimal convergence rates on curved, three-dimensional solid geometries. We then present a further industrial use case with a non-smooth solution and finally show that we can obtain results on real-world mechanical parts from the landing gear of an Airbus A380, that are in good agreement with reference solutions obtained by a classical finite element method.

1 Partition Of Unity Method

In this chapter we give a brief overview of the Partition of Unity Method (PUM) that was introduced in [8, 9] as a generalization of the FEM and is based on [7]. These basic theoretical ideas have been picked up by Schweitzer and Griebel who then developed an entire framework of algorithms around the PUM for an efficient and stable discretization [52], handling of boundary conditions [55, 107], solving of the resulting equation systems [50, 53], stable enrichment methods [105, 108] and more.

1.1 Approximation Properties

Let us consider a Lipschitz domain $\Omega \subset \mathbb{R}^d$ in d dimensions. To obtain a partition of unity space V^{PU} we first construct an *open cover* $C_\Omega = \{\omega_i \mid i = 1, \ldots, N\}$ consisting of open sets ω_i, the so-called *patches*, that cover the whole domain, i.e. $\bar{\Omega} \subset \bigcup_{i=1}^N \omega_i$. The geometric information of the patches is used to construct a partition of unity (PU) $\{\varphi_i\}_{i=1}^N$ with $\mathrm{supp}(\varphi_i) = \omega_i$ for all $i = 1, \ldots, N$. Additional local approximation spaces $V_i := V_i(\omega_i) := \mathrm{span}\langle \vartheta_i^m \rangle$ give the final ingredient to define the global space as

$$V^{\mathrm{PU}} := \sum_{i=1}^{N} \varphi_i V_i = \mathrm{span}\langle \varphi_i \vartheta_i^m \rangle$$

with m being a counting index over the local basis functions ϑ_i^m employed on each patch. Let us now take a look at the basic properties that each of those ingredients need to fullfil to show the approximation properties of the PUM as given in [9].

Definition 1.1 (Partition of Unity)**.** Let $\Omega \subset \mathbb{R}^d$ be an open set, $\{\omega_i \mid i = 1, \ldots, N\}$ be an open cover of Ω satisfying a point-wise overlap condition

$$\exists M \in \mathbb{N} \quad \text{s.t.} \quad \mathrm{card}\{i \mid \boldsymbol{x} \in \omega_i\} \leq M \quad \forall \boldsymbol{x} \in \Omega.$$

Let φ_i be a Lipschitz partition of unity subordinate to the cover $\{\omega_i\}$ satisfying

$$\mathrm{supp}(\varphi_i) \subset \mathrm{closure}(\omega_i) \quad \forall i = 1, \ldots, N,$$

$$\sum_{i=1}^{N} \varphi_i \equiv 1 \quad \text{on } \Omega,$$

$$\|\varphi_i\|_{L^\infty(\mathbb{R}^d)} \leq C_\infty \qquad \text{and}$$

$$\|\nabla \varphi_i\|_{L^\infty(\mathbb{R}^d)} \leq \frac{C_\nabla}{\operatorname{diam}(\omega_i)}$$

where C_∞ and C_∇ are two positive constants. Then $\{\varphi_i\}$ is called a (M, C_∞, C_∇) partition of unity subordinate to the cover $\{\omega_i\}$. The partition of unity $\{\varphi_i\}$ is said to be of degree $m \in \mathbb{N}_0$ if $\{\varphi_i\} \subset C^m(\mathbb{R}^d)$. The covering sets $\{\omega_i\}$ are called patches.

Definition 1.2 (Global Partition of Unity Space)**.** Let $\{\omega_i\}$ be an open cover of $\Omega \subset \mathbb{R}^d$ and let $\{\varphi_i\}$ be a (M, C_∞, C_∇) partition of unity subordinate to $\{\omega_i\}$. Let $V_i \subset H^1(\omega_i \cap \Omega)$ be given. Then the space

$$V^{\mathrm{PU}} := \sum_{i=1}^{N} \varphi_i V_i = \left\{ \sum_{i=1}^{N} \varphi_i v_i \mid v_i \in V_i \right\} \subset H^1(\Omega)$$

is called a PUM space. The PUM space V^{PU} is said to be of degree $m \in \mathbb{N}$ if $V^{\mathrm{PU}} \subset C^m(\Omega)$. The spaces V_i are referred to as local approximation spaces.

Theorem 1.1 (Approximation)**.** *Let $\Omega \subset \mathbb{R}^d$ be given. Let $\{\omega_i\}$, $\{\varphi_i\}$ and $\{V_i\}$ be as in Definitions 1.1 and 1.2. Let $u \in H^1(\Omega)$ be the function to be approximated. Assume that the local approximation spaces V_i have the following approximation properties: On each patch $\omega_i \cap \Omega$, the function u can be approximated by a function $v_i \in V_i$ such that*

$$\begin{aligned} \|u - v_i\|_{L^2(\omega_i \cap \Omega)} &\leq \hat{\epsilon}_i \quad \textit{and} \\ \|\nabla(u - v_i)\|_{L^2(\omega_i \cap \Omega)} &\leq \tilde{\epsilon}_i \end{aligned}$$

hold for all $i = 1, \ldots, N$ with some local error bounds $\hat{\epsilon}_i$ and $\tilde{\epsilon}_i$. Then the function

$$u^{PU} = \sum_{i=1}^{N} \varphi_i v_i \in V^{PU} \subset H^1(\Omega)$$

satisfies the global estimates

$$\|u - u^{PU}\|_{L^2(\Omega)} \leq \sqrt{M} C_\infty \left(\sum_{i=1}^{N} \hat{\epsilon}_i^2 \right)^{\frac{1}{2}}, \tag{1.1}$$

$$\|\nabla(u - u^{PU})\|_{L^2(\Omega)} \leq \sqrt{2M} \left(\sum_{i=1}^{N} \left(\frac{C_\nabla}{\operatorname{diam}(\omega_i)} \right) \tilde{\epsilon}_i^2 + C_\infty^2 \hat{\epsilon}_i^2 \right)^{\frac{1}{2}}. \tag{1.2}$$

Proof. See [8, 9].

If we assume that M is independent of N, i.e. $M = O(1)$[1] the equations (1.1) and (1.2) can be reformulated to show that the PUM has both an h-version as well as a p- and hp-version.
To this end, let us consider $\operatorname{diam}(\omega_i) \approx h_i$ for all $i = 1, \dots, N$ with $h_i < h$ and that all local approximation spaces V_i contain polynomials of order p. Due to the Bramble-Hilbert lemma the local approximation spaces satisfy the local error bounds

$$\begin{aligned} \|u - u_i\|_{L^2(\Omega\cap\omega_i)} &\leq Ch^{p+1}\|u\|_{H^k(\Omega\cap\omega_i)} =: \hat{\epsilon}_i, \\ \|\nabla(u - u_i)\|_{L^2(\Omega\cap\omega_i)} &\leq Ch^{p}\|u\|_{H^k(\Omega\cap\omega_i)} =: \tilde{\epsilon}_i \end{aligned} \tag{1.3}$$

with $u \in H^k(\Omega)$, $k \geq 1$ and $p \leq k - 1$. Using these local error bounds the equations (1.1) and (1.2) can be reformulated to become

$$\|u - u^{\mathrm{PU}}\|_{L^2(\Omega)} \leq MC_\infty Ch^{p+1}\|u\|_{H^k(\Omega)}, \tag{1.4}$$

$$\|\nabla(u - u^{\mathrm{PU}})\|_{L^2(\Omega)} \leq M\sqrt{2(C_\nabla + C_\infty)}Ch^{p}\|u\|_{H^k(\Omega)} \tag{1.5}$$

which resemble the error estimates for an h-version of a classical finite element method. Similarly a p-version can be constructed when we assume local error bounds of the form

$$\begin{aligned} \|u - u_i\|_{L^2(\Omega\cap\omega_i)} &\leq Ch_i p^{-\mu}\|u\|_{H^k(\Omega\cap\omega_i)}, \\ \|\nabla(u - u_i)\|_{L^2(\Omega\cap\omega_i)} &\leq Cp^{-\mu}\|u\|_{H^k(\Omega\cap\omega_i)} \end{aligned} \tag{1.6}$$

for some appropriate $\mu > 0$. These bounds again hold for local approximation spaces V_i that contain polynomials of order $p \leq k - 1$. But more generally the PUM allows to use *any* construction of the local approximation spaces V_i that is best suited to reduce the local error bounds $\hat{\epsilon}_i$ and $\tilde{\epsilon}_i$. This does especially include the possibility to exploit a priori knowledge of any local behavior of the solution to construct local *enrichment* functions that are added to the function spaces V_i and can immediately achieve much better local approximation properties than either h-refinement as given in (1.3) or p-refinement as given in (1.6). They are commonly used to, but not limited to, resolve non-smooth components of the solution. Examples for such enrichment functions are functions to resolve discontinuities in the derivatives as they arise at material interfaces or singularities as they arise at re-entrant corners or at crack tips.
Let us now turn to the stability of the method. To this end, we should note that the estimates given in Theorem 1.1 do only state how the approximations u^{PU} converge to the real solution, but the functions $u^{\mathrm{PU}} = \sum_i \varphi_i u_i$ could be non-unique. This means that our shape functions $\varphi_i\vartheta_i$ can be linearly dependent. Consider the following example:

Example 1.1. Given the domain $\Omega := (0, 1)$ and the partition of unity functions

$$\varphi_1(x) := 1 - x, \qquad \varphi_2(x) := x$$

[1] In fact this limitation can be lifted when we do additionally assume that our partition of unity functions are non-negative, i.e. $0 \leq \varphi_i(\boldsymbol{x}) \leq 1$ for all $\boldsymbol{x} \in \Omega$, $i = 1, \dots, N$. In this case Theorem 1.1 can be improved to be less dependent on M [109].

with the local polynomial basis functions

$$\vartheta_1^1(x) := 1, \qquad \vartheta_1^2(x) := x \quad \text{and}$$
$$\vartheta_2^1(x) := 1, \qquad \vartheta_2^2(x) := x.$$

This results in the global PUM space

$$V^{\mathrm{PU}} := \operatorname{span}\langle (1-x), (1-x)x, x, x^2 \rangle$$

which spans the space of all quadratic polynomials, but uses four functions to do so. Since there are only three linearly independent polynomials of degree two, our shape functions $\varphi_i \vartheta_i^m$ have to be linearly dependent. In fact in our example it is $\varphi_1(x)\vartheta_1^2(x) = \varphi_2(x)\vartheta_2^1(x) - \varphi_2(x)\vartheta_2^2(x) = x - x^2$ and hence $\langle \varphi_i \vartheta_i^m \rangle$ is a generating set for V^{PU} but not a basis.

To overcome this issue Babuška and Melenk suggested to use PU functions φ_i that are flat, i.e. $\varphi_i \equiv 1$, on some subset $\tilde{\omega}_i \subset \omega_i \cap \Omega$. Such requirements have been formalized in [106] where definitions of the *flat-top property* and an *admissible cover* are given as follows:

Definition 1.3 (Flat-top property)**.** Let $\{\varphi_i \mid i = 1, \ldots, N\}$ be a partition of unity according to Definition 1.1. Let us define the sub-patches $\omega_i^{\mathrm{FT}} \subset \omega_i$ such that $\varphi_i|_{\omega_i^{\mathrm{FT}}} \equiv 1$. The PU is said to have the flat-top property, if there exists a constant C_{FT} such that for all patches $\omega_i = \operatorname{supp}(\varphi_i)$

$$\mu(\omega_i) \leq C_{\mathrm{FT}} \mu(\omega_i^{\mathrm{FT}}) \tag{1.7}$$

where $\mu(A)$ denotes the Lebesgue measure of $A \subset \mathbb{R}^d$.

Definition 1.4 (Admissible cover)**.** Let $\Omega \subset \mathbb{R}^d$ be an open set. Let $\omega_i \subset R^d$ be open sets with $\omega_i \cap \Omega \neq \emptyset$ for $i = 1, \ldots, N$. Furthermore let us introduce the covering index $\lambda_{C_\Omega} : \Omega \to \mathbb{N}$ such that

$$\lambda_{C_\Omega}(\boldsymbol{x}) = \operatorname{card}(\{\omega_i \in C_\Omega \mid \boldsymbol{x} \in \omega_i\}) \tag{1.8}$$

with the collection $C_\Omega = \{\omega_i \mid i = 1, \ldots, N\}$ being called an admissible cover of Ω and the sets ω_i are denoted admissible patches if the following conditions are satisfied

- Global covering:
$$\bar{\Omega} \subset \bigcup_{i=1}^{N} \omega_i.$$
- Minimal overlap: Given the subset
$$\omega_i^{\mathrm{FT}} := \{\boldsymbol{x} \in \omega_i \mid \lambda_{C_\Omega}(\boldsymbol{x}) = 1\} \subseteq \omega_i, \tag{1.9}$$
there exists a constant C_{FT} such that
$$\mu(\omega_i) \leq C_{\mathrm{FT}} \mu(\omega_i^{\mathrm{FT}}). \tag{1.10}$$

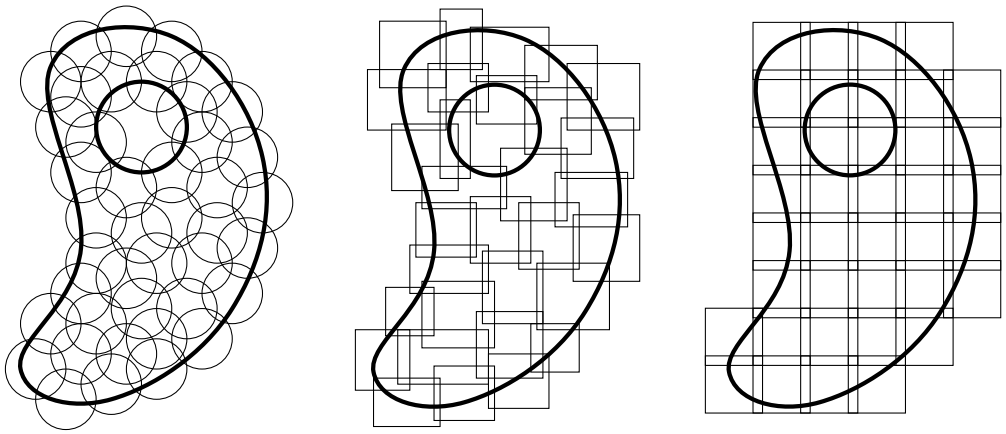

Figure 1.1: Different examples for covers C_Ω over the same domain Ω.
Left: Using circles for all patches ω_i.
Center: Using non-uniformly distributed axis-aligned rectangles for all patches ω_i.
Right: Using uniformly distributed axis-aligned squares for all patches ω_i.

- Bounded overlap: There exists a constant $M > 0$ such that for any $\boldsymbol{x} \in \Omega$, it holds that
$$\|\lambda_{C_\Omega}\|_{L^\infty(\Omega)} < M \ll N. \tag{1.11}$$
- Sufficient overlap: There exists a constant $C_S > 0$ such that for any $\boldsymbol{x} \in \Omega$ there is at least one cover patch ω_i such that $\boldsymbol{x} \in \omega_i$ and
$$\operatorname{dist}(\boldsymbol{x}, \partial\omega_i) \geq C_S \operatorname{diam}(\omega_i). \tag{1.12}$$
- Comparability of neighboring patches: A subset
$$C_i := \{\omega_j \in C_\Omega \mid \omega_j \cap \omega_i \neq \emptyset\} \subset C_\Omega \tag{1.13}$$
is called a local neighborhood or local cover of a particular cover patch $\omega_i \in C_\Omega$. There exists a constant $C_\mathcal{N} \geq 1$ such that for all patches $\omega_j, \omega_i \in C_\Omega$ the implication
$$\omega_j \cap \omega_i \neq \emptyset, \quad \operatorname{diam}(\omega_i) \geq \operatorname{diam}(\omega_j) \qquad \Longrightarrow \qquad \frac{\operatorname{diam}(\omega_i)}{\operatorname{diam}(\omega_j)} \leq C_\mathcal{N} \tag{1.14}$$
holds.

Given such an admissible cover (some examples are given in Figure 1.1) we can construct a partition of unity by utilizing Shepard functions like
$$\varphi_i := \frac{W_i(\boldsymbol{x})}{\sum_{\omega_j \in C_i} W_j(\boldsymbol{x})} \tag{1.15}$$
where C_i is the set of all geometrical neighbors ω_j of the patch ω_i as defined in equation (1.13) and W_i being non-negative weight functions, i.e. $W_i(\boldsymbol{x}) > 0$ for all $\boldsymbol{x} \in \omega_i$.

Here, we want to point out that a very important property of the PUM is that the PU functions φ_i inherit their smoothness from the weight functions W_i. Thus we can construct arbitrarily smooth function spaces V^{PU} by choosing the right weight functions W_i. This property is especially important when we want to solve higher order PDEs. To construct those weight functions we assume that our patches are d-dimensional rectangles $\omega_i := \bigotimes_{m=1}^{d}((\boldsymbol{c}_i)_m - (\boldsymbol{r}_i)_m, (\boldsymbol{c}_i)_m + (\boldsymbol{r}_i)_m)$ with $\boldsymbol{c}_i$ being the center of the patch and $\boldsymbol{r}_i$ the anisotropic radius. This allows us to construct the weight functions W_i by tensor products of one-dimensional spline functions $\mathcal{W} : [-1, 1] \to \mathbb{R}$. Utilizing affine transformations $T_{i,m} : [(\boldsymbol{c}_i)_m - (\boldsymbol{r}_i)_m, (\boldsymbol{c}_i)_m + (\boldsymbol{r}_i)_m] \to [-1, 1]$ the weight functions W_i can be written as

$$W_i(\boldsymbol{x}) := \prod_{m=1}^{d} \mathcal{W} \circ T_{i,m}(x_m). \tag{1.16}$$

Compare Figure 1.2 for an exemplary construction of some linear partition of unity functions in 1D.

Lemma 1.1. *The PU defined by* (1.15) *with weights* (1.16) *defined on an admissible cover as given in Definition 1.4 is valid according to Definition 1.1 and satisfies Definition 1.3.*

Proof. See [106].

The final ingredient for a PUM space as given in Definition 1.2 are the local approximation spaces V_i. Again following the work of [106] we choose local approximation spaces that consist of a smooth polynomial part $\mathcal{P}^{p_i} := \operatorname{span}\langle \psi_i^s \rangle$ and problem dependent enrichment functions $\mathcal{E}_i := \operatorname{span}\langle \eta_i^t \rangle$ as

$$V_i^{p_i} := \mathcal{P}^{p_i} + \mathcal{E}_i = \operatorname{span}\langle \psi_i^s, \eta_i^t \rangle \tag{1.17}$$

where the ψ_i^s are polynomials of total degree $p \leq p_i$. Since we construct our patches ω_i from d-dimensional rectangles, a natural choice for the polynomials ψ_i^s is to employ tensor products of some one-dimensional polynomials. Throughout this thesis we use one-dimensional Legendre polynomials $\mathcal{L} : [-1, 1] \to \mathbb{R}$ that allow us to define the basis functions up to the degree p_i as

$$\{\psi_i^s(\boldsymbol{x}) \mid \psi_i^s(\boldsymbol{x}) := \prod_{m=1}^{d} \mathcal{L}^{q_m} \circ T_{i,m}(x_m), \quad \|\boldsymbol{q}\|_1 \leq p_i\} \tag{1.18}$$

where $\boldsymbol{q} = (q_m)_{m=1}^{d}$ is the multi-index of the polynomial degrees q_m per dimension.

1.2 Hierarchical Cover Construction

In this section we give an overview on the construction of an admissible cover C_Ω according to Definition 1.4. To this end, we employ an algorithm based on d-binary trees similar to what has been introduced in [52] and [56].

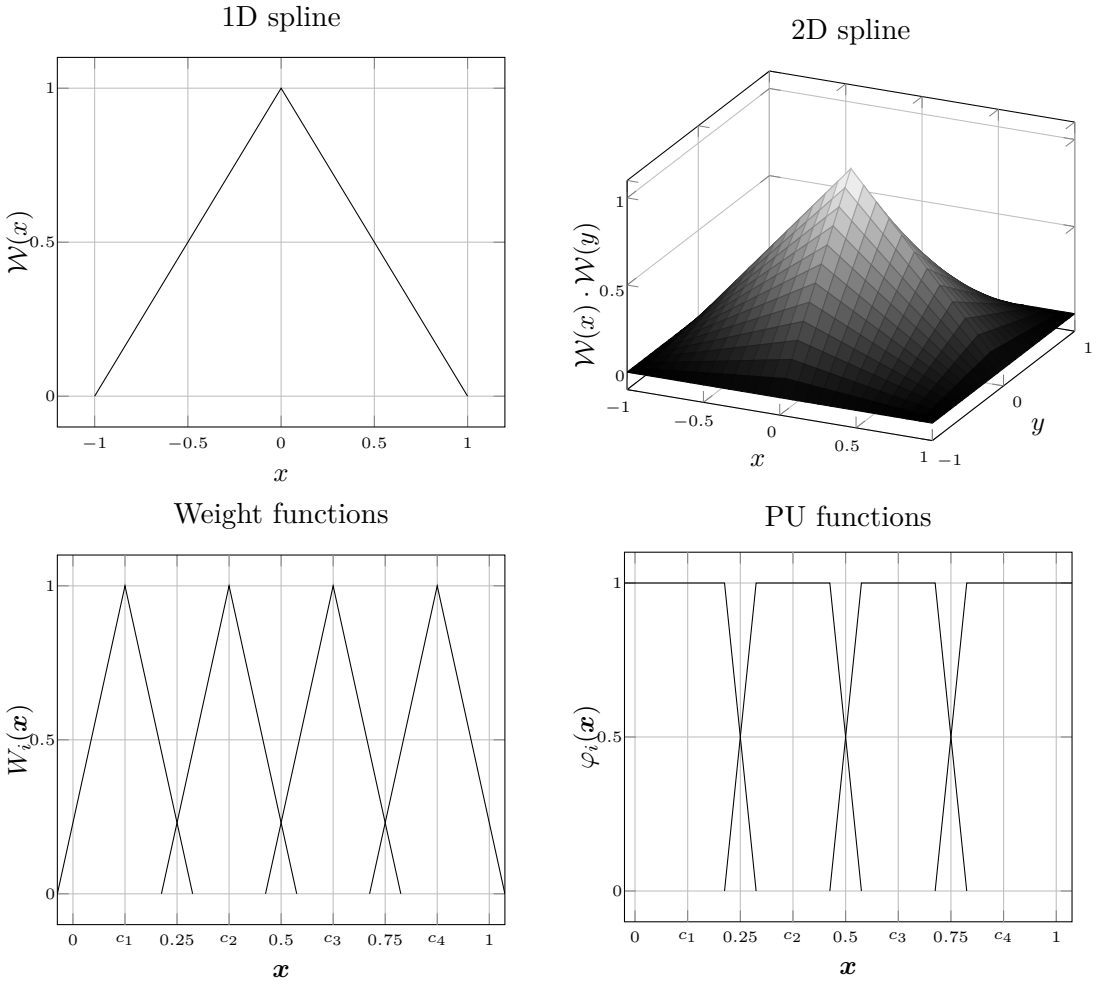

Figure 1.2: *Top, left:* One-dimensional linear splines $\mathcal{W}(x) := 1 - |x|$ on the reference domain $[-1, 1]$.
Top, right: Two-dimensional tensor product spline $\mathcal{W}(x) \cdot \mathcal{W}(y) = (1 - |x|)(1 - |y|)$ on the reference domain $[-1, 1]^2$.
Bottom, left: One-dimensional, linear weight functions W_i of four different patches with their centers at $c_1 = 0.125$, $c_2 = 0.375$, $c_3 = 0.625$ and $c_4 = 0.875$. All patches do have the same radius $r_i = 0.1625$. The weights have been constructed by transformation of the linear spline function $\mathcal{W}$ from $[-1, 1]$ to the respective domain $\omega_i := (c_i - r_i, c_i + r_i)$ of each patch.
Bottom, right: Resulting flat-top PU functions φ_i from the weights W_i by applying the Shepard construction given in equation (1.15).

Instead of creating the cover C_Ω directly, we start by creating a sequence of *disjoint covers* $\hat{C}_\Omega^k$ on levels $k \in \mathbb{N}_0$ as

$$\hat{C}_\Omega^k := \{\mathcal{C}_{i,k}\}, \quad \mathcal{C}_{i,k} \cap \mathcal{C}_{j,k} = \emptyset \qquad \forall i, j = 1, \ldots, N_k,\ i \neq j$$

with $N_k := \mathrm{card}(\hat{C}_\Omega^k)$, consisting of *disjoint, rectangular tree cells*

$$\mathcal{C}_{i,k} := \bigotimes_{m=1}^{d} ((\boldsymbol{a}_{i,k})_m, (\boldsymbol{b}_{i,k})_m)$$

with $\boldsymbol{a}_{i,k}, \boldsymbol{b}_{i,k} \in \mathbb{R}^d$ being the lower-bound and the upper-bound of the cell $\mathcal{C}_{i,k}$, respectively. These disjoint cells are then stretched into the patches $\omega_{i,k}$ of the admissible cover by applying a stretch factor $\alpha_{i,k}$ as follows

$$\omega_{i,k} := \bigotimes_{m=1}^{d} ((\boldsymbol{c}_{i,k})_m - \alpha_{i,k}\frac{(\boldsymbol{h}_{i,k})_m}{2}, (\boldsymbol{c}_{i,k})_m + \alpha_{i,k}\frac{(\boldsymbol{h}_{i,k})_m}{2}), \tag{1.19}$$

with $\boldsymbol{h}_{i,k} = \boldsymbol{b}_{i,k} - \boldsymbol{a}_{i,k}$ and $\boldsymbol{c}_{i,k} = \boldsymbol{a}_{i,k} + \frac{\boldsymbol{h}_{i,k}}{2}$. We create the disjoint covers $\hat{C}_\Omega^k$ by a tree subdivision algorithm.
For the root node with *depth* $l = 0$ we initialize its associated cell to the bounding box $R_\Omega \supset \bar{\Omega}$ of the domain Ω. By using R_Ω as the only cell $\mathcal{C}_{i,k}$ on the coarsest level $k = 0$ we can create the initial disjoint cover $\hat{C}_\Omega^0 := \{R_\Omega\} = \{\mathcal{C}_{1,0}\}$. Next we subdivide this cell uniformly into a set of smaller cells

$$S_{i,k} := \left\{ \bigotimes_{m=1}^{d} \left((\boldsymbol{a}_{i,k})_m + q_m \frac{(\boldsymbol{h}_{i,k})_m}{2}, (\boldsymbol{a}_{i,k})_m + (q_m + 1)\frac{(\boldsymbol{h}_{i,k})_m}{2} \right), \quad \|\boldsymbol{q}\|_\infty \leq 1 \right\}, \tag{1.20}$$

where $\boldsymbol{q} = (q_m)_{m=1}^d$ is the counting multi-index with $q_m \in \mathbb{N}_0$ per dimension and it is $\mathrm{card}(S_{i,k}) = 2^d$. For each cell $\mathcal{C}_j \in S_{i,k}$ we attach a child-node of depth $l+1$ to the node of $\mathcal{C}_{i,k}$ in the tree. After each refinement step we collect the cells associated with leaf nodes of the tree that have a non-empty intersection with the domain Ω into the set of cells on a level k, yielding $\hat{C}_\Omega^k$. Repeated application of the sub-division strategy to the leaves yields ever finer covers over the domain (compare Figure 1.3). Note that in each refinement step we can decide for each leaf cell whether it should be split into smaller cells or not. If a cell $\mathcal{C}_{i,k}$ is not split up, its node stays a leaf in the new tree and the associated cell will be part of the covers C_Ω^k on multiple levels $k \geq l_{i,k}$, where $l_{i,k}$ is the depth of the node in the tree corresponding to the cell $\mathcal{C}_{i,k}$. Finally we can transform all disjoint cells $\mathcal{C}_{i,k} \in \hat{C}_\Omega^k$ into patches $\omega_{i,k}$ by applying equation (1.19) which yields a cover C_Ω^k on each level (compare Figure 1.4).

Lemma 1.2. *There exists a choice of $\alpha_{i,k} \in (1, 2)$ for each cell $\mathcal{C}_{i,k}$ such that on a level k, the covers C_Ω^k constructed by the algorithm described above (and formalized below in Algorithm 1.1) are admissible covers according to Definition 1.4.*

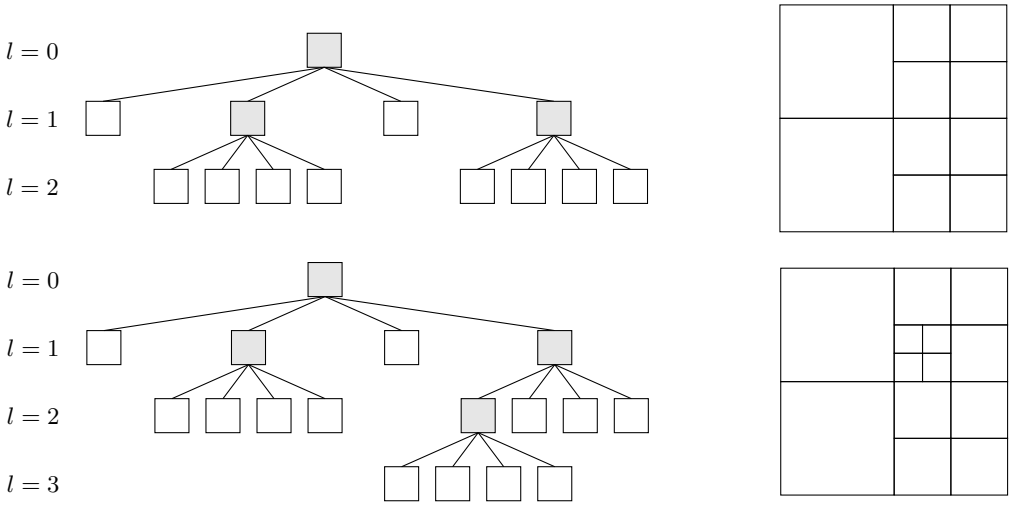

Figure 1.3: *Top left:* Adaptive quad-tree with maximum depth $l = 2$. The white leaf nodes induce a cover on level $k \geq l$ consisting of cells with depths $l = 1$ and $l = 2$.
Top right: The disjoint cover $\hat{C}_\Omega^k$ on level k induced by the leaves of the tree on the left.
Lower row: A single node of the tree has been refined, yielding new nodes of depth $l = 3$ and a cover $\hat{C}_\Omega^{k+1}$ on level $k + 1$. Most cells from the cover $\hat{C}_\Omega^k$ are cells of the cover $\hat{C}_\Omega^{k+1}$ as well. Only the refined cell is no longer present, but its children have been added to that cover.

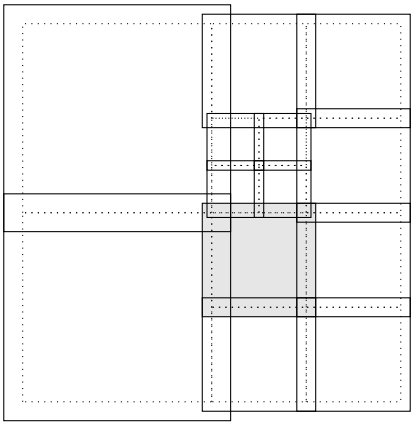

Figure 1.4: Patches $\omega_{i,k}$ of a cover C_Ω^k represented by rectangle boxes that have been created by applying a stretch factor $\alpha = 1.2$ to all cells $\mathcal{C}_{i,k}$ of the final disjoint cover $\hat{C}_\Omega^k$ depicted in Figure 1.3. The local tree cells $\mathcal{C}_{i,k}$ are represented by dotted rectangles. The gray box highlights the domain of a single patch $\omega_{j,k}$.

Proof. For ease of notation, let us drop the index k in the following and assume that all arguments are made for a single level k. For a patch ω_i stretched from $\mathcal{C}_i$ via equation (1.19) we can state that for the distance of any point $\boldsymbol{x} \in \bar{\mathcal{C}}_i$ to the boundary of its corresponding patch ω_i it holds

$$\operatorname{dist}(\boldsymbol{x}, \partial\omega_i) \geq \min_{m=1,\dots,d} \left((\alpha_i - 1) \frac{(\boldsymbol{h}_i)_m}{2} \right), \quad \boldsymbol{x} \in \bar{\mathcal{C}}_i.$$

Since $\hat{C}_\Omega$ is a disjoint partition of R_Ω and $\Omega \subset \bigcup_{i=1}^N \bar{\mathcal{C}}_i$, for each $\boldsymbol{x} \in \Omega$ there is always at least one $\mathcal{C}_j$ such that $\boldsymbol{x} \in \bar{\mathcal{C}}_j$. Then with

$$\begin{aligned} \frac{\operatorname{dist}(\boldsymbol{x}, \partial\omega_j)}{\operatorname{diam}(\omega_j)} &\geq \frac{\min_{m=1,\dots,d} \left((\alpha_j - 1) \frac{(\boldsymbol{h}_j)_m}{2} \right)}{\operatorname{diam}(\omega_j)} \\ &\geq \min_{i=1,\dots,N} \frac{\min_{m=1,\dots,d} \left((\alpha_i - 1) \frac{(\boldsymbol{h}_i)_m}{2} \right)}{\operatorname{diam}(\omega_i)} =: C_S \end{aligned}$$

it holds that $C_S > 0$ if $\alpha_i > 1$ for all $i = 1, \dots, N$ and $\operatorname{diam}(\Omega) > 0$ and thus we obtain the *sufficient overlap* property from equation (1.12) of Definition 1.4. Next we should note that the size of the tree cells $\mathcal{C}_i$ constructed by the algorithm described above, depends solely on the depth l_i of its corresponding node in the tree and can be written as

$$\operatorname{diam}(\mathcal{C}_i) = \frac{\operatorname{diam}(R_\Omega)}{2^{l_i}}. \tag{1.21}$$

Let us additionally define the *direct neighborhood* of a cell $\mathcal{C}_i$ as

$$\hat{C}_i := \{\mathcal{C}_j \in \hat{C}_\Omega \mid \bar{\mathcal{C}}_j \cap \bar{\mathcal{C}}_i \neq \emptyset\} \tag{1.22}$$

as well as the global and local *maximum depth difference*

$$L := \max_{i=1,\dots,N} L_i, \qquad L_i := \max_{\mathcal{C}_j \in \hat{C}_i} |l_i - l_j|. \tag{1.23}$$

Then for a cell $\mathcal{C}_i$ and its neighbor $\mathcal{C}_j \in \hat{C}_i$ with $\operatorname{diam}(\mathcal{C}_i) \geq \operatorname{diam}(\mathcal{C}_j)$, i.e. $l_i \leq l_j$, for $\alpha_i \in (1, 2)$ it is

$$\begin{aligned} \frac{\operatorname{diam}(\omega_i)}{\operatorname{diam}(\omega_j)} &= \frac{\alpha_i \operatorname{diam}(\mathcal{C}_i)}{\alpha_j \operatorname{diam}(\mathcal{C}_j)} = \frac{\alpha_i \frac{\operatorname{diam}(R_\Omega)}{2^{l_i}}}{\alpha_j \frac{\operatorname{diam}(R_\Omega)}{2^{l_j}}} = \frac{\alpha_i}{\alpha_j} 2^{l_j - l_i} \\ &\leq \max_{k=1,\dots,N} \frac{\alpha_k}{\alpha_j} 2^{l_j - l_k} \leq \max_{k=1,\dots,N} \frac{\alpha_k}{\alpha_j} 2^{L_k} < \max_{k=1,\dots,N} 2^{L_k + 1} \\ &\leq 2^{L+1} \end{aligned}$$

since $\frac{\alpha_k}{\alpha_j} < 2$ and we obtain the *comparability of neighboring patches* from (1.14) with $C_\mathcal{N} := 2^{L+1} \geq 1$. Let us now define a sufficient condition that no patch ω_i generated from a cell $\mathcal{C}_i$ penetrates any of its direct neighbors $\mathcal{C}_j \in \hat{C}_i \setminus \{\mathcal{C}_i\}$ to its center or further by

$$(\alpha_i - 1) \frac{(\boldsymbol{h}_i)_m}{2} < \frac{(\boldsymbol{h}_j)_m}{2} \qquad \forall m = 1, \dots, d \quad \forall \mathcal{C}_j \in \hat{C}_i \setminus \{\mathcal{C}_i\}. \tag{1.24}$$

Since equation (1.21) can be formulated equivalently to define $(\boldsymbol{h}_i)_m$ of a cell C_i from R_Ω and l_i, we get

$$\alpha_i < 1 + 2^{l_i - l_j} \qquad \forall \mathcal{C}_j \in \hat{C}_i \setminus \{\mathcal{C}_i\}.$$

Since $1 + 2^{l_i - l_j} \geq 2$ if $l_i \geq l_j$ we do only need to take the case into account where the neighbors $\mathcal{C}_j$ are smaller than $\mathcal{C}_i$. So let us define the maximum depth difference bounded towards smaller neighbors as

$$L_i^- := \max_{\mathcal{C}_j \in \hat{C}_i} |\min\{l_i - l_j, 0\}|. \tag{1.25}$$

which leads to

$$\alpha_i < 1 + 2^{-L_i^-}. \tag{1.26}$$

If we now turn the case around and look at a patch ω_i that is only surrounded by patches ω_j that have been stretched by stretch factors limited as given in equation (1.26) we can reformulate equation (1.24) to

$$(\boldsymbol{h}_i)_m - \max_{\mathcal{C}_j \in \hat{C}_i \setminus \{\mathcal{C}_i\}} ((\alpha_j - 1)(\boldsymbol{h}_j)_m) =: (\boldsymbol{h}_i^{\mathrm{FT}})_m > 0$$

where $\boldsymbol{h}_i^{\mathrm{FT}}$ is the diagonal of a domain

$$\tilde{\omega}_i^{\mathrm{FT}} := \left\{ \boldsymbol{x} \in \omega_i \mid x_m \in \left((\boldsymbol{c}_i)_m - \frac{(\boldsymbol{h}_i^{\mathrm{FT}})_m}{2}, (\boldsymbol{c}_i)_m + \frac{(\boldsymbol{h}_i^{\mathrm{FT}})_m}{2} \right) \right\}$$

located around the center $\boldsymbol{c}_i$ of the patch ω_i that is not overlapped by any other patch $\omega_j \neq \omega_i$ and thus $\lambda_{C_\Omega}(\boldsymbol{x}) = 1$ and $\tilde{\omega}_i^{\mathrm{FT}} \subseteq \omega_i^{\mathrm{FT}}$. With $\mu(\tilde{\omega}_i^{\mathrm{FT}}) := \prod_{m=1}^d (\boldsymbol{h}_i^{\mathrm{FT}})_m > 0$ and due to $\tilde{\omega}_i^{\mathrm{FT}} \subseteq \omega_i^{\mathrm{FT}}$ it holds that $\mu(\tilde{\omega}_i^{\mathrm{FT}}) \leq \mu(\omega_i^{\mathrm{FT}})$ and we can reformulate the *minimal overlap* property given in equation (1.10) from Definition 1.4 as

$$\mu(\omega_i) \leq C_{\mathrm{FT}} \mu(\tilde{\omega}_i^{\mathrm{FT}}) \leq C_{\mathrm{FT}} \mu(\omega_i^{\mathrm{FT}})$$

which holds for

$$C_{\mathrm{FT}} := \max_{i=1,\ldots,N} \frac{\mu(\omega_i)}{\mu(\tilde{\omega}_i^{\mathrm{FT}})}.$$

We are now left to prove the property of *bounded overlap* given in equation (1.11) from Definition 1.4. Given a point $\boldsymbol{x} \in \Omega$ and a patch ω_i such that $\boldsymbol{x} \in \omega_i$. The center $\boldsymbol{c}_i$ of the patch ω_i divides that patch in 2^d disjoint sections and without loss of generality, let the point be located in the upper most section, i.e.

$$(\boldsymbol{c}_i)_m + \alpha_i \frac{(\boldsymbol{h}_i)_m}{2} > \boldsymbol{x}_m \geq (\boldsymbol{c}_i)_m \qquad \forall m = 1, \ldots, d. \tag{1.27}$$

Then, assume that there is another patch ω_j such that the point is located in the same section, i.e.

$$(\boldsymbol{c}_j)_m + \alpha_i \frac{(\boldsymbol{h}_j)_m}{2} > \boldsymbol{x}_m \geq (\boldsymbol{c}_j)_m \qquad \forall m = 1, \ldots, d. \tag{1.28}$$

Combining (1.27) and (1.28) then yields

$$\begin{aligned}(\boldsymbol{c}_i)_m + \alpha_i \frac{(\boldsymbol{h}_i)_m}{2} > (\boldsymbol{c}_j)_m \qquad \forall m = 1, \dots, d \qquad \text{and} \\ (\boldsymbol{c}_j)_m + \alpha_j \frac{(\boldsymbol{h}_j)_m}{2} > (\boldsymbol{c}_i)_m \qquad \forall m = 1, \dots, d.\end{aligned} \tag{1.29}$$

Since the tree cells $\mathcal{C}_i$ do not overlap, it does additionally hold

$$|(\boldsymbol{c}_j)_m - (\boldsymbol{c}_i)_m| \geq \frac{(\boldsymbol{h}_i)_m}{2} + \frac{(\boldsymbol{h}_j)_m}{2}. \tag{1.30}$$

Without loss of generality we can assume $(\boldsymbol{c}_j)_m > (\boldsymbol{c}_i)_m$ such that the first equation from (1.29) together with (1.30) yields

$$\begin{aligned}\alpha_i \frac{(\boldsymbol{h}_i)_m}{2} &\geq (\boldsymbol{c}_j)_m - (\boldsymbol{c}_i)_m \geq \frac{(\boldsymbol{h}_i)_m}{2} + \frac{(\boldsymbol{h}_j)_m}{2} \\ \alpha_i \frac{(\boldsymbol{h}_i)_m}{2} - \frac{(\boldsymbol{h}_i)_m}{2} &\geq \frac{(\boldsymbol{h}_j)_m}{2} \\ (\alpha_i - 1) \frac{(\boldsymbol{h}_i)_m}{2} &\geq \frac{(\boldsymbol{h}_j)_m}{2}\end{aligned}$$

which contradicts (1.24). Hence, if (1.26) holds for all patches, the point $\boldsymbol{x}$ cannot be in the same section of more than a single patch and due to there being at most 2^d different sections we can finally state

$$\|\lambda_{C_\Omega}(\boldsymbol{x})\|_{L^\infty(\Omega)} \leq 2^d =: M.$$

□

Remark 1.1. As an alternative to enforce equation (1.24) that implies the flat-top property for non-uniform covers, equation (1.26) can be reformulated to

$$L_i^- < -\log_2(\alpha_i - 1), \tag{1.31}$$

which gives us a bound on the local maximum depth difference based on a given stretch factor α_i. Using that bound we can select a fixed $\alpha \in (1, 2)$ that we apply to all patches $\alpha_i = \alpha$ for all $i = 1, \dots, N$ and modify our subdivision algorithm to only yield covers where $L_i^- < L_{\max} := -\log_2(\alpha - 1)$ is fulfilled by forcing a refinement of a neighboring cell $\mathcal{C}_j$ when it is neighbor to a cell $\mathcal{C}_i$ with $l_i - l_j \geq L_{\max}$. While this method makes sure we get a valid cover as of Definition 1.4 we additionally get a cover with smooth transitions of patch sizes because the global maximum depth difference L is bounded by $L_{\max}$ as well. Note that bounding $L \leq L_{\max} = \text{const}$ does additionally establish a bound on the number of neighbors, for which we have $\text{card}(C_i) \leq 2^{d(L+2)}$, that depends on the constants $L_{\max}$ and d only. This benefits the computational cost of our method. On the other hand, we should note that for some refinements this can have a noteworthy effect on the total number of patches N, since the refinement of a single patch can lead to the refinement of its neighbors and the neighbors of the neighbors etc., which leads to the so-called *ripple-effect*.

The conditional splitting of leaf cells yields an *adaptive h*-version of the PUM. Additionally, in each refinement step we can choose to increase the polynomial degree of child patches $\omega_{j,k+1}$ of $\omega_{i,k}$ which results in adaptive *p*- and *hp*-versions depending on whether $\omega_{i,k}$ is split up, or simply transferred to the next level. In general, on each level we can define a local *refinement indicator function* r_S subject to

$$r_S : \{(\omega_{i,k}, V_{i,k}),\, i = 1, \ldots, N_k\} \to \{\text{null}, \text{h}, \text{p}, \text{hp}\}$$

that decides for all patches $\omega_{i,k}$ whether to refine their size, increase their polynomial degree, both or nothing at all. The decision how to refine a specific patch can be based on many criteria. Common examples would be simple geometric predicates for patches that are close to some specific feature of the domain Ω, e.g. a re-entrant corner or crack tips. Alternatively it can be driven by a more sophisticated local error estimate as described in [56].
Let us summarize the complete cover construction in the following algorithm:

Algorithm 1.1 (Hierarchical Cover Construction)**.**

Given: A domain Ω, a desired stretch factor $\alpha \in (1, 2)$ and an initial polynomial degree p. Optionally a bound for the maximum depth difference $L_{\max}$.

Compute the bounding box $R_\Omega \supset \bar{\Omega}$ of the domain Ω.

Initialize the tree by using R_Ω as root node with depth $l = 0$. Initialize the disjoint cover on level $k = 0$ as the set $\hat{C}_\Omega^0 := \{\mathcal{C}_{1,0}\}$ with $\mathcal{C}_{1,0} := R_\Omega$ being the only cell in that cover. Initialize C_Ω^0 by applying (1.19) to $\mathcal{C}_{1,0}$. Initialize V_0^{PU} with the local function space $V_{1,0}$ for the patch $\omega_{1,0}$ by defining a weight function $\varphi_{1,0}$ according to (1.15) and (1.16) and the local basis functions according to (1.17) and (1.18) using the initial polynomial degree p.

While the cover on level k is not fine enough:

1. Initialize the disjoint cover on the next level $\hat{C}_\Omega^{\text{next}} := \{\}$.
2. Evaluate the refinement indicator function $r_i := r_S(\omega_{i,k}, V_{i,k})$ for all patches $\omega_{i,k}$, $i = 1, \ldots, N_k$.
3. **If** the maximum depth difference is to be bounded by $L_{\max}$:
   (a) For all patches that are marked to be h-refined $\{\omega_{i,k} \mid r_i \in \{\text{h}, \text{hp}\}\}$, select all neighbors that are not marked to be h-refined and which are going to violate the max depth difference $\{\omega_{j,k} \in C_{i,k} \mid r_j \notin \{\text{h}, \text{hp}\}$ and $(l_{i,k} + 1) - l_{j,k} \geq L_{\max}\}$. Mark those patches to be h-refined as well by

$$r_j = \begin{cases} \text{h} & \text{if } r_j = \text{null} \\ \text{hp} & \text{if } r_j = \text{p} \end{cases}.$$

   Apply this step recursively on the newly h-refined patches until no patches violate the max depth difference.

4. **For all** cells on the current level $\mathcal{C}_{i,k} \in \hat{C}_\Omega^k$:
   (a) **If** $r_i \in \{\mathrm{h}, \mathrm{hp}\}$:
      i. Uniformly split the cell $\mathcal{C}_{i,k}$ of depth $l_{i,k}$ into 2^d children $\mathcal{C}_j \in S_{i,k}$ of depth $l_{i,k}+1$ according to (1.20) and attach them to the tree.
      ii. Append all children that have a non-empty intersection with the domain Ω to the disjoint cover on the new level $\hat{C}_\Omega^{\mathrm{next}} \leftarrow \hat{C}_\Omega^{\mathrm{next}} \cup \{\mathcal{C}_j \in S_{i,k} \mid \mathcal{C}_j \cap \Omega \neq \emptyset\}$.
   (b) **Else**:
      i. Append the unrefined cell to the disjoint cover on the new level $\hat{C}_\Omega^{\mathrm{next}} \leftarrow \hat{C}_\Omega^{\mathrm{next}} \cup \{\mathcal{C}_{i,k}\}$.
5. **If** a leaf cell has been refined, i.e. $\exists\, i \mid r_i \in \{\mathrm{h}, \mathrm{hp}\}$ and $l_{i,k} = k$:
   (a) Assign the generated disjoint cover to the next level $\hat{C}_\Omega^{\mathrm{k+1}} \leftarrow \hat{C}_\Omega^{\mathrm{next}}$.
   (b) Increment the level index $k \leftarrow k+1$.
6. **Else**:
   (a) Overwrite the last level with the newly created one $\hat{C}_\Omega^{\mathrm{k}} \leftarrow \hat{C}_\Omega^{\mathrm{next}}$.
7. Create the overlapping cover C_Ω^k by applying (1.19) to all cells $\mathcal{C}_{i,k} \in \hat{C}_\Omega^k$, using the same stretch factor $\alpha_{i,k} := \alpha_{j,k-1}$ that has been used to create its parent patch $\omega_{j,k-1}$.
8. Compute all neighborhoods $C_{i,k}$ by a tree traversal algorithm.
9. **If** the maximum depth difference $L_{\max}$ is not given or insufficient to guarantee the flat-top property according to (1.31):
   (a) Compute a new $\tilde{\alpha}_{i,k}$ for each patch $\omega_{i,k}$ according to (1.26) by using its neighborhood $C_{i,k}$ to evaluate the local depth difference $L_{i,k}^-$. Update all patches ω_i by applying (1.19) if $\tilde{\alpha}_{i,k} < \alpha_{i,k}$ and set $\alpha_{i,k} \leftarrow \tilde{\alpha}_{i,k}$. Update the affected neighborhoods.
10. Create V_k^{PU} by defining local function spaces $V_{i,k}$ for all patches. Increase the polynomial degree $p_{i,k}$ of all patches $\omega_{i,k}$ that were children of a patch $\omega_{j,k-1}$ with $r_j \in \{\mathrm{p}, \mathrm{hp}\}$. This step is optional for all levels k except the last one if r_S does not need the local function spaces $V_{i,k}$[2] and we do not want to use a multilevel solver which will be described in Section 1.6.

Let us now state some additional properties of the patches constructed via Algorithm 1.1 that will be important throughout the rest of this thesis:

Lemma 1.3. *For all patches $\omega_{j,k+1} \in C_\Omega^{k+1}$ that have been constructed during Algorithm 1.1 from children $\mathcal{C}_j \in S_{i,k}$ of a cell $\mathcal{C}_{i,k} \in \hat{C}_\Omega^k$ with the corresponding patch $\omega_{i,k} \in C_\Omega^k$ it holds that*

$$\omega_{j,k+1} \subseteq \omega_{i,k}.$$

[2] r_S does usually need the local function spaces $V_{i,k}$ when it is based on a local error estimator and does not need them when it is based on purely geometric predicates.

Remark 1.2. This property is very important for the computational efficiency of many aspects in our method because it allows us to use tree-traversal algorithms for geometric queries on a patch $\omega_{i,k}$. Those traversals have to visit $k_{\max}$ nodes for a full tree-descend, where $k_{\max}$ is the total depth of our tree which is usually in $O(\log(N_{k_{\max}}))$. An example is finding the neighborhoods $C_{i,k}$ of all patches in step 8 of Algorithm 1.1.

Lemma 1.4. *For each patch $\omega_{i,k} \in C_\Omega^k$ on a level $k > 0$ that has been constructed during Algorithm 1.1 there is exactly one patch $\omega_{\tilde{i},k-1} \in C_\Omega^{k-1}$ such that*

$$\omega_{\tilde{i},k-1} \supseteq \omega_{i,k}.$$

1.3 Galerkin Discretization & Numerical Integration

Up until now we have only looked into how to create PUM spaces but not how to use those spaces to solve any problem equations. To this end, let us consider an elliptic boundary value problem

$$\begin{aligned} Lu &= f \quad \text{in} \quad \Omega \subset \mathbb{R}^d \\ Bu &= g \quad \text{on} \quad \partial\Omega \end{aligned}$$

where L is a symmetric partial differential operator of second order and B expresses suitable boundary conditions. For reasons of simplicity we restrict ourselves to a simple Poisson problem in this section. The Poisson problem can be defined as

$$\begin{aligned} -\Delta u &= f \quad \text{in } \Omega \\ u &= g_{\mathrm{D}} \quad \text{on } \Gamma_{\mathrm{D}} \subset \partial\Omega \\ \nabla u \cdot \boldsymbol{n} &= g_{\mathrm{N}} \quad \text{on } \Gamma_{\mathrm{N}} = \partial\Omega \setminus \Gamma_{\mathrm{D}} \end{aligned}$$

with f being any appropriate right-hand side source term, g_{D} being Dirichlet boundary conditions on the boundary part Γ_{D} and g_{N} being Neumann boundary conditions in normal direction $\boldsymbol{n}$ on the remaining boundary part Γ_{N}.
We employ Galerkin's method to discretize the partial differential operator, i.e. we multiply the equation by appropriate test functions v and integrate both sides of the equation. After partial integration we get

$$\int_\Omega \nabla u \nabla v \, dx - \int_{\partial\Omega} (\partial_{\boldsymbol{n}} u) v \, ds = \int_\Omega f v \, dx \tag{1.32}$$

with $\partial_{\boldsymbol{n}} u := \frac{\partial u}{\partial \boldsymbol{n}}$ being the normal derivative of u on the boundary $\partial\Omega$. Using the boundary data from (1.3) we can split the surface term over $\partial\Omega$ into two disjoint parts over Γ_{D} and Γ_{N}. The Neumann boundary conditions can be enforced naturally, thus we simply replace the unknown $\partial_{\boldsymbol{n}} u$ by the given g_{N} on Γ_{N} and move that term to the right-hand side. The Dirichlet condition is enforced by a conforming splitting of the space V^{PU} that is described in Section 1.4 and thus we simply ignore that term for now. Hence we end up with the following equation

$$\int_\Omega \nabla u \nabla v \, dx = \int_\Omega f v \, dx + \int_{\Gamma_{\mathrm{N}}} g_{\mathrm{N}} v \, ds. \tag{1.33}$$

If we choose both, trial functions u and test functions v from our PUM Space V^{PU} we can reformulate (1.33) to the following left-hand side bilinear form

$$a(\varphi_i\vartheta_i^n, \varphi_j\vartheta_j^m) = \int_\Omega \nabla(\varphi_i\vartheta_i^n)\nabla(\varphi_j\vartheta_j^m)\ dx$$

and right-hand side linear form

$$l(\varphi_i\vartheta_i^n) = \int_\Omega f(\varphi_i\vartheta_i^n)\ dx + \int_{\Gamma_\mathrm{N}} g_\mathrm{N}(\varphi_j\vartheta_j^m)\ ds.$$

These allow us to assemble the stiffness matrix A and right-hand side vector $\hat{f}$ via

$$\begin{array}{lll} A = ((A_{i,j})_{n,m}), & (A_{i,j})_{n,m} & = \ a(\varphi_j\vartheta_j^m, \varphi_i\vartheta_i^n) \\ \hat{f} = ((\hat{f}_i)_n), & (\hat{f}_i)_n & = \ l(\varphi_i\vartheta_i^n) \end{array}$$

to set up the linear equation system $A\tilde{u} = \hat{f}$, with $\tilde{u}$ being the coefficient vector of the unknown function. Note that we defined A to be a sparse block-matrix consisting of the dense blocks $(A_{i,j})$. The block-sparsity pattern of A is determined by the neighborhoods C_i from (1.13) of a patch ω_i. For a given row i we have $(A_{i,j}) \equiv 0$ when $\omega_j \notin C_i$. Thus each row of A consists of $\mathrm{card}(C_i)$ non-zero blocks. On the other hand, each block $(A_{i,j})$ is determined by the basis functions ϑ_i^n and ϑ_j^m of the patches ω_i and ω_j and thus has a size of $\dim(V_i) \times \dim(V_j)$. While this block structure is not the only way to order the degrees of freedom employed by the PUM space V^{PU} it turned out to be beneficial to the ease of implementation and performance during the numerical integration and linear solving steps of the overall simulation algorithms. The employed equation system can be solved by any direct or iterative algorithm for linear equation systems. One particularly fast method for this task will be shown in Section 1.6.
We are now left with the (numerical) integration of $a(\cdot,\cdot)$ and $l(\cdot)$ to determine the entries of A and $\hat{f}$. To this end, we need to consider the regularity of our PUM shape functions $\varphi_i\vartheta_i^n \in V_i$. First, we consider the case without enrichments, i.e. V_i consists of polynomials only. Since we choose smooth polynomial functions ψ_i^s up to a degree p_i over the full local patch domain ω_i, the only source of kinks or discontinuities in $\varphi_i\psi_i^s$ can come from the PU functions φ_i. To this end, recall the construction via Shepards method from equation (1.15) given as

$$\varphi_i := \frac{W_i(\boldsymbol{x})}{\sum_{\omega_j\in C_i} W_j(\boldsymbol{x})} = \frac{W_i(\boldsymbol{x})}{\sum_{j=1}^N W_j(\boldsymbol{x})}$$

which we rewrite to

$$\varphi_i := W_i S^{-1}, \quad \text{with } S := \sum_{i=1}^N W_j.$$

There are two main sources of kinks or discontinuities in φ_i or its derivatives $\nabla\varphi_i$ in this construction: The first are kinks and/or discontinuities within the weight functions W_i or their derivatives ∇W_i that are usually inherited directly from the construction by tensor products of the univariate spline functions $\mathcal{W}$. The second source is the sum within S. Since the weight functions W_i have a support limited to their respective patches ω_i and

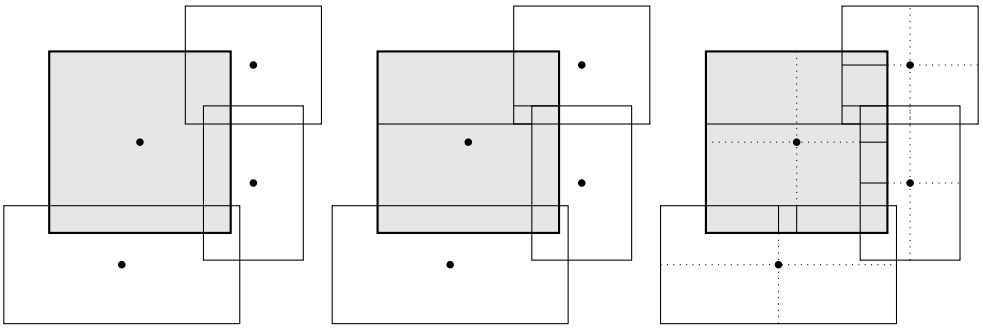

Figure 1.5: Decomposition of a patch domain ω_i into integration domains $\{\mathcal{D}_i^n\}$.
Left: A patch ω_i (gray) with its three neighboring patches. The centers of the patches are depicted by small circles.
Center: The patch ω_i has been decomposed into the intermediate integration domains $\{\tilde{\mathcal{D}}_i^m\}$. To this end, the overlaps have been resolved and the L-shaped sub-domains have been split to form a total of 8 axis-aligned rectangle domains.
Right: The intermediate integration domains have been further split into a total of 13 final integration domains $\{\mathcal{D}_i^n\}$ so that the kinks of the linear spline weights W_i and W_j on all patches are resolved correctly. The dotted lines indicate kinks in the weights W_i and W_j. Intermediate cells $\tilde{\mathcal{D}}_i^m$ that overlap the flat-top domain of ω_i have not been split along those dotted lines.

those patches overlap, S does additionally have kinks at the patch surfaces $\partial\omega_j$ (compare Figure 1.2). Therefore all integrals involving φ and especially their derivatives $\nabla\varphi$ should not be integrated by a simple quadrature rule over all of ω_i. Instead we should decompose ω_i into smaller integration domains $\{\mathcal{D}_i^n\}$ that resolve the piecewise character imposed by W_i and S.

To this end, we first consider the decomposition required to resolve the sum in S. Here, we decompose a patch into intermediate integration domains $\{\tilde{\mathcal{D}}_i^m\}$ so that the covering index λ_{C_Ω} from equation (1.8) is constant on each of these domains. This results in a decomposition where the number of iterations in the sum that evaluate to a non-zero value is constant per integration domain or in other words all patch boundaries $\partial\omega_i$ are resolved. To make sure that we can use simple quadrature rules on the resulting integration domain, all domains that are not yet axis-aligned rectangles (L-shaped polygons can occur) are further split into rectangles (compare the first step in Figure 1.5). Considering that we constructed our patches ω_i as products of one-dimensional intervals in equation (1.19) the decomposition can be carried out by a tree decomposition algorithm. The tree is initialized with the patch's domain ω_i as root node and then the neighbors $\omega_j \in C_i$ are inserted into the tree by splitting all leaves of the tree that are intersected by each neighbor. Note that this decomposition is key to the performance of the integration step of the PUM. We can easily store a list of all patches that overlap each of the cells $\tilde{\mathcal{D}}_i^m$. This list can later be used to determine which weights W_i and basis functions ϑ_i are to be evaluated at each integration point without the need to search for the patches that are non-zero at any given

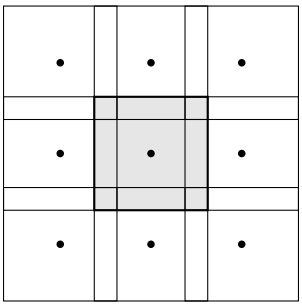

Figure 1.6: Uniform neighborhood around a patch ω_i (gray) where all patches have the same diameter and the same distance in each direction.

position x.
We are now left with resolving the kinks in W_i or its derivatives induced by the piecewise character of the spline weights $\mathcal{W}$ (compare Figure 1.2). Since the weights W_i are constructed by tensor products, the kinks at points of the univariate functions $\mathcal{W}$ can be resolved in higher dimensions by splitting ω_i into smaller axis-aligned rectangles. E.g. for a linear spline where $\mathcal{W}$ has a kink at the center of its domain, we need to split a patch in d dimensions into 2^d equally sized sub-domains. Also note that in the flat-top region of a patch ω_i it is $\varphi_i \equiv 1$, thus the kinks of $\mathcal{W}$ do not translate into kinks of φ_i in the flat-top region of a patch and we can spare to split the integration domains $\tilde{\mathcal{D}}_i^m$ where $\lambda_{C_\Omega} = 1$ (compare the second step in Figure 1.5).
It is worth mentioning that the number of required splits to incorporate $\mathcal{W}$ can be mitigated under some circumstances. Let us assume that the cover has been constructed as described in Section 1.2 and a uniform refinement strategy has been used, i.e. all patches have the same diameter and their centers have the same distance in each direction (compare Figure 1.6). Under these circumstances we can define the reference weight functions $\mathcal{W}$ dependent on the stretch factor α in such a fashion that the kinks in all W_i are co-located with the surfaces $\partial\omega_j$ of the neighboring patches ω_j (compare Figure 1.7) and thus we have $\{\tilde{\mathcal{D}}_i^m\} \equiv \{\mathcal{D}_i^m\}$.[3]
Now that we have integration domains that can be used to integrate our shape functions $\varphi_i\psi_i^s$ on any domain $\omega_i \cap \omega_j$ we still need to create integration cells that can be used to integrate the shape functions on $\omega_i \cap \omega_j \cap \Omega$. For sub domains $\omega_i \cap \omega_j \cap \partial\Omega \neq \emptyset$ this is actually a very difficult task that we will cover in detail throughout Chapter 4 for two-dimensional domains Ω and throughout Chapter 5 for three-dimensional domains. But in the case of an empty intersection with the boundary (i.e. $\omega_i \cap \omega_j \cap \partial\Omega = \emptyset$) our integration domains $\{\mathcal{D}_i^m\}$ are sufficient and we are only left with the task to select appropriate quadrature rules on each domain $\mathcal{D}_i^m$. Since both, the PU functions φ_i as

[3] In fact we can construct functions $\mathcal{W}$ that are even appropriate for non-uniform neighborhoods of a patch ω_i by moving the kinks closer to the center of the patch if it has neighbors $\omega_j \in C_i$ with $\operatorname{diam}(\omega_j) > \operatorname{diam}(\omega_i)$ or further away in the opposite case. While this has the potential to put more kinks of W_i on top of some $\partial\omega_j$ it is not possible to guarantee this property for arbitrary neighborhoods in $d > 1$. Note that this involves making use of the possibility to use different functions $\mathcal{W}_m$ in each direction $m = 1, \ldots, d$ in the tensor-product construction of the weights W_i.

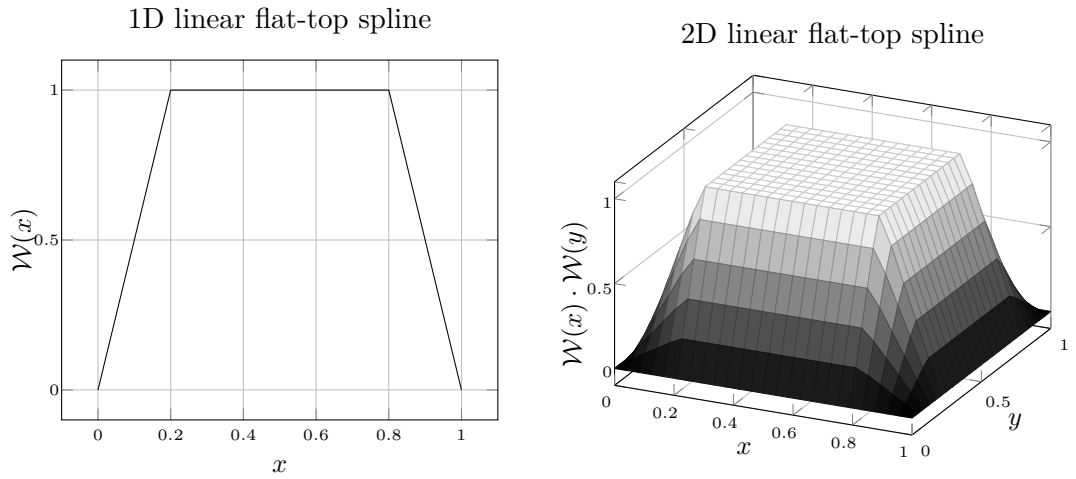

Figure 1.7: Construction of flat-top weight functions for a uniform cover.
Left: The univariate linear spline function $\mathcal{W}$ on $[0, 1]$ has been constructed in such a fashion that the kinks are located at $1 - \frac{1}{\alpha}$ and $\frac{1}{\alpha}$, where α is the stretch factor used in the construction of our patches from equation (1.19) (here $\alpha = 1.25$ has been used).
Right: Two-dimensional tensor product spline $\mathcal{W}(x)\mathcal{W}(y)$ on the reference domain $[0, 1]^2$. Transformation of this function onto a patch with a uniform neighborhood as depicted in Figure 1.6 will put the kinks onto the surface of the neighbor patches ω_j.

well as the polynomial basis functions ψ_i^s, have been constructed by tensor products, a natural choice is to use tensor-products of one-dimensional quadrature rules. Throughout this thesis we used tensor products of one-dimensional Gauss-Legendre quadrature rules on rectangular integration domains. Note that this includes the use of anisotropic tensor products where one-dimensional rules of different orders are used in each direction. If we consider the case of the uniform cover from Figure 1.6 with weights as depicted in 1.7, this means we only need to increase the order in one direction on four of the eight domains that have more than a single patch assigned to them. Additionally note that, even though we constructed them via Shepards method, the PU functions φ_i are actually polynomials most of the time for uniform covers. Therefore, we can integrate them exactly by an appropriate tensor product rule. For cells $\mathcal{D}_i^m$ where the PU functions φ_i are actually rational, higher integration orders are required to achieve a sufficient accuracy.

1.4 Dirichlet Boundary Treatment

The handling of essential boundary conditions is a non-trivial task in most meshfree methods as well as in the PUM. This is due to the fact that our shape functions $\varphi_i \vartheta_i^n$ are neither interpolatory on the boundary $\partial\Omega$ nor do they vanish on it. A common way to overcome these issues and impose Dirichlet boundary conditions in meshfree methods is to use a variational approach due to Nitsche [89]. This approach has been shown to work in the

setting of our flat-top PUM in [55]. Even though Nitsche's approach is of optimal complexity and yields an optimally convergent method it comes with some downsides. The weak formulations that include the required symmetrization and regularization terms need to be derived analytically and thus they are highly dependent on the problems and boundary conditions at hand. Additionally the computation of appropriate regularization parameters is not always an easy task. Choosing parameters that are too low might lead to the boundary conditions being imposed too weak and might even result in singular systems. While in theory very high regularization parameters overcome those issues they have bad effects on the condition number of the stiffness matrix and thus impose a hard problem on the numerical linear solvers applied to the equation system. To counter those problems a conforming treatment of Dirichlet boundary conditions has been derived from the limit case where the regularization parameter goes to infinity in [107].
The main idea of the conforming method is to employ a direct splitting

$$V_i = V_{i,K} \oplus V_{i,I}$$

of the local approximation spaces V_i, where $V_{i,K}$ is used to approximate the solution of the PDE and $V_{i,I}$ is used to approximate the boundary conditions on $\omega_i \cap \Gamma_\mathrm{D}$. This local splitting transfers into our global PUM space and induces a global splitting

$$V^\mathrm{PU} = \sum_{i=1}^{N} \varphi_i V_i = \sum_{i=1}^{N} \varphi_i V_{i,K} \oplus \sum_{i=1}^{N} \varphi_i V_{i,I} = V_K^\mathrm{PU} \oplus V_I^\mathrm{PU}.$$

The first step in the algorithm is to construct an equation

$$b_i(u_i, v_i) = g_i(v_i), \quad \forall v_i \in V_i \tag{1.34}$$

that imposes the boundary conditions on a local function u_i, utilizing a local bilinear form $b_i : V_i \times V_i \to \mathbb{R}$ and linear form $g_i : V_i \to \mathbb{R}$. We then set $V_{i,K}$ to the null space of $b_i(\cdot,\cdot)$ and $V_{i,I}$ to the orthogonal space to $V_{i,K}$ in V_i. In the case of standard Dirichlet boundary conditions, where we set $u = g_\mathrm{D}$ on $\Gamma_\mathrm{D} \subset \partial\Omega$, we use an L^2-projection for (1.34) that is given by

$$b_i(u_i, v_i) := \langle u_i, v_i \rangle_{L^2(\omega_i \cap \Gamma_\mathrm{D})}, \qquad g_i(v_i) := \langle g_\mathrm{D}, v_i \rangle_{L^2(\omega_i \cap \Gamma_\mathrm{D})}. \tag{1.35}$$

If discretized with $u_i, v_i \in \operatorname{span}\langle \vartheta_i^n \rangle$, the bilinear form $b_i(\cdot,\cdot)$ and linear form $g_i(\cdot)$ yields a local boundary trace operator matrix

$$(B_i)_{n,m} = b_i(\vartheta_i^m, \vartheta_i^n), \qquad B_i \in \mathbb{R}^{d_i \times d_i} \tag{1.36}$$

and right-hand side moment vector

$$(\hat{g}_i)_n = g_i(\vartheta_i^n), \qquad \hat{g}_i \in \mathbb{R}^{d_i} \tag{1.37}$$

with $d_i = \dim(V_i)$. To come up with the splitting of the degrees of freedom of V_i into $V_{i,K}$ and $V_{i,I}$ we employ an eigenvalue decomposition of B_i for all patches $\omega_i \cap \Gamma_\mathrm{D} \neq \emptyset$ as

$$B_i = Q_i^T D_i Q_i \quad \text{with } Q_i, D_i \in \mathbb{R}^{d_i \times d_i}$$

where the rows of Q_i contain all eigenvectors of B_i, D_i is diagonal and $\lambda_s := (D_i)_{s,s}$ with $s = 1, \dots, d_i$ are the eigenvalues of B_i. Let us assume that the eigenvalues λ_s in D_i are given in descending order, thus λ_1 being the largest eigenvalue. Then we can create a block-partitioning

$$Q_i = \begin{pmatrix} Q_{i,I} \\ Q_{i,K} \end{pmatrix}, \quad D_i = \begin{pmatrix} D_{i,I} & 0 \\ 0 & D_{i,K} \end{pmatrix}$$

where $D_{i,K}$ contains all eigenvalues $\{\lambda_s \mid \lambda_s \leq \epsilon\lambda_1\}$ that are small with respect to some $\epsilon \ll 1$ and the rows of $Q_{i,K}$ represent the eigenvectors that span the (numerical) kernel of $b_i(\cdot,\cdot)$ if ϵ is chosen subject to the machine epsilon.[4] Considering that Q_i specifies a normal basis-transformation

$$Q_i : V_i = \operatorname{span}\langle \vartheta_i^n \rangle \to V_i = \operatorname{span}\langle \tilde{\vartheta}_i^s \rangle$$

that transforms $\langle \vartheta_i^n \rangle$ into a new *separated* basis $\langle \tilde{\vartheta}_i^s \rangle$ we can use the partitioning of Q_i to define the operators

$$\Pi_{i,I} := Q_{i,I}^T Q_{i,I} = \begin{pmatrix} \mathbb{I}_{d_{i,I}} & 0 \\ 0 & 0 \end{pmatrix} \quad \text{and} \quad \Pi_{i,K} := Q_{i,K}^T Q_{i,K} = \begin{pmatrix} 0 & 0 \\ 0 & \mathbb{I}_{d_{i,K}} \end{pmatrix}$$

with $d_{i,K} = \operatorname{card}(\{\lambda_s \mid \lambda_s \leq \epsilon\lambda_1\})$ and $d_{i,I} = d_i - d_{i,K}$ that are the projections into the image of B_i and the kernel respectively. Thus we can create the sub-spaces

$$V_{i,K} := Q_{i,K}(V_i), \qquad V_{i,I} := Q_{i,I}(V_i)$$

that each are represented by parts of the new basis $\langle \tilde{\vartheta}_i^s \rangle$.
These local transformations can be used to create a global transformation by a block-diagonal operator with the entries

$$(T)_{i,j} = \begin{cases} \mathbb{I}_{d_i} & \text{if } i = j \text{ and } \omega_i \cap \Gamma_{\mathrm{D}} = \emptyset \\ Q_i & \text{if } i = j \text{ and } \omega_i \cap \Gamma_{\mathrm{D}} \neq \emptyset \\ 0 & \text{if } i \neq j \end{cases} \tag{1.38}$$

that allows us to transform any stiffness matrix assembled in the original basis $\langle \vartheta_i^n \rangle$ to the separated basis $\langle \tilde{\vartheta}_i^s \rangle$ by

$$A_{\tilde{\vartheta}} := T A_\vartheta T^T.$$

Exchanging Q_i in equation (1.38) by either $Q_{i,K}$ or $Q_{i,I}$ allows us to create rectangular transformations T_I and T_K that can be used to transform any matrix A_ϑ directly into the subspaces V_K^{PU} and V_I^{PU}.
Let us now take a look on how to use the introduced splitting to employ boundary conditions on a system to be solved. To this end, let us assume we want to solve an equation

[4] In practice we employ both a relative and an absolute epsilon to determine which eigenvalues belong to the numerical kernel.

system $A\tilde{u} = \hat{f}$ that has been assembled in the global basis $\langle \varphi_u \vartheta_i^n \rangle$ without taking boundary conditions into account. We can then split the coefficient vector $\tilde{u}$ into two parts

$$\tilde{u}_K = T_K \tilde{u} \quad \text{and} \quad \tilde{u}_I = T_I \tilde{u}$$

where $\tilde{u}_K$ now contains all *interior* degrees of freedom, subject to the original PDE and $\tilde{u}_I$ contains all *boundary* degrees of freedom, subject to the boundary condition $u = g_\mathrm{D}$ on Γ_D. We can then solve for $\tilde{u}_I$ by solving the discretized L^2-projections given by equation (1.35) for the boundary degrees of freedom as

$$(Q_{i,I} B_i Q_{i,I}^T) \tilde{u}_{i,I} = Q_{i,I} \hat{g}_i \tag{1.39}$$

for all patches $\omega_i \cap \Gamma_\mathrm{D} \neq \emptyset$. Note that equation (1.39) imposes *local* problems only and since $Q_{i,I} B_i Q_{i,I}^T = D_{i,I}$ they do only involve the inversion of a diagonal matrix and are easily parallelizable. For the remaining degrees of freedom we can then solve

$$A_{K,K} \tilde{u}_K = \hat{f}_K - A_{K,I} \tilde{u}_{i,I}$$

where $A_{K,K} = T_K A T_K^T$, $\hat{f}_K = T_K \hat{f}$ and $A_{K,I} = T_K A T_I^T$. To retrieve a coefficient vector $\tilde{u}$ that is represented in the original global basis $\langle \varphi_u \vartheta_i^n \rangle$ we can simply apply our transformation T to get

$$\tilde{u} = T^T \begin{pmatrix} \tilde{u}_K \\ \tilde{u}_I \end{pmatrix}.$$

1.5 Stability

Let us now take a closer look at the stability of our method. We introduced the flat-top property in Definition 1.3 to make sure that we cannot get linear dependencies between the shape functions $\varphi_i \vartheta_i^n$ on different patches ω_i, thus we get global stability of our PUM space V^PU from local stability of V_i. But we are still left with the question of whether our local spaces V_i are stable, i.e. whether the functions $\langle \vartheta_i^n \rangle$ form a *basis* of V_i and not only a generating set. In fact we can observe multiple cases where $\langle \vartheta_i^n \rangle$ is **not** a valid basis for V_i. Reconsider our construction of $V_i := \mathrm{span}\langle \psi_i^s, \eta_i^t \rangle$ in which we use polynomials ψ_i^s and problem dependent enrichment functions η_i^t. First of all, since an important property of the PUM is that we want to use *arbitrary* enrichment functions η_i^t we cannot (or do not want to) impose any restrictions on the functions used, thus $\langle \eta_i^t \rangle$ by itself may be a generating set only. This is especially true if we consider that for V_i the enrichment functions are imposed locally on ω_i only. Thus if we use enrichment functions that are linearly independent globally, they might still be (nearly) linearly dependent on the restricted domain ω_i. Additionally even if we have enrichment functions that are locally linearly independent and $\langle \eta_i^t \rangle$ does indeed form a basis, the polynomials could be able to approximate the enrichment functions well and thus $\langle \psi_i^s, \eta_i^t \rangle$ would be ill-conditioned. This again can be common if the enrichments get restricted onto small domains ω_i or if the enrichment functions are not given by analytical functions, but instead are the solutions

to some other numerical method. A last case of possible instabilities can arise in the pure polynomial part $\langle \psi_i^s \rangle$. While our choice of tensor products of Legendre polynomials for ψ_i^s makes sure that $\langle \psi_i^s \rangle$ is a orthogonal basis on ω_i, the restriction $\omega_i \cap \Omega$ onto small intersections with the global domain can lead to numerical near linear dependencies that either make our discretized stiffness matrix singular or at least make it very ill-conditioned and hence difficult to solve by numerical solvers.
To this end, a local stabilization method has been proposed in [105] that transforms a potential generating set $\langle \vartheta_i^n \rangle$ into a stable basis $\langle \tilde{\vartheta}_i^n \rangle$ for V_i. Let the local mass matrix be given by

$$(M_i)_{n,m} := \langle \vartheta_i^n, \vartheta_i^m \rangle_{L^2(\omega_i \cap \Omega)}, \quad m, n = 1, \dots, d_i$$

with $d_i = \dim(V_i)$. We can now compute the eigenvalue decomposition

$$M_i = Q_i^T D_i Q_i \quad \text{with } Q_i, D_i \in \mathbb{R}^{d_i \times d_i}$$

where the rows of Q_i contain all eigenvectors of M_i, D_i is diagonal and $\lambda_n := (D_i)_{n,n}$ with $n = 1, \dots, d_i$ are the eigenvalues of M_i. We assume that the eigenvalues are given in descending order, i.e. $\lambda_n \geq \lambda_{n+1}$ so that we can create a block-partitioning

$$Q_i = \begin{pmatrix} \tilde{Q}_i \\ K_i \end{pmatrix}, \quad D_i = \begin{pmatrix} \tilde{D}_i & 0 \\ 0 & \kappa_i \end{pmatrix}$$

where κ_i contains all eigenvalues $\{\lambda_n \mid \lambda_n \leq \epsilon \lambda_1\}$ that are small with respect to some cutoff $\epsilon \ll 1$ and the rows of K_i represent the eigenvectors associated with small eigenvalues. Since $(\tilde{D}_i)_{n,n} > \epsilon \lambda_1$ and hence $(\tilde{D}_i)_{n,n} > 0$ we can define the projection

$$S_i := \tilde{D}_i^{-1/2} \tilde{Q}_i$$

which can be used to remove the near-null space of M_i as

$$S_i M_i S_i^T = \tilde{D}_i^{-1/2} \tilde{Q}_i M_i \tilde{Q}_i^T \tilde{D}_i^{-1/2} = \mathbb{I}_{\tilde{d}_i}$$

with $\tilde{d}_i := \text{card}(\{\lambda_n \mid \lambda_n > \epsilon \lambda_1\})$. Hence we created a local basis transformation

$$S_i : V_i = \text{span}\langle \vartheta_i^n \rangle \to \text{span}\langle \tilde{\vartheta}_i^n \rangle \approx V_i$$

that transforms a potentially ill-conditioned generating set $\langle \vartheta_i^n \rangle$ to a stable basis $\langle \tilde{\vartheta}_i^n \rangle$ that is optimally conditioned. Similar to how we constructed global basis transformation operators T, T_K and T_I from the local transformation matrices Q_i in (1.38) for the conforming boundary treatment we can create a global block-transformation matrix

$$(S)_{i,j} = \begin{cases} \mathbb{I}_{d_i} & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$$

for the stable transformation. Again, this allows us to assemble the stiffness matrix A_ϑ and right-hand side vector $\hat{f}_\vartheta$ employing the original basis $\langle \vartheta_i^n \rangle$ and then transform that equation system with

$$A_{\tilde{\vartheta}} := S A_\vartheta S^T, \qquad \hat{f}_{\tilde{\vartheta}} := S \hat{f}_\vartheta$$

into the stable basis given by

$$A_{\tilde{\vartheta}}\tilde{u}_{\tilde{\vartheta}} = \hat{f}_{\tilde{\vartheta}}.$$

A coefficient vector in the original basis can then be retrieved with $\tilde{u}_\vartheta = S^T\tilde{u}_{\tilde{\vartheta}}$ if it is required for any further computations or post-processing purposes.

Remark 1.3. The method presented above stabilizes the complete set of functions $\langle\vartheta_i^n\rangle = \langle\psi_i^s, \eta_i^t\rangle$ in one step. This means that after the stabilization there is no separation of the stable basis functions $\langle\tilde{\vartheta}_i^n\rangle$ into the smooth polynomial part and the enrichment part anymore. Hence, we have no information on whether the instabilities arise due to linear dependencies among the enrichments, among the polynomials or due to the enrichments being well approximable by the polynomials. To this end, an improved stabilization approach has been presented in [108] where in a first step stable bases for $\mathcal{P}^{p_i} = \operatorname{span}\langle\psi_i^s\rangle$ and $\mathcal{E}_i = \operatorname{span}\langle\eta_i^t\rangle$ are created separately and in a second step the enrichment space is additionally stabilized against the polynomials, which means we do only remove the part of the stabilized enrichment space that is well approximated by the polynomials. Hence, we end up with a direct splitting

$$V_i \approx \tilde{\mathcal{P}}^{p_i} \oplus \hat{\mathcal{E}}_i$$

where $\tilde{\mathcal{P}}^{p_i} := \operatorname{span}\langle\tilde{\psi}_i^s\rangle \approx \mathcal{P}^{p_i}$ is the stabilized polynomial space and $\hat{\mathcal{E}}_i \approx \tilde{\mathcal{E}}_i \setminus \tilde{\mathcal{P}}^{p_i}$ is the part of the stabilized enrichment space $\tilde{\mathcal{E}}_i := \operatorname{span}\langle\tilde{\eta}_i^t\rangle \approx \mathcal{E}_i$ without the part that can be approximated by polynomials. Note that for patches where $\langle\psi_i^s\rangle$ is already a stable basis it is $\tilde{\mathcal{P}}^{p_i} = \mathcal{P}^{p_i}$. Observe that this holds at least for all patches in the interior of Ω.

1.6 Multilevel Solver

For a PUM space on the cover C_Ω where we use polynomials of degree p_i on each patch $\omega_i \in C_\Omega$ we have $\text{dof} = \dim(V^{\text{PU}}) \simeq Np^d$ with $N = \operatorname{card}(C_\Omega)$ and $p = \max_i p_i$. Hence, the system of linear equations $A\tilde{u} = \hat{f}$ consists of the sparse block-matrix A of dimension $\text{dof} \times \text{dof}$, the coefficient vector $\tilde{u}$ of length dof and the right-hand side moment vector $\hat{f}$ of the same length. Solving such an equation system with a direct solver like Gaussian elimination, LU- or Cholesky-decompositions usually needs storage in $O(\text{dof}^2)$ and a number of operations in $O(\text{dof}^3)$. Even more advanced direct solvers for sparse matrices can only improve those complexities to a limited extend and hence a usual approach is to employ *iterative* solvers on larger problems with many degrees of freedom. Iterative solvers like the Jacobi or Gauss-Seidel method are able to keep a storage complexity of $O(\text{dof})$ but since usually for an increasing number of degrees of freedom not only the required operations per iteration increases, but the number of iterations till convergence increases as well, they cannot yield optimal complexity in general. One class of solvers that can yield an optimal complexity in both storage as well as operations count are so-called *multilevel* or *multigrid* methods.

The driving motivation in the construction of any multilevel solver is the observation that classical iterative solvers like Jacobi or Gauss-Seidel methods are fast at reducing the high oscillatory error components but do not converge well on the remaining smooth errors. The idea is to move those smooth error components onto a coarser level where they can

be approximated well and the formerly smooth errors on the fine level do now resemble higher oscillations relative to the coarse level. Thus, the classical iterative solvers are again suitable to reduce the errors on the coarser levels. After the errors have been reduced on the coarse level the coarse information is transferred back to the fine level and applied as correction to the fine solution. Recursive application of additional coarsening steps to this two level approach yields a *multilevel solver*. Let us now formalize the multilevel algorithm. Given a sequence of function spaces V_k on levels $k = k_{\min}, \dots, k_{\max}$ with $k_{\max} \geq k_{\min}$ we denote the discretization of the problem dependent bilinear-form $a(\cdot, \cdot)$ on each level by A_k. Given additional linear *inter-level transfer operators*

$$I_{k-1}^k : V_{k-1} \to V_k, \qquad I_k^{k-1} : V_k \to V_{k-1}$$

and linear smoothing operators

$$S_k^{\text{pre}} : V_k \times V_k \to V_k, \qquad S_k^{\text{post}} : V_k \times V_k \to V_k$$

on all but the coarsest level $k = k_{\min} + 1, \dots, k_{\max}$, we can define the recursive multilevel algorithm as follows:

Algorithm 1.2 (Multilevel method $M(k, x_k, b_k)$)**.**

if $k > k_{\min}$
  for $i = 1, \dots, \nu_k^{\text{pre}}$ ▷ Pre-smoothing
    $x_k \leftarrow S_k^{\text{pre}}(x_k, b_k)$
  $r_{k-1} \leftarrow I_k^{k-1}(b_k - A_k x_k)$ ▷ Restrict residual
  $c_{k-1} \leftarrow 0$
  for $i = 1, \dots, \gamma$ ▷ Compute coarse-level correction
    $c_{k-1} \leftarrow M(k-1, c_{k-1}, r_{k-1})$
  $x_k \leftarrow x_k + I_{k-1}^k c_{k-1}$ ▷ Prolongate & apply correction
  for $i = 1, \dots, \nu_k^{\text{post}}$ ▷ Post-smoothing
    $x_k \leftarrow S_k^{\text{post}}(x_k, b_k)$
else
  $x_k \leftarrow A_k^{-1} b_k$ ▷ Solve on coarsest level

Here the parameters ν_k^{pre} and ν_k^{post} allow us to control how many pre- or post-smoothing steps should be performed on each level. The parameter γ controls the *shape* of the multilevel iteration, where $\gamma = 1$ results in what is commonly known as the *V-cycle* and $\gamma = 2$ yields the *W-cycle*. Note that we still need to solve the equation system $x_k \leftarrow A_k^{-1} b_k$ on the coarsest level $k_{\min}$. Usually the goal is to create a coarsest level function space $V_{k_{\min}}$ that has only a very small amount of degrees of freedom und thus the high memory and runtime complexities of a direct solver like the LU-decomposition do not impose any significant problems.

Let us now take a look how we can apply this multilevel method to our PUM setting. In Section 1.2 we already gave an algorithm that yields covers C_Ω^k on levels $k = 0, \dots, k_{\max}$ that can be used to create function spaces V_k^{PU} so the first step is already done and we

can discretize the operators A_k for each of those function spaces. The next ingredient we need to provide are the inter-level transfer operators I_{k-1}^k and I_k^{k-1}. Due to the fact that our hierarchical function space construction yields non-nested function spaces $V_0^{\mathrm{PU}} \not\subset V_1^{\mathrm{PU}} \not\subset \cdots \not\subset V_{k_{\max}}^{\mathrm{PU}}$ and that additionally our shape functions $\varphi_{i,k}\vartheta_{i,k}^s$ are non-interpolatory in general we cannot create the transfers by natural injection, nor by simple interpolation. Nevertheless it has been suggested to create the inter-level transfer operators by L^2-projections in [53] and we briefly summarize that approach in the following. An inter-level L^2-projection $\Pi_{k-1}^k : V_{k-1}^{\mathrm{PU}} \to V_k^{\mathrm{PU}}$ can be defined by utilizing the mass matrices

$$\begin{aligned}(M_k^k)_{i,j} &:= \langle \varphi_{j,k}\vartheta_{j,k}^t, \varphi_{i,k}\vartheta_{i,k}^s \rangle_{L^2}, \\ (M_{k-1}^k)_{i,j} &:= \langle \varphi_{j,k-1}\vartheta_{j,k-1}^t, \varphi_{i,k}\vartheta_{i,k}^s \rangle_{L^2}\end{aligned}$$

to construct the global L^2-projection as

$$\Pi_{k-1}^k := (M_k^k)^{-1}(M_{k-1}^k).$$

We can identify some issues with this global construction: First of all the inter-level mass matrix contains a significant amount of non-zero entries. The reason for this is that the sparsity pattern of that matrix is determined by the intersections of patches over multiple levels $\omega_{j,k-1} \cap \omega_{i,k} \neq \emptyset$ which for uniform refined covers yields 4^d neighbors to a coarse patch $\omega_{j,k-1}$ compared to the usual $3^d - 1$ neighbors within a single level. But not only the inter-level mass matrix is rather dense. The mass matrix M_k^k has the same sparsity pattern as the stiffness matrix A which already doubles the space requirement of our method on the finest level only. But the final and most problematic issue is that we need to invert the global mass matrix M_k^k, which renders the global projection as currently constructed unsuitable in practice. Two modifications that solve each of the mentioned issues to a varying extend have been proposed. The first modified version, denoted the *global-to-local* L^2-projection, is to replace the mass matrices by versions that are localized on the fine level and that are given by

$$\begin{aligned}(\tilde{M}_k^k)_{i,i} &:= \langle \vartheta_{i,k}^t, \vartheta_{i,k}^s \rangle_{L^2(\omega_{i,k}\cap\Omega)}, \\ (\hat{M}_{k-1}^k)_{i,j} &:= \langle \varphi_{j,k-1}\vartheta_{j,k-1}^t, \vartheta_{i,k}^s \rangle_{L^2(\omega_{i,k}\cap\Omega)}\end{aligned} \tag{1.40}$$

where $\tilde{M}_k^k$ is now a block-diagonal matrix and hence easy to invert in the construction for the final projection operator

$$\hat{\Pi}_{k-1}^k := (\tilde{M}_k^k)^{-1}(\hat{M}_{k-1}^k). \tag{1.41}$$

We can show that this localization does not impose any significant loss of accuracy. Given the fact that in the PUM the global error is bounded by the sum of our local errors according to (1.1) we have

$$\|v - v^{\mathrm{PU}}\|_{L^2(\Omega)}^2 \leq C \sum_{i=1}^N \|v - v_i\|_{L^2(\Omega\cap\omega_i)}^2$$

where $v^{\mathrm{PU}} := \sum_{i=1}^{N} \varphi_i \sum_{n=1}^{d_i} u_i^s \vartheta_i^n$ and $v_i := \sum_{n=1}^{d_i} u_i^s \vartheta_i^n$ with $d_i := \dim(V_i)$. For the localized inter-level projection from the coarse to the fine level we have $v = u_{k-1}^{\mathrm{PU}}$ and $v^{\mathrm{PU}} = I_{k-1}^k u_{k-1}^{\mathrm{PU}}$ which results in

$$\|u_{k-1}^{\mathrm{PU}} - I_{k-1}^k u_{k-1}^{\mathrm{PU}}\|_{L^2(\Omega)}^2 \leq C \sum_{i=1}^{N} \|u_{k-1}^{\mathrm{PU}} - u_{i,k}\|_{L^2(\Omega \cap \omega_{i,k})}^2 \tag{1.42}$$

as an error bound of the inter-level transfer problem that states that the global projection error is bounded by the errors on the fine level patches $\omega_{i,k}$. Hence it is sufficient to approximate the global function u_{k-1}^{PU} by the local basis functions $\vartheta_{i,l}^n$ of $V_{i,k}$ only.
While the *global-to-local* projection solves the biggest problem (i.e. we needed to invert the matrix M_k^k) and thus it makes it suitable for the use in practice, the sparsity pattern of the localized inter-level transfer matrix $\hat{M}_{k-1}^k$ did not change compared to the one of M_{k-1}^k. To this end, an additional localization step was proposed where we want to additionally exploit the PUM construction of the coarse level space V_{k-1}^{PU}. Using Lemma 1.4 which stated that for each patch $\omega_{i,k}$ we have exactly one coarse patch $\omega_{\tilde{i},k-1}$ such that $\omega_{i,k} \subseteq \omega_{\tilde{i},k-1}$, we can introduce a term of the coarse level solution $-u_{\tilde{i},k-1} + u_{\tilde{i},k-1}$ into the local error expressions from the right-hand side of 1.42 and then apply the triangle inequality to get

$$\begin{aligned} \|u_{k-1}^{\mathrm{PU}} - u_{i,k}\|_{L^2(\Omega \cap \omega_{i,k})} \quad \leq \quad & \|u_{k-1}^{\mathrm{PU}} - u_{\tilde{i},k-1}\|_{L^2(\Omega \cap \omega_{i,k})} + \\ & \|u_{\tilde{i},k-1} - u_{i,k}\|_{L^2(\Omega \cap \omega_{i,k})} \end{aligned} \tag{1.43}$$

for the local error bounds. Note that the first term on the right-hand side of (1.43) is small by construction and thus it is sufficient to be concerned with the control of the errors $\|u_{\tilde{i},k-1} - u_{i,k}\|_{L^2(\Omega \cap \omega_{i,k})}$. This leads to the construction of the *local-to-local* projection by utilizing the fully localized mass matrices

$$\begin{aligned} (\tilde{M}_k^k)_{i,i} &:= \langle \vartheta_{i,k}^t, \vartheta_{i,k}^s \rangle_{L^2(\omega_{i,k} \cap \Omega)}, \\ (\tilde{M}_{k-1}^k)_{i,j} &:= \begin{cases} \langle \vartheta_{j,k-1}^t, \vartheta_{i,k}^s \rangle_{L^2(\omega_{i,k} \cap \Omega)} & \text{if } j = \tilde{i} \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{1.44}$$

to end up with the projection operator

$$\tilde{\Pi}_{k-1}^k := (\tilde{M}_k^k)^{-1} (\tilde{M}_{k-1}^k). \tag{1.45}$$

This new fully local projection eliminated all of the weaknesses of the initial global projection. The matrix $\tilde{M}_k^k$ is block diagonal and thus sufficiently sparse and easy to invert. Additionally the matrix $\tilde{M}_{k-1}^k$ is just as sparse: For each row i there is only a single block $(\tilde{M}_{k-1}^k)_{i,j} \neq 0$ for the patch where $\omega_{j,k-1} \supseteq \omega_{i,k}$ holds. Additionally it should be mentioned that the partition of unity functions $\varphi_{i,k}$ and $\varphi_{i,k-1}$ do not play any role during the assembly of both $\tilde{M}_k^k$ as well as $\tilde{M}_{k-1}^k$. Therefore we can employ a decomposition of Ω into integration cells that do not take the kinks and/or discontinuities of weights $W_{i,k}$ into account and additionally we can use lower quadrature rules on cells that are located in the overlap of two or more patches.

2 Domain Respecting Cover Construction

In this chapter we revisit the cover construction that was introduced in Section 1.2 and take a closer look on its properties with respect to stability under the influence of complex geometries Ω. We can identify two main shortcomings where a stiffness matrix resulting from a discretization using a function space V^{PU} that has been constructed as described in Section 1.2 can still be ill-conditioned or even singular:

- Even though we chose polynomials ψ_i^s that are orthogonal on ω_i, they might not be (numerically) linearly independent when we restrict them to a subset $\tilde{\omega}_i \subset \omega_i$ (compare Table 2.1). This can be particularly problematic for patches at the boundary since in that case it is $\omega_i \cap \Omega =: \tilde{\omega}_i \subset \omega_i$.

- Even though the flat-top property from Definition 1.3 guarantees that $\langle \varphi_i \vartheta_i^n \rangle$ provides a stable basis for the function space V^{PU} over $\mathcal{D}_{C_\Omega} := \bigcup_{i=1}^N \omega_i$ when all local functions $\langle \vartheta_i^n \rangle$ are linearly independent, this flat-top property is no longer sufficient to induce global stability from local stability when we restrict our functions to $\Omega \subset \mathcal{D}_{C_\Omega}$.

To overcome those issues we will propose a post-processing step, applied to each level of the original, hierarchical cover construction, in the following.

2.1 Cover Post-Processing

Let us first take a look at the second item from the list above, that deals with global instabilities and illustrate the problem by a simple example:

Example 2.1. Given two overlapping patches $\omega_1 := (-1, 1)$ and $\omega_2 := (0, 2)$ with respective partition of unity functions

$$\varphi_1(x) := \begin{cases} 1 & \text{for } x \in (-1, 0] \\ 1 - x & \text{for } x \in (0, 1) \\ 0 & \text{elsewhere} \end{cases}, \qquad \varphi_2(x) := \begin{cases} x & \text{for } x \in (0, 1) \\ 1 & \text{for } x \in [1, 2) \\ 0 & \text{elsewhere} \end{cases}$$

Table 2.1: Condition numbers of local mass matrices $M := \langle \mathcal{L}^p, \mathcal{L}^p \rangle_{L^2(\tilde{\omega})}$ with $p \leq p_{\max}$ where $\mathcal{L}^p : [-1, 1] \to \mathbb{R}$ are the one-dimensional Legendre polynomials of degree p and $\tilde{\omega} := [-h, h] \subseteq [-1, 1]$ for $0 < h \leq 1$. While M stays well conditioned for $h = 1$ even when using higher-order polynomials, the condition number reaches machine-precision limits for $p_{\max} = 5$ when $\tilde{\omega}$ covers only $\sim 6\%$ of the original domain of $\mathcal{L}^p$.

	$p_{\max}$					
h	0	1	2	3	4	5
1	$1.0 \cdot 10^0$	$3.0 \cdot 10^0$	$5.0 \cdot 10^0$	$7.0 \cdot 10^0$	$9.0 \cdot 10^0$	$1.1 \cdot 10^1$
0.5	$1.0 \cdot 10^0$	$1.2 \cdot 10^1$	$1.0 \cdot 10^2$	$1.2 \cdot 10^3$	$1.3 \cdot 10^4$	$1.7 \cdot 10^5$
0.25	$1.0 \cdot 10^0$	$4.8 \cdot 10^1$	$1.9 \cdot 10^3$	$1.0 \cdot 10^5$	$5.5 \cdot 10^6$	$3.2 \cdot 10^8$
0.125	$1.0 \cdot 10^0$	$1.9 \cdot 10^2$	$3.2 \cdot 10^4$	$7.2 \cdot 10^6$	$1.6 \cdot 10^9$	$3.8 \cdot 10^{11}$
0.0625	$1.0 \cdot 10^0$	$7.7 \cdot 10^2$	$5.1 \cdot 10^5$	$4.7 \cdot 10^8$	$4.3 \cdot 10^{11}$	$4.1 \cdot 10^{14}$

and the local polynomial basis functions

$$\vartheta_1^1(x) := 1, \qquad \vartheta_1^2(x) := x \quad \text{and}$$
$$\vartheta_2^1(x) := 1, \qquad \vartheta_2^2(x) := x.$$

Both PU functions clearly fullfil the flat-top property as given in Definition 1.3 on $(-1, 0]$ and $[1, 2)$ respectively and local linear independence of $\langle \vartheta_i^n \rangle$ implies that $\langle \varphi_i \vartheta_i^n \rangle$ is a basis for V^{PU} over $(-1, 1)$. But when we restrict the problem onto the domain $\Omega := (0, 1)$, the flat-top regions are cut away and we end up in the same case as illustrated by Example 1.1. Especially we again have $\varphi_1(x)\vartheta_1^2(x) = \varphi_2(x)\vartheta_2^1(x) - \varphi_2(x)\vartheta_2^2(x) = x - x^2$ for all $x \in \Omega$.

This problem can be overcome when we tighten the requirements of the flat-top property and require that not only every patch has a sufficient flat-top domain ω_i^{FT}, but that its intersection with Ω does not vanish as well. To this end, we define the *strict flat-top property* as follows:

Definition 2.1 (Strict flat-top property)**.** Let $\{\varphi_i \mid i, \ldots, N\}$ be a partition of unity according to Definition 1.1. Let us define the sub-patches $\omega_i^{\mathrm{FT}} \subset \omega_i$ such that $\varphi_i|_{\omega_i^{\mathrm{FT}}} \equiv 1$. The PU is said to have the strict flat-top property, if there exists a constant C_{FT} such that for all patches $\omega_i = \mathrm{supp}(\varphi_i)$

$$\mu(\omega_i) \leq C_{\mathrm{FT}} \mu(\omega_i^{\mathrm{FT}} \cap \Omega) \tag{2.1}$$

where $\mu(A)$ denotes the Lebesgue measure of $A \subset \mathbb{R}^d$.

Note that Definition 2.1 introduced only a minor change compared to the original flat-top specification of Definition 1.3. For the strict flat-top property we now take $\omega_i^{\mathrm{FT}} \cap \Omega$ into account only, compared to ω_i^{FT} in (1.7) from the original definition. The tree-based cover construction algorithm introduced in Section 1.2 only creates an admissible cover according to Definition 1.3, but not necessarily according to Definition 2.1 (compare

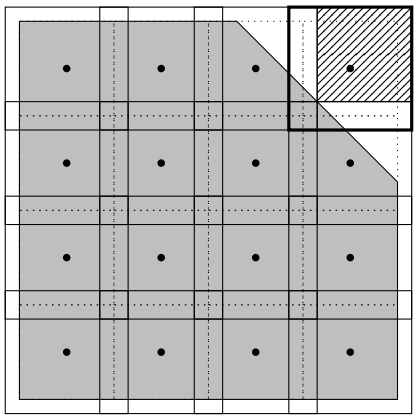

Figure 2.1: Cut-corner domain Ω (gray area) with a uniform disjoint cover $\hat{C}_\Omega$ (dashed rectangles) on level $k = 2$ and the respective overlapping cover C_Ω (solid rectangles) as constructed by Algorithm 1.1. Although the disjoint tree cell $\mathcal{C}_i$ for the upper right patch has a non-empty intersection with the domain and thus the corresponding patch ω_i is added into the cover C_Ω, the flat-top region of that patch (diagonally striped region) has an empty intersection with the domain Ω.

Figure 2.1). Hence, we introduce a post-processing step that transforms a cover created by Algorithm 1.1 into one that fulfills the strict flat-top property. To this end, we should note that for all patches where $\omega_i \subset \Omega$ holds, the original flat-top property and the strict version are equivalent and thus we are only concerned with patches at the boundary for which $\omega_i \cap \partial\Omega \neq \emptyset$ holds. We can further reduce the set of problematic patches by testing whether $\boldsymbol{c}_i \in \Omega$ holds for the center $\boldsymbol{c}_i$ of a patch ω_i. In this case the strict flat-top property has to be fulfilled for the patch since the choice of the stretch factors α_i and/or the bound $L_{\max}$ on the depth differences in Algorithm 1.1 guarantee that $\boldsymbol{c}_i$ is not included in any other patch ω_j with $j \neq i$. For all remaining boundary patches we need to check whether the intersection $\omega_i^{\mathrm{FT}} \cap \Omega$ is empty or not. If the intersection is empty we want to simply eliminate those patches from C_Ω. A simple elimination of a patch ω_i with $\omega_i^{\mathrm{FT}} \cap \Omega = \emptyset$ is valid since in this case the part of ω_i that overlaps the domain omega is completely covered by its neighbors, i.e. $(\omega_i \cap \Omega) \subset \bigcup_{\omega_j \in C_i} \omega_j$, and hence ω_i is not required to obtain a complete covering of Ω. Note that the arguments given before do not hold when we consider a complete covering of $\bar{\Omega}$ instead of just Ω in cases that the flat-top region of an eliminated patch touches the domain, i.e. $\partial\omega_i^{\mathrm{FT}} \cap \partial\Omega \neq \emptyset$. These cases can be mitigated by enlarging the neighboring patches by a small amount, which we will explain in more detail later.

At this point it is important to note that the shape of ω_i^{FT} does not only depend on ω_i itself but especially on the neighbors $\omega_j \in C_i$ of ω_i. This has two implications: First of all, the shape is not a simple axis-aligned box in general and hence checking whether $\omega_i^{\mathrm{FT}} \cap \Omega = \emptyset$ is computationally more involved in general. Secondly, the elimination of a patch ω_i has an influence on the flat-top property of all its neighbors $\omega_j \in C_i$ since ω_i has to be removed from all neighborhoods C_j as well. This makes the elimination step inherently sequential:

We cannot check concurrently for each patch ω_i whether it needs to be eliminated or not since the prior elimination of a patch ω_j could have enlarged ω_i^{FT} and thus eliminating ω_i could not only no longer be necessary but could even lead to a part of the domain Ω not being covered anymore. Luckily, the elimination of a patch ω_i always only enlarges the flat-top regions ω_j^{FT} of its neighboring patches, but never reduces them. This means that when we have checked whether a patch ω_i needs to be eliminated and found that $\omega_i^{\mathrm{FT}} \cap \Omega \neq \emptyset$, any subsequent elimination of a patch ω_j never changes the fact that ω_i does not need to be eliminated. Hence the elimination step is *sequential* but does not need to be *iterated*. To speed up the check whether a patch has an empty flat-top intersection with the domain we utilize the decomposition into the intermediate integration domains $\{\tilde{\mathcal{D}}_i^m\}$ from Section 1.3 as depicted in Figure 1.5. We discard all domains $\tilde{\mathcal{D}}_i^m$ where any patch ω_j other than ω_i is present. For the remaining domains $\tilde{\mathcal{D}}_i^m$ (which are axis-aligned d-dimensional boxes) we can check whether the intersection with Ω is empty or not. As soon as we found the first non-empty intersection we can stop and the patch does not need to be eliminated. Only if there is no $\tilde{\mathcal{D}}_i^m \subseteq \omega_i^{\mathrm{FT}}$ with a non-empty intersection the patch needs to be eliminated.

Let us now take a closer look at what happens to our PU functions φ_i when we eliminate patches from the cover C_Ω. To this end, we consider the case where we have four patches of the same size and distance and one of them is eliminated from the cover which results in something we call an *L-constellation* (compare the upper left in Figure 2.2). Those constellations introduce a singularity into the gradients of the shepard partition of unity functions φ_i. The singularity arises when flat-top regions "touch" each other. This can be explained because for a patch ω_i with flat-top region ω_i^{FT}, for its corresponding shepard PU function φ_i it must hold that $\varphi_i(x) = 1$ on the boundary of the flat-top region, i.e. for all $\boldsymbol{x} \in \bar{\omega}_i^{\mathrm{FT}}$. On the other hand, it has to hold $\varphi_i(x) = 0$ for all $\boldsymbol{x} \in \bar{\omega}_j^{\mathrm{FT}}$ with $j \neq i$. At locations $\chi := \bar{\omega}_i^{\mathrm{FT}} \cap \bar{\omega}_j^{\mathrm{FT}}$ these conditions contradict and in the vicinity of those locations it is $\lim_{\mathrm{dist}(\boldsymbol{x},\chi)\to 0} \nabla\varphi_i = \infty$ (compare Figure 2.2). Note that $\chi \neq \emptyset$ can only happen if $d > 1$. In 2D those sets can only contain isolated points. In 3D those sets can contain points or edges, but no facets or volumes. In general we can say that only geometric entities of co-dimension ≥ 2 can be elements of χ. Additionally note that in general those singular locations are not a problem by themselves. E.g. we can still represent smooth functions u^{PU} by linear combinations of all shape functions $\varphi_i \vartheta_i^m$. The real problem is that the accuracy of our numerical quadrature of the gradients $\nabla\varphi_i$ is highly influenced by those locations and we would have to employ adaptive quadrature rules towards all $\boldsymbol{x} \in \chi$ to get a sufficiently good result. To mitigate that issue we want to make sure that the problematic locations are far away from Ω. Moving the singularities away from $\partial\Omega$ can be achieved by enlarging the neighboring patches $\omega_j \in C_i$ when ω_i has been eliminated (compare Figure 2.3). While simply increasing the stretch factors α_j for the neighboring patches might seem like the most simple solution it comes with some major downsides: First of all α_j could already be the maximum allowed value according to (1.26) and thus choosing a higher value could lead to a patch in the neighborhood of ω_j being completely covered by its own neighbors, and thus its flat-top region will be empty. But even when we are not limited by (1.26) yet, stretching a patch ω_j equally into all directions reduces

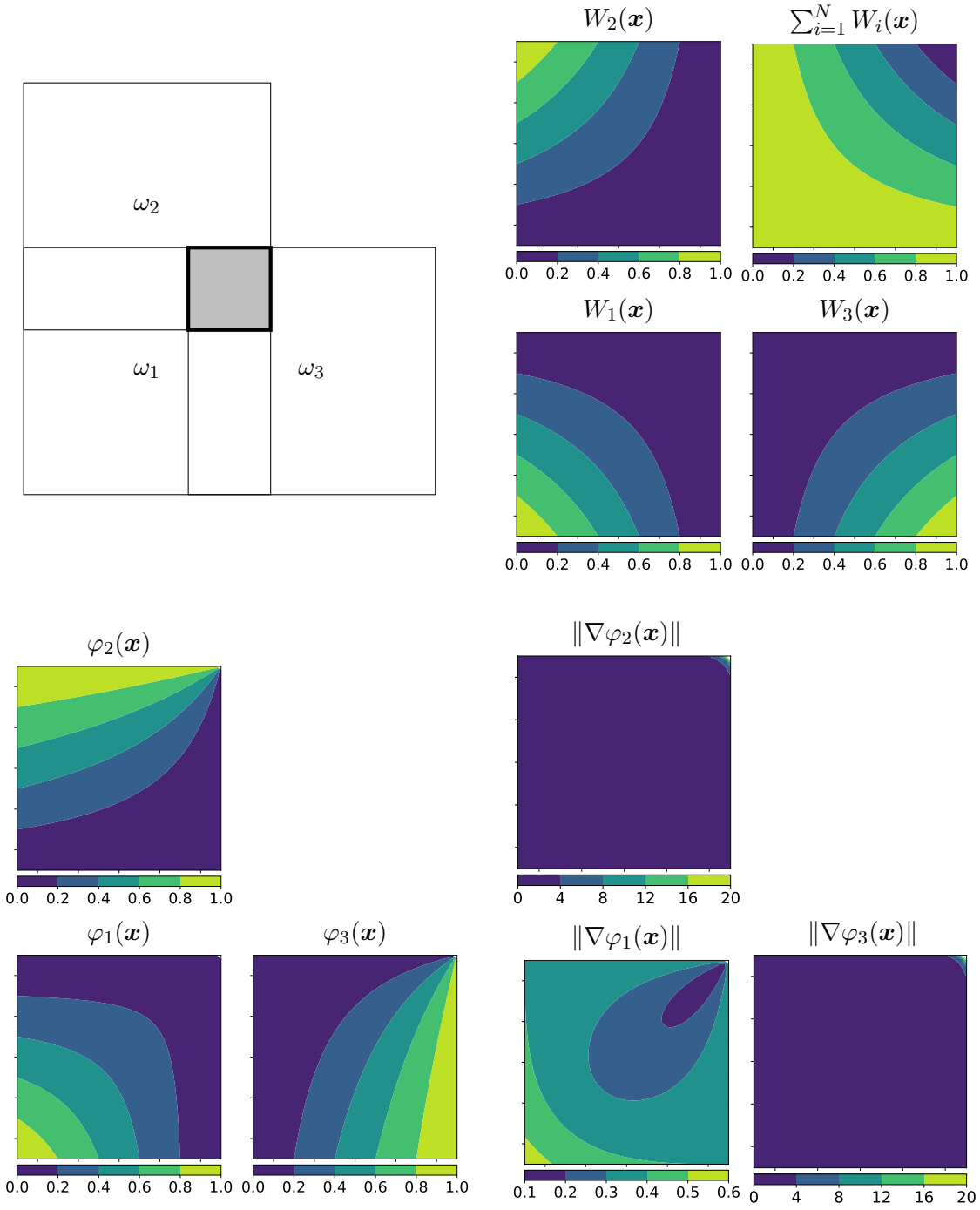

Figure 2.2: *Upper left:* Setting of three patches ω_1, ω_2 and ω_3 where the forth patch in the upper right corner is not present. The overlap region $\omega_1 \cap \omega_2 \cap \omega_3$ of all three patches is highlighted as gray box.
Upper right: Contour plot of the linear flat-top spline weight functions $W_i(\boldsymbol{x})$ (compare Figure 1.7) corresponding to the patches ω_i. Only the overlap region with $\boldsymbol{x} \in \omega_1 \cap \omega_2 \cap \omega_3$ is shown. Note that the sum of all weight functions is zero in the upper right corner of the overlap region.
Lower left: Rational shepard partition of unity functions φ_i according to (1.15) resulting from the weights W_i. The PUs are undefined in the upper right corner.
Lower right: Magnitude of the gradients $\nabla\varphi_i$. The gradients of the diagonally located patches ω_2 and ω_3 go to infinity when $\boldsymbol{x}$ approaches the upper right corner (the color-bar has been capped at 20).

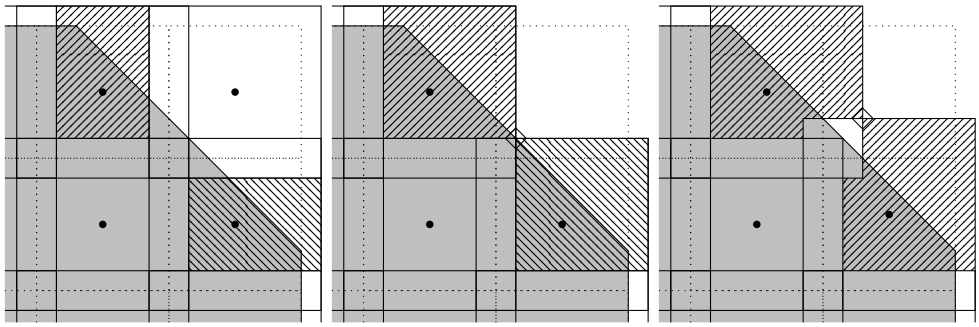

Figure 2.3: *Left:* Zoom into the four upper right patches from Figure 2.1. The flat-top regions of the patches left and below the invalid patch are highlighted.
Center: The invalid patch has been eliminated from the cover and the flat-top regions of the neighboring patches now protrude further into the region of the eliminated patch. The point where the flat-top regions touch is marked by a diamond.
Right: The domains of the two neighboring patches have been extended anisotropically into the region of the eliminated patch. The point where the flat-top regions touch has been moved further away from the domain Ω. The new flat-top regions of the neighboring patches are a superset of the flat-top regions before the patch elimination took place. No flat-top regions of other patches have been changed.

the flat-top regions ω_k^{FT} of its neighboring patches $\omega_k \in C_j$ and could thus lead to the fact that we end up with $\omega_k^{\mathrm{FT}} \cap \Omega_j = \emptyset$ for such a neighbor. This would then nullify our previous property that eliminating a patch does not lead to any other patch being required to be eliminated and hence we would need to apply the elimination step repeatedly over all patches until no patches change anymore. To overcome those issues we do not enlarge all neighboring patches $\omega_j \in C_i$ after the elimination ω_i and we do not stretch them by the same amount into all directions. Instead we only enlarge those neighboring patches whose tree cells $\mathcal{C}_j$ have a common facet with $\mathcal{C}_i$ (e.g. in 2D these are the cells left, right, below and above $\mathcal{C}_i$, but not the diagonal neighbors) and we enlarge those patches anisotropically towards the center of ω_i only. This makes sure that they are expanded into ω_i only and hence cannot overlap any region that was not previously overlapped by ω_i itself. This means we do not reduce the flat-top property of any other patch and the feature that the elimination step does not need to be repeated after a single iteration of all patches stays valid (compare Figure 2.3). Note that while this holds for all uniform covers, it does not necessarily hold for adaptively refined covers when a patch ω_i gets eliminated and we need to enlarge one of its neighbors ω_j that belongs to a tree cell from higher in the cover tree than the patch to be eliminated, i.e. the neighbor ω_j is larger than the patch ω_i. Hence, we do only enlarge a neighboring patch into the direction of the eliminated patch if the neighbors diameter is smaller or equal to the diameter of the eliminated patch. Otherwise we skip enlarging the patch and the potentially remaining singularity has to be resolved by adaptive refinement of the integration cells.
We are now left with the question about the amount by which the neighboring patches

should be enlarged. To answer that questions we should note that we do not only get singularities in the derivatives of the PU functions when a patch is missing in the cover due to the newly introduced elimination post-processing step. They can arise whenever a disjoint tree cell $\mathcal{C}_i$ does not intersect the domain and thus no patch ω_i is created for it. The difference is that in those cases the singularity is always located outside of $\bar{\Omega}$. Let us define the set of singular locations that arise from the elimination (or absence in general) of a patch ω_i as

$$\chi_i := \bigcup_{\omega_j,\omega_k \in C_i} \left(\bar{\omega}_j^{\mathrm{FT}} \cap \bar{\omega}_k^{\mathrm{FT}} \right), \qquad j \neq k.$$

Then, when ω_i is absent from the cover due to $\mathcal{C}_i \cap \Omega = \emptyset$ for its corresponding tree cell $\mathcal{C}_i$, it is

$$\operatorname{dist}(\boldsymbol{x}, \partial\Omega) \geq \min_{\mathcal{C}_j \in \hat{C}_i} \min_{m=1,\dots,d} \left((\alpha_j - 1) \frac{(\boldsymbol{h}_j)_m}{2} \right) \quad \forall \boldsymbol{x} \in \chi_i$$

with $(\boldsymbol{h}_j)_m$ being the diameter of $\mathcal{C}_j$ in the coordinate direction m. Experiments have shown that those singularities do not impose any problems for the accuracy of the numerical quadrature in the overall discretization process and hence moving the singularities that arise from a patch elimination in the post-processing step away from $\partial\Omega$ by a similar amount should be sufficient. We thus propose to extend a patch ω_j into the direction of ω_i by

$$(\alpha_j - 1) \frac{(\boldsymbol{h}_j)_m}{2}.$$

Experimental verification that this choice is sufficient to retrieve optimal convergence rates without the need to adjust the local quadrature will be presented in Section 2.2. An additional benefit is that this guarantees that a patch $\omega_{i,k}$ on a level k still completely covers all its children on level $k+1$, even when the children have been enlarged and thus Lemma 1.3 still holds. On the other hand, Lemma 1.4 that states that for each patch $\omega_{i,k}$ there is exactly one patch $\omega_{\tilde{i},k-1}$ such that $\omega_{\tilde{i},k-1} \supseteq \omega_{i,k}$ holds, is not necessarily true anymore. This can happen when $\omega_{\tilde{i},k-1}$ has been eliminated from the cover on level $k-1$ but at least one of its children is part of the cover on level k (compare Figure 2.3 and 2.4). In such a case we cannot make use of the *local-to-local* projection as given in (1.44) and (1.45) in the construction of our multilevel preconditioner anymore. One option would be to fall back to the *global-to-local* projection as given in (1.40) and (1.41), but since the number of eliminated patches on any level k is usually small compared to the total amount of patches N_k, considering that elimination can only occur for patches intersecting the boundary, we do not need to impose the cost of using all hierarchical neighbors everywhere. Instead let us first denote the set of all patches $\omega_{i,k}$ that have been eliminated on a level k by C_k^E. Then we can define the *mixed-to-local* projection

$$\begin{aligned} (\tilde{M}_k^k)_{i,i} &:= \langle \vartheta_{i,k}^t, \vartheta_{i,k}^s \rangle_{L^2(\omega_{i,k} \cap \Omega)}, \\ (\hat{\tilde{M}}_{k-1}^k)_{i,j} &:= \begin{cases} \langle \vartheta_{j,k-1}^t, \vartheta_{i,k}^s \rangle_{L^2(\omega_{i,k} \cap \Omega)} & \text{if } \omega_{\tilde{i},k-1} \notin C_{k-1}^E \text{ and } j = \tilde{i} \\ \langle \varphi_{j,k-1} \vartheta_{j,k-1}^t, \vartheta_{i,k}^s \rangle_{L^2(\omega_{i,k} \cap \Omega)} & \text{if } \omega_{\tilde{i},k-1} \in C_{k-1}^E \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{2.2}$$

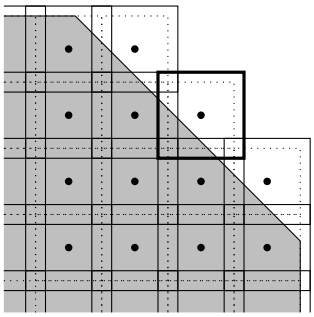

Figure 2.4: The cover from Figure 2.1 has been uniformly refined to level $k = 3$. The four tree cells $\mathcal{C}_i$ in the upper right corner from Figure 2.3 resulted in 16 smaller cells from which 13 have been stretched into patches of the cover C_Ω on level 3. No patches have been created for the other cells since their tree cells $\mathcal{C}_i$ do not intersect the domain. All created patches have a flat-top region that intersects the domain and no patches need to be eliminated. Note that one of the patches (highlighted by a thicker line) is a child of the cell which's corresponding patch has been eliminated on level 2.

to end up with the projection operator

$$\hat{\tilde{\Pi}}^k_{k-1} := (\tilde{M}^k_k)^{-1}(\hat{\tilde{M}}^k_{k-1}). \tag{2.3}$$

Here, $\hat{\tilde{M}}^k_{k-1}$ (and hence $\hat{\tilde{\Pi}}^k_{k-1}$) have a single block that is not zero in most rows, but in rows i where the parent patch $\omega_{\tilde{i},k-1}$ has been eliminated, we have a non-zero block for all patches $\omega_{j,k-1}$ where $\omega_{j,k-1} \cap \omega_{i,k} \cap \Omega \neq 0$. Note that in most practical cases these are still only a few non-zero entries, since the intersection $\omega_{i,k} \cap \Omega$ is usually small for patches where the parent has been eliminated and since those patches are at the boundary the number of patches $\omega_{j,k-1}$ for which $\omega_{j,k-1} \cap \omega_{i,k} \neq 0$, is not very high as well since many tree cells $\mathcal{C}_{j,k-1}$ in the vicinity of $\omega_{i,k}$ were already outside of Ω.
With the changes proposed above we do now have covers C_Ω that can be used to create a partition of unity $\{\varphi_i\}$ that fulfills the strict flat-top property according to Definition 2.1, with PU functions $\varphi_i(x)$ that can be integrated numerically over Ω with sufficient accuracy and efficiency, and we can still construct a fast multilevel preconditioner using the sequence of covers C^k_Ω. We are now left to solve the problem mentioned in the first item at the beginning of this section, about instabilities of the local polynomial spaces $\langle \psi^n_i \rangle$ when patches do only have small overlaps with Ω. In theory those instabilities do not impose any major problem to the overall simulation since the stable transformation that was described in Section 1.5 does create an orthogonal basis subject to the $L2$-norm restricted to each ω_i and remove the numerical kernel from the original basis. In cases where the intersection $\omega_i \cap \Omega$ is very small this could potentially include eliminating a part of the polynomial space $\text{span}\langle \psi^n_i \rangle$. While this would be sufficient to make sure any globally discretized operator matrix A would not be ill-conditioned, our local error bounds given in (1.3) require that the local function spaces V_i contain all polynomials up to degree p. Hence, as soon as the stable transformation starts to eliminate any part of the space $\text{span}\langle \psi^n_i \rangle$ on a patch ω_i, it

is not guaranteed that we are able to obtain optimal convergence rates anymore. In fact, in the worst case, the error of the approximation u_k^{PU} could be larger than the error of the approximation u_{k-1}^{PU} if we applied some h-refinement when refining from level $k-1$ to level k, and due to small overlaps on level k, the stable transformation started to reduce some function spaces $V_{i,k}$. We propose a very simple solution to this problem: For each patch with $\omega_i \cap \partial\Omega \neq \emptyset$ we compute the intersection $\omega_i \cap \Omega$ and then the minimal bounding box $\mathcal{B}_{\omega_i} := \bigotimes_{m=1}^{d} \left((\boldsymbol{r}_i)_m, (\boldsymbol{s}_i)_m\right)$ of that intersection such that $\mathcal{B}_{\omega_i} \supseteq (\omega_i \cap \Omega)$. Assume ω_i is currently given as $\omega_i := \bigotimes_{m=1}^{d} \left((\boldsymbol{a}_i)_m, (\boldsymbol{b}_i)_m\right)$. Then we compute a smaller domain $\tilde{\omega}_i$ with $\tilde{\omega}_i \subseteq \omega_i$ as

$$\tilde{\omega}_i := \bigotimes_{m=1}^{d} \Big(\max\left\{(\boldsymbol{a}_i)_m, (\boldsymbol{r}_i)_m - (\boldsymbol{e}_i)_m\right\}, \min\left\{(\boldsymbol{b}_i)_m, (\boldsymbol{s}_i)_m + (\boldsymbol{e}_i)_m\right\} \Big)$$

where $(\boldsymbol{e}_i)_m > 0$ is some safety distance that we add to the minimal bounding box. We need to choose $(\boldsymbol{e}_i)_m > 0$ to ensure that not only the open set Ω is fully enclosed within all patches, but its boundary $\partial\Omega$ is covered as well. Additionally choosing $(\boldsymbol{e}_i)_m$ too small might move singular locations $\boldsymbol{x} \in \chi$ closer to Ω again. Hence, in practice we use $(\boldsymbol{e}_i)_m := (\alpha_i - 1)\frac{(\boldsymbol{h}_i)_m}{2}$ which is exactly the distance by which we wanted singularities to be kept away from $\partial\Omega$ in the elimination step. Finally Figure 2.5 depicts how an invalid cover that resulted from Algorithm 1.1 is altered by the proposed post-processing step by first eliminating patches violating Definition 2.1 and then shrinking boundary patches as described above.

2.2 Numerical Experiments

In this section we want to validate that the post-processing step that was proposed in the previous section does indeed yield function spaces V_k^{PU} that are formed by bases that are linearly independent and stable in the sense that the discretized operator matrices are well-conditioned and the resulting equation systems can thus be solved efficiently by numerical solvers. Additionally, we show that we can obtain optimal convergence rates towards known reference solutions in two and three space dimensions. The ability to obtain those optimal rates shows that the employed local function spaces do indeed provide the necessary approximation power so that Theorem 1.1 and its h-version (1.4) hold. In particular, this has thus to be true for patches at the boundary that have been adjusted by the proposed post-processing step. Additionally, it shows that all numerical integrals can be computed with a sufficient precision such that the overall method works as expected. Throughout this first section that presents results of numerical experiments, all domains Ω are described by either linear polygons in 2D or linear polyhedra in 3D. Numerical experiments for curved simulation domains in 2D will be presented later in Section 4.4 of Chapter 4 and for domains in 3D in Section 5.4 of Chapter 5.
Let us first introduce the relative errors subject to several norms by

$$e_{L^\infty} := \frac{\|u^* - u^{\mathrm{PU}}\|_{L^\infty}}{\|u^*\|_{L^\infty}}, \quad e_{L^2} := \frac{\|u^* - u^{\mathrm{PU}}\|_{L^2}}{\|u^*\|_{L^2}}, \quad e_{H^1} := \frac{\|\nabla(u^* - u^{\mathrm{PU}})\|_{L^2}}{\|\nabla u^*\|_{L^2}} \tag{2.4}$$

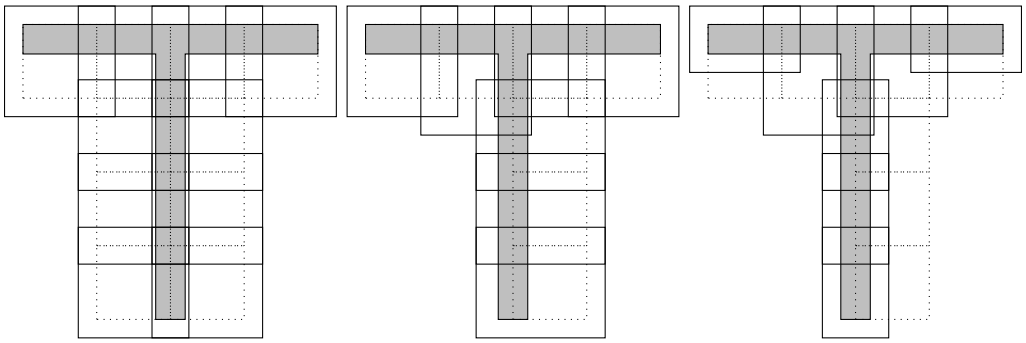

Figure 2.5: *Left:* T-domain Ω (gray area) with a uniform disjoint cover $\hat{C}_\Omega$ (dashed rectangles) on level $k = 2$ and the respective overlapping cover C_Ω (solid rectangles) as constructed by Algorithm 1.1. The flat-top regions of all six lower patches have an empty intersection with the domain.
Center: Three of the six patches have been eliminated and their neighbors have been enlarged. The flat-top regions of the resulting patches do all have a non-empty intersection with the domain. The cover C_Ω still covers the whole domain, i.e. $\bar{\Omega} \subset \bigcup_{i=1}^{N} \omega_i$.
Right: All patches on the boundary have been shrunk to reduce the area that has no intersection with the domain Ω. We can see that the original disjoint tree cells $\mathcal{C}_i$ by which we constructed the patches ω_i are no longer a subset of those patches, i.e. it is no longer $\mathcal{C}_i \subset \omega_i$ for all patches.

where u^* is the (analytical) reference solution. The corresponding convergence rates from a level $k-1$ to a level k are given by

$$\rho := -\frac{\log\left(\frac{\|u^* - u_k^{\mathrm{PU}}\|}{\|u^* - u_{k-1}^{\mathrm{PU}}\|}\right)}{\log\left(\frac{\mathrm{dof}_k}{\mathrm{dof}_{k-1}}\right)} \tag{2.5}$$

where $\mathrm{dof}_k := \dim(V_k^{\mathrm{PU}}) = \sum_{i=1}^{N_k} \dim(V_{i,k})$ is the number of degrees of freedom on a level k. The specific convergence rates ρ_{L^∞}, ρ_{L^2} and ρ_{H^1} are formed by employing norms in (2.5) that correspond to the norms used in (2.4). Under the assumption that the function u^* is sufficiently smooth we expect the optimal convergence rates to be $\rho_{L^2} = \frac{p+1}{d}$ and $\rho_{H^1} = \frac{p}{d}$ for a uniformly h-refined cover over $\Omega \subset \mathbb{R}^d$ with $p_{i,k} = p$ being a fixed polynomial degree of the involved local function spaces $V_{i,k}$ for all $i = 1, \ldots, N_k$ and $k = 0, \ldots, k_{\max}$. In this section the reference solution u^* is always given analytically. Hence, employing a quadrature rule with integration cells that have been constructed for a function space V_k^{PU} as described in Section 1.3 might not be sufficient to integrate the norms of u^* and especially not to evaluate the norms of the errors $e := u^* - u^{\mathrm{PU}}$. Hence, the integration cells to evaluate all norms involved in (2.4) have been constructed for a function space V_k^{PU} but the intermediate integration domains $\mathcal{D}_i^m$ have been refined uniformly and the employed orders of the Gaussian quadrature rules have been increased, such that the computed norms of the reference solution u^* were at least three decimal places more accurate than

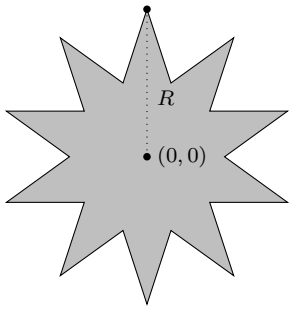

Figure 2.6: |10/4|-star polygon domain with outer radius $R = 15$.

the one of the approximated solution u_k^{PU}. Note that the L^∞ norm of the usually highly oscillating errors e is very difficult to compute and thus it is approximated by using the maximum of e evaluated at all integration points. This includes all integration points within Ω as well as on the boundary $\partial\Omega$. To reduce the influence of rounding issues from limited machine precision during the computation of the numerical integrals required for the errors in the L^2- and H^1-norms, a modified Kahan-Babuška summation algorithm according to [88] has been implemented.
Throughout this section we consider a scalar Helmholtz type of equation given by

$$\begin{aligned} -\Delta u + cu &= f && \text{in } \Omega \\ u &= g_{\mathrm{D}} && \text{on } \Gamma_{\mathrm{D}} \subset \partial\Omega \\ \nabla u \cdot \boldsymbol{n} &= g_{\mathrm{N}} && \text{on } \Gamma_{\mathrm{N}} = \partial\Omega \setminus \Gamma_{\mathrm{D}} \end{aligned} \tag{2.6}$$

with f being a right-hand side source term, g_{D} being Dirichlet and g_{N} Neumann boundary conditions imposed on the boundary parts Γ_{D} and Γ_{N}, respectively.

Example 2.2 (2D star domain)**.** In this example we consider equation (2.6) with $c = 1$ on a star shaped polygon domain (compare Figure 2.6). Since our main concern is to show the stability and approximation properties of the local function spaces $V_{i,k}$ for patches at the boundary we are going to impose Neumann boundary conditions only, i.e. $\Gamma_{\mathrm{N}} := \partial\Omega$ and $\Gamma_{\mathrm{D}} := \emptyset$. This makes sure all degrees of freedom are actually involved in the solution of the PDE itself and we do not need to fix any degrees of freedom for the Dirichlet boundary conditions by applying the conforming boundary transformation as described in Section 1.4. Note that using $c = 1$ in (2.6) guarantees that the assembled operator matrix is symmetric positive definite and has a unique solution, even though we are not imposing any Dirichlet boundary conditions. We choose the right-hand side function f and the Neumann boundary conditions g_{N} in such a fashion that the reference solution is given by

$$u^* := x\cos(y) + y\sin(x). \tag{2.7}$$

We first show that the many small intersections of the patches $\omega_{i,k}$ with the star-shaped domain Ω do indeed lead to local instabilities as mentioned in Section 2.1. Since the computation of the eigenvalues (end especially the smallest eigenvalue) of the assembled operator matrices is computationally very expensive and non-practical for large matrices,

Table 2.2: Number of solver iterations n until the residual has reached an absolute tolerance of 10^{-15} for Example 2.2 (star domain) using the original hierarchical cover construction **without** post-processing and **without** local stabilization (DEFAULT), **without** post-processing and **with** local stabilization (STABLE TRAFO) and **with** post-processing but **without** local stabilization (POST-PROCESS) employing polynomials of degree $p = 1$, $p = 3$ and $p = 5$ in the construction of all local function spaces $V_{i,k}$. Cases where the solver did not reach the desired residual tolerance after 1000 iterations are marked with a dagger †. The employed solver is a preconditioned CG method using a multilevel-preconditioner with $\gamma := 1$, $\nu_k^{\text{pre}} := \nu_k^{\text{post}} := 1$, global-to-local transfer according to (1.41) and a block factorized sparse approximate inverse (block-FSAI) smoother.

	DEFAULT			STABLE TRAFO			POST-PROCESS		
k	$n_{p=1}$	$n_{p=3}$	$n_{p=5}$	$n_{p=1}$	$n_{p=3}$	$n_{p=5}$	$n_{p=1}$	$n_{p=3}$	$n_{p=5}$
3	2	2	2	2	2	2	2	2	2
4	8	9	12	7	9	12	8	8	11
5	10	8	11	11	9	12	10	8	11
6	13	10	555	13	10	12	14	10	11
7	14	10	1,000†	14	11	12	14	11	11
8	14	10	1,000†	14	11	12	14	10	11
9	14	10	1,000†	15	11	15	14	10	11
10	14	10	–	15	11	–	16	11	–
11	14	10	–	16	12	–	15	10	–
12	15	10	–	16	11	–	15	10	–

we use the number of iterations that a numerical solver requires to solve the discretized equation system as an indicator for the condition of the operator matrix. We employ a preconditioned conjugate gradient (CG) method with the multilevel-preconditioner from Section 1.6 to solve the discretized equation system on multiple h-refined covers for different polynomial degrees $p_{i,k} =: p$. We first perform the experiments with the original cover construction presented in Algorithm 1.1 without applying a local stable transformation as presented in Section 1.5 and without the post-processing step as presented in Section 2.1. Then we repeat the experiment, once with the local stabilization enabled (using $\epsilon := 10^{-14}$ for the partitioning in (1.5)) and once with the post-processing step enabled. From the results shown in Table 2.2 we can see that while for lower polynomial degrees all matrices can be inverted efficiently, for higher polynomial degrees the naive cover construction without stabilization results in an ill-conditioned operator matrix. Additionally, the fact that applying the local stabilization fixes the arising ill-conditioning indicates that the ill-conditioning is a result of local (numerical) linear dependencies of the polynomials on some patches but not global linear dependencies resulting from the strict flat-top property being violated for some boundary patches. Furthermore, we can see that applying the post-processing step from Section 2.1 results in well-conditioned operator matrices and does not require a local stabilization of the polynomial spaces. Besides that, we can observe that the number of iterations for $p = 5$ is lower for the post-processing case than for the stabilized case. This can be explained by two things: First, we do use a relatively small cut-off factor ϵ in the stabilization so that the condition of a single patch

Table 2.3: Relative errors $e.$, convergence rates $\rho.$ and number of eliminated degrees of freedom dof_{el} for Example 2.2 (star domain) employing the original cover construction **with local stabilization**, using a global polynomial degree $p = 1$.

k	dof	dof_{el}	N	e_{L^∞}	ρ_{L^∞}	e_{L^2}	ρ_{L^2}	e_{H^1}	ρ_{H^1}
1	12	0	4	1.17_{0}	–	9.46_{-1}	–	9.67_{-1}	–
2	48	0	16	6.71_{-1}	0.21	6.98_{-1}	0.20	8.54_{-1}	0.11
3	156	0	52	3.66_{-1}	0.47	2.82_{-1}	0.77	5.61_{-1}	0.36
4	444	0	148	1.38_{-1}	0.93	1.30_{-1}	0.74	4.16_{-1}	0.29
5	1,512	0	504	4.79_{-2}	0.86	4.42_{-2}	0.88	2.54_{-1}	0.40
6	5,508	0	1,836	1.49_{-2}	0.91	1.26_{-2}	0.97	1.38_{-1}	0.47
7	21,108	0	7,036	4.35_{-3}	0.91	3.35_{-3}	0.99	7.13_{-2}	0.49
8	82,332	0	27,444	1.17_{-3}	0.97	8.55_{-4}	1.00	3.62_{-2}	0.50
9	325,116	0	108,372	3.07_{-4}	0.97	2.15_{-4}	1.00	1.82_{-2}	0.50
10	1,288,692	0	429,564	8.50_{-5}	0.93	5.40_{-5}	1.00	9.14_{-3}	0.50
11	5,131,332	0	1,710,444	2.36_{-5}	0.93	1.35_{-5}	1.00	4.58_{-3}	0.50
12	20,478,180	0	6,826,060	6.56_{-6}	0.92	3.38_{-6}	1.00	2.29_{-3}	0.50

Table 2.4: Relative errors $e.$, convergence rates $\rho.$ and number of eliminated degrees of freedom dof_{el} for Example 2.2 (star domain) employing the original cover construction **with local stabilization**, using a global polynomial degree $p = 3$.

k	dof	dof_{el}	N	e_{L^∞}	ρ_{L^∞}	e_{L^2}	ρ_{L^2}	e_{H^1}	ρ_{H^1}
1	40	0	4	8.36_{-1}	–	7.04_{-1}	–	8.71_{-1}	–
2	160	0	16	2.76_{-1}	0.78	2.28_{-1}	0.82	4.36_{-1}	0.50
3	520	0	52	5.17_{-2}	1.41	3.18_{-2}	1.67	1.21_{-1}	1.09
4	1,480	0	148	2.84_{-3}	2.77	2.48_{-3}	2.44	1.70_{-2}	1.87
5	5,040	0	504	2.65_{-4}	1.94	1.94_{-4}	2.08	2.22_{-3}	1.66
6	18,360	0	1,836	2.07_{-5}	1.97	1.38_{-5}	2.05	2.91_{-4}	1.57
7	70,360	0	7,036	1.46_{-6}	1.97	9.12_{-7}	2.02	3.75_{-5}	1.53
8	274,440	0	27,444	9.48_{-8}	2.01	5.86_{-8}	2.02	4.76_{-6}	1.52
9	1,083,720	0	108,372	5.96_{-9}	2.01	3.71_{-9}	2.01	5.99_{-7}	1.51
10	4,295,636	4	429,564	3.29_{-6}	−4.58	2.43_{-9}	0.31	5.17_{-6}	−1.56
11	17,104,144	296	1,710,444	7.78_{-6}	−0.62	1.63_{-8}	−1.38	5.80_{-5}	−1.75
12	68,259,328	1,272	6,826,060	2.37_{-5}	−0.80	6.00_{-8}	−0.94	2.23_{-4}	−0.97

matrix-block $(A_{i,i})$ within the global operator matrix can still be quite bad and we can expect the condition of the global operator matrix to suffer from that. Second, due to the patch elimination that is performed during the post-processing step the overall number of degrees of freedom is smaller in the post-processing case than in the stabilized case, even when we are considering the eliminated degrees of freedom during the local stabilization. Next we take a closer look at the approximation properties of the spaces resulting from the post-processed cover construction compared to the stabilized space. Errors and convergence rates for the stabilized case with polynomials of degree $p = 1$, $p = 3$ and $p = 5$ are given in Table 2.3, 2.4 and 2.5 respectively and are depicted in Figure 2.7. The results for the post-processed case are given in Table 2.6, 2.7 and 2.6 and are depicted in Figure 2.8. Let us first note that the prescribed solution u^* is smooth everywhere, even on the star-shaped domain at hand that has re-entrant corners and hence we expect the approximations u_k^{PU} on h-refined covers to converge with optimal rates. Looking at the results using the local stabilization but not the post-processing step we can see that we

Table 2.5: Relative errors $e.$, convergence rates $\rho.$ and number of eliminated degrees of freedom dof_{el} for Example 2.2 (star domain) employing the original cover construction **with local stabilization**, using a global polynomial degree $p = 5$.

k	dof	dof_{el}	N	e_{L^∞}	ρ_{L^∞}	e_{L^2}	ρ_{L^2}	e_{H^1}	ρ_{H^1}
1	84	0	4	4.60_{-1}	–	3.15_{-1}	–	5.51_{-1}	–
2	336	0	16	6.68_{-2}	1.38	5.67_{-2}	1.24	1.38_{-1}	1.00
3	1,092	0	52	2.05_{-3}	2.96	1.44_{-3}	3.12	6.32_{-3}	2.62
4	3,108	0	148	3.97_{-5}	3.77	3.33_{-5}	3.60	2.87_{-4}	2.96
5	10,556	28	504	7.31_{-7}	3.27	6.30_{-7}	3.25	1.12_{-5}	2.65
6	38,524	32	1,836	1.51_{-7}	1.22	1.09_{-8}	3.14	4.66_{-7}	2.46
7	147,580	176	7,036	4.19_{-7}	−0.76	8.77_{-9}	0.16	1.82_{-6}	−1.01
8	575,848	476	27,444	1.70_{-6}	−1.03	1.66_{-8}	−0.47	6.19_{-6}	−0.90
9	2,273,964	1,848	108,372	1.35_{-5}	−1.51	3.14_{-8}	−0.46	2.52_{-5}	−1.02

Table 2.6: Relative errors $e.$, convergence rates $\rho.$, number of eliminated patches N_{el} and number of shrinked patches N_{sh} for Example 2.2 (star domain) employing the **post-processed** cover construction without local stabilization, using a global polynomial degree $p = 1$.

k	dof	N	N_{el}	N_{sh}	e_{L^∞}	ρ_{L^∞}	e_{L^2}	ρ_{L^2}	e_{H^1}	ρ_{H^1}
1	12	4	0	4	1.02_{0}	–	9.42_{-1}	–	9.66_{-1}	–
2	48	16	0	8	1.02_{0}	-2.69_{-4}	7.02_{-1}	0.21	8.73_{-1}	7.33_{-2}
3	156	52	0	20	3.74_{-1}	0.85	2.82_{-1}	0.77	5.61_{-1}	0.37
4	432	144	4	48	1.42_{-1}	0.95	1.28_{-1}	0.78	4.10_{-1}	0.31
5	1,440	480	24	112	6.30_{-2}	0.67	4.43_{-2}	0.88	2.55_{-1}	0.39
6	5,412	1,804	32	244	2.14_{-2}	0.82	1.26_{-2}	0.95	1.38_{-1}	0.47
7	20,784	6,928	108	492	5.42_{-3}	1.02	3.35_{-3}	0.98	7.14_{-2}	0.49
8	81,696	27,232	212	992	1.50_{-3}	0.94	8.55_{-4}	1.00	3.62_{-2}	0.50
9	322,212	107,404	968	1,980	4.84_{-4}	0.82	2.15_{-4}	1.01	1.82_{-2}	0.50
10	1,286,208	428,736	828	3,976	9.95_{-5}	1.14	5.40_{-5}	1.00	9.14_{-3}	0.50
11	5,126,220	1,708,740	1,704	7,964	2.55_{-5}	0.98	1.35_{-5}	1.00	4.58_{-3}	0.50
12	20,467,944	6,822,648	3,412	15,964	6.64_{-6}	0.97	3.38_{-6}	1.00	2.29_{-3}	0.50

Table 2.7: Relative errors $e.$ and convergence rates $\rho.$, number of eliminated patches N_{el} and number of shrinked patches N_{sh} for Example 2.2 (star domain) employing the **post-processed** cover construction without local stabilization, using a global polynomial degree $p = 3$.

k	dof	N	N_{el}	N_{sh}	e_{L^∞}	ρ_{L^∞}	e_{L^2}	ρ_{L^2}	e_{H^1}	ρ_{H^1}
1	40	4	0	4	8.36_{-1}	–	7.04_{-1}	–	8.71_{-1}	–
2	160	16	0	8	2.44_{-1}	0.86	2.18_{-1}	0.85	4.16_{-1}	0.53
3	520	52	0	20	5.14_{-2}	1.32	3.17_{-2}	1.64	1.20_{-1}	1.06
4	1,440	144	4	48	2.84_{-3}	2.84	2.36_{-3}	2.55	1.64_{-2}	1.95
5	4,800	480	24	112	3.08_{-4}	1.84	1.98_{-4}	2.06	2.24_{-3}	1.65
6	18,040	1,804	32	244	2.57_{-5}	1.88	1.37_{-5}	2.02	2.90_{-4}	1.54
7	69,280	6,928	108	492	1.75_{-6}	1.99	9.15_{-7}	2.01	3.76_{-5}	1.52
8	272,320	27,232	212	992	1.15_{-7}	1.99	5.86_{-8}	2.01	4.76_{-6}	1.51
9	1,074,040	107,404	968	1,980	6.68_{-9}	2.07	3.71_{-9}	2.01	6.00_{-7}	1.51
10	4,287,360	428,736	828	3,976	4.58_{-10}	1.94	2.33_{-10}	2.00	7.53_{-8}	1.50
11	17,087,400	1,708,740	1,704	7,964	9.65_{-11}	1.13	1.72_{-11}	1.89	9.43_{-9}	1.50
12	68,226,480	6,822,648	3,412	15,964	1.83_{-10}	−0.46	1.78_{-10}	−1.69	1.22_{-9}	1.48

Table 2.8: Relative errors $e.$ and convergence rates $\rho.$, number of eliminated patches N_{el} and number of shrinked patches N_{sh} for Example 2.2 (star domain) employing the **post-processed** cover construction without local stabilization, using a global polynomial degree $p = 5$.

k	dof	N	N_{el}	N_{sh}	e_{L^∞}	ρ_{L^∞}	e_{L^2}	ρ_{L^2}	e_{H^1}	ρ_{H^1}
1	84	4	0	4	5.88_{-1}	–	3.15_{-1}	–	5.51_{-1}	–
2	336	16	0	8	6.90_{-2}	1.55	5.58_{-2}	1.25	1.36_{-1}	1.01
3	1,092	52	0	20	2.00_{-3}	3.01	1.40_{-3}	3.13	6.21_{-3}	2.62
4	3,024	144	4	48	4.14_{-5}	3.81	3.29_{-5}	3.68	2.84_{-4}	3.03
5	10,080	480	24	112	9.32_{-7}	3.15	6.30_{-7}	3.29	1.13_{-5}	2.68
6	37,884	1,804	32	244	1.90_{-8}	2.94	1.06_{-8}	3.08	3.87_{-7}	2.55
7	145,488	6,928	108	492	3.49_{-10}	2.97	1.72_{-10}	3.07	1.26_{-8}	2.54
8	571,872	27,232	212	992	6.08_{-12}	2.96	2.72_{-12}	3.03	4.01_{-10}	2.52
9	2,255,484	107,404	968	1,980	5.81_{-12}	3.3_{-2}	1.63_{-12}	0.37	2.64_{-11}	1.98

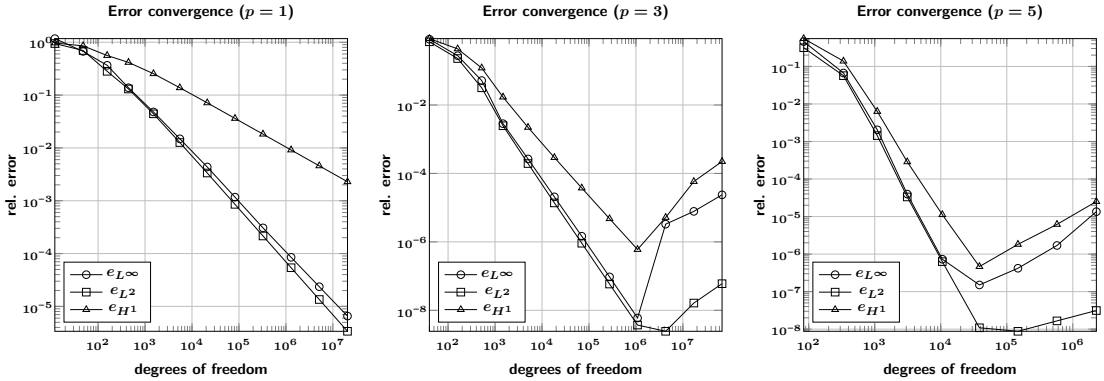

Figure 2.7: Error convergence for Example 2.2 (star domain) employing the original cover construction **with local stabilization**, using global polynomial degrees $p = 1$, $p = 3$ and $p = 5$ (left to right).

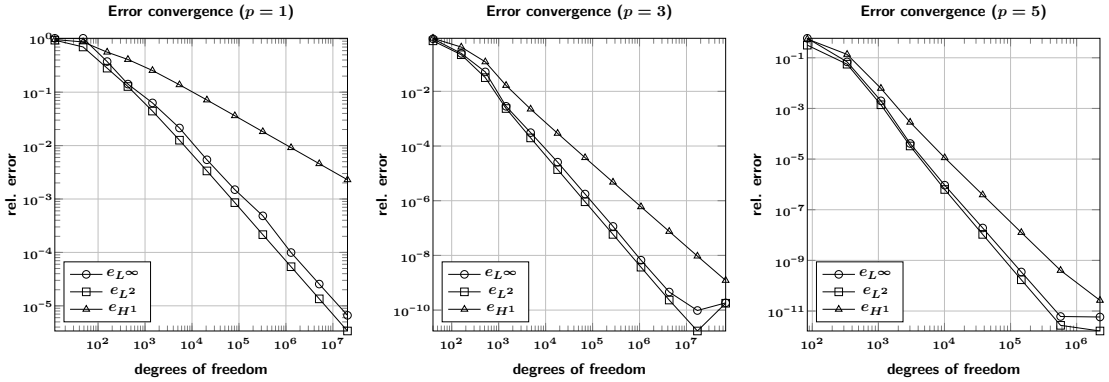

Figure 2.8: Error convergence for Example 2.2 (star domain) employing the **post-processed** cover construction without local stabilization, using global polynomial degrees $p = 1$, $p = 3$ and $p = 5$ (left to right).

Table 2.9: Number of solver iterations n until the residual has reached an absolute tolerance of 10^{-15} for Example 2.2 (star domain) with different types of transfer matrices. The employed solver is a preconditioned CG method using a multilevel-preconditioner with $\gamma := 1$, $\nu_k^{\text{pre}} := \nu_k^{\text{post}} := 1$ and a block factorized sparse approximate inverse (block-FSAI) smoother. Results for transfer according to (1.41) (GLOBAL-TO-LOCAL), according to (1.45) (LOCAL-TO-LOCAL) and according to (2.3) (MIXED-TO-LOCAL).

	GLOBAL-TO-LOCAL			LOCAL-TO-LOCAL			MIXED-TO-LOCAL		
k	$n_{p=1}$	$n_{p=3}$	$n_{p=5}$	$n_{p=1}$	$n_{p=3}$	$n_{p=5}$	$n_{p=1}$	$n_{p=3}$	$n_{p=5}$
3	2	2	2	2	2	2	2	2	2
4	8	8	11	8	10	17	8	10	17
5	10	8	11	12	12	18	12	12	18
6	14	10	11	16	16	22	15	14	21
7	14	11	11	22	21	31	15	14	24
8	14	10	11	27	24	34	15	14	26
9	14	10	11	35	29	42	15	15	28
10	16	11	12	130	117	131	16	14	28
11	15	10	–	70	58	–	16	14	–
12	15	10	–	99	77	–	16	14	–

obtain close to optimal rates $\rho_{L^2} = 1$ and $\rho_{H^1} = \frac{1}{2}$ for $p = 1$ on all levels $k \geq 6$. Note that for linear polynomials, no degrees of freedom have been eliminated anywhere and thus this is expected. But for $p = 3$ we can only obtain the optimal rates $\rho_{L^2} = 2$ and $\rho_{H^1} = \frac{3}{2}$ for $4 \leq k \leq 9$. Starting from level $k = 10$ the convergence does not only slow down but the errors do actually increase and keep increasing on all following levels. We should note that this happens as soon as the stable transformation starts to eliminate even just very few degrees of freedom. For $p = 5$ where the expected rates would be $\rho_{L^2} = 3$ and $\rho_{H^1} = \frac{5}{2}$ we can see the same problem starting at level $k = 7$. Additionally note that due to this effect the errors that we get on the final level $k = 9$ with polynomials of degree $p = 5$ are actually larger than the errors we get on the same level for $p = 3$. For the post-processed case we do not run into those problems and we can obtain the optimal convergence rates until the values of the relative errors e_{L^∞}, e_{L^2} and e_{H^1} reach $\approx 10^{-11}$ where rounding errors due to the limited machine precision make further reductions rather challenging.[1] To further illustrate those problems with the local stabilization we the errors $|u^* - u_k^{\text{PU}}|$ are depicted in Figure 2.9 for $p = 5$ on levels $k = 5$, $k = 6$ and $k = 7$, where the stable transformation starts to eliminate some degrees of freedom. We can clearly see that the error is evenly distributed over the whole domain on level $k = 5$ but starting from level $k = 6$ the overall error is dominated by some patches at the boundary of the domain. These are exactly those patches where some degrees of freedom have been eliminated due to small overlaps with the domain and thus the local approximation spaces $\tilde{V}_{i,k} := \text{span}\langle \tilde{\vartheta}_{i,k}^n \rangle$ after the stable transformation are not capable of representing all polynomials of degree up to $p = 5$ anymore. Again we can show that the post-processed cover does not suffer from these issues and the error keeps being distributed evenly over all patches.

[1] Experiments on simple square and cube domains show that relative errors $< 10^{-11}$ cannot be obtained there as well and the same limitations have been encountered in [106].

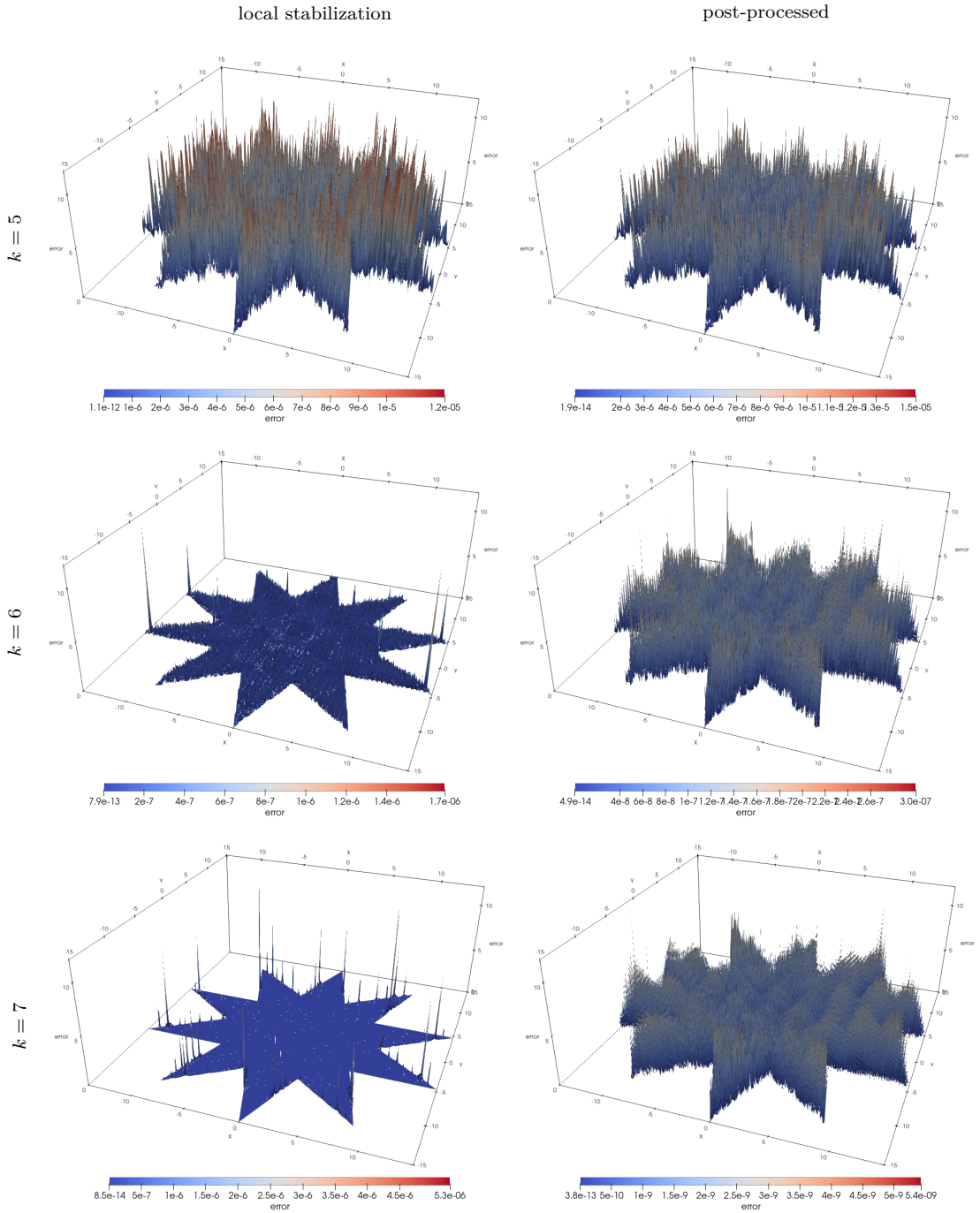

Figure 2.9: Warped contour plots of the errors $|u^* - u_k^{\mathrm{PU}}|$ using polynomials of degree $p = 5$ on levels $k = 5, 6, 7$ (top to bottom).
Left column: Applying local stabilization according to Section 1.5 without the patch shrinking post-processing step from Section 2.1. Starting on level 6 the errors for some patches on the boundary do not get reduced anymore.
Right column: Applying the cover post-processing step from Section 2.1. The errors are evenly distributed over all patches on all levels.

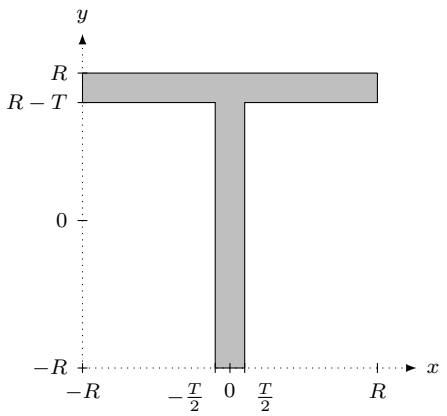

Figure 2.10: T-domain with radius $R = 15$ and thickness $T = 3$.

We finally demonstrate that the mixed multilevel transfer operators as given in (2.2) and (2.3) are capable of yielding a multilevel solver that requires a nearly constant number of iterations over all levels, even when the post-processing step did eliminate parent patches and hence Lemma 1.4 does not hold anymore. Table 2.9 depicts the number of iterations that were required to solve the problem with the different transfer types. As expected, the global-to-global transfer yields a very robust solver but it has the highest cost. On the other hand, the local-to-local transfer is not capable of retaining a constant number of iterations due to the patches with an eliminated parent not receiving any coarse level correction information anymore. Finally, the mixed-to-local transfer is again capable of retaining a constant number of iterations for polynomial degrees $p = 1$ and $p = 3$ and eventually for $p = 5$ as well. For lower polynomial degrees, the required number of iterations is very close to the number we retrieve by applying to global-to-global transfer, but the mixed-to-local transfer is both cheaper to construct as well as cheaper to apply. Here, it is additionally very important to note that it is far from self-evident that we can retrieve a similar number of iterations for different polynomial degrees on the same level. This does usually require the construction of sophisticated smoothers as has been demonstrated in [50]. The here employed block factorized sparse approximate inverse (block-FSAI) smoother has not yet been thoroughly analyzed with respect to this property.

Example 2.3 (2D T-domain)**.** In this example we again consider the Helmholtz type of equation from (2.6) with $c = 1$ but this time on a T-shaped polygon domain (compare Figure 2.10). Again we choose the right-hand side f and the Neumann boundary conditions g_{N} such that the solution is given by (2.7). Again we first want to take a look at the behavior of the linear solver when we try to solve the discretized equation systems from the original cover construction, with the local stabilization and with the post-processing step enabled. The results are shown in Table 2.10. If we compare those results to the ones from the previous example we can observe the following differences: First of all using the original cover construction does not only result in ill-conditioned operator matrices for high polynomial degrees of $p = 5$, but this already happens for cubic and even linear

Table 2.10: Number of solver iterations n until the residual has reached an absolute tolerance of 10^{-15} for Example 2.3 (T-domain) using the original hierarchical cover construction **without** post-processing and **without** local stabilization (DEFAULT), **without** post-processing and **with** local stabilization (STABLE TRAFO) and **with** post-processing but **without** local stabilization (POST-PROCESS) employing polynomials of degree $p = 1$, $p = 3$ and $p = 5$ in the construction of all local function spaces $V_{i,k}$. Cases where the solver did not reach the desired residual tolerance after 1000 iterations are marked with a dagger †. The employed solver is a preconditioned CG method using a multilevel-preconditioner with $\gamma := 1$, $\nu_k^{\text{pre}} := \nu_k^{\text{post}} := 1$, global-to-local transfer according to (1.41) and a block factorized sparse approximate inverse (block-FSAI) smoother.

	DEFAULT			STABLE TRAFO			POST-PROCESS		
k	$n_{p=1}$	$n_{p=3}$	$n_{p=5}$	$n_{p=1}$	$n_{p=3}$	$n_{p=5}$	$n_{p=1}$	$n_{p=3}$	$n_{p=5}$
1	3	1,000†	1,000†	2	1,000†	1,000†	2	2	2
2	1,000†	1,000†	1,000†	1,000†	1,000†	1,000†	7	11	14
3	6	1,000†	1,000†	6	15	1,000†	6	13	16
4	6	60	1,000†	6	12	17	6	12	16
5	9	15	1,000†	9	16	31	9	14	29
6	11	16	1,000†	11	17	36	12	14	27
7	12	14	1,000†	12	15	32	12	14	30
8	12	13	1,000†	13	14	33	12	13	30
9	12	15	1,000†	13	17	37	13	13	30
10	12	15	–	13	17	–	12	14	–
11	12	13	–	13	15	–	12	13	–
12	12	13	–	13	15	–	12	13	–

polynomials. Additionally the stable transformation is not capable of solving those issues on all levels. Especially on the lower levels $k = 1, 2, 3$ the linear solvers cannot converge to the desired residual tolerance. These observations can be explained by the fact that for this domain the violation of the strict flat-top property according to Definition 2.1 does actually lead to linear dependencies of the polynomials of neighboring patches and such global linear dependencies cannot be resolved by the local stable transformation. On the other hand, the post-processing step resolves both the local linear dependencies by shrinking boundary patches and global linear dependencies by guaranteeing the strict flat-top property and hence the equation systems on all levels can be solved efficiently.

Convergence results for the post-processed cover construction and employing polynomial degrees $p = 1$, $p = 3$ and $p = 5$ are given in Table 2.11, 2.12 and 2.13 respectively and are plotted in Figure 2.11. We can see that we are again able to obtain optimal rates for all polynomial degrees until the relative errors reach an absolute value of about 10^{-11}.

Example 2.4 (3D small stellated dodecahedron)**.** In this final example of the section, we show that the cover post-processing does not only work for two-dimensional domains, but does in fact extend to three-dimensions as well. To this end, we are solving (2.6) in $d = 3$ space dimensions with $c = 1$ on a small stellated dodecahedron domain as given in

Table 2.11: Relative errors $e.$ and convergence rates $\rho.$, number of eliminated patches N_{el} and number of shrinked patches N_{sh} for Example 2.3 (T-domain) employing the **post-processed** cover construction without local stabilization, using a global polynomial degree $p = 1$.

k	dof	N	N_{el}	N_{sh}	e_{L^∞}	ρ_{L^∞}	e_{L^2}	ρ_{L^2}	e_{H^1}	ρ_{H^1}
1	9	3	1	1	1.29_{0}	–	9.34_{-1}	–	8.91_{-1}	–
2	21	7	3	5	9.03_{-1}	0.42	7.69_{-1}	0.23	8.42_{-1}	6.58_{-2}
3	66	22	0	18	4.59_{-1}	0.59	2.06_{-1}	1.15	5.32_{-1}	0.40
4	180	60	0	40	1.44_{-1}	1.16	9.66_{-2}	0.75	3.74_{-1}	0.35
5	636	212	28	84	6.47_{-2}	0.63	4.41_{-2}	0.62	2.61_{-1}	0.28
6	2,370	790	114	170	1.78_{-2}	0.98	1.37_{-2}	0.89	1.44_{-1}	0.45
7	9,822	3,274	0	342	5.03_{-3}	0.89	3.47_{-3}	0.96	7.32_{-2}	0.48
8	37,908	12,636	0	688	1.33_{-3}	0.98	9.02_{-4}	1.00	3.76_{-2}	0.49
9	150,252	50,084	460	1,380	3.47_{-4}	0.98	2.28_{-4}	1.00	1.90_{-2}	0.50
10	598,242	199,414	1,842	2,762	8.84_{-5}	0.99	5.75_{-5}	1.00	9.52_{-3}	0.50
11	2,398,494	799,498	0	5,526	2.23_{-5}	0.99	1.43_{-5}	1.00	4.75_{-3}	0.50
12	9,571,860	3,190,620	0	11,056	5.63_{-6}	1.00	3.58_{-6}	1.00	2.38_{-3}	0.50

Table 2.12: Relative errors $e.$ and convergence rates $\rho.$, number of eliminated patches N_{el} and number of shrinked patches N_{sh} for Example 2.3 (T-domain) employing the **post-processed** cover construction without local stabilization, using a global polynomial degree $p = 3$.

k	dof	N	N_{el}	N_{sh}	e_{L^∞}	ρ_{L^∞}	e_{L^2}	ρ_{L^2}	e_{H^1}	ρ_{H^1}
1	30	3	1	1	5.52_{-1}	–	5.32_{-1}	–	7.14_{-1}	–
2	70	7	3	5	3.50_{-1}	0.54	2.66_{-1}	0.82	4.72_{-1}	0.49
3	220	22	0	18	6.60_{-2}	1.46	5.49_{-2}	1.38	1.52_{-1}	0.99
4	600	60	0	40	3.93_{-3}	2.81	3.51_{-3}	2.74	2.08_{-2}	1.98
5	2,120	212	28	84	5.49_{-4}	1.56	2.57_{-4}	2.07	2.95_{-3}	1.55
6	7,900	790	114	170	2.51_{-5}	2.35	1.38_{-5}	2.22	3.45_{-4}	1.63
7	32,740	3,274	0	342	1.14_{-6}	2.17	6.34_{-7}	2.17	3.80_{-5}	1.55
8	126,360	12,636	0	688	7.10_{-8}	2.06	3.95_{-8}	2.06	4.77_{-6}	1.54
9	500,840	50,084	460	1,380	7.07_{-9}	1.68	2.54_{-9}	1.99	6.04_{-7}	1.50
10	1,994,140	199,414	1,842	2,762	3.98_{-10}	2.08	1.59_{-10}	2.01	7.57_{-8}	1.50
11	7,994,980	799,498	0	5,526	7.28_{-11}	1.22	4.90_{-11}	0.85	9.39_{-9}	1.50
12	31,906,200	3,190,620	0	11,056	6.05_{-11}	0.13	4.71_{-11}	2.92_{-2}	1.17_{-9}	1.50

Table 2.13: Relative errors $e.$ and convergence rates $\rho.$, number of eliminated patches N_{el} and number of shrinked patches N_{sh} for Example 2.3 (T-domain) employing the **post-processed** cover construction without local stabilization, using a global polynomial degree $p = 5$.

k	dof	N	N_{el}	N_{sh}	e_{L^∞}	ρ_{L^∞}	e_{L^2}	ρ_{L^2}	e_{H^1}	ρ_{H^1}
1	63	3	1	1	5.17_{-1}	–	5.13_{-1}	–	6.73_{-1}	–
2	147	7	3	5	9.48_{-2}	2.00	9.80_{-2}	1.95	1.98_{-1}	1.44
3	462	22	0	18	2.23_{-3}	3.27	1.20_{-3}	3.85	7.17_{-3}	2.90
4	1,260	60	0	40	4.00_{-5}	4.01	3.19_{-5}	3.61	3.15_{-4}	3.11
5	4,452	212	28	84	2.21_{-6}	2.30	1.17_{-6}	2.62	1.79_{-5}	2.27
6	16,590	790	114	170	3.48_{-8}	3.15	2.26_{-8}	3.00	6.51_{-7}	2.52
7	68,754	3,274	0	342	6.34_{-10}	2.82	3.71_{-10}	2.89	2.09_{-8}	2.42
8	265,356	12,636	0	688	1.19_{-11}	2.94	6.35_{-12}	3.01	6.88_{-10}	2.53
9	1,051,764	50,084	460	1,380	2.66_{-12}	1.09	2.18_{-12}	0.77	2.22_{-11}	2.49

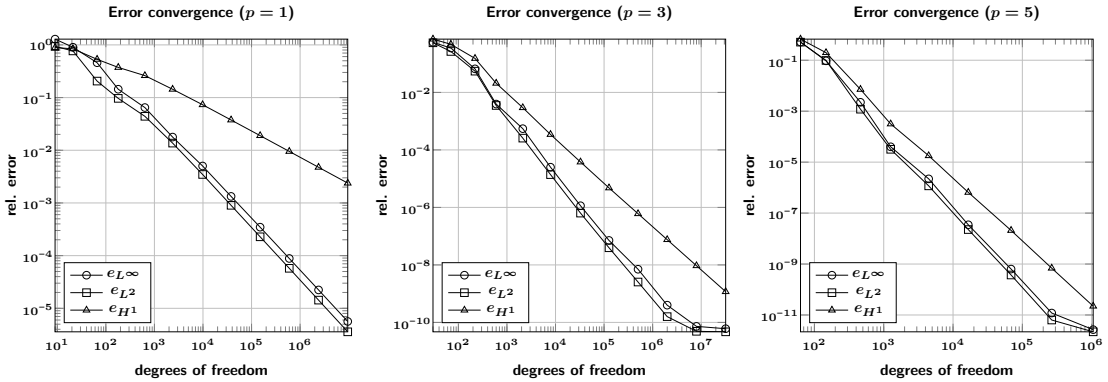

Figure 2.11: Error convergence for Example 2.3 (T-domain) employing the **post-processed** cover construction without local stabilization, using global polynomial degrees $p = 1$, $p = 3$ and $p = 5$ (left to right).

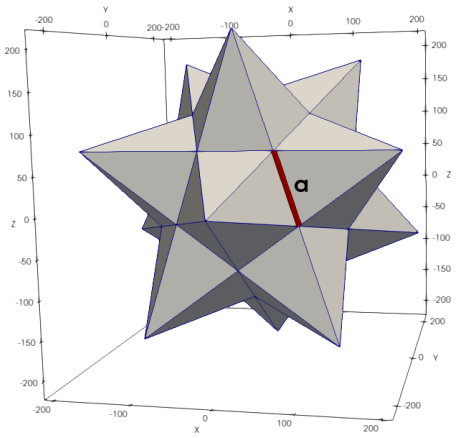

Figure 2.12: Small stellated dodecahedron domain with edge length $a = 100$.

Figure 2.12. This time the reference solution is given by

$$u^* := \lambda x \cos(\lambda y) + \lambda y \sin(\lambda x) + \lambda(x + y) \sin(\lambda z) \tag{2.8}$$

with $\lambda = 0.1$. The convergence results for the post-processed cover construction using polynomial degrees $p = 1, 3, 5$ are presented in Table 2.14, 2.15 and 2.16 and are depicted in Figure 2.13. We are again able to obtain optimal convergence rates. In the three-dimensional case those are given by $\rho_{L^2} = \frac{2}{3}$ and $\rho_{H^1} = \frac{1}{3}$ for $p = 1$, $\rho_{L^2} = \frac{4}{3}$ and $\rho_{H^1} = 1$ for $p = 3$ and $\rho_{L^2} = 2$ and $\rho_{H^1} = \frac{5}{3}$ for $p = 5$.

Table 2.14: Relative errors $e.$ and convergence rates $\rho.$, number of eliminated patches N_{el} and number of shrinked patches N_{sh} for Example 2.4 (stellated dodecahedron) employing the **post-processed** cover construction without local stabilization, using a global polynomial degree $p = 1$.

k	dof	N	N_{el}	N_{sh}	e_{L^∞}	ρ_{L^∞}	e_{L^2}	ρ_{L^2}	e_{H^1}	ρ_{H^1}
1	32	8	0	8	9.32_{-1}	–	9.45_{-1}	–	9.64_{-1}	–
2	112	28	4	20	1.04_{0}	-8.85_{-2}	7.76_{-1}	0.16	9.52_{-1}	9.84_{-3}
3	496	124	20	72	5.30_{-1}	0.45	2.95_{-1}	0.65	7.61_{-1}	0.15
4	2,740	685	115	269	1.60_{-1}	0.70	1.14_{-1}	0.56	5.28_{-1}	0.21
5	17,584	4,396	272	1,048	7.05_{-2}	0.44	3.76_{-2}	0.60	3.06_{-1}	0.29
6	120,848	30,212	2,784	3,784	2.05_{-2}	0.64	1.08_{-2}	0.64	1.61_{-1}	0.33
7	919,184	229,796	10,696	13,480	8.81_{-3}	0.42	2.83_{-3}	0.66	8.19_{-2}	0.33
8	7,167,408	1,791,852	41,760	53,200	1.95_{-3}	0.73	7.17_{-4}	0.67	4.12_{-2}	0.33
9	56,717,600	14,179,400	134,968	241,680	6.72_{-4}	0.52	1.80_{-4}	0.67	2.07_{-2}	0.33

Table 2.15: Relative errors $e.$ and convergence rates $\rho.$, number of eliminated patches N_{el} and number of shrinked patches N_{sh} for Example 2.4 (stellated dodecahedron) employing the **post-processed** cover construction without local stabilization, using a global polynomial degree $p = 3$.

k	dof	N	N_{el}	N_{sh}	e_{L^∞}	ρ_{L^∞}	e_{L^2}	ρ_{L^2}	e_{H^1}	ρ_{H^1}
1	160	8	0	8	1.44_{0}	–	5.92_{-1}	–	8.37_{-1}	–
2	560	28	4	20	4.35_{-1}	0.95	2.34_{-1}	0.74	5.70_{-1}	0.31
3	2,480	124	20	72	1.59_{-1}	0.68	3.55_{-2}	1.27	2.02_{-1}	0.70
4	13,700	685	115	269	1.18_{-2}	1.52	2.50_{-3}	1.55	2.86_{-2}	1.14
5	87,920	4,396	272	1,048	9.47_{-4}	1.36	1.89_{-4}	1.39	3.47_{-3}	1.13
6	604,240	30,212	2,784	3,784	5.72_{-5}	1.46	1.59_{-5}	1.29	4.30_{-4}	1.08
7	4,595,920	229,796	10,696	13,480	6.58_{-6}	1.07	1.21_{-6}	1.27	5.26_{-5}	1.04
8	35,837,040	1,791,852	41,760	53,200	3.90_{-7}	1.38	8.37_{-8}	1.30	6.57_{-6}	1.01

Table 2.16: Relative errors $e.$ and convergence rates $\rho.$, number of eliminated patches N_{el} and number of shrinked patches N_{sh} for Example 2.4 (stellated dodecahedron) employing the **post-processed** cover construction without local stabilization, using a global polynomial degree $p = 5$.

k	dof	N	N_{el}	N_{sh}	e_{L^∞}	ρ_{L^∞}	e_{L^2}	ρ_{L^2}	e_{H^1}	ρ_{H^1}
1	448	8	0	8	8.44_{-1}	–	1.51_{-1}	–	3.47_{-1}	–
2	1,568	28	4	20	2.83_{-1}	0.87	6.45_{-2}	0.68	2.30_{-1}	0.33
3	6,944	124	20	72	2.23_{-2}	1.71	2.07_{-3}	2.31	1.29_{-2}	1.93
4	38,360	685	115	269	3.25_{-4}	2.47	5.09_{-5}	2.17	5.89_{-4}	1.81
5	246,176	4,396	272	1,048	4.80_{-6}	2.27	1.03_{-6}	2.10	2.14_{-5}	1.78
6	1,691,872	30,212	2,784	3,784	5.86_{-8}	2.28	1.96_{-8}	2.05	7.14_{-7}	1.76
7	12,868,576	229,796	10,696	13,480	1.69_{-9}	1.75	3.35_{-10}	2.01	2.29_{-8}	1.70

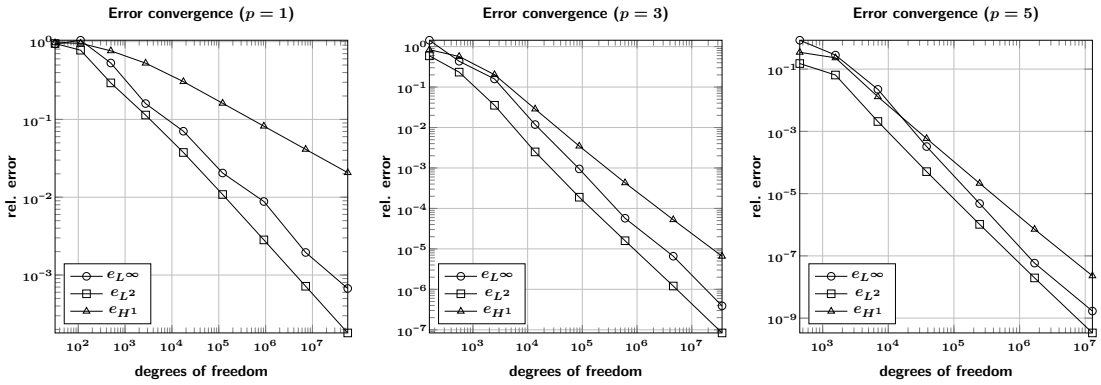

Figure 2.13: Error convergence for Example 2.4 (stellated dodecahedron) employing the **post-processed** cover construction without local stabilization, using global polynomial degrees $p = 1$, $p = 3$ and $p = 5$ (left to right).

3 Geometry Representation

To perform numerical simulations on real-world geometries we need to decide how these geometries are to be represented. There are many different ways to describe geometries and the choice of the representation format heavily influences how complex the geometries we can support will be. Within the industry and especially within the fields of Computer-Aided Design (CAD) and Computer-Aided Engineering (CAE) there have been efforts to standardize how geometries are to be represented and how to exchange those representations between different software programs involved in the product's development cycle. The predominant result of those efforts is the *STandard for the Exchange of Product model data (STEP)* described in the ISO standards document *ISO 10303*. This standard consists of many parts dealing with different tasks during the development process from the initial design, engineering, manufacturing up to maintenance. In this thesis we are most interested in *Part 42 - Geometric and topological representation* [62] of the STEP standard. That part describes the *Boundary Representation (BREP)* which we summarize in this chapter and that is used by the most common CAD kernels like ParaSolid, ACIS, C3D, OpenCASCADE and many more.

3.1 Boundary Representation

The *Boundary Representation (BREP or B-Rep)* describes geometries in two and three dimensions by their limits. I.e. a volumetric domain $\Omega \subset \mathbb{R}^d$ with $d \in \{2, 3\}$ is not described directly, but instead a description for its (oriented) boundary $\partial\Omega$ is provided and the volume is then defined as all points that lie within that boundary description. It is a very powerful representation designed with the goal to be able to describe all geometries that can be physically manufactured. Following the definition from part 42 of the ISO 10303 standard [62] the description of BREPs is split into two parts, the *topology* (mainly represented by *vertices*, *edges*, *faces*, *shells* and *solids*) and the *geometry* (mainly represented by *points*, *curves* and *surfaces*).
The topology defines how the basic entities are connected with each other. Edges are connected by vertices and form *paths* or *loops*, faces are connected along edges and form *shells*. Shells that are closed (there is no edge that is connected to only a single face) can

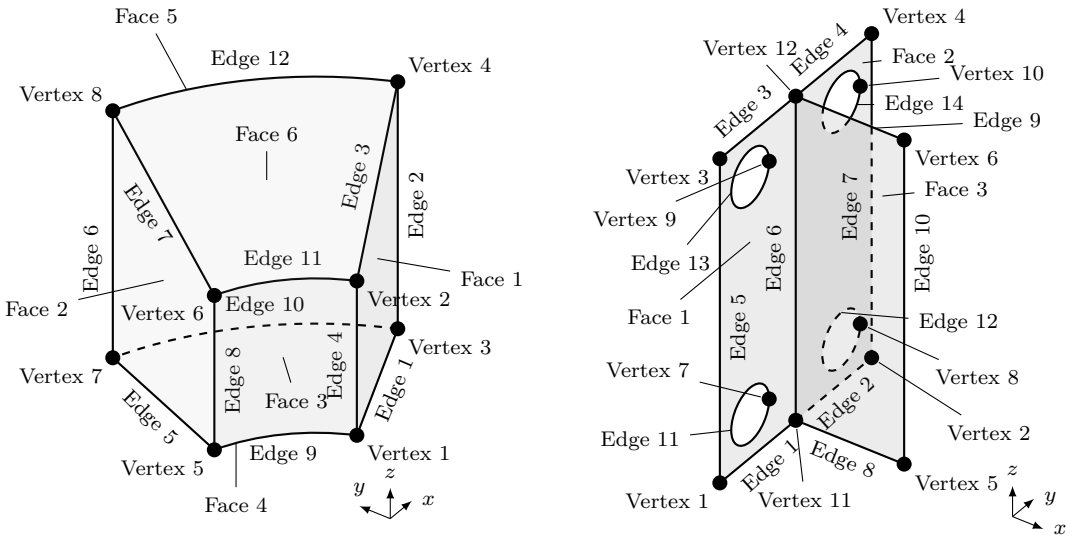

Figure 3.1: Exemplary BREP topologies.
Left: A cylindrical wedge solid model, represented by a *closed* shell consisting of six faces. Each edge is connected to exactly two faces.
Right: A T-joint shell model consisting of three faces. The edge 6 is connected to three faces and thus making it a *non-manifold* shell. All other edges are connected to a single face only, thus making it an *open* shell.

be used to define a *solid*. Any solid is required to consist of at least one *outer* shell, but can have additional *inner* shells that describe voids. Those inner shells need to be fully enclosed in the outer shell and cannot be nested. In general shells can describe either manifold or non-manifold entities, depending on whether an edge is allowed to connect more than two faces with each other or not. While only closed manifold shells can be used to define a solid, open or non-manifold shells are often used to describe thin geometries where plate theory can be applied for the numerical simulations (compare Figure 3.1 for some exemplary models described by BREPs). Note that STEP forbids faces from a shell to intersect anywhere except along edges or vertices that connect them, even if they categorize as non-manifold shells. This includes self-intersections.
While the topology describes the model's structure, the geometry is used to describe its actual shape. To this end each topological entity is assigned some geometrical representation. Vertices are represented by points, edges by curves and faces by surfaces. All geometric objects are defined through a parametrization. A curve $\boldsymbol{C}$ in two or three dimensions is defined by a function

$$\boldsymbol{C} : \mathcal{P}_{\boldsymbol{C}} \to \mathbb{R}^d, \quad u \mapsto \boldsymbol{x} \tag{3.1}$$

with $\mathcal{P}_{\boldsymbol{C}} := (u_a, u_b) \subseteq \mathbb{R}$ and $d \in \{2, 3\}$. Similarly a surface $\boldsymbol{S}$ is defined by

$$\boldsymbol{S} : \mathcal{P}_{\boldsymbol{S}} \to \mathbb{R}^3, \quad (u, v) \mapsto \boldsymbol{x} \tag{3.2}$$

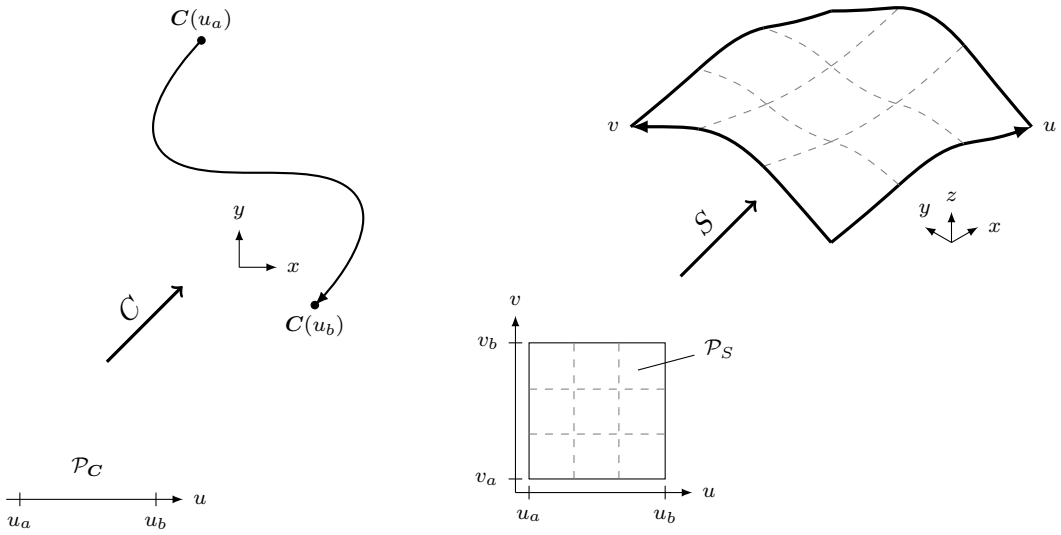

Figure 3.2: *Left:* A curve $\boldsymbol{C}$ maps from the one-dimensional interval $\mathcal{P}_{\boldsymbol{C}} := (u_a, u_b) \subset \mathbb{R}$ into the two dimensional world space $\mathbb{R}^2$.
Right: A surface $\boldsymbol{S}$ maps from the two-dimensional tensor product domain $\mathcal{P}_{\boldsymbol{S}} := (u_a, u_b) \times (v_a, v_b) \subset \mathbb{R}^2$ into the three dimensional world space $\mathbb{R}^3$.

with $\mathcal{P}_{\boldsymbol{S}} := (u_a, u_b) \times (v_a, v_b) \subseteq \mathbb{R}^2$ (compare Figure 3.2). The STEP standard describes multiple entities to define such parametrized curves and surfaces. The most fundamental of those entities are *elementary* representations for curves (e.g. lines or circles) and surfaces (e.g. planes, spheres, cylindrical, conical or toroidal surfaces). These representations usually describe infinite domains i.e. any of u_a, u_b, v_a or v_b can be $\pm\infty$. Additionally note that they can be periodic. E.g., while the parameter domain $\mathcal{P}_{\boldsymbol{C}}$ of a circular curve is unbounded, a single turn of such a curve spans over an interval of length 2π. The other of the most common representations are *bounded* curves and surface that are mainly described by *non-uniform rational B-splines (NURBS)* which will be introduced in more detail in Section 3.2. For surfaces there is the additional category of *swept* surfaces, like linear extrusions of curves along a direction vector or surfaces of revolution. Those swept surfaces as well as all elementary representations can be converted to NURBS curves and surfaces that represent the same geometric objects, as long as the infinite surfaces are restricted to a finite part of the parameter domains $\mathcal{P}_{\boldsymbol{C}}$ and $\mathcal{P}_{\boldsymbol{S}}$ [94]. Hence, without loss of generality, we assume that all curves and surfaces are described by NURBS throughout the remainder of this thesis.[1]

If we consider our current description of curves and surface and compare that to the

[1]Since edition 5 (2014) of ISO 10303-42, the STEP standard does additionally support *(rational) locally refined spline* curves, surfaces and volumes. These have been introduced to better support *Isogeometric Analysis (IGA)* and can represent things commonly known as *hierarchical B-splines* [40], *T-splines* [117] or *LR B-splines* [30]. To the best of our knowledge they are still unsupported by most CAD kernels and hence are not of importance for this thesis.

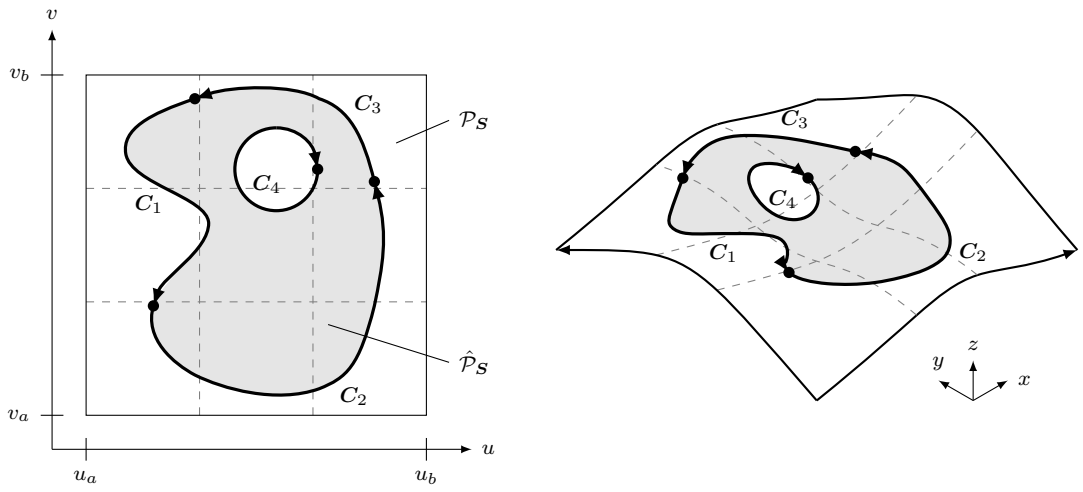

Figure 3.3: *Left:* The parameter domain $\mathcal{P}_{\boldsymbol{S}} := (u_a, u_b) \times (v_a, v_b)$ of a surface $\boldsymbol{S}$ is restricted to $\hat{\mathcal{P}}_{\boldsymbol{S}}$ where $\partial\hat{\mathcal{P}}_{\boldsymbol{S}}$ is represented by two rings $\{R_1, R_2\}$. The counter-clock-wise oriented outer ring R_1 consists of the curves $\boldsymbol{C}_1$, $\boldsymbol{C}_2$ and $\boldsymbol{C}_3$. The clock-wise oriented inner ring R_2 consists of a single curve $\boldsymbol{C}_3$ and defines a hole.
Right: The restricted parameter domain transfers onto the surface $\boldsymbol{S}$ and represents a trimmed domain of the full tensor-product surface in the world space.

example geometries given in Figure 3.1 it is worth to notice that it would not be possible to build the example geometries with the given geometric object descriptions yet. E.g. planes and cylindrical surfaces are elementary and thus have an infinite extend. Additionally all other surfaces have a tensor product shape and there is no way to express holes like in the shell example on the right in Figure 3.1. To overcome those limitations STEP allows to restrict the parameter domains and define $\hat{\mathcal{P}}_{\boldsymbol{C}} \subseteq \mathcal{P}_{\boldsymbol{C}}$ and $\hat{\mathcal{P}}_{\boldsymbol{S}} \subseteq \mathcal{P}_{\boldsymbol{S}}$, which results in so-called *trimmed* or *bounded* curves and surfaces. While the trimming of curves is simple, since we just need to pick any interval $\hat{\mathcal{P}}_{\boldsymbol{C}} := (\hat{u}_a, \hat{u}_b)$ that is defined by the two scalar values $\hat{u}_a, \hat{u}_b \in \bar{\mathcal{P}}_{\boldsymbol{C}}$ only, the trimming of surfaces is a lot more involved. The parameter domain bounds for surfaces are provided by a two-dimensional boundary representation and hence we have given a description for $\partial\hat{\mathcal{P}}_{\boldsymbol{S}}$ instead of $\hat{\mathcal{P}}_{\boldsymbol{S}}$ being given directly. The boundary $\partial\hat{\mathcal{P}}_{\boldsymbol{S}}$ is formed by closed rings consisting of curves, similar to how a solid is formed by closed shells consisting of faces (compare Figure 3.3). To this end the bounds are defined by a set of *rings* (also called *loops*) $\{R_1, \ldots, R_k\}$ with $k \geq 1$, where R_1 is the counter-clock-wise oriented outer ring and $\{R_2, \ldots, R_k\}$ are optional clock-wise oriented inner rings that describe the boundary of holes. It is then

$$\partial\hat{\mathcal{P}}_{\boldsymbol{S}} := \bigcup_{i=1}^{k} \bar{R}_i.$$

Each ring R_i is not allowed to intersect any other ring R_j. Note that this also disallows self-intersection in the case $j = i$. A ring itself consists of a set of one or more two-dimensional

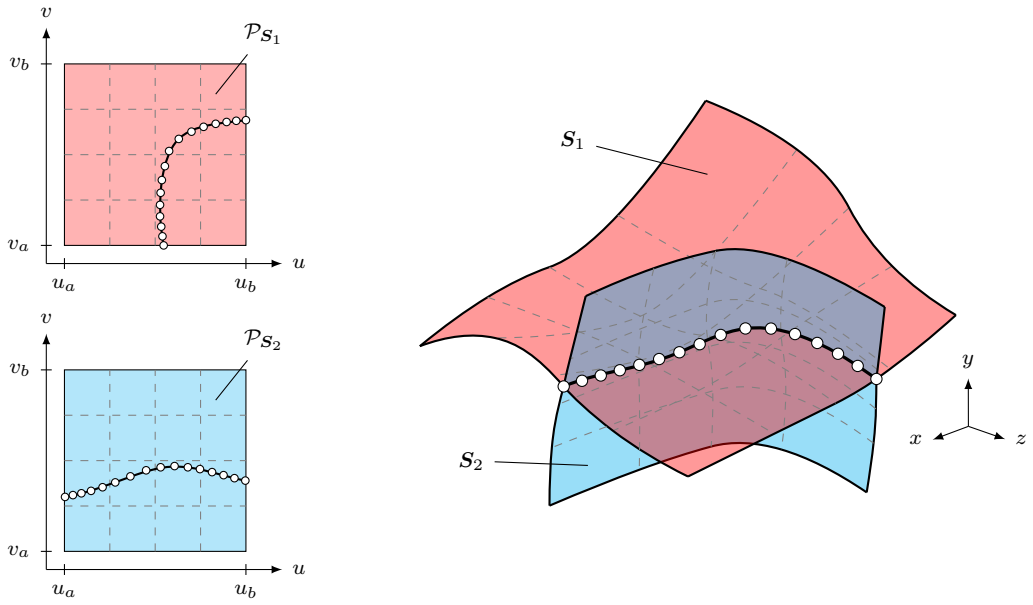

Figure 3.4: Intersection of two surfaces $\boldsymbol{S}_1$ and $\boldsymbol{S}_2$. The intersection curve has two-dimensional representations in each of the surfaces parameter domains $\mathcal{P}_{\boldsymbol{S}_1}$ and $\mathcal{P}_{\boldsymbol{S}_2}$ (left) and another three-dimensional representation in the world-space (right).

curves $\{\boldsymbol{C}_1, \ldots, \boldsymbol{C}_n\}$ with $n \geq 1$ that are connected and closed, i.e.

$$\boldsymbol{C}_i(u_{b,i}) = \boldsymbol{C}_j(u_{a,j}) \quad \text{with} \quad j = (i \bmod n) + 1 \qquad \forall i = 1, \ldots, n. \tag{3.3}$$

Note that the parameter bounds description above actually encloses a connected two-dimensional volume and thus is additionally used to represent the simulation domain Ω for two-dimensional simulation problems.
To understand the implications that trimmed surfaces have on the overall geometric objects and the numerical simulations to be performed on those geometries it is necessary to look at the procedures that are performed during the modelling of the geometries that lead to those trimmed surfaces. One of the most common ways to create BREP geometries in CAD is by incorporating *constructive solid geometry (CSG)* ideas into the build process. I.e. new BREP objects can be constructed by applying *boolean operations* (in the sense of set-theoretical *union*, *intersection* and *difference*) of two or more input BREP models [98]. To perform those operations one of the most fundamental steps is to find the intersection curves of two surfaces. Those intersection curves are then used to describe the trimming curves of a newly created trimmed surface of the resulting BREP geometry. Unfortunately the computation and representation of those intersection curves turns out to be a problem that is hard to solve in general. A famously cited example by Sederberg et. al. shows that the intersection of bi-cubic surfaces patches results in a curve of degree 324 [113]. Those high degrees of the intersection curves make exact representations of them unsuitable in practice and the common solution is to use low degree approximations

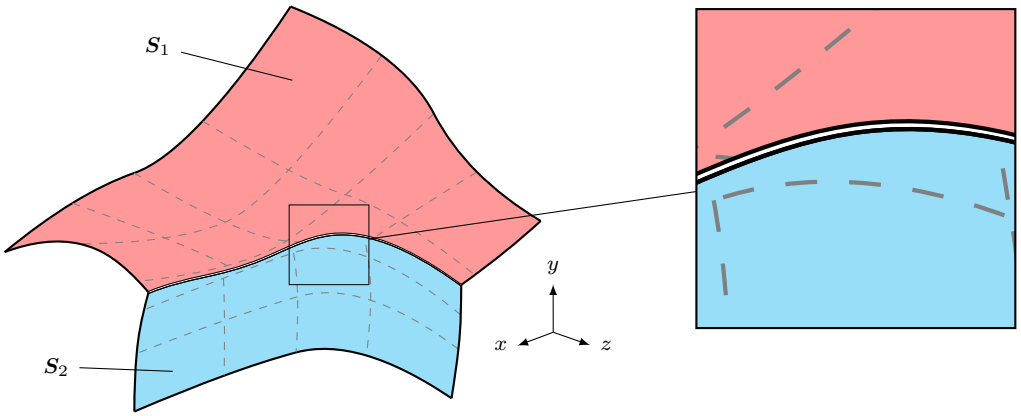

Figure 3.5: The surfaces $\boldsymbol{S}_1$ and $\boldsymbol{S}_2$ have been trimmed at their intersecting curve. Due to the separate approximations of the intersection curve in each of the parameter domains, the surfaces do not match exactly along the intersection curve in the world space. A *gap* can be seen between the surfaces.

of those curves instead. There are many different methods to compute the intersection curves and an interested reader is referred to [91] and [92] that give good overviews over those methods. Pretty much all of those methods end up presenting a set of sampling points in the parameter space of one (or both) of the surfaces that are then used to form low degree curves and need to be transferred into the world space (compare Figure 3.4). The exact representation of a curve of degree d on a tensor product surface patch of degrees m and n is of a degree $d(m+n)$ which again leads to the use of an approximate representation of the three-dimensional curve in the world space in practice [97, 139]. A fundamental problem that arises from those approximate representations is that the three geometrical representations of an edge (one in each of the parameter spaces and the one in the world space) do usually not match exactly. Since it is common that the approximation or interpolation points used to create the distinct curve representations are not the same, it is not even guaranteed that the curve representations agree on a finite set of points. Quite the contrary, it is even allowed and very common that a STEP file that stores a BREP geometry does only contain a single geometric representation for each edge (either a parametric representation or the three-dimensional one) and the other representations are re-created when the file is read or on-demand when required. Hence, trimmed surfaces usually lead to BREP representations that are called *non-watertight* which means that there are (small) gaps between neighboring surfaces (compare Figure 3.5). In fact, this non-matching property does not only apply to surfaces connected over edges, but applies to the connection of two consecutive trim curves as well. I.e. equation (3.3) that states that consecutive curves need to agree at their endpoints actually becomes

$$\|\boldsymbol{C}_i(u_{b,i}) - \boldsymbol{C}_j(u_{a,j})\| \leq \epsilon \quad \text{with} \quad j = (i \bmod n) + 1 \qquad \forall i = 1, \ldots, n.$$

in practice, where ϵ is a tolerance that depends on the tolerances that have been used during the approximate construction of the involved trim curves.

3.2 Non-Uniform Rational B-Splines

Since NURBS are used to describe the curves and surfaces within a BREP we will summarize the mathematical theory behind NURBS while following [94].
A NURBS curve $\boldsymbol{C}$ of degree p in d-dimensions is defined for $u \in [u_a, u_b]$, with $u_a, u_b \in \mathbb{R}$ by

$$\boldsymbol{C}(u) = \frac{\sum_{i=0}^{n} N_{i,p}(u) w_i \boldsymbol{P}_i}{\sum_{i=0}^{n} N_{i,p}(u) w_i} \tag{3.4}$$

where $\{\boldsymbol{P}_i\}$ are the d-dimensional *control points*, $\{w_i\}$ are real *weights* and $\{N_{i,p}\}$ are the *B-spline basis functions* of degree p. The basis functions $N_{i,p}$ are defined recursively by

$$\begin{aligned} N_{i,0}(u) &= \begin{cases} 1, & \text{if } u_i \leq u \leq u_{i+1} \\ 0, & \text{otherwise} \end{cases} \\ N_{i,p}(u) &= \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u) \end{aligned} \tag{3.5}$$

with $U := \{u_0, \ldots, u_{n+p}\}$ being the so-called *knot vector*. We require the knot vector to be non-decreasing, i.e. $u_i \leq u_{i+1}$. The number of consecutive knots that are equal to a knot u_i is called the *knot multiplicity* of u_i and can be defined as $k_i := \text{card}(\{u_j \mid u_j = u_i\})$. We additionally require that $k_i \leq p+1$ for the knots u_0 and u_{n+p} and $k_i \leq p$ for all inner knots $u_i \in \{u \in U \mid u \neq u_0 \text{ and } u \neq u_{n+p}\}$, which guarantees that the curve is at least C^0 continuous for $u \in [u_0, u_{n+p}]$. In general a NURBS curve is C^{p-k_i} continuous at the knots u_i. If the multiplicities k_0 or k_{n+p} are equal to $p+1$ the curve is called *clamped* at any of those ends. A curve that is clamped at its beginning is discontinuous at u_0 and it interpolates the first control point at that location, i.e. $\boldsymbol{C}(u_0) = \boldsymbol{P}_0$. Respectively it is $\boldsymbol{C}(u_{n+p}) = \boldsymbol{P}_n$ for a curve that is clamped at its end.
Similarly a NURBS surface $\boldsymbol{S}$ of degree p in the u-direction and degree q in the v-direction is defined by

$$\boldsymbol{S}(u) = \frac{\sum_{i=0}^{n} \sum_{j=0}^{m} N_{i,p}(u) N_{j,q}(v) w_{i,j} \boldsymbol{P}_{i,j}}{\sum_{i=0}^{n} \sum_{j=0}^{m} N_{i,p}(u) N_{j,q}(v) w_{i,j}}$$

where the $\{\boldsymbol{P}_{i,j}\}$ are the three-dimensional control points, $\{w_{i,j}\}$ are real weights and $\{N_{i,p}\}$ and $\{N_{j,q}\}$ are the basis functions from (3.5) and are utilizing the knot vectors $U := \{u_0, \ldots, u_{n+p}\}$ and $V := \{v_0, \ldots, v_{m+q}\}$ respectively.
Throughout this thesis we assume that all curve weights $\{w_i\}$ and surface weights $\{w_{i,j}\}$ are positive which is a restriction that the STEP standard imposes on all curves and surfaces it supports as well.

4 Exact Geometry & Integration (2D)

In this chapter we are concerned with the treatment of two-dimensional domains Ω for the PUM that are represented by bounds like the trimming regions of faces described in Chapter 3, especially on the left of Figure 3.3. Here, we want to point out that a domain with $\Omega \subset \mathbb{R}^2$ does not necessarily mean that we have a problem at hand that would commonly be categorized as being two-dimensional. On the contrary, many of the practically relevant problems that we will be able to solve with the methods presented throughout this chapter are so-called *shell problems* where a three-dimensional domain that consists of thin geometric structures is represented by its *mid-surface* only. Given some assumption about the physical behavior of those problems, the three-dimensional PDEs can be transformed into problems that are solved in the two-dimensional parametrizations of those mid-surfaces. An example with a more detailed mathematical description for such a shell problem will be given in Example 4.4.
In the PUM, the part that is most affected by the geometry is the assemble step. There, integrals over parts of Ω and its boundary $\partial\Omega$ need to be evaluated as shown in Section 1.3: The intersections of a patch ω_i and its neighbors ω_j have been split into integration domains $\mathcal{D}_i^m$ that resolve all discontinuities and kinks of the employed basis functions and that are axis-aligned bounding boxes. If ω_i does not intersect the boundary, those integration domains are already sufficient. But in general we need to evaluate integrals over $\mathcal{D}_i^m \cap \Omega$ and $\mathcal{D}_i^m \cap \partial\Omega$. Here, the idea is to split the intersections $\mathcal{D}_i^m \cap \Omega$ into smaller *integration cells* like triangles (in 2D) or tetrahedra (in 3D) where we can employ known Gaussian quadrature rules afterwards. Hence, we need to solve the following tasks:

1. Compute the intersection between the integration domains $\mathcal{D}_i^m$ and the simulation domain Ω.

2. Create a **local** splitting of that intersection domain into simpler integration cells like triangles or tetrahedra.

3. Perform Gaussian quadrature on those cells by using appropriate quadrature rules.

The remainder of this chapter is concerned with how these tasks can be solved for domains $\Omega \subset \mathbb{R}^2$.

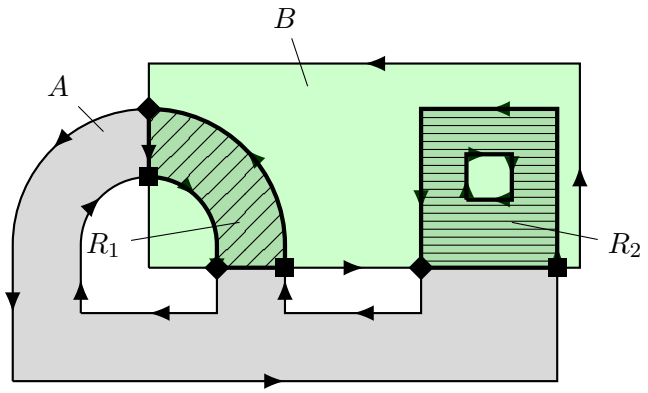

Figure 4.1: Intersection of a curved polygon A (gray) and an axis-aligned bounding box B (green). The intersection points have been categorized into *entry* points (rectangles) and *exit* points (diamonds) of the polygon A (categorization for the polygon B is not depicted). The intersection $R := A \cap B$ consists of two disjoint (curved) polygons R_1 (diagonally stripped) and R_2 (horizontally stripped), where R_2 includes one of the holes from A.

4.1 Robust & Efficient Intersection Computations

To compute the intersections $\mathcal{D}_i^m \cap \Omega$, we employ a modified version of the Greiner-Hormann polygon clipping algorithm [49]. This algorithm was originally designed to intersect two arbitrary polygons, which in particular includes self-intersecting polygons. Given two polygons A and B with holes, the algorithm can be summarized as follows:

1. Compute all intersection points of the boundaries of the polygons A and B.

2. Mark all intersection points as either *entry* or *exit* points for the polygons A and B. At an entry point of the polygon A we need to traverse along the boundary of the polygon A in *forward* direction to get to the next intersection point. At an exit point, we need to traverse in *reversed* direction.

3. Generate the resulting polygon by starting at any intersection point. If that point is an entry point of A, follow the boundary of A in forward direction until the next intersection point is reached. Otherwise follow B. At the next point, switch to the polygon B (or A) and traverse its boundary in forward/reversed direction, corresponding to its entry/exit flag. Whenever the initial intersection point is reached, all added curves form an outer ring of a resulting polygon. If any intersection point has not yet been reached, start forming a new ring from such an intersection point until no intersection points are left. If any holes of A or B are located completely within the resulting polygon without intersecting its boundary, their inner rings are copied to the resulting polygon.

Compare Figure 4.1 for an example of the intersection of two (curved) polygons A and B. While this algorithm has initially been introduced to compute intersections of arbitrary

polygons with piecewise linear boundaries, it can be easily modified to support the intersections we need: Patch domains ω_i and the integration domains $\mathcal{D}_i^m$ are always described by axis-aligned bounding boxes (AABB) and thus are a very simple case of a convex polygon. The domain Ω on the other hand is a polygon with the generalization that instead of having piecewise linear boundaries, we have boundaries of connected curve parts, described by NURBS curves. The nice property of the Greiner-Hormann polygon clipping algorithm is that this only affects step 1. of the algorithm: Instead of looking for the intersection points between two straight line segments, we need to find the intersections of axis-aligned line segments (the boundary segments of the AABB) and 2D NURBS curves (the boundary segments of the domain Ω).
But one major downside of the original version of the Greiner-Hormann clipping algorithm is that it cannot handle degenerate intersections where the boundaries of the polygons touch in individual points or partially overlap. This is due to how the entry/exit flags are computed: Given all intersection points $\boldsymbol{p}_i$, $i = 1, \ldots, N$, ordered along a ring of polygon A, we just need to check whether the first vertex of polygon A lies within polygon B or not. If that point lies within polygon B, the intersection point $\boldsymbol{p}_0$ is an exit point, otherwise it is an entry point. Then, all following intersection points $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_N$ are marked alternating as entry and exit points. But this alternating assignment of the entry/exit flags is not valid anymore when the polygons can have touching or overlapping intersections. Among others, solutions to that problem have been proposed in [68] and [42]. For this thesis a solution similar to what has been proposed in the latter has been implemented. The basic idea is to simply ignore intersection points where the boundary of polygon A does not cross the boundary of polygon B (and vice versa), but instead they are just *touching*. To detect those cases let us assume that $\boldsymbol{p}_i$ is the intersection point in question. We then check whether a point on the boundary of A between $\boldsymbol{p}_i$ and $\boldsymbol{p}_{i-1}$ and another point between $\boldsymbol{p}_i$ and $\boldsymbol{p}_{i+1}$ lies within the polygon B. Only if one of the points lies within B and the other one is not, the intersection point $\boldsymbol{p}_i$ is in fact a *turn* point and otherwise it is just a touching point. Overlapping intersections can simply be considered "stretched intersection points" where we need to check a point before the overlapping interval starts and an additional point after the end of it, to detect whether the whole overlapping segment is only touching or crossing. Note that we do not need to perform the additional point-within-polygon checks for all intersection points. Touching intersections can only arise when one of the curves has a kink at that location (a common case for this is at the endpoints of curve parts) or when the tangents of the curves are parallel at the intersection points. Additionally it is worth mentioning that in our use case one of the polygons is just an axis-aligned bounding box, and hence most of the point-within-polygon checks are just very simply point-within-AABB checks.
Hence the most difficult step that remains to be solved is to find all intersection points. To this end, we want to start with a more formal description of the problem at hand: Let an axis-aligned line segment in direction m be given by

$$\boldsymbol{A}(t) = t\alpha\boldsymbol{e}_m + \boldsymbol{c} \tag{4.1}$$

with $t \in [0, 1]$, $\boldsymbol{e}_m$ the canonical unit vector in direction m, α the length of the segment and $\boldsymbol{c}$ the base point of the segment. Computing intersection points of such a line segment and

a NURBS curve $\boldsymbol{C}$ as given by (3.4) can be described as solving the system of equations

$$\boldsymbol{A}(t) - \boldsymbol{C}(u) = 0. \tag{4.2}$$

Unfortunately solving equation (4.2) might be very difficult in general:

1. Since $\boldsymbol{C}$ is a NURBS curve described by a possibly rational, higher order basis, it is a **nonlinear** equation in general.
2. The equation can have **no solution** if the NURBS curve and the axis segment do not intersect.
3. The equation can have **more than one solution** if there are multiple intersection points.
4. The equation can have an **infinite number of solutions** if the NURBS curve and the axis do (partially) overlap.

While this problem is far from new and basically all CAD kernels need to solve these or similar equations, the difficulty in our case are the extremely high robustness and performance requirements to the solutions of this equation that are imposed by our use case. In general we have no control over how patches are located relative to the domain and the curves that are used to describe them. This makes very unlucky locations of patches that barely intersect or barely do not intersect specific curves not only likely, but basically guarantees that such cases will arise on higher levels with millions of patches on even just slightly more complex input geometries. Common approaches try to separate all roots of (4.1) when high robustness is required, but the root separation per intersection computation is very expensive and thus makes it unpractical for use cases like ours, where many intersections need to be computed. On the other hand, approximate approaches where a tolerance parameter is used to stop searching for further intersection points fail to guarantee that all intersection points are found. While a CAD kernel can commonly apply such approximate approaches and give the control back to the CAD engineer, who can then validate the outcome and take steps (like moving one of the geometries to be intersected slightly) in case of an incorrect computation, we do not have that luxury. To this end, we have to find a more robust approach to solve equation (4.2).
To achieve the required robustness and performance for our use case, we split the boundary of the simulation domain into parts where the number of intersection points with an infinite axis-aligned line is known a priori, i.e. we split the parameter domain $\mathcal{P}$ of all NURBS curves $\boldsymbol{C}$ into parts $\mathcal{P}_i$ where equation (4.2) has either no solution, exactly one solution or the curve and the axis-aligned line segment overlap over the complete interval, i.e. at all points $\boldsymbol{C}(u)$ for $u \in \mathcal{P}_i$. To this end, we have to split each NURBS curve on the boundary into x- and y-monotone parts. Splitting of NURBS curves into monotone parts has been done before to solve different kinds of problems. Rockwood et al. use monotone curve parts to split trimmed NURBS surface regions into smaller, monotone regions [99]. Qin et al. perform it to accelerate distance queries in the context of texture rendering [95]. Schollmeyer and Fröhlich pre-process NURBS trim bounds into monotone parts to speed up point classification queries that arise during ray-tracing of trimmed NURBS surfaces [104].

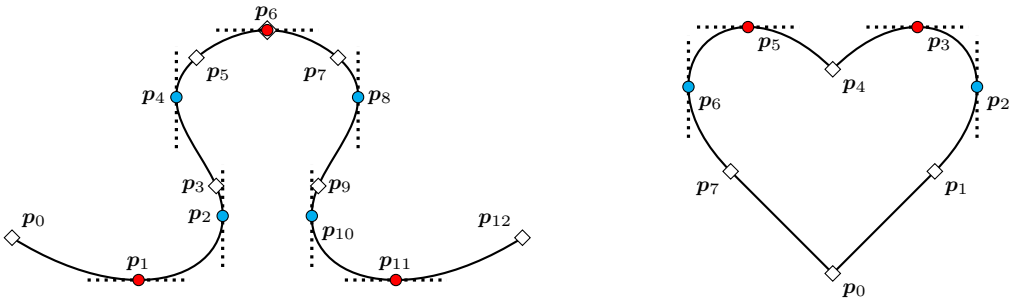

Figure 4.2: NURBS curves that have been split into bi-monotone parts. The split locations where $C'(u)_x = 0$ are denoted by blue circles and the split locations where $C'(u)_y = 0$ are denoted by red circles. Locations between two consecutive Bézier spans where the knot multiplicity is $\geq p$ are denoted by diamonds.
Left: The curve has been split into eight bi-monotone parts between the points $\{\boldsymbol{p}_0, \boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_4, \boldsymbol{p}_6, \boldsymbol{p}_8, \boldsymbol{p}_{10}, \boldsymbol{p}_{11}, \boldsymbol{p}_{12}\}$.
Right: The closed curve has been split into six bi-monotone parts between the points $\{\boldsymbol{p}_0, \boldsymbol{p}_2, \boldsymbol{p}_3, \boldsymbol{p}_4, \boldsymbol{p}_5, \boldsymbol{p}_6, \boldsymbol{p}_0\}$.

4.1.1 Monotone decomposition

Let us first start with a formal definition of a monotone curve:

Definition 4.1 (Monotone curve)**.** Let $\boldsymbol{C} : \mathcal{P} \to \mathbb{R}^2$ with $\mathcal{P} := [u_a, u_b] \subset \mathbb{R}$ be the parametrization of a two-dimensional curve with $\boldsymbol{C} \in C^0(\mathcal{P})$. Then the curve is called *monotone* in the direction $m \in \{1, 2\}$ over the interval $[\tilde{u_a}, \tilde{u_b}] \subseteq \mathcal{P}$ when

$$\begin{aligned} C'(u)_m &\geq 0 \qquad \forall u \in [\tilde{u_a}, \tilde{u_b}] \quad \text{or} \\ C'(u)_m &\leq 0 \qquad \forall u \in [\tilde{u_a}, \tilde{u_b}] \end{aligned} \tag{4.3}$$

holds. It is called *bi-monotone* if (4.3) holds for both $m = 1$ and $m = 2$. Similarly the curve is called *strictly (bi-)monotone* when

$$\begin{aligned} C'(u)_m &> 0 \qquad \forall u \in (\tilde{u_a}, \tilde{u_b}) \quad \text{or} \\ C'(u)_m &< 0 \qquad \forall u \in (\tilde{u_a}, \tilde{u_b}) \end{aligned} \tag{4.4}$$

holds.

Our goal is to split the parameter domain $\mathcal{P} := [u_a, u_b]$ of any NURBS curve $\boldsymbol{C}$ into parts $\mathcal{P}_i$ such that the curve is bi-monotone according to Definition 4.1 over each interval $\mathcal{P}_i$. To this end, we first find locations $\hat{u} \in \mathcal{P}$ where the curve has kinks so that the curve is at least C^1-continuous between all those locations. Then we search all those C^1-continuous sub-curves to find all locations $\bar{u} \in \mathcal{P}$ where $C'(\bar{u})_m = 0$ for $m \in \{1, 2\}$. Then the curve parts between all locations $\hat{u}$ and $\bar{u}$ are bi-monotone (compare Figure 4.2). The first step in the monotone decomposition is to convert all NURBS curves into an equal representation of piecewise Bézier curves. This can be achieved by repeated *knot insertion* which is often denoted as *knot refinement* [94]. It allows us to insert new knots into the knot vector U

by adjusting the control points $\boldsymbol{P}_i$ and weights w_i such that we get a new representation of the same NURBS curve. For example if we have a NURBS curve of degree $p = 3$ with knot vector $U := \{0, 0, 0, 0, 1, 2, 2, 3, 3, 3, 3\}$ we would insert the knots 1, 1 and 2 so that all inner knots have a multiplicity of p and the first and last knot have a multiplicity of $p+1$. After that conversion each span $u \in [u_k, u_{k+1}]$ between two NURBS knots $u_k \neq u_{k+1}$ can be represented by a rational Bézier curve of the form

$$\boldsymbol{C}_k(\tilde{u}) = \frac{\sum_{i=0}^{p} B_{i,p}(\tilde{u}) w_{\tilde{i}} \boldsymbol{P}_{\tilde{i}}}{\sum_{i=0}^{p} B_{i,p}(\tilde{u}) w_{\tilde{i}}} \tag{4.5}$$

with $\tilde{u} := \frac{u-u_k}{u_{k+1}-u_k}$ being the transformation of $u \in [u_k, u_{k+1}]$ into $\tilde{u} \in [0, 1]$, $\tilde{i} = i + k - p$ being the shifted index to select the required weights and control points from the original NURBS curve and $B_{i,p}$ being the Bernstein polynomial basis functions that are given by

$$B_{i,p}(\tilde{u}) = \binom{p}{i} \tilde{u}^i (1 - \tilde{u})^{p-i}.$$

Even though a NURBS curve that has been converted to piecewise Bézier form has a knot vector where all inner knots have a multiplicity of p and thus is only guaranteed to be C^0 at all those inner knots, some of those locations might actually still be of higher smoothness. E.g. at all knots that had a lower multiplicity before the conversion into piecewise Bézier form, the curve has to be of higher smoothness. To this end, we check the derivative of the two consecutive Bézier curves at each inner knot $u_k \in U$ where $u_{k+1} \neq u_k$ that connects them and do only split the curve $\boldsymbol{C}$ at u_k when $\boldsymbol{C}'_k(1) \neq \boldsymbol{C}'_{k+p}(0)$ or $\boldsymbol{C}'_k(1) = 0$.[1]
Now that we have detected all kink locations $\hat{u}$, we are left to find the extrema $\bar{u}$. To this end, we need to investigate the derivatives $\boldsymbol{C}'_k$ of the Bézier sub-curves. Let us first take a look at the special case where the NURBS curves are actually non-rational, i.e. where it is $w_i = 1$ for all $i = 0, \ldots, p$. In this case the denominator of (4.5) is equal to one everywhere and the numerator is a simple polynomial of degree p. Hence, the derivatives of such curves are polynomials of degree $p-1$ which can be represented by a Bézier curve of degree $p-1$, that is called its *hodograph*. For rational Bézier curves, the derivatives can be transformed into a rational function where the numerator is given by a Bézier curve of degree $2p-2$ [116] and is given by

$$\boldsymbol{C}'_{\text{bezier}}(u) = \frac{\sum_{i=0}^{2p-2} B_{i,2p-2}(u) \boldsymbol{R}_i}{(\sum_{i=0}^{p} B_{i,p}(u) w_i)^2} \tag{4.6}$$

with the control points

$$\boldsymbol{R}_i = \frac{\sum_{k=\max(0,i-p+1)}^{\lfloor i/2 \rfloor} (i - 2k + 1) \binom{p}{k} \binom{p}{i-k+1} w_k w_{i-k+1} (\boldsymbol{P}_{i-k+1} - \boldsymbol{P}_k)}{\binom{2p-2}{i}}.$$

Since the denominator in (4.6) is positive, it is sufficient to find only the roots of the numerator, which is a non-rational Bézier curve and thus a polynomial of known degree

[1] Note that for the monotone decomposition it would be sufficient to split at an inner knot u_k when there is an actual sign change in the derivatives, i.e. $\text{sgn}(\boldsymbol{C}'_k(1)) \neq \text{sgn}(\boldsymbol{C}'_{k+p}(0))$.

$2p-2$. This means, to find the roots of the derivatives of any NURBS curve we need an algorithm to find all roots of a polynomial of known degree. In the past a large number of algorithms to find all real roots of a polynomial have been developed. In our case are especially interested in algorithms to find the roots of polynomials that are given in the Bernstein basis since (4.6) is a Bézier curve and it has been shown that not converting those polynomials to a power basis for root finding is beneficial for the stability of the root finding problem [36]. A thorough overview over many such algorithms can be found in [126]. Nevertheless, most of those algorithms are especially designed to handle cases that are not well conditioned, as they can arise when dealing with root finding problems imposed when trying to find the intersections of two general Bézier curves. In contrast to that, we need to find the roots of the derivative of a single Bézier curve, and hence we usually have to deal with cases where the root finding problem is far better conditioned. To this end, we briefly summarize a very simple algorithm to find the required roots in the following. That algorithm has provided sufficient performance and robustness in all our test cases so far.

Let $f : [u_a, u_b] \to \mathbb{R}$, $u \mapsto f(u)$ be a polynomial of degree p. We want a simple algorithm to find all locations u_r where $f(u_r) = 0$ with $u_r \in [u_a, u_b]$. To this end we employ an algorithm similar to a variant of *real-root-isolation* based on a sequence of derivatives presented in [22]. We successively find the roots of the k-derivative, starting from the highest non-trivial derivative up to the function itself. First, for $k = p-1$ the degree of $f^{(k)}$ is at most one, so there will be at most one $u_{k,r} \in [u_a, u_b]$ where $f^{(k)}(u_{k,r}) = 0$. Whether there is an actual root or not can be verified by applying Bolzano's theorem. If the root exists we can find it by applying an algorithm like bisection and the root splits the interval into two parts $[u_a, u_{k,r}]$ and $[u_{k,r}, u_b]$. In both of these intervals the one-lower derivative $f^{(k-1)}$ is strictly monotone and hence can again have at most one root per interval. We continue to compute all roots $u_{k,r}$ for all $k = p-1, \ldots, 0$ until for $k = 0$ we have found the roots of f itself. Note that this algorithm is robust for cases where the actual degree of f is less than assumed, so it is of a degree $q < p$. In this case the derivatives $k = p-1, \ldots, q$ are all zero on the complete interval $[u_a, u_b]$. When applying Bolzano's theorem it will hence be $f^{(k)}(u_a) = f^{(k)}(u_b) = 0$ which is not a sign change and we thus do not report a root in $[u_a, u_b]$. This makes sure that the intervals of the next iteration will not be split up and we will keep the single interval $[u_a, u_b]$ until we reach $k = q-1$.

Remark 4.1. The monotone decomposition of a NURBS curve $\boldsymbol{C}$ described above does actually not only split the curve into bi-monotone curve parts, but into parts that are either *strictly* bi-monotone or *parallel* to either the x- or the y-axis. The reason for this is that we have split all Bézier curves at locations where $C'_k(\bar{u})_m = 0$, hence in the intervals between those locations the derivatives are non-zero and thus the curves are strictly monotone in direction m. The only exception to this is when $C'_k(\tilde{u})_m = 0$ for all $\tilde{u} \in [0, 1]$ in which case it is $C_k(\tilde{u})_m = \text{const}$ and hence the curve is parallel to the axis orthogonal to the direction m.

4.1.2 Intersection queries

Next, we will use the (bi-)monotone property of all curve parts to distinguish the cases where equation (4.2) has no, exactly one or an infinite amount of solutions. To this end, we first state some lemmas about the properties of a monotone curve:

Lemma 4.1. *Let $\boldsymbol{C} : [u_a, u_b] \to \mathbb{R}^2$, $u \mapsto \boldsymbol{C}(u)$ be a curve between the points $\boldsymbol{p}_a := \boldsymbol{C}(u_a)$ and $\boldsymbol{p}_b := \boldsymbol{C}(u_b)$ that is monotone in the direction m according to Definition 4.1. Let*

$$I_m := [\min\{(\boldsymbol{p}_a)_m, (\boldsymbol{p}_b)_m\}, \max\{(\boldsymbol{p}_a)_m, (\boldsymbol{p}_b)_m\}] \tag{4.7}$$

be the space in direction m between the endpoints of the curve. Then it holds that

$$C(u)_m \in I_m \qquad \forall u \in [u_a, u_b].$$

Proof. Without loss of generality, we consider the case $(\boldsymbol{p}_a)_m \leq (\boldsymbol{p}_b)_m$. Let us then assume that there is a location $\hat{u} \in (u_a, u_b)$ such that $C(\hat{u})_m < (\boldsymbol{p}_a)_m$. Since the curve is continuous it follows from the mean value theorem that there has to be a location $u^- \in (u_a, \hat{u})$ where $\boldsymbol{C}'(u^-) = \frac{C(\hat{u})_m - (\boldsymbol{p}_a)_m}{\hat{u} - u_a} < 0$ and another location $u^+ \in (\hat{u}, u_b)$ where $\boldsymbol{C}'(u^+) = \frac{(\boldsymbol{p}_b)_m - C(\hat{u})_m}{u_b - \hat{u}} > 0$. This contradicts the definition of a monotone curve that has either only non-negative or only non-positive derivatives and hence such locations $\hat{u}$ cannot exist. The case for a point with $C(\hat{u})_m > (\boldsymbol{p}_b)_m$ follows by symmetry, just like the cases for $(\boldsymbol{p}_a)_m \geq (\boldsymbol{p}_b)_m$. □

Remark 4.2. From Lemma 4.1 it especially follows that a bi-monotone curve $\boldsymbol{C}$ with $u \in [u_a, u_b]$ is contained within the box $B := I_1 \times I_2$ that is the bounding box of its endpoints $\boldsymbol{p}_a := \boldsymbol{C}(u_a)$ and $\boldsymbol{p}_b := \boldsymbol{C}(u_b)$.

Lemma 4.2. *Let $\boldsymbol{C} : [u_a, u_b] \to \mathbb{R}^2$, $u \mapsto \boldsymbol{C}(u)$ be a curve between the points $\boldsymbol{p}_a := \boldsymbol{C}(u_a)$ and $\boldsymbol{p}_b := \boldsymbol{C}(u_b)$ that is monotone in the direction m according to Definition 4.1. Then the following implication holds*

$$(\boldsymbol{p}_a)_m = (\boldsymbol{p}_b)_m =: \xi \qquad \Longrightarrow \qquad C(u)_m = \xi \quad \forall u \in [u_a, u_b].$$

Proof. This follows directly from Lemma 4.1, since in this case it is $I_m := [(\boldsymbol{p}_a)_m, (\boldsymbol{p}_b)_m] = \{\xi\}$. □

Lemma 4.3. *Let $\boldsymbol{C} : [u_a, u_b] \to \mathbb{R}^2$, $u \mapsto \boldsymbol{C}(u)$ be a curve between the points $\boldsymbol{p}_a := \boldsymbol{C}(u_a)$ and $\boldsymbol{p}_b := \boldsymbol{C}(u_b)$ that is strictly monotone in the direction m according to Definition 4.1. Then, for all $\tilde{u}, \hat{u} \in [u_a, u_b]$ with $\tilde{u} \neq \hat{u}$ it holds that*

$$C(\tilde{u})_m \neq C(\hat{u})_m.$$

Proof. Since $\boldsymbol{C}$ is differentiable for $u \in [u_a, u_b]$ we can express any point along the curve by

$$C(u)_m = C(u_a)_m + \int_{u_a}^{u} C'(\tau)_m \, d\tau.$$

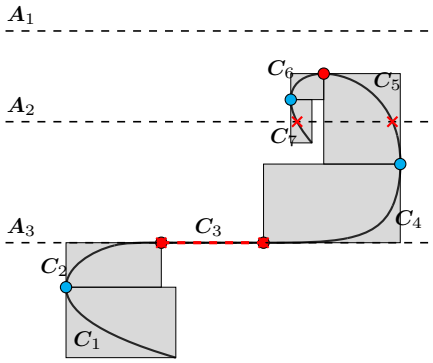

Figure 4.3: A NURBS curve $\boldsymbol{C}$ has been split into seven bi-monotone curve parts $\{\boldsymbol{C}_1, \boldsymbol{C}_2, \ldots, \boldsymbol{C}_7\}$. The infinite line parallel to the x-axis $\boldsymbol{A}_1$ does not intersect any of the bounding boxes of the curve parts and thus does not intersect any of the curves. The line $\boldsymbol{A}_2$ intersects the bounding boxes of the curves $\boldsymbol{C}_5$ and $\boldsymbol{C}_7$ and thus has exactly one intersection with each of the curves and no intersections with any other curve. The line $\boldsymbol{A}_3$ intersects the (closed set) bounding boxes of $\boldsymbol{C}_2$, $\boldsymbol{C}_3$, and $\boldsymbol{C}_4$. Since both endpoints of the bi-monotone curve $\boldsymbol{C}_3$ are located on $\boldsymbol{A}_3$, the curve overlaps the line over its entire length. For the curves $\boldsymbol{C}_2$ and $\boldsymbol{C}_4$ the line has exactly one intersection at the end and start of the curves respectively.

Given some $\tilde{u}, \hat{u} \in [u_a, u_b)$ with $\tilde{u} \neq \hat{u}$. Without loss of generality, assume $\tilde{u} < \hat{u}$. Then, we can state that

$$\begin{aligned} C(\hat{u})_m &= C(u_a)_m + \int_{u_a}^{\hat{u}} C'(\tau)_m \, d\tau \\ &= C(u_a)_m + \int_{u_a}^{\tilde{u}} C'(\tau)_m \, d\tau + \int_{\tilde{u}}^{\hat{u}} C'(\tau)_m \, d\tau \\ &= C(\tilde{u})_m + \int_{\tilde{u}}^{\hat{u}} C'(\tau)_m \, d\tau. \end{aligned}$$

Let us first assume that the curve is strictly monotone *increasing* in the direction m, i.e. $C'(u)_m > 0$ for all $u \in (u_a, u_b)$. Then it is $\int_{\tilde{u}}^{\hat{u}} C'(\tau)_m \, d\tau > 0$ and hence $C(\hat{u})_m \neq C(\tilde{u})_m$. By symmetry the same holds for a strictly monotone *decreasing* curve, i.e. $C'(u)_m < 0$ and hence it holds for any curve that is strictly monotone in direction m. □

Let us now consider a line A according to (4.1), parallel to the axis in direction $m \in \{1, 2\}$ with the base point $\boldsymbol{c}$ and $k := (m \bmod 2) + 1$ being the orthogonal direction to m in the two-dimensional space. Furthermore, let us assume that the parameter domain $\mathcal{P}$ of the curve $\boldsymbol{C}$ has been split into parts $\mathcal{P}_i := [a_i, b_i]$ by the monotone decomposition algorithm described in the previous paragraph. Then, from Lemma 4.1 it follows that a curve $\boldsymbol{C}$ has no intersection with A for $u \in \mathcal{P}_i$ when

$$c_k \notin I_k \tag{4.8}$$

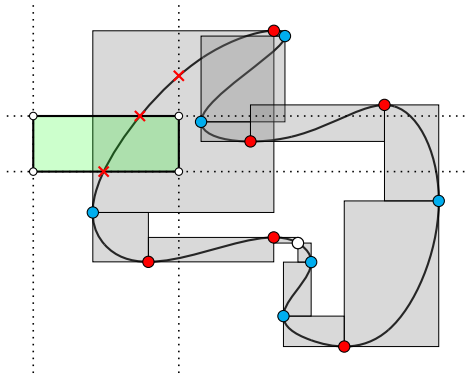

Figure 4.4: The boundary segments of an axis-aligned bounding box (green) have been extended to infinity (dotted lines). Even though the infinite lines have multiple intersections with different curve parts, we need to compute only three intersection points (red crosses) since the boundary segments do not intersect the bounding boxes of the other curves. One of the intersection points will be discarded, since it is outside of the original segment's domain.

holds, where I_k is the curve's bounds in direction k according to (4.7). On the other hand, from Lemma 4.2 in follows that the curve $\boldsymbol{C}$ overlaps A over the entire interval $\mathcal{P}_i$ when

$$C(a_i)_k = C(b_i)_k = c_k \tag{4.9}$$

holds. Since equation (4.9) detects all intervals $\mathcal{P}_i$ in which the curve is parallel to the axis in direction m, from Remark 4.1 the curve is strictly monotone in all remaining intervals $\mathcal{P}_i$. Hence, from Lemma 4.3 it follows that equation (4.2) has exactly one solution for $u \in \mathcal{P}_i$ when neither (4.8) nor (4.9) hold. Compare Figure 4.3 for examples of curve-axis intersections.
Additionally note that since we need to find the intersection point with an axis-aligned line, we only need to take a single component of the two-dimensional curve equation for $\boldsymbol{C}$ into account. Hence, we need to find $\hat{u} \in \mathcal{P}_i$ such that

$$C(\hat{u})_k - c_k = 0. \tag{4.10}$$

Finding those roots is now an easy task and can be performed by classic methods such as a Newton solver or simple bisection. We should additionally note that equation (4.10) is strictly monotone for $u \in \mathcal{P}_i$ which makes it an particularly easy problem to solve and we only need to evaluate a single component of the curve's equation, which makes it even cheaper to compute. When we found a solution $\hat{u}$, the parameter $\hat{t}$ of the axis where $\boldsymbol{C}(\hat{u}) = \boldsymbol{A}(\hat{t})$, is given by $\hat{t} := \frac{C(\hat{u})_m - c_m}{\alpha}$ and we can discard intersection points where $\hat{t} \notin [0, 1]$ that are not located on the original segment's domain.
To reduce the number of intersection points that are computed and then discarded we can perform an initial filtering: For any axis-aligned segment we first check whether the segment intersects the bounding box of any bi-monotone curve part. Only in the case that

there is such an intersection, the segment and the curve can have an intersection (compare Remark 4.2 and Figure 4.4). Note that such a filtering returns a subset of the curves for which (4.8) holds and thus for each curve returned by that filtering we have exactly one intersection point (or an overlap over the full curve domain) with the infinite extension of the axis segment and no intersection points for all other curve parts that have been filtered out. To speed up the filtering we can construct an acceleration structure like an R-tree [57]. Note that due to the fact that we can easily first compute all monotone curve parts and then construct the tree, we can use packing algorithms to build the tree, which should result in good balancing and thus sufficient query times [43, 80]. Updates of the tree after its initial construction are not required.

4.2 Local Decomposition into Integration Cells

Now that we can build the intersection domains of Ω with an axis-aligned bounding box we need to decompose these intersection domains into simpler geometric primitives that we have known quadrature rules for. The approach of choice in this thesis is to decompose the potentially curved domains into triangles, with potentially curved edges. To this end, we first create a linear triangulation of the curved domain, while keeping track of edges that are actually curved. In a post-processing step we make sure that the curved triangulation is valid, i.e. it does not contain any self-intersecting outer, curved or inner, linear edges.
The problem of generating linear triangulations for polygons or similar bounded domains has been well studied in the past and multiple types of algorithms have been developed. The predominant approaches in practice are *iterative* Delaunay construction algorithms. A Delaunay triangulation is a triangulation where no point is located within the circumcircle of any triangle, which is called the *Delaunay criterion*. Such triangulations maximize the minimum angle in the triangulation. Some noteworthy basics on how to add a new point into an already existing Delaunay triangulation have been developed independently by Bowyer [15] and Watson [134] and are nowadays known as the *Bowyer-Watson-algorithm*. Lawson layed down the fundamentals to describe algorithms that allow to transform non-Delaunay triangulations into Delaunay triangulations by performing simple flips [79]. Sloan combined those ideas and developed a fast algorithm to compute the Delaunay triangulation of any input point-set [125]. Algorithms which do not only triangulate a given point-set, but ensure that edges from the input polygons are represented within the set of edges of the triangulation can mainly be divided into two groups: *conforming* triangulations where the Delaunay criterion is fulfilled everywhere and *constrained* triangulations that allow the Delaunay criterion to be violated locally at input edges.
Conforming methods make sure that all input edges are represented in the triangulation by inserting additional points and utilize the Delaunay criterion itself to make sure all required edges are present [121]. One benefit of those methods is that *Delaunay refinement* algorithms (like *Chew's second algorithm* [21] or *Ruppert's algorithm* [100]) to improve the quality of the resulting triangles often require that all triangles are Delaunay and thus they can be applied easily on conforming triangulations.
On the other hand, the benefit of constrained methods is that no additional points need to

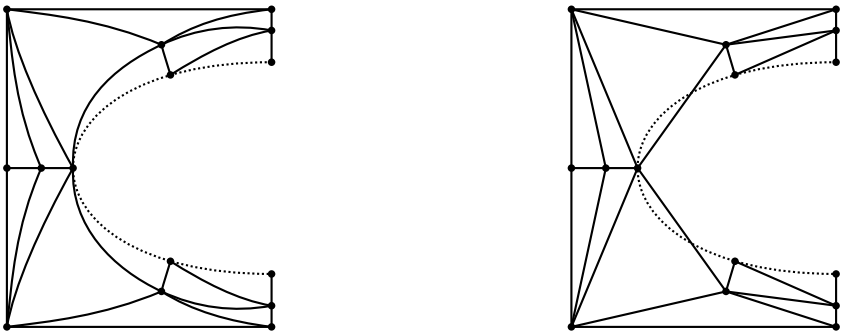

Figure 4.5: Meshes for a domain with one curved edge (dotted).
Left: Generated by an *a posteriori* meshing method potentially all edges in the mesh are curved and thus all cells have to be higher order curved triangles.
Right: Generated by a *direct mapping* method only the edges belonging to the curved input polygon are curved. Invalid triangles can arise where a curved input edge intersects an inner edge.

be inserted into the triangulation, thus they usually result in triangulations with a lower number of final triangles. The main idea behind constrained triangulations is to first create a Delaunay triangulation of only the input points. In the next step all constrained edges are inserted. For each constraint we need to find all triangles that are intersected by them. Those triangles are first removed from the triangulation and the resulting cavity is split along the constrained edge to be inserted into two sub-polygons that then need to be re-triangulated [27]. After that step the inserted edge might not fullfil the Delaunay criterion so that it needs to be flagged to make sure it will not be flipped in any subsequent insertion steps. There are different approaches on how to re-triangulate the cavity during the edge insertion. Some possibilities are given in [122, 132] where the latter has been implemented for this thesis.

For curved mesh generation techniques one of the main application fields is to generate higher order meshes for finite element methods, so that good convergence rates of an *hp*-FEM can be attained on curved geometries [6]. Some of the most common methods to generate higher order meshes for the finite element analysis start with a linear mesh and then deform the cells by increasing the degree of the triangles via solving various physical equations and are called *a posteriori* methods. Common methods to achieve this are by solving a linear elastic equation [2], non-linear elastic equations [136] or the Winslow equation [41, 72]. All these methods have in common that the deformation is done on the hole body, thus curved triangles will not only be present at the boundary, but throughout the triangulation. Additionally since the curving is represented by polynomial higher order finite elements, the original curved boundaries of the input geometries are usually still approximated only and especially non-polynomial boundaries like circles or ellipsis cannot be represented exactly. A benefit of those methods is that due to the deformation of not only the edges on the surface of the original geometry, but all inner edges as well, invalid triangles where a curved surface edge intersects one of the inner edges (as depicted

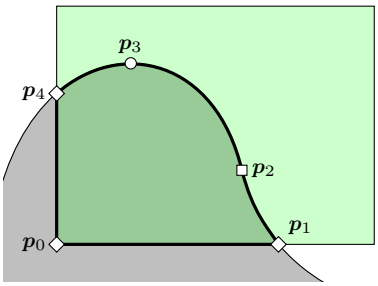

Figure 4.6: Points to be added into a local triangulation for the intersection of the curved domain Ω (gray) and an integration domain $\mathcal{D}_i^m$ (green). The points $\boldsymbol{p}_0$, $\boldsymbol{p}_1$ and $\boldsymbol{p}_4$ are endpoints of the segments belonging to $\partial\mathcal{D}_i^m$ and are of category 1. The point $\boldsymbol{p}_3$ is an extrema of a curve belonging to $\partial\Omega$ and is of category 1. as well. The point $\boldsymbol{p}_2$ is an inflection point of a curve belonging to $\partial\Omega$ and is of category 2.

on the right in Figure 4.5) do usually not arise.
Direct mapping methods start with a linear mesh as well but then create higher order elements only at the boundary by applying an analytical map [29, 44, 58]. While many of these methods still only use polynomial approximations to boundaries described by rational functions, a method to keep the original NURBS curves and surfaces for the geometry approximation was introduced in [118]. Since direct methods only use curved edges directly at the original curved surface of the geometry, they all have to deal with curved edges intersecting inner edges of the triangulation as depicted in figure 4.5.
The method presented here falls into the category of direct mapping methods as well. We start with a linear triangulation via an iterative constrained Delaunay triangulation. An algorithm that is particularly easy to implement and showed sufficient performance for our use cases is the one presented by Anglada [132]. Furthermore, data-structures to represent the triangulation similar to the ones used by CGAL and described by Boissonnat et al. [14] have been used. To locate the triangle which contains a point to be inserted, we implemented a *line-walk* algorithm according to Devillers et al. [28]. To ensure consistent results of geometric predicates with floating-point arithmetic, we implemented adaptive floating-point predicates as shown in the well-known paper of Shewchuk [120].

4.2.1 Input point-set

First we need to specify the input point-set to be used when constructing the initial linear triangulation. To this end, we add the following points (compare Figure 4.6):

1. Endpoints of bi-monotone curve parts, i.e. the curve's extrema in x and y direction and kinks.

2. Points where the sign of the curvature changes, i.e. the curve's inflection points.

3. Points where two consecutive input curves are connected.

Adding points from the category 1. gives us the guarantee that each curve that connects two points from the triangulation is smooth and bi-monotone. Additionally adding the points of category 2. makes sure that all those curve parts have constant sign curvature as well. The properties of bi-monotonicity and constant sign curvature will be important later on when we replace the initial linear edges with their original curved representation. The points of category 3. are not actually necessary if two consecutive curve parts are connected smoothly, i.e. they are not at kinks and do not additionally fall into category 1. We add them nevertheless since this way each edge in the triangulation will only belong to a single curve of the input geometry. This property makes it easier if we need to evaluate integrals on a specific input curve only (e.g. to apply local boundary conditions on that input curve).
If we consider that we want to triangulate the intersection of our initial input geometry and an axis-aligned bounding box we can note the following: The points from category 1. are the split points that we computed for the initial monotone decomposition of the input curves, thus they are readily available and do not need to be recomputed for the curve parts belonging to the input geometry. Since the other curves that are parts of the axis-aligned bounding box are always straight lines, they do not have any extrema and their curvature is zero, so we do only need to take their endpoints into account.
The only points that are missing are the inflection points where the sign of the curvature of a curve changes. The signed curvature of a plane curve $\boldsymbol{C}$ is defined as

$$\kappa(u) = \frac{\boldsymbol{C}'(u) \times \boldsymbol{C}''(u)}{\|\boldsymbol{C}'(u)\|^3} = \frac{\det\left(\boldsymbol{C}'(u), \boldsymbol{C}''(u)\right)}{\|\boldsymbol{C}'(u)\|^3}. \tag{4.11}$$

If we do only consider intervals $u \in (a, b)$ for which the curve is bi-monotone, (4.11) is always defined since we have split the curve at locations where $C'(u)_x = 0$ and $C'(u)_y = 0$ and hence $\|C'(u)\| \neq 0$ for $u \in (a, b)$ holds. Since the denominator is always positive for $u \in (a, b)$ we need to be concerned with the roots of the numerator only and hence we need to find all $\tilde{u}$ where

$$\boldsymbol{C}'(\tilde{u}) \times \boldsymbol{C}''(\tilde{u}) = C'(\tilde{u})_x C''(\tilde{u})_y - C''(\tilde{u})_x C'(\tilde{u})_y = 0 \tag{4.12}$$

holds. For non-rational curves (4.12) is a polynomial of degree $2p-3$. For rational curves, in contrast to the monotone decomposition that we have presented earlier, we are now additionally concerned with the second derivatives $\boldsymbol{C}''$. To this end, we first introduce an alternative representation for the hodograph of a rational Bézier curve compared to (4.6). Kim et. al. introduced a closed form for the hodograph in [69] such that the hodograph itself is another rational Bézier curve of degree $2p$ that is given by

$$\boldsymbol{C}'(u) = \frac{\sum_{i=0}^{2p} B_{i,2p}(u) \bar{w}_i \bar{\boldsymbol{P}}_i}{\sum_{i=0}^{2p} B_{i,2p}(u) \bar{w}_i} \tag{4.13}$$

with the weights given by

$$\bar{w}_i = \sum_{j=0}^{p} w_{i-j} w_j c_{p,p,i-j,j} \tag{4.14}$$

with

$$c_{n,m,i,j} = \begin{cases} \frac{\binom{n}{i}\binom{m}{j}}{\binom{n+m}{i+j}} & \text{if } 0 \le i \le n \text{ and } 0 \le j \le m \\ 0 & \text{otherwise} \end{cases}$$

and the control points given by

$$\begin{aligned} \bar{\boldsymbol{P}}_i = \Big(\sum_{j=0}^{p} \big(&jw_{i-j}(w_j(P_j - P_{i-j}) - w_{j-1}(P_{j-1} - P_{i-j})) \\ &+ (p-j)w_{i-j}(w_{j+1}(P_{j+1} - P_{i-j}) - w_j(P_j - P_{i-j}))\big) c_{p,p,i-j,j}\Big) / \bar{w}_i. \end{aligned} \tag{4.15}$$

While in this representation the numerator of (4.13) is of degree $2p$ instead of $2p-2$ as given by (4.6), the benefit is that it can be applied recursively to derive higher order derivatives of a rational curve. Hence, the second derivative is given by

$$\boldsymbol{C}''(u) = \frac{\sum_{i=0}^{4p} B_{i,4p}(u)\bar{\bar{w}}_i \bar{\bar{\boldsymbol{P}}}_i}{\sum_{i=0}^{4p} B_{i,4p}(u)\bar{\bar{w}}_i}$$

where $\bar{\bar{w}}_i$ and $\bar{\bar{\boldsymbol{P}}}_i$ can be obtained by applying (4.14) and (4.15) to the weights $\bar{w}_i$ and control points $\bar{\boldsymbol{P}}_i$ of the first derivative. Introducing the short hand notation

$$\boldsymbol{C}'(u) = \frac{\sum_{i=0}^{2p} B_{i,2p}(u)\bar{w}_i \bar{\boldsymbol{P}}_i}{\sum_{i=0}^{2p} B_{i,2p}(u)\bar{w}_i} = \frac{\bar{\boldsymbol{N}}(u)}{\bar{D}(u)} \quad \text{and} \quad \boldsymbol{C}''(u) = \frac{\sum_{i=0}^{4p} B_{i,4p}(u)\bar{\bar{w}}_i \bar{\bar{\boldsymbol{P}}}_i}{\sum_{i=0}^{4p} B_{i,4p}(u)\bar{\bar{w}}_i} = \frac{\bar{\bar{\boldsymbol{N}}}(u)}{\bar{\bar{D}}(u)}$$

we can now rewrite (4.12) as

$$\boldsymbol{C}'(\tilde{u}) \times \boldsymbol{C}''(\tilde{u}) = \frac{\bar{N}(u)_x \bar{\bar{N}}(u)_y - \bar{\bar{N}}(u)_x \bar{N}(u)_y}{\bar{D}(u)\bar{\bar{D}}(u)}. \tag{4.16}$$

Since the denominator of that expression is the product of two positive functions $\bar{D}(u)$ and $\bar{\bar{D}}(u)$, it is positive as well and to find the roots of (4.16) we need to find the roots of the numerator only. By applying arithmetic multiplication and addition for polynomials given in Bernstein basis from [37] on the numerator of (4.16) we can further rewrite the root finding problem to find the roots of a non-rational scalar valued polynomial in Bernstein form

$$\bar{N}(u)_x \bar{\bar{N}}(u)_y - \bar{\bar{N}}(u)_x \bar{N}(u)_y = \sum_{i=0}^{6p} B_{i,6p}(u) c_i = 0 \tag{4.17}$$

with the coefficients

$$c_i = \sum_{j=\max(0,i-4p)}^{\min(2p,i)} \frac{\binom{2p}{j}\binom{4p}{i-j}}{\binom{6p}{i}} \left(\bar{w}_j(\bar{P}_j)_x \bar{\bar{w}}_{i-j}(\bar{\bar{P}}_{i-j})_y - \bar{\bar{w}}_{i-j}(\bar{\bar{P}}_{i-j})_x \bar{w}_j(\bar{P}_j)_y\right).$$

Note that the final equation (4.17) for rational curves is of the same structure as the original equation (4.12) for non-rational curves. Hence, the transformation of the equation into a

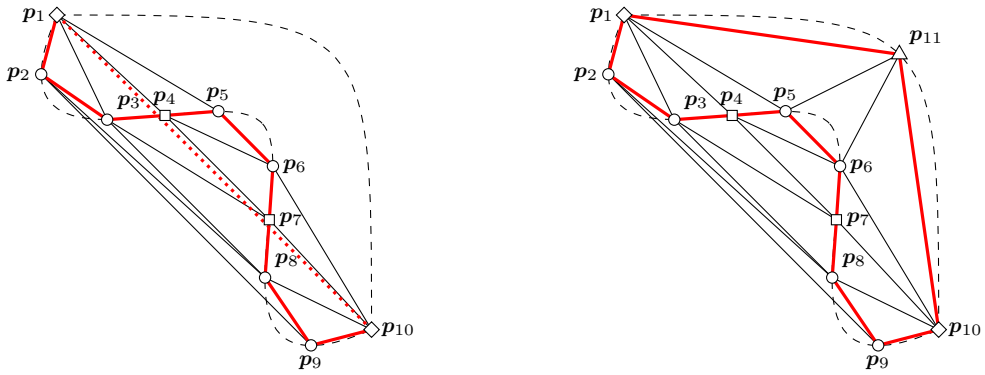

Figure 4.7: Steps to create a constrained triangulation of a boomerang domain.
Left: The initial triangulation has been generated from the input points $\{\boldsymbol{p}_1, \ldots, \boldsymbol{p}_{10}\}$ and the segments $\{(\boldsymbol{p}_1, \boldsymbol{p}_2), (\boldsymbol{p}_2, \boldsymbol{p}_3), \ldots, (\boldsymbol{p}_9, \boldsymbol{p}_{10})\}$ have been inserted as constrained edges (solid red) into the triangulation. The next constraint $(\boldsymbol{p}_{10}, \boldsymbol{p}_1)$ (dotted red) cannot be added since it intersects the already added constraints $(\boldsymbol{p}_3, \boldsymbol{p}_4)$ and $(\boldsymbol{p}_7, \boldsymbol{p}_8)$. *Right:* The curve belonging to the edge $(\boldsymbol{p}_{10}, \boldsymbol{p}_1)$ has been refined. To this end a new point $\boldsymbol{p}_{11}$ has been added to the triangulation. The new constraints $(\boldsymbol{p}_{10}, \boldsymbol{p}_{11})$ and $(\boldsymbol{p}_{11}, \boldsymbol{p}_1)$ have been added without intersecting any other already added constraint.

single, scalar valued polynomial in Bernstein basis can also be applied to (4.12) for non-rational curves and we end up with a root finding problem for polynomials in Bernstein form that are of degree $2p - 3$ for non-rational curves and $6p$ for rational curves.[2] These roots can be found by the same polynomial real root isolation algorithm that has already been used for the monotone decomposition.
Note that the inflection points, just like the extrema required for the monotone decomposition, depend on Ω only and are independent of the integration domains $\mathcal{D}_i^m$ we intersected Ω with. Hence, the inflection points need to be computed only once in the beginning for Ω and can be reused for the triangulations of any domain $\Omega \cap \mathcal{D}_i^m$.

4.2.2 Initial linear triangulation

Now that we have specified all required points we first create an initial Delaunay triangulation of that point set and then insert constrained edges between all consecutive points that belong to an input curved edge. Since the constrained edges are straight and thus do only approximate the original curves it can happen that those constraints intersect, even though the original curves do not intersect (compare Figure 4.7). We resolve these cases by a simple subdivision algorithm. Consider a curve $\boldsymbol{C}_1$, $u \mapsto \boldsymbol{C}_1(u)$ and two points $\boldsymbol{p}_a := \boldsymbol{C}_1(u_a)$ and $\boldsymbol{p}_b := \boldsymbol{C}_1(u_b)$ along that curve that are part of the initial triangulation.

[2] The actual degree of the polynomial for rational curves is most likely of a slightly lower degree, since we used an equation where the numerator in the rational hodograph was of degree $2p$, but it is known that the degree of that numerator is actually $2p - 2$ only.

Consider a second curve $\boldsymbol{C}_2$, $v \mapsto \boldsymbol{C}_2(v)$ with the points $\boldsymbol{q}_a := \boldsymbol{C}_2(v_a)$ and $\boldsymbol{q}_b := \boldsymbol{C}_2(v_b)$. Assuming that the edge $(\boldsymbol{p}_a, \boldsymbol{p}_b)$ has already been added to the triangulation and the insertion of the edge $(\boldsymbol{q}_a, \boldsymbol{q}_b)$ fails since it intersects the edge $(\boldsymbol{p}_a, \boldsymbol{p}_b)$. We then add a new point $\boldsymbol{q}_c := \boldsymbol{C}_2(\frac{v_a+v_b}{2})$ into the triangulation and try to add the constrained edges $(\boldsymbol{q}_a, \boldsymbol{q}_c)$ and $(\boldsymbol{q}_c, \boldsymbol{q}_b)$. If this fails as well we remove the constrained edge $(\boldsymbol{p}_a, \boldsymbol{p}_b)$ from the triangulation, add another point $\boldsymbol{p}_c := \boldsymbol{C}_1(\frac{u_a+u_b}{2})$ along the first curve $\boldsymbol{C}_1$ and add the constrained edges $(\boldsymbol{p}_a, \boldsymbol{p}_c)$ and $(\boldsymbol{p}_c, \boldsymbol{p}_b)$ before trying to add the edges $(\boldsymbol{q}_a, \boldsymbol{q}_c)$ and $(\boldsymbol{q}_c, \boldsymbol{q}_b)$ again. This subdivision is then applied recursively whenever the insertion of a constrained edge fails, until all constrained edges have been added without any intersection problems. Using our restriction that the input curves $\boldsymbol{C}_i$ are not allowed to intersect, this algorithm has to terminate at some point, since the approximations of the curves $\boldsymbol{C}_1$ and $\boldsymbol{C}_2$ by the points p and q will convergence towards the original intersection-free curves. But as we have learned in Chapter 3, curves in STEP files are often only approximations to actual intersection curves of two faces. Hence, in practice it can still happen that two input curves do actually intersect. It is easy to tolerate such intersections for us: The splitting algorithm to resolve self-intersections presented above can be interpreted as a bisection algorithm to find those invalid intersection points. Hence, we simply introduce a check that tests for $\min\{|\boldsymbol{p}_a - \boldsymbol{p}_c|, |\boldsymbol{p}_b - \boldsymbol{p}_c|\} < \epsilon$ after inserting the point $\boldsymbol{p}_c$ in-between $\boldsymbol{p}_a$ and $\boldsymbol{p}_b$ (an similarly for $\boldsymbol{q}_c$, $\boldsymbol{q}_a$ and $\boldsymbol{q}_b$) with some ϵ dependent on the machine precision. If this check is hit, this means that we have reached a point where $\boldsymbol{p}_c$ is barely distinguishable from $\boldsymbol{p}_a$ and $\boldsymbol{p}_b$. Hence $\boldsymbol{p}_c$ should be a good approximation of the intersection point of the curves $\boldsymbol{C}_1$ and $\boldsymbol{C}_2$. Therefore we insert that point into the triangulation and add constrained edges $(\boldsymbol{p}_a, \boldsymbol{p}_c)$, $(\boldsymbol{p}_c, \boldsymbol{p}_b)$, $(\boldsymbol{q}_a, \boldsymbol{p}_c)$ and $(\boldsymbol{p}_c, \boldsymbol{q}_b)$ and stop the recursive subdivision after that point.[3]

4.2.3 Curving of triangles at the boundary

To create the final curved triangulation we have to replace all constrained edges in the linear triangulation with the input curve parts that they belong to. As with all direct mapping methods it can now happen that those curved edges intersect inner, non-constrained edges of the triangulation or a triangle gets inverted (compare Figure 4.8). To resolve these issues we use the property that all curves belonging to a single constrained edge are bi-monotone and have constant sign curvature which is guaranteed by the initial selection of the points we inserted into the triangulation. To this end, we first formally define a curve with constant sign curvature, similar to Definition 4.1 for monotone curves. We then show that given a curve $\boldsymbol{C}$ that is bi-monotone and has constant sign curvature, we can find a tight bounding triangle $\mathcal{T}_{\boldsymbol{C}}$ such that the complete curve is contained in that triangle. Hence, for any triangle $(\boldsymbol{p}_a, \boldsymbol{p}_b, \boldsymbol{p}_c)$ of the curved triangulation where the segment $(\boldsymbol{p}_a, \boldsymbol{p}_b)$ has to be replaced by the curve $\boldsymbol{C}$ we can quickly test whether the other segments $(\boldsymbol{p}_b, \boldsymbol{p}_c)$ or $(\boldsymbol{p}_c, \boldsymbol{p}_a)$ intersect the bounding triangle $\mathcal{T}_{\boldsymbol{C}}$ to detect invalid self-intersections

[3] Note that this is different (and in fact easier) than finding all intersection points of all involved curves $\boldsymbol{C}_i$ beforehand, since the algorithm presented here is not guaranteed to actually find all intersection points. For two intersecting curves, the initial linear constrained edges could actually not intersect at all, and hence the recursive sub-division would never be initiated.

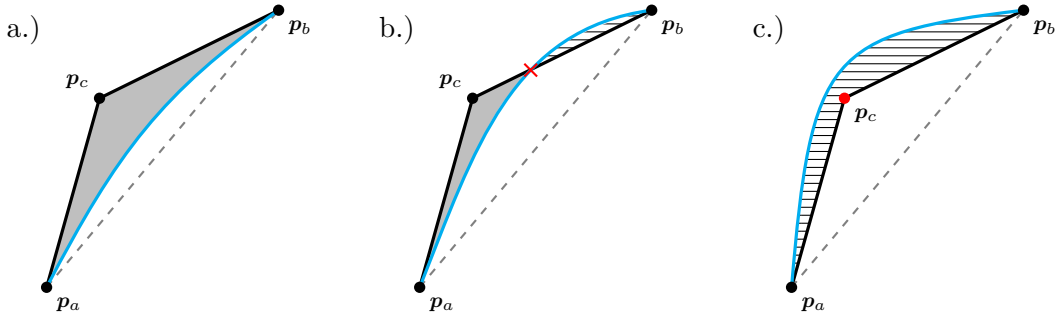

Figure 4.8: A triangle $(\boldsymbol{p}_a, \boldsymbol{p}_b, \boldsymbol{p}_c)$ with a single curved edge (blue) that is bi-monotone and has constant sign curvature. Case a.) is a valid configuration, where the volume of the curved triangle is well defined. Cases b.) and c.) are invalid. Either the curved edge intersects an inner, non-curved edge (case b.)) or the point $\boldsymbol{p}_c$ is underneath the curved edge (case c.)) and the whole triangle is inverted.

and inversions. We finally show how such cases can be resolved after they have been detected robustly.

Definition 4.2 (Constant sign curvature)**.** Let $\boldsymbol{C} : \mathcal{P} \to \mathbb{R}^2$ with $\mathcal{P} := [u_a, u_b] \subset \mathbb{R}$ be the parametrization of a two-dimensional curve with $\boldsymbol{C} \in C^0(\mathcal{P})$. Then the curve is said to have *constant sign curvature* if

$$\begin{aligned} \kappa(u) &\geq 0 \qquad \forall u \in [u_a, u_b] \quad \text{or} \\ \kappa(u) &\leq 0 \qquad \forall u \in [u_a, u_b] \end{aligned} \tag{4.18}$$

holds.

Additionally the direction of the tangent along a curve can be represented by a function $\phi(u) : [u_a, u_b] \to \mathbb{R}$ that describes the angle between the tangent and the x-axis and is denoted the *tangential angle*, defined by

$$\frac{\boldsymbol{C}'(u)}{\|\boldsymbol{C}'(u)\|} = \begin{pmatrix} \cos\phi(u) \\ \sin\phi(u) \end{pmatrix}. \tag{4.19}$$

The tangential angle and the curvature are related by $\phi'(u) = \|\boldsymbol{C}'(u)\|\kappa(u)$ [48, p. 20] and hence it is

$$\phi(u) = \int_{u_a}^{u} \|\boldsymbol{C}'(\tau)\|\kappa(\tau)\, d\tau + \phi_0$$

with $\phi_0 = \phi(u_a)$ being the angle of the initial tangent $\frac{\boldsymbol{C}'(u_a)}{\|\boldsymbol{C}'(u_a)\|}$. While $\phi(u)$ is always the angle relative to the x-axis, we can shift that function by the initial angle at u_a to get

$$\tilde{\phi}(u) := \phi(u) - \phi_0 = \int_{u_a}^{u} \|\boldsymbol{C}'(\tau)\|\kappa(\tau)\, d\tau. \tag{4.20}$$

where $\tilde{\phi}(u)$ describes the *change of the tangential angle* (compare Figure 4.9).

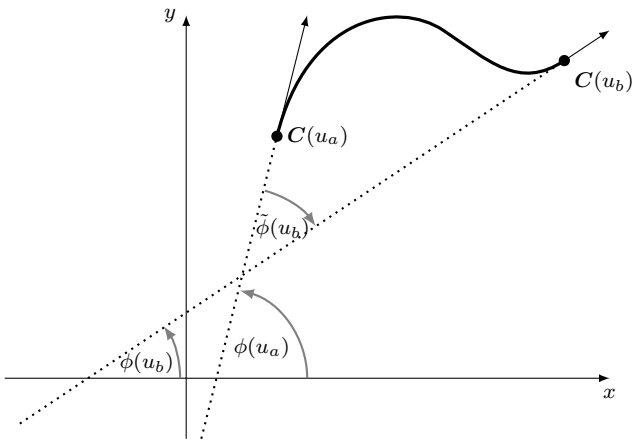

Figure 4.9: A curve $\boldsymbol{C}$ with the tangential angles $\phi(u_a)$ and $\phi(u_b)$ as well as the final change of the tangential angle $\tilde{\phi}(u_b)$.

Lemma 4.4. *Let $\boldsymbol{C} : [u_a, u_b] \to \mathbb{R}^2$, $u \mapsto \boldsymbol{C}(u)$ be a curve that is strictly bi-monotone according to Definition 4.1. Then the tangential angle $\phi(u)$ cannot change by more than $\frac{\pi}{2}$, i.e.*

$$|\tilde{\phi}(u)| \leq \frac{\pi}{2}$$

holds for all $u \in [u_a, u_b]$.

Proof. Let us first note that due to the fact that the curve is strictly bi-monontone we know that

$$C'(u)_m \neq 0$$

for all $u \in (u_a, u_b)$ and all $m \in \{1, 2\}$. Using (4.19) we can hence state that

$$\cos\phi(u) \neq 0 \qquad \text{and} \qquad \sin\phi(u) \neq 0 \tag{4.21}$$

must hold for all $u \in (u_a, u_b)$. Using the fact that $\cos x$ and $\sin x$ have a root at $x = k\pi + \frac{\pi}{2}$ and $x = k\pi$ respectively for all $k \in \mathbb{Z}$, so either of those is zero at any $x = k\frac{\pi}{2}$. Additionally since $\boldsymbol{C}'(u)$ is continuous for $u \in [u_a, u_b]$ the tangential angle $\phi(u)$ given by (4.19) is continuous as well. To fulfill (4.21) it then follows that

$$\exists \hat{k} \qquad \text{s.t.} \qquad \forall u \in [u_a, u_b], \quad \phi(u) \in [\hat{k}\frac{\pi}{2}, (\hat{k}+1)\frac{\pi}{2}]$$

holds. Given that choice for $\hat{k}$ and due to $\phi(u) = \phi(u_a) + \tilde{\phi}(u)$ we can further state that

$$\hat{k}\frac{\pi}{2} \leq \phi(u_a) + \tilde{\phi}(u) \leq (\hat{k}+1)\frac{\pi}{2} \qquad \forall u \in [u_a, u_b]$$

must hold and thus it must finally hold that

$$|\tilde{\phi}(u)| \leq \frac{\pi}{2} \qquad \forall u \in [u_a, u_b].$$

□

In the following lemma we need to distinguish whether a point is to the left or to the right of a given oriented line segment. Let us hence define the well-known 2D-orientation test

$$\gamma(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{x}) := \begin{vmatrix} a_1 - x_1 & a_2 - x_2 \\ b_1 - x_1 & b_2 - x_2 \end{vmatrix} \tag{4.22}$$

that returns a negative value if the point $\boldsymbol{x}$ is located to the left of the directed line $(\boldsymbol{a}, \boldsymbol{b})$, a positive value if it is located to the right and zero if $\boldsymbol{a}$, $\boldsymbol{b}$ and $\boldsymbol{x}$ all lie on the same line.

Lemma 4.5. *Let $\boldsymbol{C} : [u_a, u_b] \to \mathbb{R}^2$, $u \mapsto \boldsymbol{C}(u)$ be a curve that is strictly bi-monotone according to Definition 4.1 and has constant sign curvature according to Definition 4.2. Let $\boldsymbol{p}_a := \boldsymbol{C}(u_a)$ be the point where the curve starts and $\boldsymbol{q}_a := \boldsymbol{p}_a + \boldsymbol{C}'(u_a)$ the point at the end of the tangent vector applied to $\boldsymbol{p}_a$. Then a curve that turns left at at least one point, lies completely left to its initial tangent vector and a curve that turns right once, lies completely right to its initial tangent vector, i.e. the implications*

$$\begin{aligned} &\exists \tilde{u} \in [u_a, u_b] \ s.t. \ \kappa(\tilde{u}) > 0 \quad &\Longrightarrow \quad &\gamma(\boldsymbol{p}_a, \boldsymbol{q}_a, \boldsymbol{C}(u)) < 0 \quad \forall u \in (u_a, u_b] \quad \textit{and} \\ &\exists \tilde{u} \in [u_a, u_b] \ s.t. \ \kappa(\tilde{u}) < 0 \quad &\Longrightarrow \quad &\gamma(\boldsymbol{p}_a, \boldsymbol{q}_a, \boldsymbol{C}(u)) > 0 \quad \forall u \in (u_a, u_b] \end{aligned}$$

hold.

Proof. Let us first consider the case where the curve turns right at at least one point, so we are looking at the case where

$$\exists \tilde{u} \in [u_a, u_b] \text{ s.t. } \kappa(\tilde{u}) < 0.$$

Since the curve has constant sign curvature this is equivalent to

$$\kappa(u) \leq 0 \quad \forall u \in [u_a, u_b]$$

and thus the curve can only turn right everywhere. We now show that such a curve cannot reach any point to the left of the segment $(\boldsymbol{p}_a, \boldsymbol{q}_a)$ when it is additionally bi-monotone. Since $\boldsymbol{C}$ is differentiable for $u \in [u_a, u_b]$ we can express any point along the curve by

$$\boldsymbol{C}(u) = \boldsymbol{C}(u_a) + \int_{u_a}^{u} \boldsymbol{C}'(\tau) \, d\tau. \tag{4.23}$$

This shows that the curve can only progress in tangential direction and thus a necessary condition to reach any point to the left of the segment $(\boldsymbol{p}_a, \boldsymbol{q}_a)$ is that there is a tangent vector $\boldsymbol{C}'(u)$ that points to the left of $(\boldsymbol{p}_a, \boldsymbol{q}_a)$. By introducing (4.19) and (4.20) into (4.23), it can be reformulated to

$$\begin{aligned} \boldsymbol{C}(u) &= \boldsymbol{C}(u_a) + \int_{u_a}^{u} \|\boldsymbol{C}'(\tau)\| \begin{pmatrix} \cos\phi(\tau) \\ \sin\phi(\tau) \end{pmatrix} \, d\tau \\ &= \boldsymbol{C}(u_a) + \int_{u_a}^{u} \|\boldsymbol{C}'(\tau)\| \begin{pmatrix} \cos(\phi_0 + \tilde{\phi}(\tau)) \\ \sin(\phi_0 + \tilde{\phi}(\tau)) \end{pmatrix} \, d\tau. \end{aligned}$$

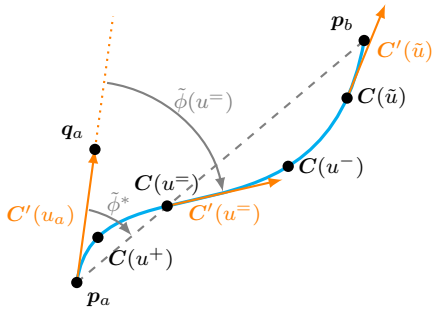

Figure 4.10: A bi-monotone curve $\boldsymbol{C}$ between two points $\boldsymbol{p}_a$ and $\boldsymbol{p}_b$ where there is a point $\boldsymbol{C}(u^+)$ that is to the left of $(\boldsymbol{p}_a, \boldsymbol{p}_b)$ and a point $\boldsymbol{C}(u^-)$ that is to the right of $(\boldsymbol{p}_a, \boldsymbol{p}_b)$ cannot have constant sign curvature.

Hence, to get a tangent vector that points to the left of $(\boldsymbol{p}_a, \boldsymbol{q}_a)$ we need that there exists a $\hat{u} \in [u_a, u_b]$ such that $\tilde{\phi}(\hat{u}) \in (2k\pi, 2k\pi + \pi)$ for any $k \in \mathbb{Z}$. Since $\tilde{\phi}(u)$ is continuous and $\tilde{\phi}(u_a) = 0$ it is sufficient to consider the interval for $k = 0$, i.e. $(0, \pi)$, in the following. Let us recall the definition of $\tilde{\phi}(u)$ from (4.20)

$$\tilde{\phi}(u) := \int_{u_a}^{u} \|\boldsymbol{C}'(\tau)\| \kappa(\tau) \, d\tau.$$

We can clearly see that $\tilde{\phi}(u)$ is a monotone decreasing function when $\kappa(u) \leq 0$ and due to Lemma 4.4 we can then state that

$$-\frac{\pi}{2} \leq \tilde{\phi}(u) \leq 0$$

holds for all $u \in [u_a, u_b]$ which in particular means there is no $\hat{u} \in [u_a, u_b]$ such that $\tilde{\phi}(\hat{u}) \in (0, \pi)$ and hence we cannot reach any point to the left of the segment $(\boldsymbol{p}_a, \boldsymbol{q}_a)$. The case were $\exists \tilde{u} \in [u_a, u_b]$ s.t. $\kappa(\tilde{u}) > 0$ follows by symmetry. □

Lemma 4.6. *Let $\boldsymbol{C} : [u_a, u_b] \to \mathbb{R}^2$, $u \mapsto \boldsymbol{C}(u)$ be a curve between the points $\boldsymbol{p}_a := \boldsymbol{C}(u_a)$ and $\boldsymbol{p}_b := \boldsymbol{C}(u_b)$ that is bi-monotone according to Definition 4.1 and has constant sign curvature according to Definition 4.2. Let $\boldsymbol{q}_a := \boldsymbol{p}_a + \boldsymbol{C}'(u_a)$ be the point at the end of the tangent vector applied to $\boldsymbol{p}_a$. Then, if $\boldsymbol{q}_a$ is to the left of $(\boldsymbol{p}_a, \boldsymbol{p}_b)$, the complete curve is to the left of $(\boldsymbol{p}_a, \boldsymbol{p}_b)$ and if $\boldsymbol{q}_a$ is to the right of $(\boldsymbol{p}_a, \boldsymbol{p}_b)$, the complete curve is to the right of $(\boldsymbol{p}_a, \boldsymbol{p}_b)$, i.e. the implications*

$$\begin{aligned} \gamma(\boldsymbol{p}_a, \boldsymbol{p}_b, \boldsymbol{q}_a) < 0 \quad &\Longrightarrow \quad \gamma(\boldsymbol{p}_a, \boldsymbol{p}_b, \boldsymbol{C}(u)) < 0 \quad \forall u \in (u_a, u_b) \quad \textit{and} \\ \gamma(\boldsymbol{p}_a, \boldsymbol{p}_b, \boldsymbol{q}_a) > 0 \quad &\Longrightarrow \quad \gamma(\boldsymbol{p}_a, \boldsymbol{p}_b, \boldsymbol{C}(u)) > 0 \quad \forall u \in (u_a, u_b) \end{aligned}$$

hold.

Proof. A graphical representation that corresponds to the notation introduced in the following proof is depicted in Figure 4.10. Let us first consider the case where the initial

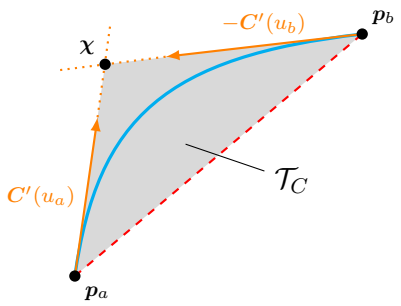

Figure 4.11: A curve $\boldsymbol{C}$ that is bi-monotone and has constant sign curvature between the points $\boldsymbol{p}_a := \boldsymbol{C}(u_a)$ and $\boldsymbol{p}_b := \boldsymbol{C}(u_b)$ is contained in the triangle $\mathcal{T}_C$ defined by the points $(\boldsymbol{p}_a, \boldsymbol{p}_b, \boldsymbol{\chi})$ where $\boldsymbol{\chi}$ is the point at which the extensions of the tangent vectors $\boldsymbol{C}'(u_a)$ and $-\boldsymbol{C}'(u_b)$ intersect.

tangent $\boldsymbol{C}'(u_a)$ points to the left of $(\boldsymbol{p}_a, \boldsymbol{p}_b)$, i.e. $\gamma(\boldsymbol{p}_a, \boldsymbol{p}_b, \boldsymbol{q}_a) < 0$. This is equal to the statement that the point $\boldsymbol{p}_b$ is to the right of $(\boldsymbol{p}_a, \boldsymbol{q}_a)$ and by an argumentation along the lines of the proof of Lemma 4.5 it is

$$-\frac{\pi}{2} \leq \tilde{\phi}(u) \leq 0 \qquad \forall u \in [u_a, u_b]$$

since we need at least one tangent vector that points to the right of $(\boldsymbol{p}_a, \boldsymbol{q}_a)$ to reach the point $\boldsymbol{p}_b$. Then, let $\tilde{\phi}^* \in (-\pi, 0)$ be the angle between the vectors $\boldsymbol{C}'(u_a)$ and $\boldsymbol{p}_b - \boldsymbol{p}_a$. Now, due to the initial tangent $\boldsymbol{C}'(u_a)$ pointing to the left of $(\boldsymbol{p}_a, \boldsymbol{p}_b)$ there is at least one u^- such that its point on the curve $\boldsymbol{C}(u^-)$ is to the left of $(\boldsymbol{p}_a, \boldsymbol{p}_b)$, i.e. where $\gamma(\boldsymbol{p}_a, \boldsymbol{p}_b, \boldsymbol{C}(u^-)) < 0$ holds. Let us then additionally assume that there is a u^+ where the corresponding point $\boldsymbol{C}(u^+)$ is to the right, i.e. where $\gamma(\boldsymbol{p}_a, \boldsymbol{p}_b, \boldsymbol{C}(u^+)) > 0$ holds. Due to the curve being continuous, the intermediate value theorem states that there has to be at least one $u^= \in (u^-, u^+)$ where the point $\boldsymbol{C}(u^=)$ is on the segment $(\boldsymbol{p}_a, \boldsymbol{p}_b)$, i.e. where $\gamma(\boldsymbol{p}_a, \boldsymbol{p}_b, \boldsymbol{C}(u^=)) = 0$ holds and to be able to actually reach the point $\boldsymbol{C}(u^+)$ from such an $\boldsymbol{C}(u^=)$, we need the tangent $\boldsymbol{C}'(u^=)$ to point to the right of $(\boldsymbol{p}_a, \boldsymbol{p}_b)$, i.e. $\tilde{\phi}(u^=) < \tilde{\phi}^*$ holds for the tangential angle at $u^=$. On the other hand, to be able to reach the point $\boldsymbol{p}_b$ from $\boldsymbol{C}(u^+)$ we need that there is a $\tilde{u} \in [u^+, u_b)$ such that $\tilde{\phi}(\tilde{u}) > \tilde{\phi}^*$. But since $u^= < u^+ \leq \tilde{u}$, the statements $\tilde{\phi}(u^=) < \tilde{\phi}^*$ and $\tilde{\phi}(\tilde{u}) > \tilde{\phi}^*$ contradict that the change of the tangential angle $\tilde{\phi}$ according to (4.20) is monotone for curves with constant sign curvature. Hence, the assumption that there is a point $\boldsymbol{C}(u^+)$ that is to the right of $(\boldsymbol{p}_a, \boldsymbol{p}_b)$ does not hold. The case where $\gamma(\boldsymbol{p}_a, \boldsymbol{p}_b, \boldsymbol{q}_a) > 0$ follows by symmetry. □

Theorem 4.1. *Let $\boldsymbol{C} : [u_a, u_b] \to \mathbb{R}^2$, $u \mapsto \boldsymbol{C}(u)$ be a curve between the points $\boldsymbol{p}_a := \boldsymbol{C}(u_a)$ and $\boldsymbol{p}_b := \boldsymbol{C}(u_b)$ that is bi-monotone according to Definition 4.1 and has constant sign curvature according to Definition 4.2. Let $\boldsymbol{\chi}$ be the intersection point of the infinitely extended tangent vectors at $\boldsymbol{p}_a$ and $\boldsymbol{p}_b$, i.e. where*

$$\boldsymbol{\chi} := \boldsymbol{p}_a + \alpha \boldsymbol{C}'(u_a) = \boldsymbol{p}_b - \beta \boldsymbol{C}'(u_a)$$

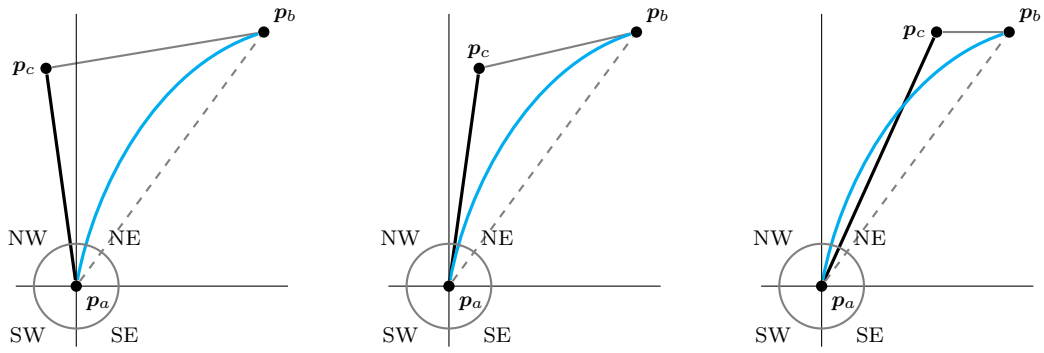

Figure 4.12: A triangle $(\boldsymbol{p}_a, \boldsymbol{p}_b, \boldsymbol{p}_c)$. The constrained edge $(\boldsymbol{p}_a, \boldsymbol{p}_b)$ (dashed) has been replaced by the curve $\boldsymbol{C}$ (blue) that is bi-monotone and has constant sign curvature. The space around the point $\boldsymbol{p}_a$ has been divided into four sectors NE, NW, SE and SW. The point $\boldsymbol{p}_b$ is located in the NE sector relative to $\boldsymbol{p}_a$.
Left: The point $\boldsymbol{p}_c$ is located in the NW sector relative to $\boldsymbol{p}_a$ and thus cannot intersect the curved edge.
Center: The point $\boldsymbol{p}_c$ is located in the NE sector relative to $\boldsymbol{p}_a$ but does still not intersect the curved edge.
Right: The point $\boldsymbol{p}_c$ is located in the NE sector relative to $\boldsymbol{p}_a$ and does intersect the curved edge.

holds for some $\alpha, \beta \in [0, \infty)$. *Then the complete curve* $\boldsymbol{C}$ *lies within the closed triangle* $\mathcal{T}_{\boldsymbol{C}}$ *formed by the points* $(\boldsymbol{p}_a, \boldsymbol{p}_b, \boldsymbol{\chi})$ *(compare Figure 4.11).*

Proof. In this proof, we show that the curve $\boldsymbol{C}$ is bounded by the segments $(\boldsymbol{p}_a, \boldsymbol{\chi})$ and $(\boldsymbol{p}_b, \boldsymbol{\chi})$ defined by the tangents at its endpoints. Due to Lemma 4.6 it is additionally bounded by $(\boldsymbol{p}_a, \boldsymbol{p}_b)$ and hence it lies within the triangle $(\boldsymbol{p}_a, \boldsymbol{p}_b, \boldsymbol{\chi})$.
Let us first assume that $\boldsymbol{p}_b$ is to the right of the segment $(\boldsymbol{p}_a, \boldsymbol{\chi})$. We did already state in the proof of Lemma 4.5 that for a curve to reach a point that is to the right of $(\boldsymbol{p}_a, \boldsymbol{q}_a)$ with $\boldsymbol{q}_a := \boldsymbol{p}_a + \boldsymbol{C}'(u_a)$ and hence right of $(\boldsymbol{p}_a, \boldsymbol{\chi})$, it needs to turn right at least once, so there has to exist at least one $\tilde{u} \in [u_a, u_b]$ such that $\kappa(\tilde{u}) < 0$. Hence, from Lemma 4.5 we know that the complete curve has to be right of $(\boldsymbol{p}_a, \boldsymbol{\chi})$. Let us then consider the reversed curve $\boldsymbol{C}_\mathrm{R}$, $t \mapsto \boldsymbol{C}_\mathrm{R}(t)$ with $t \in [-u_b, -u_a]$ and $t = -u$ such that $\boldsymbol{C}_\mathrm{R}(t) = \boldsymbol{C}(u)$. Since that reversed curve starts at $\boldsymbol{p}_b$ and ends in $\boldsymbol{p}_a$ and with $\boldsymbol{C}'_\mathrm{R}(-u_b) = -\boldsymbol{C}'(u_b)$ we can analogously apply Lemma 4.5 to state that the complete curve has to be left of $(\boldsymbol{p}_b, \boldsymbol{\chi})$.
The case where we initially assume that $\boldsymbol{p}_b$ is to the left of the segment $(\boldsymbol{p}_a, \boldsymbol{\chi})$ follows by symmetry. □

Let us now discuss how we can use Theorem 4.1 to implement a fast test whether a given triangle $(\boldsymbol{p}_a, \boldsymbol{p}_b, \boldsymbol{p}_c)$ with one or more curved edges is self-intersecting or inverted as depicted in Figure 4.8. Let us assume that the edge $(\boldsymbol{p}_a, \boldsymbol{p}_b)$ of the triangle needs to be replaced by the curve $\boldsymbol{C}$. Then we need to check whether any of the segments $(\boldsymbol{p}_a, \boldsymbol{p}_c)$ or $(\boldsymbol{p}_b, \boldsymbol{p}_c)$ does intersect the curve or whether $\boldsymbol{p}_c$ is located between $(\boldsymbol{p}_a, \boldsymbol{p}_b)$ and the curve.

A naive implementation would be to compute the intersection point $\boldsymbol{\chi}$ of the tangents of $\boldsymbol{C}$ at $\boldsymbol{p}_a$ and $\boldsymbol{p}_b$ to form the triangle $\mathcal{T}_{\boldsymbol{C}}$ as given in Figure 4.11. Then we can test whether any of the segments $(\boldsymbol{p}_a, \boldsymbol{p}_c)$ or $(\boldsymbol{p}_b, \boldsymbol{p}_c)$ intersects that triangle. Note that in the case that $\boldsymbol{p}_c$ lies within the triangle $\mathcal{T}_{\boldsymbol{C}}$, either $(\boldsymbol{p}_a, \boldsymbol{p}_c)$ or $(\boldsymbol{p}_b, \boldsymbol{p}_c)$ intersect the curve, or $\boldsymbol{p}_c$ is located between the curve and $(\boldsymbol{p}_a, \boldsymbol{p}_b)$ and hence the triangle is inverted. We present a slightly simpler solution that does not require to compute $\boldsymbol{\chi}$ and avoids a full segment-triangle intersection test.
We first assume that we are looking at a curve $\boldsymbol{C}$ with $u \in [u_a, u_b]$ between the points $\boldsymbol{p}_a = \boldsymbol{C}(u_a)$, $\boldsymbol{p}_b = \boldsymbol{C}(u_b)$ and we want to check whether a segment $(\boldsymbol{p}_a, \boldsymbol{p}_c)$ intersects the curve $\boldsymbol{C}$. To this end, we define a classification function

$$\sigma(a, b) := \begin{cases} \text{NE} & \text{if } b_1 \geq a_1 \text{ and } b_2 \geq a_2 \\ \text{NW} & \text{if } b_1 < a_1 \text{ and } b_2 \geq a_2 \\ \text{SE} & \text{if } b_1 \geq a_1 \text{ and } b_2 < a_2 \\ \text{SW} & \text{if } b_1 < a_1 \text{ and } b_2 < a_2 \end{cases}$$

that determines in which sector (NE, NW, SE or SW) a point b is located relative to a point a. Due to Lemma 4.1 and Remark 4.2 we do know that the complete curve is in the axis-aligned bounding box of $\boldsymbol{p}_a$ and $\boldsymbol{p}_b$, which means it is completely within one sector relative to $\boldsymbol{p}_a$ and thus

$$\sigma(\boldsymbol{p}_a, \boldsymbol{C}(u)) = \sigma(\boldsymbol{p}_a, \boldsymbol{p}_b)$$

holds for all $u \in (u_a, u_b]$. Hence, a necessary condition for the curve to have an intersection with the edge $(\boldsymbol{p}_a, \boldsymbol{p}_c)$ or for $\boldsymbol{p}_c$ to be within $\mathcal{T}_{\boldsymbol{C}}$ is that

$$\sigma(\boldsymbol{p}_a, \boldsymbol{p}_c) = \sigma(\boldsymbol{p}_a, \boldsymbol{p}_b) \tag{4.24}$$

holds (compare Figure 4.12). Equation (4.24) gives us a first fast test to dismiss cases where $\boldsymbol{p}_c$ is located relative to $\boldsymbol{p}_a$ such that an invalid case is impossible. We are now left to distinguish the cases where $\boldsymbol{p}_c$ and $\boldsymbol{p}_b$ are located in the same sector relative to $\boldsymbol{p}_a$. To this end we want to check whether the point $\boldsymbol{p}_c$ is in-between the edges $(\boldsymbol{p}_a, \boldsymbol{p}_b)$ and $(\boldsymbol{p}_a, \boldsymbol{q}_a)$. If this is the case the segment $(\boldsymbol{p}_a, \boldsymbol{p}_c)$ does definitely intersect $\mathcal{T}_{\boldsymbol{C}}$ and hence we have detected an invalid case (compare Figure 4.13). We employ the orientation test from (4.22) with which

$$\gamma(\boldsymbol{p}_a, \boldsymbol{p}_b, \boldsymbol{p}_c) \cdot \gamma(\boldsymbol{p}_a, \boldsymbol{q}_a, \boldsymbol{p}_c) < 0 \tag{4.25}$$

returns whether the point $\boldsymbol{p}_c$ is in-between of the segments $(\boldsymbol{p}_a, \boldsymbol{p}_b)$ and $(\boldsymbol{p}_a, \boldsymbol{q}_a)$ and thus intersects $\mathcal{T}_{\boldsymbol{C}}$. We can now additionally check the other end of the curve by swapping $\boldsymbol{p}_a$ and $\boldsymbol{p}_b$ and use $\boldsymbol{q}_b = \boldsymbol{p}_b - \boldsymbol{C}'(u_b)$ instead of $\boldsymbol{q}_a$ in (4.24) and (4.25) and so we are finally able to detect all invalid triangles with a curved edge.
We are now left to resolve all those invalid triangles. Given the curve $\boldsymbol{C}$ between $\boldsymbol{p}_a$ and $\boldsymbol{p}_b$ that is part of an invalid, curved triangle. We first remove the constrained edge $(\boldsymbol{p}_a, \boldsymbol{p}_b)$ from the triangulation. Then we insert a new point $\boldsymbol{q} := \boldsymbol{C}(\tilde{u})$ at the center of the parametrization $\tilde{u} = \frac{u_a + u_b}{2}$ into the triangulation and insert the edges $(\boldsymbol{p}_a, \boldsymbol{q})$ and $(\boldsymbol{q}, \boldsymbol{p}_b)$ as new constraints (compare Figure 4.14). Note that any of those two new constraints can

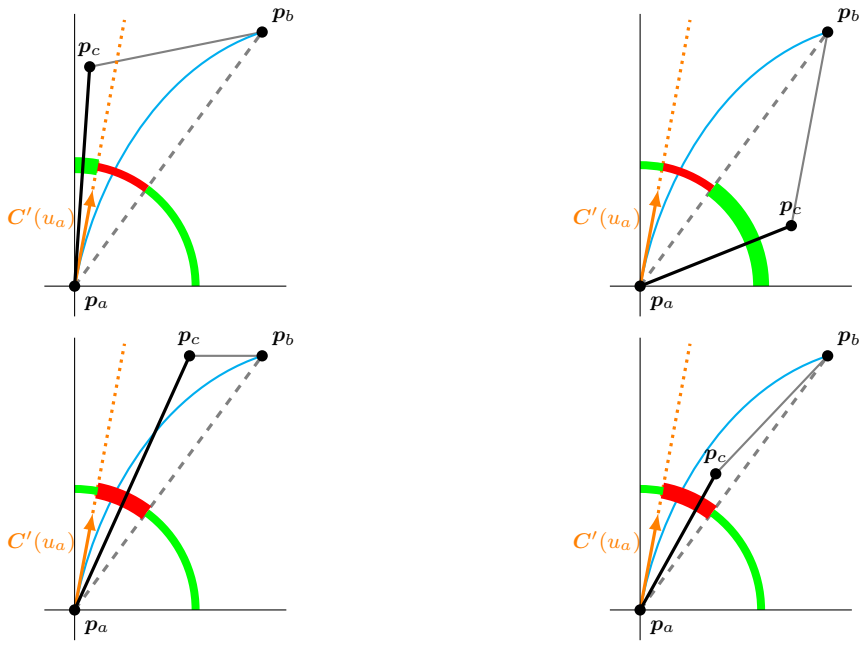

Figure 4.13: A triangle $(\boldsymbol{p}_a, \boldsymbol{p}_b, \boldsymbol{p}_c)$. The constrained edge $(\boldsymbol{p}_a, \boldsymbol{p}_b)$ (dashed) has been replaced by the curve $\boldsymbol{C}$ (blue) that is bi-monotone and has constant sign curvature. The NE sector around the point $\boldsymbol{p}_a$ has been divided into three smaller sectors. The space between the edge $(\boldsymbol{p}_a, \boldsymbol{p}_b)$ and the curve's tangent vector $\boldsymbol{C}'(u_a)$ (red) and the remaining space (green) between the x-axis and the edge $(\boldsymbol{p}_a, \boldsymbol{p}_b)$ and the tangent vector and the y-axis.
Upper: The point $\boldsymbol{p}_c$ is located in the green zones, either left of the tangent vector (left) or right of the edge $(\boldsymbol{p}_a, \boldsymbol{p}_b)$ (right).
Lower: The point $\boldsymbol{p}_c$ is located in the red zone, either above the curve $\boldsymbol{C}$ which results in an intersection with the edge $(\boldsymbol{p}_a, \boldsymbol{p}_c)$ (left) or below the curve where the whole triangle is inverted (right).

intersect any other already added constraint, which will be resolved by adding even more points along either the curve $\boldsymbol{C}$ or the curves belonging to the other constraints, just like it was done when creating the initial constrained triangulation. When we are done and we have a valid linear constrained triangulation, we check all curve parts that belong to newly added constrained edges for an invalid triangle by (4.24) and (4.25) and continue to split the curves until all triangles are valid. Even though it seems very likely, it is still open to prove that this algorithm will terminate at some point. Since we cannot guarantee the termination yet, we stop trying to curve an edge after 50 subdivisions and then just keep linear edges for the remaining curve parts, that are still next to invalid triangles. Note that after 50 subdivisions the diameter of the parameter range $[u_a, u_b]$ of any remaining curve part is $\frac{1}{2^{50}} \approx 9 \cdot 10^{-16}$ of the original curve's parameter range which is close to the epsilon of a double precision floating-point number. In practice we have never encountered

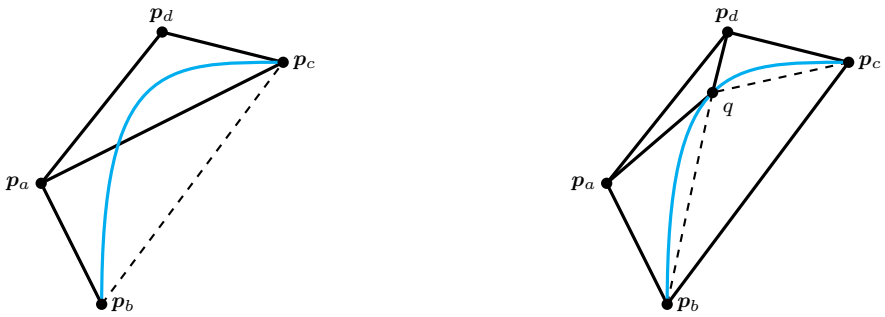

Figure 4.14: *Left:* Two triangles $(\boldsymbol{p}_a, \boldsymbol{p}_b, \boldsymbol{p}_c)$ and $(\boldsymbol{p}_a, \boldsymbol{p}_b, p_d)$ with the edge $(\boldsymbol{p}_a, \boldsymbol{p}_b)$ being replaced by the curve $\boldsymbol{C}$ (blue) that intersects the inner edge $(\boldsymbol{p}_b, \boldsymbol{p}_c)$.
Right: A new point q has been inserted along the curve $\boldsymbol{C}$. After the insertion the edges $(\boldsymbol{p}_a, q)$ and $(q, \boldsymbol{p}_b)$ have been added as constraints and the Delaunay property has been restored. The self-intersection is resolved.

any case where more than 3 iterations were necessary to resolve all invalid triangles.

4.3 Quadrature by Transfinite Interpolation

Now that we can compute the intersections of the integration domains $\mathcal{D}_i^m$ with the simulation domain Ω and decompose the resulting domains into potentially curved triangles $\mathcal{T}_i^n$, we are left to find quadrature rules for those triangles. Let us drop the patch index i and the specific triangle index n in the following. Given a transformation $\boldsymbol{\Psi} : \mathcal{R} \to \mathcal{T}$ that transforms any point $\boldsymbol{\xi}$ from some reference domain $\mathcal{R} \subset \mathbb{R}^2$ into the target triangle $\mathcal{T}$, we can carry out the integration of any function $f : \mathcal{T} \to \mathbb{R}$, $\boldsymbol{x} \mapsto f(\boldsymbol{x})$ in the reference domain $\mathcal{R}$ by

$$\int_{\mathcal{T}} f(\boldsymbol{x})\ d\boldsymbol{x} = \int_{\mathcal{R}} f(\boldsymbol{\Psi}(\boldsymbol{\xi})) \cdot |\det J_{\boldsymbol{\Psi}}(\boldsymbol{\xi})|\ d\boldsymbol{\xi} \tag{4.26}$$

where $J_{\boldsymbol{\Psi}}$ is the Jacobian of the transformation $\boldsymbol{\Psi}$. If we then choose a reference domain $\mathcal{R}$, where we have a known quadrature rule $Q_{\mathcal{R}}^n := \{(\boldsymbol{\xi}_i, w_i)\}_{i=1}^n$ consisting of n quadrature points $\boldsymbol{\xi}_i \in \mathcal{R}$ and weights $w_i \in \mathbb{R}$ we can approximate the integral by

$$\int_{\mathcal{T}} f(\boldsymbol{x})\ d\boldsymbol{x} \approx \sum_{i=1}^{n} f(\boldsymbol{\Psi}(\boldsymbol{\xi}_i)) \cdot w_i |\det J_{\boldsymbol{\Psi}}(\boldsymbol{\xi}_i)|. \tag{4.27}$$

Assuming that the target function f as well as the transformation $\boldsymbol{\Psi}$ can be described by polynomials, we can usually find an n-point quadrature rule $Q_{\mathcal{R}}^n$ that yields an exact result and hence (4.27) becomes an actual equality. To this end, we need a quadrature rule that is capable of integrating polynomials up to a degree

$$\begin{aligned} \bar{p} :=\ & \deg\Big(f(\boldsymbol{\Psi}(\boldsymbol{\xi})) \cdot |\det J_{\boldsymbol{\Psi}}(\boldsymbol{\xi})|\Big) \\ =\ & \deg(f)\deg(\boldsymbol{\Psi}) + \deg(\det J_{\boldsymbol{\Psi}}) \end{aligned} \tag{4.28}$$

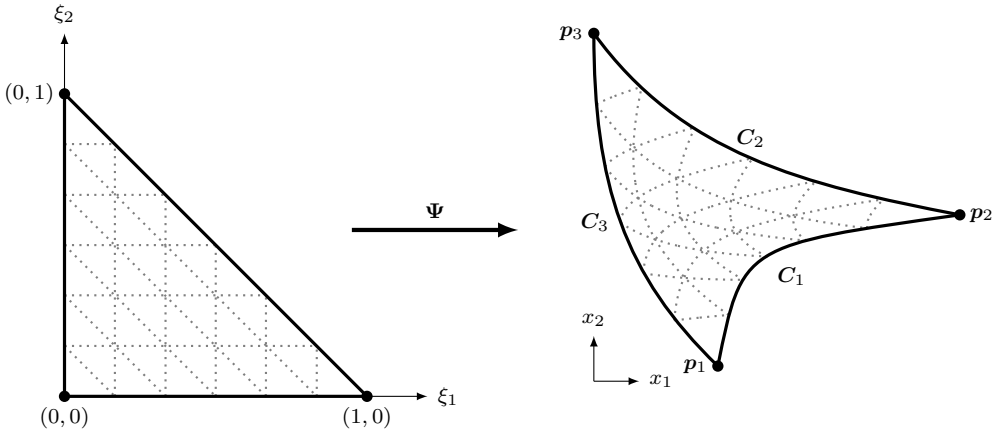

Figure 4.15: The transformation $\boldsymbol{\Psi}$ transforms points from $\boldsymbol{\xi} \in \mathcal{R}$ described by $\big((0,0),(1,0),(0,1)\big)$ to a curved triangle $(\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3)$ with the curved edges $\boldsymbol{C}_1$, $\boldsymbol{C}_2$ and $\boldsymbol{C}_3$.

exactly on the reference domain $\mathcal{R}$.
In our setting $\mathcal{T}$ is a triangle described by the points $\boldsymbol{p}_1$, $\boldsymbol{p}_2$ and $\boldsymbol{p}_3$ with potentially curved edges $\boldsymbol{C}_1$, $\boldsymbol{C}_2$ and $\boldsymbol{C}_3$. Hence a natural choice for the reference domain is a simple, linear unit triangle described by the points $(0,0)$, $(1,0)$ and $(0,1)$ (compare Figure 4.15). There are many sources for quadrature rules defined over this (or similar) linear triangles. E.g. rules to integrate polynomials up to degree 50 with only positive weights w_i and where all quadrature points $\boldsymbol{\xi}_i$ are in the interior of the reference triangle can be found in [135]. We construct the transformation $\boldsymbol{\Psi}$ by a method denoted as *transfinite interpolation* which was introduced in [46] and is highly related to the well-known *Coons patches* [23]. The specific transformation for curved triangles given below was introduced in [20] and [93] and has been used to generate quadrature point in the PUM in [26]. We start by defining the barycentric coordinates of a point $\boldsymbol{\xi}$ in the reference triangle by

$$\boldsymbol{\lambda}(\boldsymbol{\xi}) = (\lambda_1, \lambda_2, \lambda_3)^T := (1 - \xi_1 - \xi_2, \xi_1, \xi_2)^T.$$

Then, let $\boldsymbol{C}_i : [u_{i,a}, u_{i,b}] \to \mathbb{R}^2$ with $i \in \{1,2,3\}$ and $\boldsymbol{C}_i(u_{i,a}) = \boldsymbol{p}_1$, $\boldsymbol{C}_i(u_{i,b}) = \boldsymbol{p}_{(i \bmod 3)+1}$ be the parametrized curves that represent the i-th edges of $\mathcal{T}$. We first introduce the mappings $g_i : [0,1] \to [u_{i,a}, u_{i,b}]$, $\hat{u} \mapsto u_{i,a} + (u_{i,b} - u_{i,a})\hat{u}$ that allow us to define the curves $\hat{C}_i := \boldsymbol{C}_i \circ g_i$ which can be evaluated over the unit interval $[0,1]$. Then the transfinite interpolation from the reference triangle $\mathcal{R}$ to $\mathcal{T}$ is given by

$$\begin{aligned}\boldsymbol{\Psi}^{\mathrm{TFI}}(\boldsymbol{\lambda}) := \lambda_1(\hat{C}_1(\lambda_2) + \hat{C}_3(1-\lambda_3) - \boldsymbol{p}_1) \\ +\lambda_2(\hat{C}_2(\lambda_3) + \hat{C}_1(1-\lambda_1) - \boldsymbol{p}_2) \\ +\lambda_3(\hat{C}_3(\lambda_1) + \hat{C}_2(1-\lambda_2) - \boldsymbol{p}_3).\end{aligned} \tag{4.29}$$

Assuming that all curves $\boldsymbol{C}_i$ are representable by polynomials we can introduce

$$p_{\mathcal{T}} := \max\{\deg(\boldsymbol{C}_1), \deg(\boldsymbol{C}_2), \deg(\boldsymbol{C}_3)\}$$

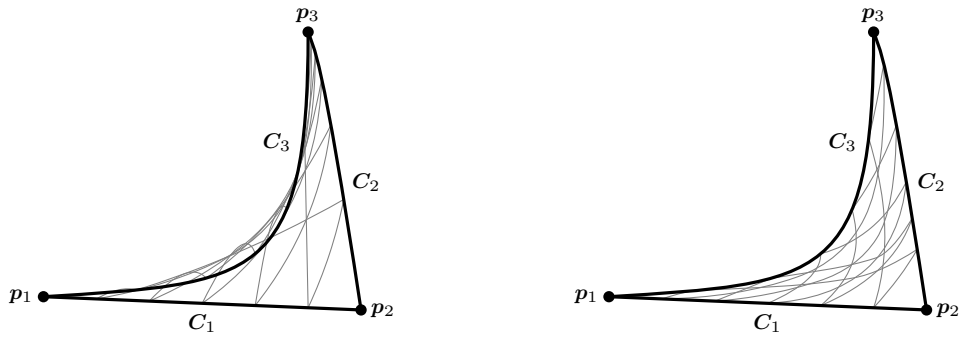

Figure 4.16: *Left:* The transfinite interpolation $\boldsymbol{\Psi}^{\mathrm{TFI}}$ is not a diffeomorphism, due to the parametrization of the curve $\boldsymbol{C}_2$ being concentrated towards the point $\boldsymbol{p}_3$.
Right: The curve $\boldsymbol{C}_2$ has been reparameterized to be close to arc-length parametrized. The "overspill" has been resolved.

as the highest polynomial degree of those curves. Using $p_{\mathcal{T}}$, the polynomial degree of the transfinite interpolation and its Jacobian are given by

$$\begin{aligned} \deg(\boldsymbol{\Psi}^{\mathrm{TFI}}) &= p_{\mathcal{T}} + 1 \\ \deg(\det J_{\boldsymbol{\Psi}^{\mathrm{TFI}}}) &= p_{\mathcal{T}}. \end{aligned} \tag{4.30}$$

By combining (4.28) and (4.30) we can state that to integrate a polynomial function f over the curved triangle $\mathcal{T}$ exactly we need to employ a quadrature rule for polynomials of degree

$$\bar{p}^{\mathrm{TFI}} := \deg(f)(p_{\mathcal{T}} + 1) + p_{\mathcal{T}}. \tag{4.31}$$

Note that (4.31) gives an upper bound for the worst-case. E.g. we know that if all curves $\boldsymbol{C}_i$ are actually just straight line segments, i.e. $\deg(\boldsymbol{C}_i) = 1$ for all $i \in \{1, 2, 3\}$ and hence $p_{\mathcal{T}} = 1$, it is sufficient to use a quadrature rule for polynomials of degree $\deg(f)$, but (4.31) would suggest that we require a rule for polynomials of degree $2\deg(f) + 1$. Additionally (4.31) does only give a bound when all curves $\boldsymbol{C}_i$ are actually described by polynomials, but the NURBS formulation allows $\boldsymbol{C}_i$ to be described by rational basis functions as well. Simple tests indicated that if $\mathcal{T}$ consists of two straight edges and one curved edge that describes the quarter of a circle, i.e. a rational NURBS curve, a quadrature rule for polynomials of degree $\deg(f) + 19$ was required to achieve an integration accuracy up to machine precision.[4] Nevertheless, in practice it is usually not required to integrate up to machine precision accuracy and hence much lower quadrature rules can be employed.

Remark 4.3. For equation (4.26) to be valid, it is actually required that $\boldsymbol{\Psi}$ is a diffeomorphism. This does especially mean that $\boldsymbol{\Psi}$ needs to be invertible, i.e. $\det J_{\boldsymbol{\Psi}}(\boldsymbol{\xi}) \neq 0$ for all $\boldsymbol{\xi} \in \mathcal{R}$ needs to hold. While this is hard to proof in general, some of the most common cases where a transformation $\boldsymbol{\Psi}^{\mathrm{TFI}}$ as given in (4.29) is not invertible is when the curves $\boldsymbol{C}_i$ do intersect or when they are "too wavy". Both of those cases are mitigated in our

[4] Using IEEE double precision floating-point values.

scenario. Intersections of two input curves $\boldsymbol{C}_i$ are forbidden by definition in the STEP standard, and cases where that definition is violated are usually resolved as described in Section 4.2.1 and hence the curves that end up in a single triangle do not intersect. Cases where an input curve $\boldsymbol{C}_i$ is intersected by an internal edge of the triangulation or where a curved edge inverts the whole triangle were the main topic throughout Section 4.2.3 and are thus not present in the final integration cells. The "waviness" of the curved edges is additionally restricted, since the change of the tangential angle is bounded by $\frac{\pi}{2}$ (see Lemma 4.4) and the curve parts do have a constant sign curvature. To the best of our knowledge the only case left where $\boldsymbol{\Psi}^{\text{TFI}}$ might be non-invertible is when any involved curve is far away from being *arc-length* parametrized[5] (compare the left of Figure 4.16). These cases could be resolved by reparameterizing the curves to arc-length. Usually an approximate arc-length parametrization is sufficient.[6] One method to compute such a parametrization is given in [133] and an application of that reparameterization to the invalid case depicted on the left of Figure 4.16 is shown on the right of the same figure. To the best of our knowledge such extreme cases are very uncommon in practice and hence we do not apply the reparameterization in any of our examples and no problems due to the "overspill" phenomena have been observed in any of them. Note that even when the reparameterization is to be applied, an approximate arc-length parametrization does only need to be computed for all curves of the input geometry once in a pre-processing step and can be reused for all curved triangles that include those curves.

4.4 Numerical Experiments

In this section we take a look at numerical examples to validate that the methods presented in this chapter give the expected results. To this end, we are mainly concerned with the approximation properties of the involved function spaces and the robustness of geometry operations in general and the numerical integration in particular. We start by measuring error norms as given in (2.4) and convergence rates as given in (2.5) against analytical and numerical reference solutions. Then we continue to show that the approach presented is performant, even when dealing with complicated geometries. Finally we conduct an experiment on an industrial grade CAD geometry to show the robustness when confronted with real-world geometries. During all experiments in this section we did employ the cover post-processing step from Chapter 2 to guarantee linear independent function spaces V^{PU}.

Example 4.1 (Curved half-circle domain)**.** In this example we show that our PUM does convergence with optimal rates to a smooth solution, even when we consider a non-linear domain. To this end, we again consider the Helmholtz type of equation from (2.6) with $c = 1$ and choose the right-hand side function such that the reference solution is given by (2.7). But this time we use a domain Ω that has one curved edge. In particular we use a

[5] An arc-length parametrization of a curve is when moving a distance δu in the parametrization of the curve is equal to moving the same distance along the curve in the world space.

[6] In general, it is impossible to give an exact arc-length parametrization by rational functions for real rational curves other than the straight line [38].

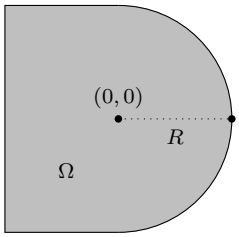

Figure 4.17: Curved simulation domain where the right edge of a rectangle $[-R, 0] \times [-R, R]$ has been replaced by a half-circle around $(0, 0)$ with radius $R = 15$.

Table 4.1: Relative errors $e.$ and convergence rates $\rho.$ for Example 4.1 (half-circle) using a global polynomial degree $p = 1$

k	dof	N	e_{L^∞}	ρ_{L^∞}	e_{L^2}	ρ_{L^2}	e_{H^1}	ρ_{H^1}
1	12	4	2.75_{0}	–	2.49_{0}	–	1.16_{0}	–
2	48	16	1.19_{0}	0.60	9.52_{-1}	0.69	9.47_{-1}	0.15
3	186	62	5.12_{-1}	0.63	3.40_{-1}	0.76	6.61_{-1}	0.27
4	708	236	2.06_{-1}	0.68	1.54_{-1}	0.59	4.67_{-1}	0.26
5	2,796	932	6.88_{-2}	0.80	5.12_{-2}	0.80	2.77_{-1}	0.38
6	11,094	3,698	1.91_{-2}	0.93	1.40_{-2}	0.94	1.46_{-1}	0.46
7	44,136	14,712	4.94_{-3}	0.98	3.60_{-3}	0.99	7.44_{-2}	0.49
8	176,022	58,674	1.32_{-3}	0.95	9.06_{-4}	1.00	3.74_{-2}	0.50
9	703,050	234,350	3.54_{-4}	0.95	2.27_{-4}	1.00	1.87_{-2}	0.50
10	2,810,352	936,784	9.42_{-5}	0.96	5.68_{-5}	1.00	9.36_{-3}	0.50
11	11,237,016	3,745,672	2.49_{-5}	0.96	1.42_{-5}	1.00	4.68_{-3}	0.50
12	44,939,430	14,979,810	6.50_{-6}	0.97	3.55_{-6}	1.00	2.34_{-3}	0.50

Table 4.2: Relative errors $e.$ and convergence rates $\rho.$ for Example 4.1 (half-circle) using a global polynomial degree $p = 3$

k	dof	N	e_{L^∞}	ρ_{L^∞}	e_{L^2}	ρ_{L^2}	e_{H^1}	ρ_{H^1}
1	40	4	1.87_{0}	–	1.18_{0}	–	1.01_{0}	–
2	160	16	3.81_{-1}	1.15	3.53_{-1}	0.87	6.01_{-1}	0.37
3	620	62	5.10_{-2}	1.49	3.79_{-2}	1.65	1.50_{-1}	1.02
4	2,360	236	4.48_{-3}	1.82	2.90_{-3}	1.92	1.99_{-2}	1.51
5	9,320	932	3.61_{-4}	1.83	2.23_{-4}	1.87	2.50_{-3}	1.51
6	36,980	3,698	2.38_{-5}	1.97	1.49_{-5}	1.96	3.13_{-4}	1.51
7	147,120	14,712	1.52_{-6}	1.99	9.48_{-7}	2.00	3.91_{-5}	1.50
8	586,740	58,674	9.57_{-8}	2.00	5.96_{-8}	2.00	4.90_{-6}	1.50
9	2,343,500	234,350	5.99_{-9}	2.00	3.73_{-9}	2.00	6.12_{-7}	1.50
10	9,367,840	936,784	3.72_{-10}	2.01	2.33_{-10}	2.00	7.65_{-8}	1.50
11	37,456,720	3,745,672	1.70_{-10}	0.56	1.69_{-11}	1.89	9.57_{-9}	1.50

Table 4.3: Relative errors $e.$ and convergence rates $\rho.$ for Example 4.1 (half-circle) using a global polynomial degree $p = 5$

k	dof	N	e_{L^∞}	ρ_{L^∞}	e_{L^2}	ρ_{L^2}	e_{H^1}	ρ_{H^1}
1	84	4	9.04_{-1}	–	7.57_{-1}	–	8.82_{-1}	–
2	336	16	8.22_{-2}	1.73	7.71_{-2}	1.65	1.91_{-1}	1.10
3	1,302	62	2.16_{-3}	2.69	1.89_{-3}	2.74	7.85_{-3}	2.36
4	4,956	236	5.83_{-5}	2.70	3.76_{-5}	2.93	3.32_{-4}	2.37
5	19,572	932	1.43_{-6}	2.70	6.00_{-7}	3.01	1.16_{-5}	2.44
6	77,658	3,698	1.81_{-8}	3.17	9.42_{-9}	3.01	3.76_{-7}	2.49
7	308,952	14,712	3.34_{-10}	2.89	1.48_{-10}	3.01	1.19_{-8}	2.50
8	1,232,154	58,674	2.94_{-11}	1.76	2.32_{-12}	3.00	3.74_{-10}	2.50
9	4,921,350	234,350	5.42_{-12}	1.22	1.39_{-12}	0.37	1.46_{-11}	2.34

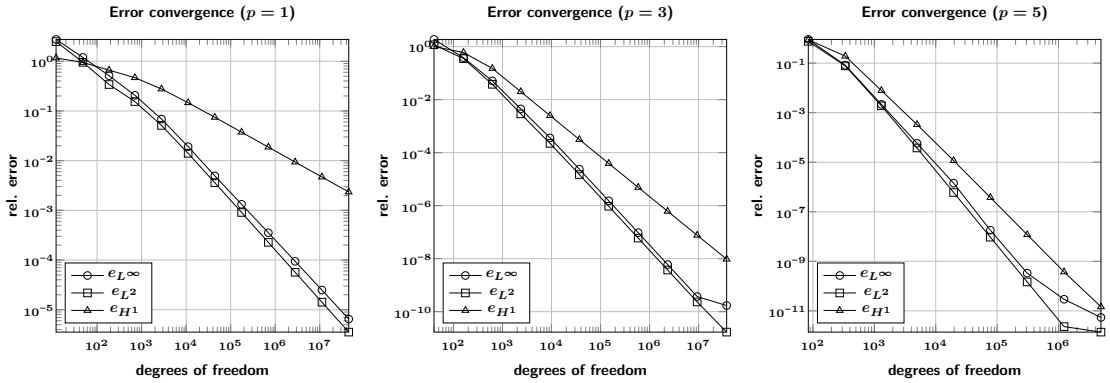

Figure 4.18: Error convergence for Example 4.1 (half-circle) using global polynomial degrees $p = 1$, $p = 3$ and $p = 5$ (left to right).

rectangle and replace the right edge with a half-circle as depicted in Figure 4.17. This half-circle is represented by a rational NURBS curve of degree 3. The convergence results are shown in Table 4.1, 4.2 and 4.3 for polynomials of degree $p = 1, 3$ and 5 respectively and are plotted in Figure 4.18. We can see that we are still able to obtain optimal convergence rates until we reach machine precision. Note that this does in particular show that our post-processed cover construction and numerical integration are accurate enough on the curved domain. Since the reference solution as well as the function spaces employed are independent of the specific shape of the domain (except that the patches need to cover all of the domain) we do expect the same convergence rates on *any* domain.

Example 4.2 (Linear Elastic Hole)**.** In this example we show the importance of an exact representation of the simulation domain, compared to an approach where we approximate the domain by a linear polygon. We did already mention above that when we are prescribing an analytical solution, we expect our method to always convergence to that solution, no mather on what kind of domain we will solve the problem. But for most problems in practice the results solved on an approximated domain will converge to a different solution than the results that are solved on the original curved domain. To this end, we consider the equations of linear elasticity as a boundary value problem for the unknown displacement

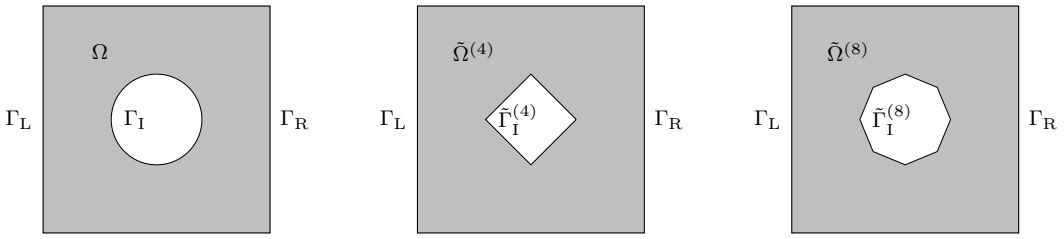

Figure 4.19: Square with hole domain Ω where the boundary of the hole Γ_{I} is a smooth circle, represented by a rational NURBS curve and approximations $\tilde{\Omega}^{(4)}$ and $\tilde{\Omega}^{(8)}$ where Γ_{I} is approximated by 4 and 8 line segments respectively.

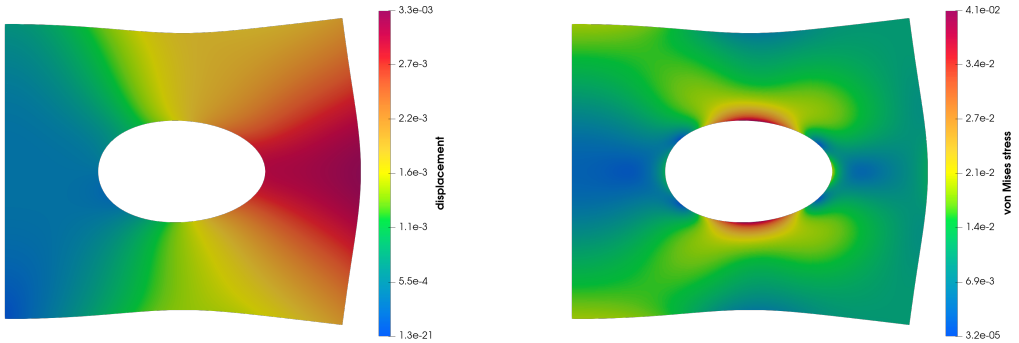

Figure 4.20: Magnitude of the displacement (left) and von Mises stress (right) of the solution of the linear elastic problem imposed in Example 4.2. The domain has been warped by the displacement field scaled by a factor of 100.

field $\boldsymbol{u}$ subject to given Dirichlet boundary conditions $\boldsymbol{g}_{\mathrm{D}}$, Neumann boundary conditions $\boldsymbol{g}_{\mathrm{N}}$ and a body force per unit volume $\boldsymbol{f}$. Hence, the generic problem is given by

$$\begin{aligned} -\operatorname{div}\sigma(\boldsymbol{u}) &= \boldsymbol{f} && \text{in } \Omega \\ \boldsymbol{u} &= \boldsymbol{g}_{\mathrm{D}} && \text{on } \Gamma_{\mathrm{D}} \subset \partial\Omega \\ \sigma(\boldsymbol{u})\cdot\boldsymbol{n} &= \boldsymbol{g}_{\mathrm{N}} && \text{on } \Gamma_{\mathrm{N}} \subset \partial\Omega \\ \sigma(\boldsymbol{u})\cdot\boldsymbol{n} &= 0 && \text{on } \partial\Omega \setminus (\Gamma_{\mathrm{D}} \cup \Gamma_{\mathrm{N}}) \end{aligned} \tag{4.32}$$

with $\sigma(\boldsymbol{u}) = \mathcal{C}\epsilon(\boldsymbol{u})$ being the symmetric Cauchy stress tensor with the material dependent stiffness tensor $\mathcal{C}$ and the infinitesimal strain tensor $\epsilon(\boldsymbol{u}) = \frac{1}{2}(\nabla\boldsymbol{u} + (\nabla\boldsymbol{u})^T)$. We limit ourselves to isotropic materials so we have $\sigma(\boldsymbol{u}) = \mathcal{C}\epsilon(\boldsymbol{u}) = 2\mu\epsilon(\boldsymbol{u}) + \lambda \operatorname{trace}(\epsilon(\boldsymbol{u}))\mathbb{I}$ with λ and μ being the Lamé constants. In this example the domain consists of a box $[-1,1]^2$ with a circular hole of radius 0.4 at its center, i.e. $\Omega := [-1,1]^2 \setminus \{\boldsymbol{x} \mid x_1^2 + x_2^2 < 0.4\}$ (compare first image in Figure 4.19). Let us define the left part of the boundary as $\Gamma_{\mathrm{L}} := \{\boldsymbol{x} \in \partial\Omega \mid x_1 = -1\}$ and similarly the right part as $\Gamma_{\mathrm{R}} := \{\boldsymbol{x} \in \partial\Omega \mid x_1 = 1\}$. Then we impose the following boundary conditions to the problem

$$\begin{aligned} u_1 &= 0 && \text{on } \Gamma_{\mathrm{L}}, \\ \sigma(\boldsymbol{u})\cdot\boldsymbol{n} &= (0.01, 0)^T && \text{on } \Gamma_{\mathrm{R}}, \\ \sigma(\boldsymbol{u})\cdot\boldsymbol{n} &= 0 && \text{on } \partial\Omega \setminus (\Gamma_{\mathrm{L}} \cup \Gamma_{\mathrm{R}}). \end{aligned}$$

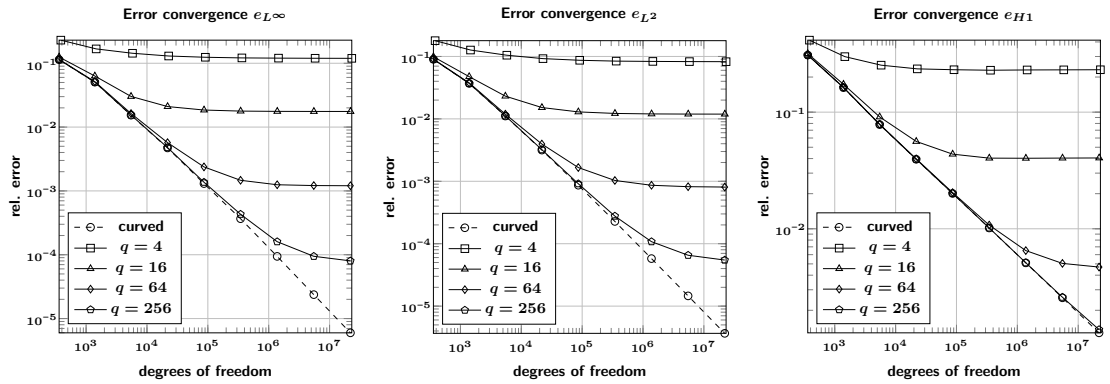

Figure 4.21: Error convergence for Example 4.2 (elastic hole) on the original domain Ω using curved integration cells and on static approximations of the domain $\Omega^{(q)}$ for $q \in \{4, 16, 64, 256\}$.

To make sure we have a unique solution to the problem we do additionally fix the displacement in y-direction, i.e. $u_2 = 0$, in the lower left corner at $(-1, -1)^T$.
Since we do not have an analytical solution for this problem we compute errors against a numerical reference solution that has been computed by utilizing a PUM space over a uniformly refined cover on level $k_{\max} = 11$ consisting of $N_{k_{\max}} = 3\,668\,380$ patches. We did employ polynomials of degree $p = 3$ on all patches which resulted in a function space with $\text{dof}_{k_{\max}} = 73\,367\,600$ degrees of freedom. Plots of the reference solution are depicted in Figure 4.20. All error norms have been computed on integration cells that resolve both, the cover of the reference solution as well as the respective cover of the solution it is compared with. One additional uniform refinement of the integration domains $\mathcal{D}_i^m$ has been performed and an increased Gaussian quadrature rule was used to make sure all error were computed with a sufficient accuracy.
To demonstrate the error introduced when we are not using an actual circle to represent the hole, but use a static approximation by linear edges we additionally solve the problem on domains $\tilde{\Omega}^{(q)}$ where we approximate the circular hole by q line segments of equal length (compare Figure 4.19). Convergence of solutions employing a PUM space with global polynomial degree $p = 1$ on the different domains can be seen in Figure 4.21. We can see that only the solutions on the original curved domain do converge to the reference solution with optimal rates. The solutions on the static domain do still converge, but they do so to a solution that is different to the one on the curved domain. The closer the domain approximation gets to the curved domain, the closer the solutions get to the curved solution, but even with $q = 256$ linear segments used for the approximation of the circle, the solution cannot match the reference solution in the L^2-norm by more than 4 decimal places. In fact this shows that on approximated domains we have two sources of errors: the error imposed by the domain approximation and the error imposed by the approximation power of the employed function spaces. Only if both errors get reduced in an overall refinement scheme we can be confident that our solution to a given problem is actually correct. Note that in general it is hard to estimate which of the two error sources

Figure 4.22: Integration domains $\mathcal{D}_i^m$ (boxes) and the resulting integration cells (gray triangles and boxes) of a single patch ω_i that intersects the boundary of a circular hole. *Left:* Original integration domains $\mathcal{D}_i^m$ resolving neighboring patches that overlap ω_i. *Right:* All integration domains $\mathcal{D}_i^m$ have been refined once. The resulting triangular integration cells interpolate the circle at more points and thus form a better approximation of the curved domain.

will dominate the overall error. Hence, trying to employ a static approximation where we have to make a choice for q before the simulation is initiated is rather challenging. While we can always chose q very high so that it is very *likely* to be sufficient for the desired accuracy it is still hard to be guaranteed. Additionally choosing a high value for q increases the computational cost of the integration step significantly, especially on coarser levels where we do only have a few number of patches and we would expect our function space approximation error to dominate the domain approximation error anyways. These observations motivate the idea that we have to balance the domain approximation error with the function space approximation power on each level to obtain a method that convergence to the solution on the original domain, even when using only linear approximations to the domain. To this end, we additionally conduct an experiment where we solve the given elasticity problem using the original curved domain to create the integration cells but then we do not use the curved triangles and instead only use the linear triangles during the integration. I.e. we simply disable the use of the transfinite interpolation to create the quadrature points and use a simple affine map instead. Due to the fact that the size of the integration domains $\mathcal{D}_i^m$ is dependent on the size of the patches and we compute the intersections of those integration domains with the original curved domain Ω, this means that our domain approximation is adaptive to h-refinement of our PUM space. We can then employ additional refinement of the linear cells to improve the domain approximation. An easy way to achieve this is by simply uniformly refining the integration domains $\mathcal{D}_i^m$ before we compute the intersection of them with the domain Ω. This automatically yields linear triangles that form a better approximation of the curved domain (compare Figure 4.22). Convergence results of experiments comparing only linear triangles with the fully curved triangles are depicted in Figure 4.23. The experiments have been conducted with different polynomial degrees $p = 1, 2, 3$. We can see that for linear polynomials the convergence behavior when using curved integration cells and the one using only linear triangles is identical. Hence, in this case it is likely that the approximation error of the function spaces dominates the one of the geometry approximation. But when using quadratic or

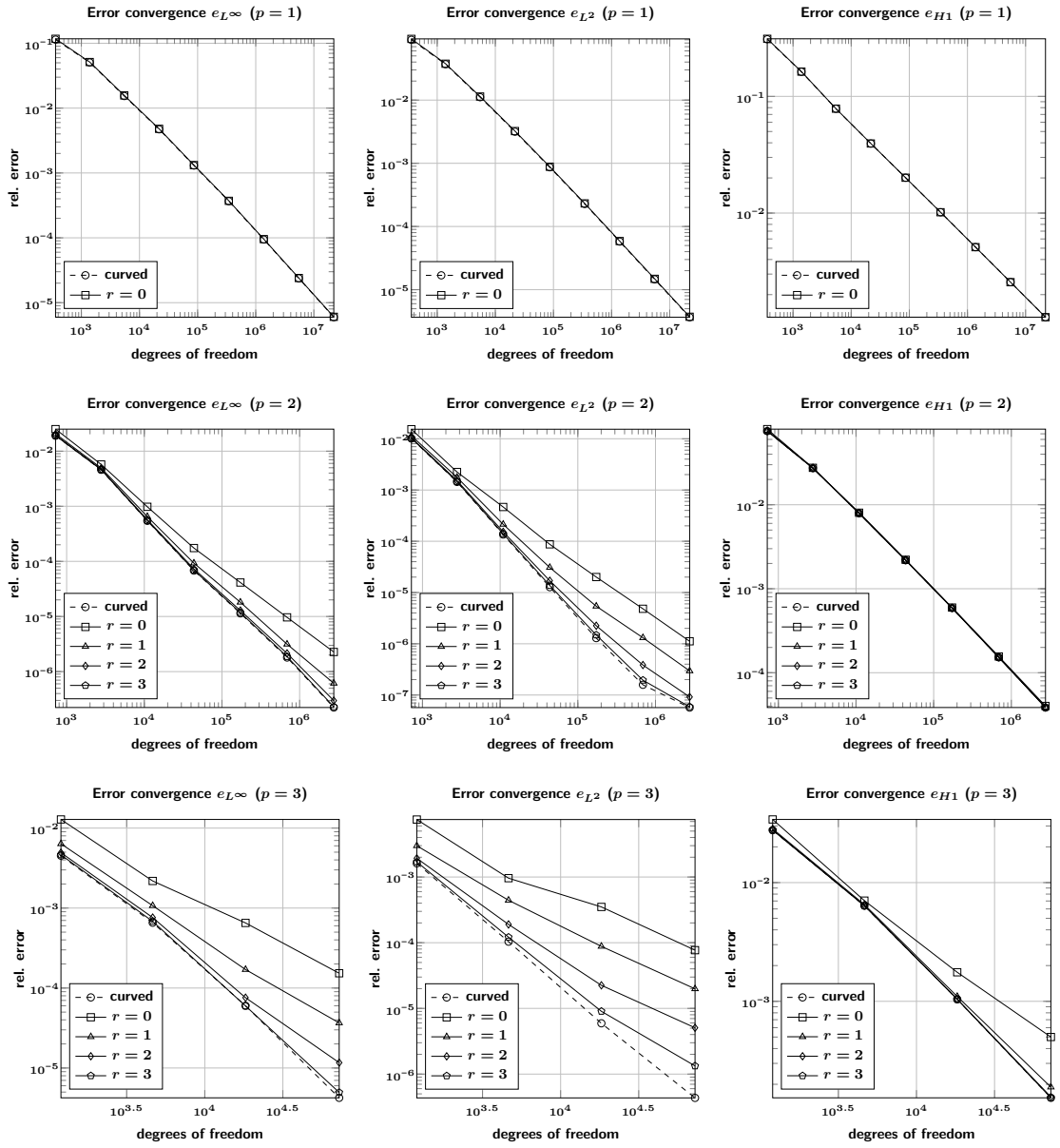

Figure 4.23: Error convergence for Example 4.2 on the original domain Ω for polynomial degrees $p = 1$, $p = 2$ and $p = 3$ (top to bottom). There are plots with enabled curved integration cells and with not curved integration cells where the integration domains $\mathcal{D}_i^m$ of boundary patches have been uniformly refined r times.

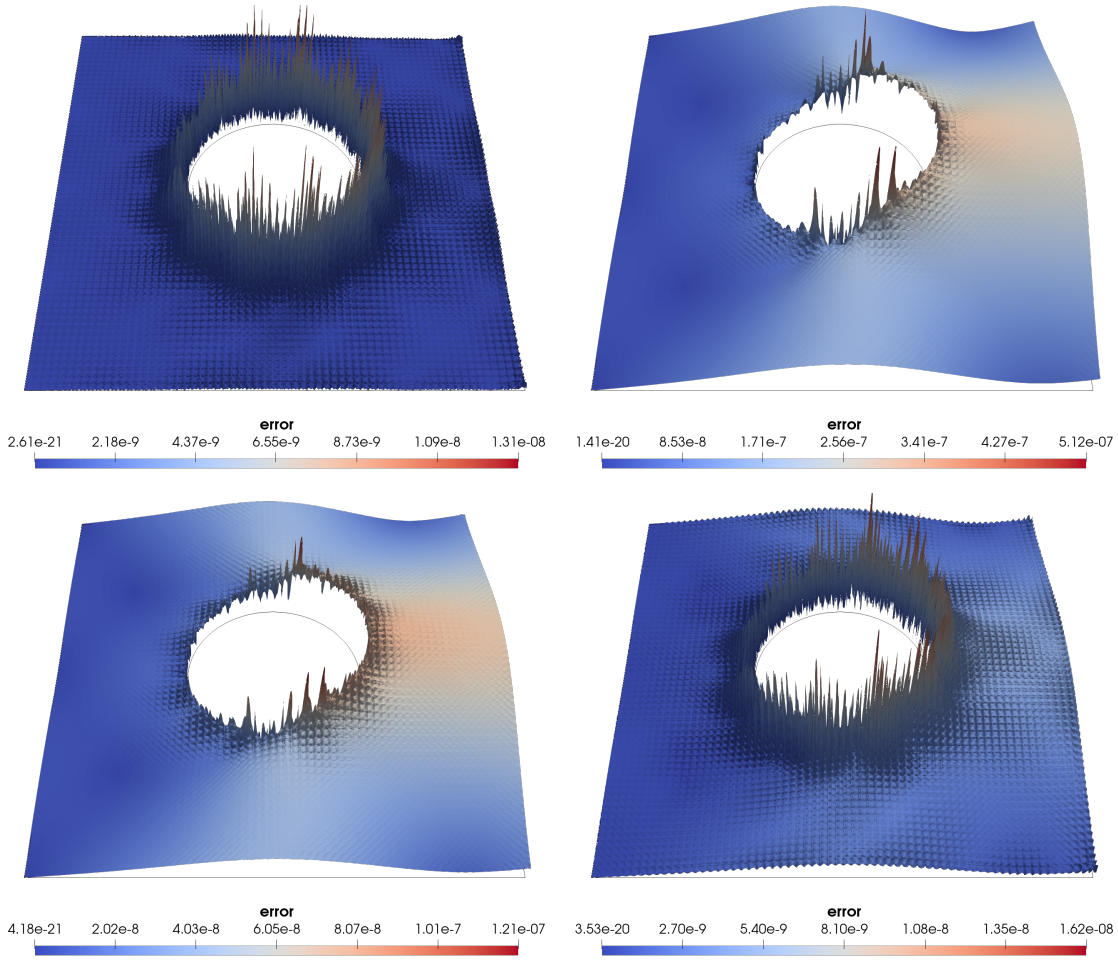

Figure 4.24: Warped contour plots of the errors $\|u^* - u_k^{\mathrm{PU}}\|$ using polynomials of degree $p = 3$ on level $k = 6$ for the solution on the original curved domain employing curved integration cells (upper left) and for solutions employing only linear integration cells with $r = 0$ uniform refinements of the integration domains $\mathcal{D}_i^m$ (upper right), $r = 1$ refinements (lower left) and $r = 3$ refinements (lower right). While the error of the solution with curved integration cells is highly oscillatory with respect to the patch size when using curved integration cells, the errors of the solutions using linear integration cells include error components with low oscillations that indicate that the solution converges to a function that is different than u^*. The low oscillatory error components decrease with higher values of r (and hence with better domain approximations). Note that the warp factor has been adjusted to best fit each plot separately and is hence not the same in all four plots.

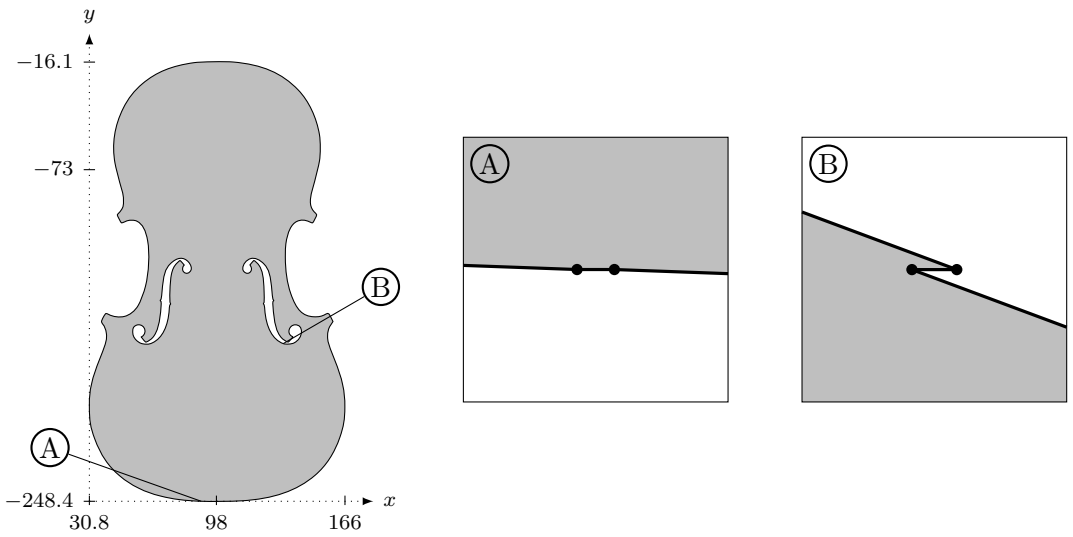

Figure 4.25: Violin domain with zooms into two locations where the boundary has very small features. The zooms are magnifications by a factor of 10^6.

cubic polynomials on all patches the behavior changes and using only linear triangles results in reduced convergence rates. Note that only the convergence rate is being reduced, but the solutions do still convergence to the original curved solution on all levels. The reduced convergence rates are due to the fact that in those cases the geometry approximation error dominates the function space approximation error and thus we are already in the case where further refinements of the involved function spaces lead to convergence towards incorrect solutions (compare Figure 4.24). Refining the integration domains $\mathcal{D}_i^m$ uniformly, which refines the geometry approximation relative to each patch, allows us to obtain better convergence behavior for the linear integration cells. Using three refinement steps results in close to optimal convergence rates for quadratic polynomials, but is still not quite sufficient for cubic polynomials. Note that choosing an insufficient refinement of the linear cells in this scenario is still better than choosing an insufficient static domain approximation a priori since even though the convergence rates are suffering, the method does still always converge closer to the curved solution when we employ any h-refinement. Hence, using only linear integration cells can sometimes be beneficial since we do not need to take the transfinite interpolation into account during the numeric integration which can drastically reduce the number of integration points that are required. Nevertheless only enabling the full curved integration scheme guarantees convergence with optimal rates.

Example 4.3 (Violin)**.** In this example we carry out a simulation on a domain resembling the front-plate of a violin[7] as depicted on the left in Figure 4.25. To demonstrate the robustness with respect to "dirty" input domains the violin has **not** been constructed directly within a CAD modeler. Instead we did extract the contour lines from a simple

[7]In fact it is the front-plate of the Stradivarius "Messiah".

bitmap of the violin via a graphics processor (using *Inkscape*[8]) which resulted in an SVG image that has then been imported into a CAD modeler to convert it into a STEP file (using *FreeCAD*[9]). This process resulted in a domain that consists of 114 curves. The outer ring consists of 57 curves and the two inner rings consist of 31 and 26 curves respectively. Most of these curves are non-rational cubic NURBS curves (or thus simple cubic B-spline curves) but 10 out of the 114 curves are just linear line segments. While the domain's bounding box has an extend of $\sim$135 in x-direction and $\sim$232 in y-direction there are two curves in the input geometry that do only have a length of $\sim 1.7 \cdot 10^{-5}$ and $\sim 1.4 \cdot 10^{-5}$ respectively and hence are about seven magnitudes smaller than the overall geometry (compare Figure 4.25). In a mesh-based method those small domain features would usually dictate the size of the smallest triangle and to retrieve a mesh where the minimal angle is bounded, this would drastically increase the total number of triangles in the mesh. Hence, for mesh-based methods it would usually be desired to remove such small artificial features in the domain before generating a mesh or to employ a meshing algorithm that is designed to not preserve small features. In the PUM however, we do not need to remove those features since the triangular decomposition is used for integration only and hence sharp triangles do not impose any problems to our basis functions. The total number of integration cells is only slightly increased when keeping those features and gets negligible on finer levels of our covers.
To demonstrate that we retrieve good performance on this geometry, we consider the time-dependent scalar wave equation in the time interval $[0, T]$, given by

$$\begin{array}{rcll} \ddot{u}(\boldsymbol{x}, t) & = & \Delta u(\boldsymbol{x}, t) & \text{for } (\boldsymbol{x}, t) \in \Omega \times (0, T] \\ u(\boldsymbol{x}, 0) & = & 0 & \text{for } \boldsymbol{x} \in \Omega \\ \dot{u}(\boldsymbol{x}, 0) & = & \gamma(x) & \text{for } \boldsymbol{x} \in \Omega \\ u(\boldsymbol{x}, t) & = & 0 & \text{for } \boldsymbol{x} \in \partial\Omega,\ t > 0 \end{array} \tag{4.33}$$

where $\dot{u} = \frac{\partial u}{\partial t}$ and $\ddot{u} = \frac{\partial^2 u}{\partial t^2}$ are the first and second time derivative of u and γ is an initial velocity field that is given by

$$\gamma(\boldsymbol{x}) = -\beta \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{c}\|^2}{\sigma^2}\right) \qquad \text{with } \boldsymbol{c} = (98, -73)^T,\ \sigma = 10 \text{ and } \beta = 10.$$

We employ central differences with $t_{n+1} = t_n + \delta t$ for the time discretization of (4.33) which yields

$$u(\boldsymbol{x}, t_{n+1}) = (\delta t)^2 \Delta u(\boldsymbol{x}, t_n) + 2u(\boldsymbol{x}, t_n) - u(\boldsymbol{x}, t_{n-1}) \quad \text{for } \boldsymbol{x} \in \Omega$$

as the spatial problem. The Galerkin discretization in V^{PU} of that equation then reads: Find $u(\boldsymbol{x}, t_{n+1}) \in V^{\mathrm{PU}}$ such that

$$\int_\Omega u(\boldsymbol{x}, t_{n+1}) v \, dx = (\delta t)^2 \int_\Omega \Delta u(\boldsymbol{x}, t_n) v \, dx + \int_\Omega 2u(\boldsymbol{x}, t_n) v \, dx + \int_\Omega u(\boldsymbol{x}, t_{n-1}) v \, dx$$

[8] A free and open source vector graphics editor. `https://inkscape.org`
[9] A general purpose parametric 3D CAD modeler. `https://www.freecadweb.org`

holds for all $v \in V^{\mathrm{PU}}$. After partial integration, eliminating terms determined by the Dirichlet boundary conditions and some rearrangement this becomes

$$\langle u(\boldsymbol{x}, t_{n+1}), v\rangle_{L^2(\Omega)} = \langle 2u(\boldsymbol{x}, t_n) - u(\boldsymbol{x}, t_{n-1}), v\rangle_{L^2(\Omega)} - (\delta t)^2 \langle \nabla u(\boldsymbol{x}, t_n), \nabla v\rangle_{L^2(\Omega)}.$$

With $\tilde{u}_n$ being the coefficient vector of the unknown function $u_n^{\mathrm{PU}} \in V^{\mathrm{PU}}$ at time t_n, we can obtain the algebraic representation

$$M\tilde{u}_{n+1} = M(2\tilde{u}_n - \tilde{u}_{n-1}) - (\delta t)^2 A\tilde{u}_n$$

with the mass matrix M given by

$$(M_{i,j})_{s,t} := \langle \varphi_j \vartheta_j^t, \varphi_i \vartheta_i^s\rangle_{L^2(\Omega)} \tag{4.34}$$

with $i, j = 1, \ldots, N$, $s = 1, \ldots, \dim(V_i)$ and $t = 1, \ldots, \dim(V_j)$ and the stiffness matrix A given by

$$(A_{i,j})_{s,t} := \langle \nabla\varphi_j \vartheta_j^t, \nabla\varphi_i \vartheta_i^s\rangle_{L^2(\Omega)}. \tag{4.35}$$

Hence, one time-stepping iteration to obtain the new solution at time t_{n+1} from the previous solutions is given by

$$\tilde{u}_{n+1} = 2\tilde{u}_n - \tilde{u}_{n-1} - (\delta t)^2 M^{-1} A\tilde{u}_n.$$

By introducing $\tilde{v}_n = \frac{\tilde{u}_n - \tilde{u}_{n-1}}{\delta t}$ as the coefficient vector of the velocity field we get

$$\begin{aligned} \tilde{u}_{n+1} &= \tilde{u}_n + \delta t\tilde{v}_n - (\delta t)^2 M^{-1} A\tilde{u}_n \\ &= \tilde{u}_n + \delta t\tilde{v}_{n+1} \end{aligned} \tag{4.36}$$

with

$$\tilde{v}_{n+1} = \tilde{v}_n - \delta t M^{-1} A\tilde{u}_n. \tag{4.37}$$

The discretization of the initial velocity condition additionally yields a simple projection problem which's algebraic representation is given by

$$\tilde{v}_0 = M^{-1}\hat{g} \tag{4.38}$$

where $\hat{g}$ is the right-hand side moment vector given by

$$(\hat{g}_i)_s = \langle \gamma, \varphi_i \vartheta_i^s\rangle_{L^2(\Omega)} \qquad i = 1, \ldots, N \quad s = 1, \ldots, \dim(V_i). \tag{4.39}$$

In principle both (4.38) as well as (4.37) require us to invert the mass matrix M but this is too expensive when required in each time step to be practical in general. Hence a common approach is to replace the conforming mass matrix M with a so-called *lumped mass matrix* $\bar{M}$ that ideally is a diagonal matrix. Such an approach was introduced for our PUM in [109] and $\bar{M}$ is given by

$$(\bar{M}_{i,j})_{s,t} := \begin{cases} \langle \vartheta_j^t, \varphi_i \vartheta_i^s\rangle_{L^2(\Omega)} & i = j \\ 0 & \text{otherwise} \end{cases} \tag{4.40}$$

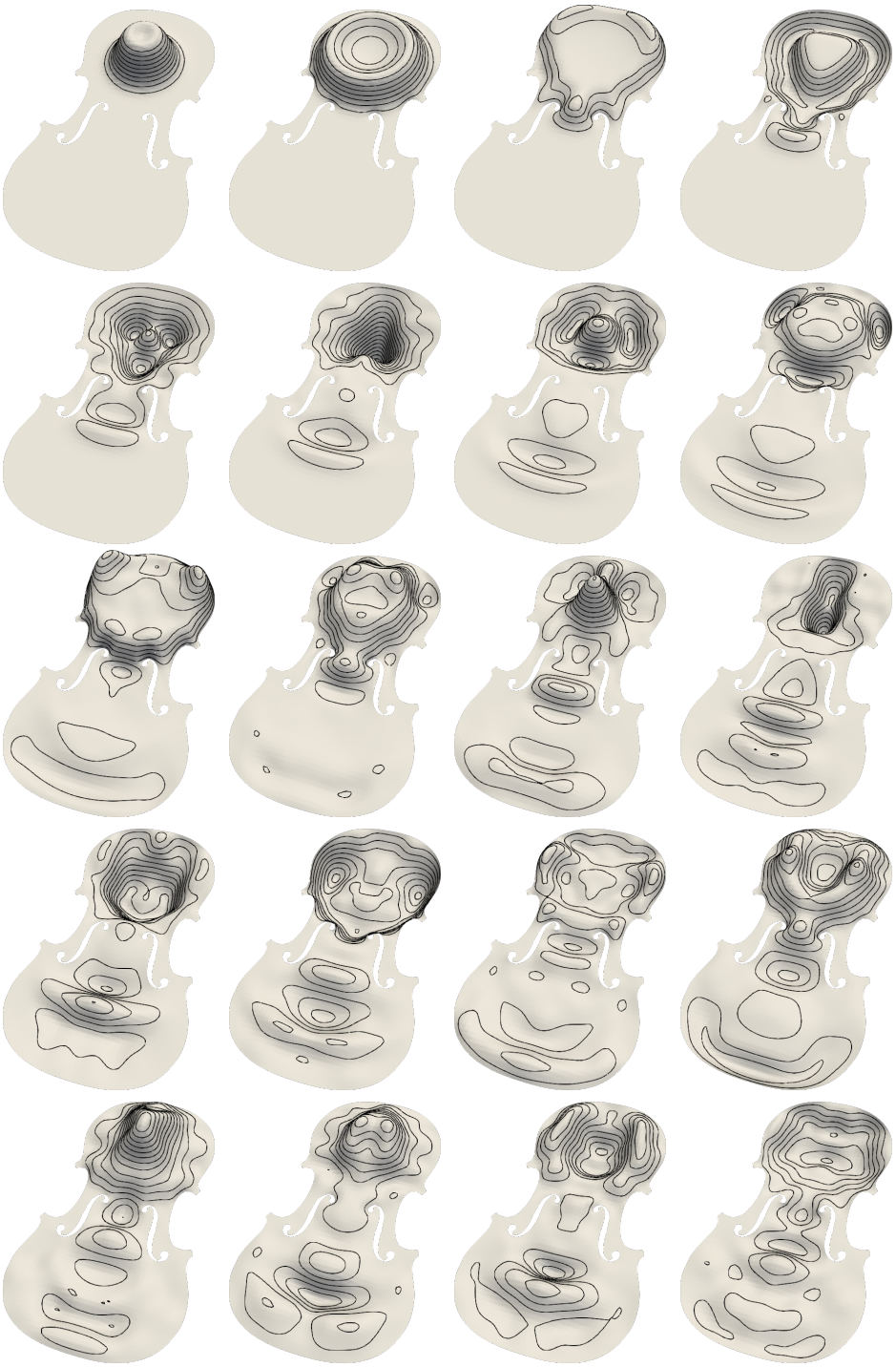

Figure 4.26: Surface plots of the displacement field $u(\boldsymbol{x}, t_n)$ at specific time points $t_n = 1500k\delta t$ for $n = 1, \ldots, 20$ (row-wise, top left to bottom right).

which does have a block-diagonal form. Hence, the inverse $\bar{M}^{-1}$ can be formed explicitly and all steps (4.38), (4.37) and (4.36) end up being simple linear algebra operations. Snapshots of the solution $u(\boldsymbol{x}, t_n)$ at specific moments in time t_n are depicted in Figure 4.26.
To evaluate the performance of our method (and particularly all geometric operations) we only take a look at the initialization step of the simulation and ignore the actual time stepping, since in each time step we do perform linear algebra operations only. Hence we measure the time of the four main tasks that take place before the time-stepping starts, which are:

1. T_{C}: Constructing the cover C_Ω^k on a given level k.

2. T_{I}: Creating the integration cells for all boundary patches of that cover.

3. T_{A}: Assembling the stiffness matrix A according to (4.35), the lumped mass matrix $\bar{M}$ according to (4.40) and the initial velocity moment vector $\hat{g}$ according to (4.39).

4. T_{S}: Explicit computation of the inverse $\bar{M}^{-1}$ via a block-wise Cholesky decomposition.

Since we are mainly interested in the performance of the geometry operations, let us first discuss to which extend each of the tasks T_{C}, T_{I}, T_{A} and T_{S} include geometric operations. The task T_{S} is purely algebraic and thus does not include any geometric operations. On the other hand, the task T_{I} needs to compute the intersections $\mathcal{D}_i^m \cap \Omega$ and the decomposition of those domains into integration cells, which are the operations we are most interested in. But the task T_{C} includes geometric operations as well: First of all we need to check whether $\mathcal{C}_{i,k} \cap \Omega = \emptyset$ to figure out which local tree cells can be discarded and need not to be stretched into a patch of C_Ω^k. Then this step includes the categorization of all patches into the subsets of patches at the boundary and patches in the interior, so we need to check for $\omega_{i,k} \cap \partial\Omega \neq \emptyset$. Finally, the cover post-processing algorithm from Chapter 2 includes checks for $\tilde{\mathcal{D}}_i^m \cap \Omega \neq \emptyset$ to detect patches with empty flat-top domains and requires to compute the bounding box of $\omega_{i,k} \cap \Omega$ to shrink patches at the boundary. The task T_{A} does not need to perform any expensive geometric operations: Since the integration cells for boundary patches that are expensive to compute have already been computed in task T_{I} the only geometric operations that are left are to compute the decompositions $\tilde{\mathcal{D}}_i^m$ for interior patches which do not involve any intersections with Ω or its boundary. We do not pre-compute the quadrature points for any integration cell, hence the task T_{A} does additionally include the evaluation of the transformations $\boldsymbol{\Psi}$ that translate the quadrature points from the reference domains into the actual integration cells. Regardless of that, the main computational tasks in T_{A} are the evaluation of the partition of unity functions φ_i, the local basis functions ϑ_i^n and the bi-linear and linear forms to integrate as well as computing the sums of the Gaussian quadrature and storing the results into the destination block matrices and vectors.
To further discuss the expected runtime for each of those tasks let us split the set of all

Table 4.4: Total number of patches N_k, number of boundary patches N_k^{B} and number of interior patches N_k^{I} on levels $k = 0, \ldots, 12$ for covers over the violin domain. Numbers are given for a uniformly refined cover and an adaptively refined cover where only boundary patches from $C^k_{\Omega,\mathrm{B}}$ are refined in each refinement step.

k	0	1	2	3	4	5	6	7	8	9	10	11	12
UNIFORM REFINEMENT													
N_k	1	4	10	36	132	488	1851	7218	28456	112849	449652	1794741	7171429
N_k^{B}	1	4	10	30	66	156	329	669	1357	2712	5419	10840	21676
N_k^{I}	0	0	0	6	66	332	1522	6549	27099	110137	444233	1783901	7149753
BOUNDARY ADAPTIVE REFINEMENT													
N_k	1	4	10	36	114	284	729	1728	3733	7909	16524	33240	67585
N_k^{B}	1	4	10	30	66	156	329	669	1357	2712	5419	10840	21676
N_k^{I}	0	0	0	6	48	128	400	1059	2376	5197	11105	22400	45909

patches in a cover C^k_Ω into the sets

$$C^k_{\Omega,\mathrm{B}} := \{\omega_{i,k} \in C^k_\Omega \mid \omega_{i,k} \cap \partial\Omega \neq \emptyset\} \quad \text{and}$$
$$C^k_{\Omega,\mathrm{I}} := \{\omega_{i,k} \in C^k_\Omega \mid \omega_{i,k} \cap \partial\Omega = \emptyset\}$$

where $C^k_{\Omega,\mathrm{B}}$ is the set of *boundary patches* and $C^k_{\Omega,\mathrm{I}}$ the set of *interior patches*. Let us further introduce $N_k^{\mathrm{B}} := \mathrm{card}(C^k_{\Omega,\mathrm{B}})$ and $N_k^{\mathrm{I}} := \mathrm{card}(C^k_{\Omega,\mathrm{I}})$ for the number of patches in each set and note that $\mathrm{card}(C^k_\Omega) =: N_k = N_k^{\mathrm{B}} + N_k^{\mathrm{I}}$.

Let us take a look at what this means for the expected runtime of our tasks on a specific level k: The task T_{C} uses the hierarchical tree construction and hence involves operations on all patches on all levels $0, \ldots, k$. Since the number of nodes in a d-binary tree with N_k leaves is less than $2N_k$, the cover construction is at least in $O(N_k)$.[10] The task T_{A} needs to assemble blocks for all patches and hence it is in $O(N_k)$ as well, although it is worth mentioning that the work is probably not distributed equally among all patches. The assembly of entries related to patches $\omega_{i,k} \in C^k_{\Omega,\mathrm{B}}$ does usually lead to more integration cells and due to the potentially curved triangle cells we need more quadrature points on many of those integration cells compared to the assembly of entries only related to patches $\omega_{i,k} \in C^k_{\Omega,\mathrm{I}}$. On the other hand, the integration cells for patches $\omega_{i,k} \in C^k_{\Omega,\mathrm{B}}$ have already been precomputed in task T_{I}, while the integration cells involving only patches $\omega_{i,k} \in C^k_{\Omega,\mathrm{I}}$ are be computed on the fly. The task T_{S} for this example does only involve the inversion of a block-diagonal matrix and is thus in $O(N_k)$ as well. Finally the task T_{I} does only involve the computation of integration cells for patches $\omega_{i,k} \in C^k_{\Omega,\mathrm{B}}$ and hence it is $O(N_k^{\mathrm{B}})$ only. To understand why this is crucial to the runtime behavior of the method, let us first assume that the cover is refined uniformly, i.e. all patches get h-refined in each refinement

[10] Since the cover construction involves some steps where we need to traverse the tree for each patch (e.g. the neighborhood construction), a more correct estimate of the complexity would be $O(N_k \log N_k)$ but how distinct the logarithmic term can be observed in measurements depends heavily on the constants involved in the complexities for the specific tasks during the cover construction.

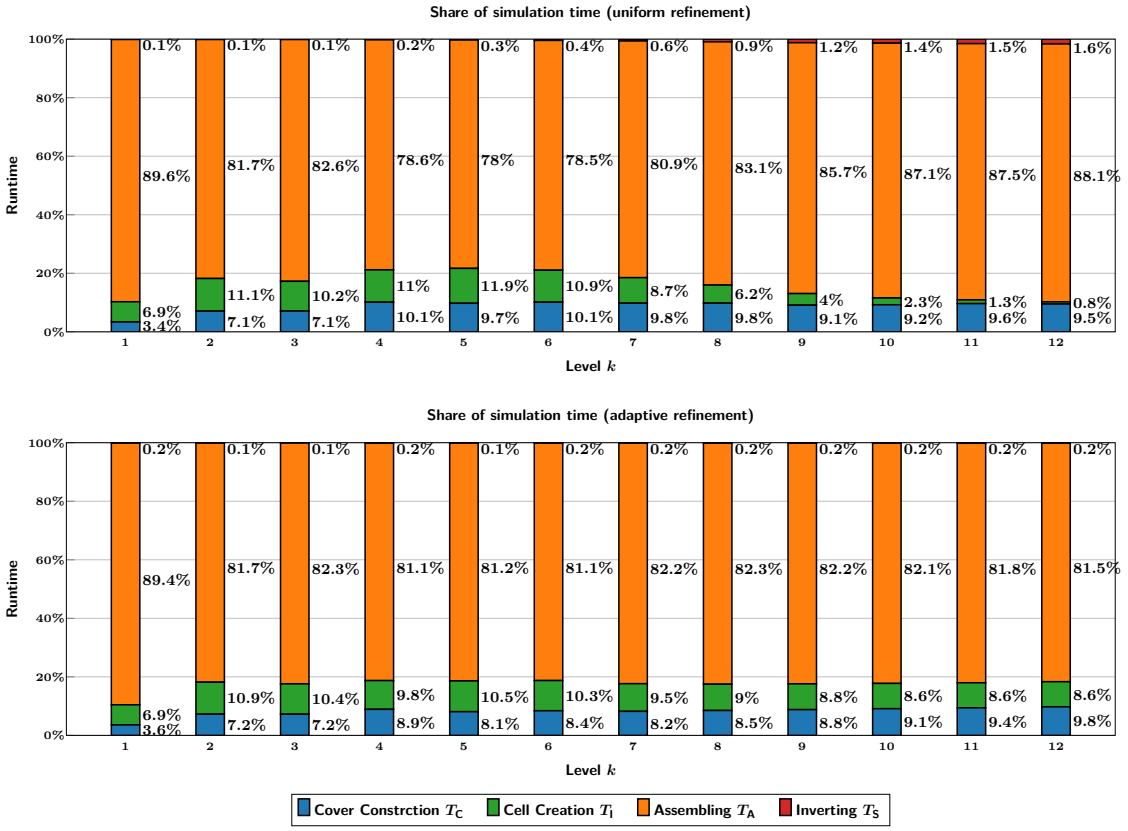

Figure 4.27: Stacked bar charts depicting the share of the simulation set-up time of the tasks T_{C}, T_{I}, T_{A} and T_{S} on levels $k = 1, \ldots, 12$. All measurements have been repeated five times and mean values are presented. *Top:* Uniformly refined cover on all levels. *Bottom:* Adaptively refined cover. On each level only patches intersecting the boundary (i.e. where $\omega_i \cap \partial\Omega \neq \emptyset$) have been refined.

step. On the first few levels we expect $N_k^{\mathrm{B}} = N_k$ and $N_k^{\mathrm{I}} = 0$ since all patches intersect the boundary. But as soon as we have reached a level $\hat{k}$ where $N_{\hat{k}}^{\mathrm{I}} > 0$ we expect N_k^{I} for $k > \hat{k}$ to grow at least by a factor of 2^d in each level since all child patches of an interior patch are in the interior as well. In contrast to that we expect the growth of N_k^{B} to slow down and eventually reach a growth rate of about 2^{d-1} since the boundary $\partial\Omega$ is a geometric entity of dimension $d-1$. That is why, asymptotically, only half the children of a boundary patch will intersect the boundary as well, with the other half eventually being equally distributed to patches being either outside of the domain or completely inside of it (compare the top half of Table 4.4). Hence, on ever finer levels we can expect the runtime of a task that is in $O(N_k^{\mathrm{B}})$ to vanish compared any task that is in $O(N_k)$ or even just $O(N_k^{\mathrm{I}})$. Thus it especially holds that the work of the task T_{I} has to vanish compared to the work of the other tasks when we employ a uniform refinement scheme. In fact this is confirmed by the runtime results depicted in the top half of Figure 4.27. First there is a short initialization

phase up until level $k = 5$ where the portion of the runtime for task T_I is increasing. On the first few levels the asymptotic complexities of the distinct tasks do not yet hold. E.g. on coarser levels a single patch can see a lot of complexity of the domain, thus we need to create comparatively many integration cells per patch on those coarse levels. Additionally it is important to note that level $k = 5$ is the first level where we indeed have $N_k^\mathrm{I} > N_k^\mathrm{B}$ and that the total runtime on that level is less than 0.1 seconds. But from $k = 6$ onwards we can see that the runtime portion of the task T_I starts to decline compared to the total runtime until eventually on levels $k \geq 11$ it even takes less time than the relatively cheap task T_S. Note that the experiment has been carried out employing polynomials of degree $p = 1$. Employing higher order polynomials would further shift the runtime towards the tasks T_A and T_S since the required number of integration points would increase and we would need to evaluate more basis functions that result in larger blocks to be assembled and inverted.

Let us now construct a case that is less favorable towards the task T_I. Instead of a uniformly refined cover, we employ a refinement strategy where we refine only patches $\omega_{i,k} \in C^k_{\Omega,\mathrm{B}}$ on each level. Note that this does not necessarily mean that we have $N_k^\mathrm{B} > N_k^\mathrm{I}$ for high enough levels. We did already note that refining a boundary patch will most likely result in some of its children becoming interior patches that are added to the interior patches already present from the last level. Additionally due us employing the maximum depth difference L_max according to (1.31) our refinement algorithm does occasionally refine interior patches close to the boundary to retain a smooth transition of patch sizes. To this end we do rather expect $N_k^\mathrm{B} \sim N_k^\mathrm{I}$ (compare the second half of Table 4.4). The runtime results for such a refinement scheme are depicted in the bottom half of Figure 4.27. We can clearly see that in contrast to the uniform refinement the runtime portion of the task T_I does not vanish on higher levels. Instead it seems to converge to a state where about 8.6% of the runtime per level will be spend for the creation of integration cells in this example. This shows that even for refinement strategies that tend to focus work around $\partial\Omega$, the algorithms presented throughout this chapter are efficient enough so that they do not dominate the overall simulation time. A fundamental necessity for this to work is that the work per boundary patch should be independent of the level k. This is only partially true though, since the costs per intersection computation $\mathcal{D}_i^m \cap \Omega$ and the decomposition of the results thereof into (curved) triangular integration cells does heavily depend on the complexity of the domain's boundary $\partial\Omega$ where it overlaps a given patch $\omega_{i,k}$ and hence it does heavily depend on the size and location of that patch. Luckily the sizes of our patches $\omega_{i,k}$ do decrease on higher levels k and hence we expect the computation of the integration cells to become less expensive per patch on each level until eventually all intersections of patches with the boundary become as simple as each patch being cut by a single, close to linear edge. Figure 4.28 depicts the time of the task T_I per boundary patch split into the time to compute $\mathcal{D}_i^m \cap \Omega$ and the time to create the resulting integration cells. We can make two important observations: First of all, both runtime components seem to converge to a state where the amount of work is fixed per boundary patch, which confirms the assumptions given above. Additionally we can see that the runtime behavior of the intersection and the decomposition steps behave quite different over the levels. In case of the decomposition,

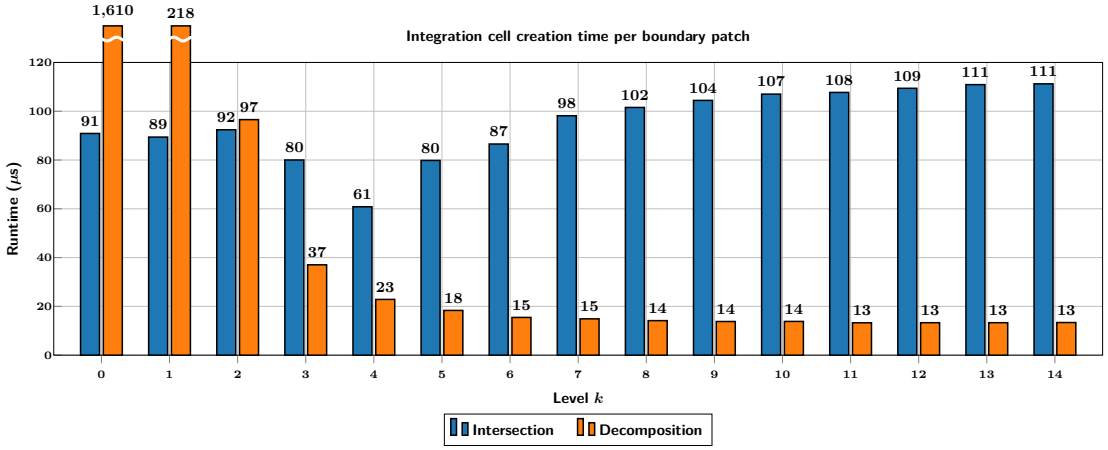

Figure 4.28: Partition of the runtime of the integration cell creation task (T_I) into the time to compute the intersections $\mathcal{D}_i^m \cap \Omega$ and the time to decompose those domains into the (curved) triangular integration cells $\mathcal{T}_i^n$. All times are reported per boundary patch (i.e. the total time per level has been divided by the number of boundary patches N_k^B). All measurements have been repeated five times and mean values are presented.

level $k = 0$ is very special, since there we have a single patch overlapping the whole domain and hence we need to create a global triangulation of the domain. Note that simulations on level 0 are probably never relevant in practice since the approximation power of V_k^PU is very limited and barely useful anyways. Even if only used as coarser levels for a multilevel level solver, we can usually start on finer levels since the respective discretized equation systems do easily fit a direct solver, both in memory and runtime consumption. On finer levels the time for the decomposition declines constantly, since the intersections $\mathcal{D}_i^m \cap \Omega$ get ever simpler and finally most decompositions end up in resulting in a single or up to only three triangles. For the intersection task, level $k = 0$ is special as well, in the sense that the intersection does actually result in all of Ω. Hence the intersection does mainly include work to figure out that there are no intersection points and then copying Ω into the result. Then, for a few levels the intersection costs decrease at first, then slightly increase before eventually they reach a constant value per patch. The explanation here is manifold. One aspect is that the costs per intersection depends on where exactly the patch is located, since the complexity of $\partial\Omega$ is not distributed evenly across the whole domain. Thus it takes a while until a steady state is reached. Additionally intersections where $\mathcal{D}_i^m \cap \partial\Omega = \emptyset$ (i.e. either $\mathcal{D}_i^m \cap \Omega = \mathcal{D}_i^m$ or $\mathcal{D}_i^m \cap \Omega = \emptyset$ holds) tend to be slightly more expensive than intersections where $\mathcal{D}_i^m \cap \partial\Omega \neq \emptyset$ and on lower levels the latter cases are more likely to happen since the average size of $\mathcal{D}_i^m$ is bigger. To understand why intersections where the boundary $\partial\Omega$ is not actually intersected are more expensive, we have to look at a few things: Since we are only looking at integration domains $\mathcal{D}_i^m$ resulting from patches at the boundary, we do expect the integration domains to be always *close* to the boundary $\partial\Omega$, even when they do not intersect it. Hence, we first compute the intersection points of the

infinite extensions of the boundary segments of $\mathcal{D}_i^m$ with $\partial\Omega$ according to the algorithm described in Section 4.1.2, only to discard all of those intersection points afterwards. In contrast to the case $\mathcal{D}_i^m \cap \partial\Omega \neq \emptyset$ where we could now start to build the intersection result, we do need to figure out whether $\mathcal{D}_i^m$ is completely within Ω or outside of it. Hence we need to perform an additional point-within-BREP test that requires to compute the intersections of a ray with all of $\partial\Omega$. Nevertheless, since the amount of domains where $\mathcal{D}_i^m \cap \partial\Omega \neq \emptyset$ compared to the amount of domains where $\mathcal{D}_i^m \cap \partial\Omega = \emptyset$ should eventually converge to a steady state and all elementary operations do only depend on the complexity of $\partial\Omega$ and not on the current cover, we achieve a constant time per patch on higher levels k.

In a final experiment we want to take a look at how the runtime of our method behaves in the case of more complex simulation domains Ω. To this end, we artificially increase the number of boundary curves that represent $\partial\Omega$ of our violin domain by recursively subdividing all curves q times. This means after q subdivision steps the domain is described by $114 \cdot 2^q$ curves instead of just the original 114 curves. Note that geometries where seemingly simple boundary curves are described by many small, low-order Bézier curves are not uncommon in practice. In fact, since the intersection curves of two three-dimensional surfaces are often approximated by piecewise cubic B-splines as described in Chapter 3, such boundary descriptions are frequently encountered when dealing with the two-dimensional parameter domains of industrial shell geometries. Let us first only consider the influence of the refined boundary description on the task T_I. Especially we do again take a look at the time to compute the intersections $\mathcal{D}_i^m \cap \Omega$ and the time to decompose those cells into the final integration cells per boundary patch. Results for $q = 1$, $q = 5$ and $q = 10$ are depicted in Figure 4.29. We first discuss the time to compute the intersections. We can note that on level $k = 0$ the time increases significantly with higher q as it is expected. Copying the whole geometry should have a runtime that scales linearly with the complexity of the geometry itself. Hence, for $q = 10$ a runtime increase of a factor of about 1024 compared to $q = 0$ would not be surprising and indeed we can see an increase to 74 373 μs for $q = 10$ compared 91 μs for the original geometry ($q = 0$) from Figure 4.28. On higher levels the time for the intersections decreases rapidly until on the finest levels the intersection time per patch for higher values of q is even lower than the time for low values of q. The fact that intersections on fine levels can be performed faster for higher values of q can be a sign that the acceleration R-Tree data structure that has been built for all the monotone curve parts could be improved for the original unrefined geometry by some additional splits: For the refined geometries the leaves of the tree that contain the curve parts are smaller and thus more intersection queries can be filtered out by the tree traversal algorithm. The numbers presented here seem to indicate that this outweighs the increased number of total nodes in the tree. Let us now turn the focus on the time to compute the local decomposition into the triangular integration cells. We can again observe that on low levels k the time increases significantly for higher values of q but the times improve rapidly on higher levels until on level $k = 14$ we get comparable times for all values of q. We should note that on level $k = 0$ the triangular decomposition does actually create a *global* triangulation of the whole geometry. The performance of that case could probably be improved significantly by choosing a better algorithm for the cavity

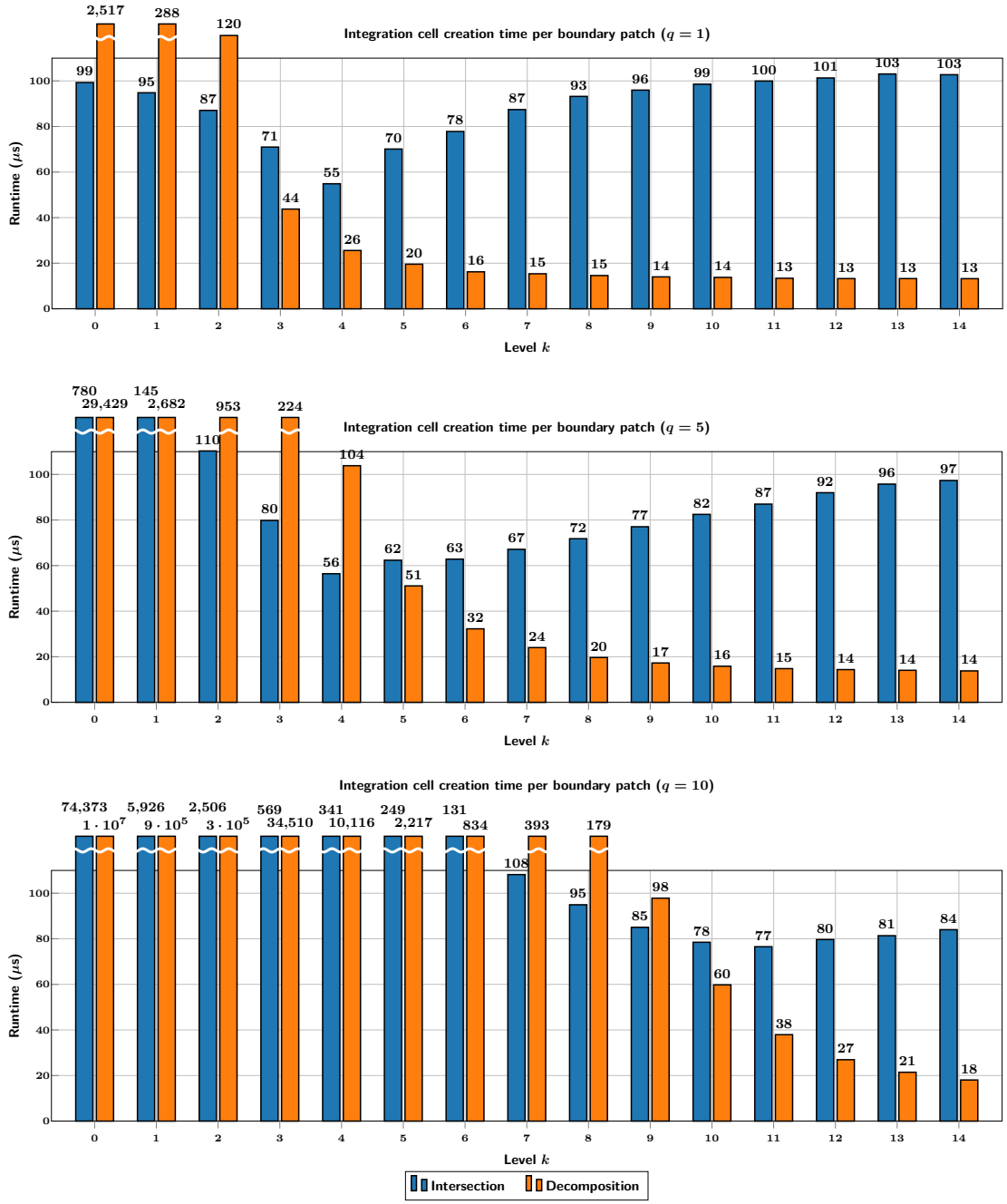

Figure 4.29: Partition of the runtime of task T_{I} into the time to compute the intersections $\mathcal{D}_i^m \cap \Omega$ and the time to decompose those domains into the (curved) triangular integration cells $\mathcal{T}_i^n$. All times are reported per boundary patch (i.e. the total time per level has been divided by the number of boundary patches N_k^{B}). All measurements have been repeated five times and mean values are presented. The boundary curves of the input geometry have been subdivided $q = 1, 5, 10$ times (top to bottom).

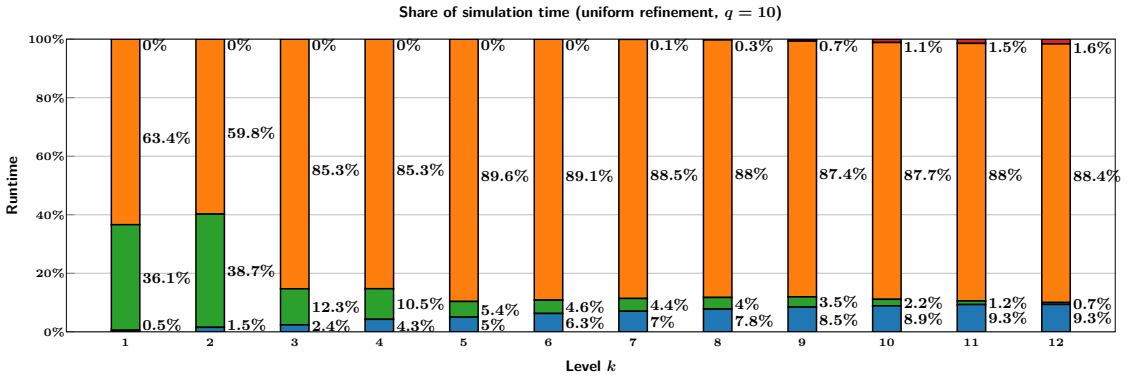

Figure 4.30: Stacked bar charts depicting the share of the simulation set-up time of the tasks T_{C}, T_{I}, T_{A} and T_{S} on levels $k = 1, \ldots, 12$. All measurements have been repeated five times and mean values are presented. The boundary curves of the input geometry have been subdivided $q = 10$ times.

triangulation during the constrained Delaunay triangulation algorithm. As mentioned in Section 4.2 we did implement the algorithm presented in [132], which is easy to implement and fast in most cases, but can have a worst-case complexity of $O(N^2)$ where N is the number of vertices in the triangulation. The choice of that algorithm was based on the fact that in practice discretizations on level $k = 0$ (or very coarse levels in general) are very uncommon in practice, especially when considering complex geometries. The numbers presented here do in fact show that algorithm is sufficiently performant when we consider simple geometries or higher levels on complex geometries. Finally we take a brief look at the runtime portion the complete task T_{I} takes in the overall simulation when considering complex geometries. Results for $q = 10$ are depicted in Figure 4.30. We can clearly see that even on level $k = 0$ the task T_{I} does not dominate the overall runtime. This is due to the fact that the more complex geometry results in more triangular integration cells that need to be processed during the assemble task T_{A} an hence that steps runtime increases as well. If we compare the runtime portions on the final level $k = 12$ for $q = 10$ to the values for the original geometry as they were depicted in Figure 4.27 we can see that the more complex geometry does not have any significant impact on the distribution of the runtime across the tasks. In fact even the total runtime does not differ significantly on the finest level. For the original geometry the mean runtime of five runs on level $k = 12$ was 497.08s, while for $q = 10$ we measured 503.08s which is only about 1.2% increase.

Example 4.4 (Door of a car)**.** In this final experiment of the chapter we consider a continuum mechanical problem where the geometry is that of the door of a car[11] (compare Figure 4.31). We only consider the *inner* shell of the door in the following. As it is common for most body parts of a car, the door is built from sheet metal. For such kinds of parts it is possible to make some kinematic assumptions that allow to reduce

[11]Door model has been taken from `https://grabcad.com/library/car-front-door-1` by author "Ciprian Dragne" (`https://grabcad.com/ciprian.dragne`). *Accessed: 2022-09-29.*

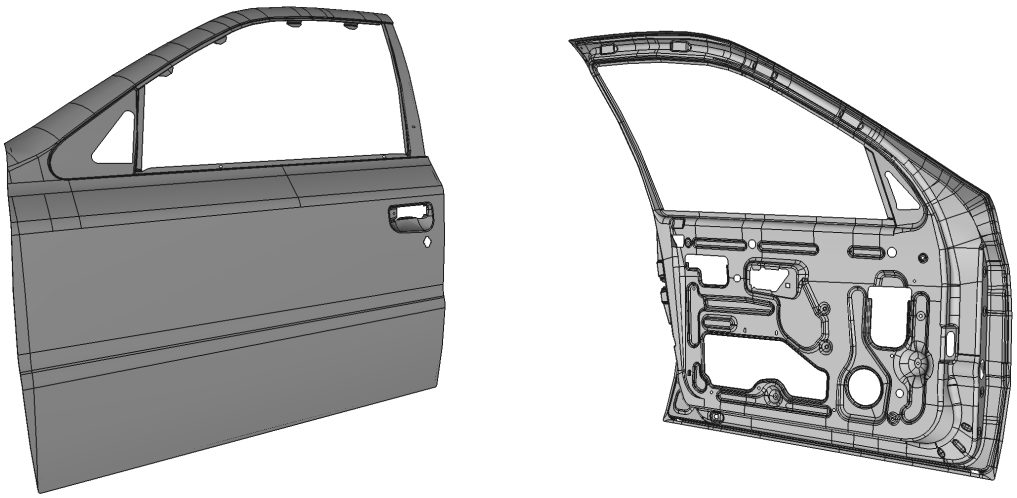

Figure 4.31: Full door model consisting of two shells. The outer shell (left) consists of 383 faces, the inner shell (right) consists of 2469 faces.

the three-dimensional solid mechanics problem to a two dimensional problem. There are different kind of assumptions one can make that result in different kind of formulations of the problem at hand. Commonly those descriptions are referred to as *plate theories* and the two most widespread representatives are the *Reissner-Mindlin plate theory* and the *Kirchhoff-Love plate theory*. In both theories only the *mid-surface* is used to represent the originally three-dimensional problem. But while the former takes transverse shear forces into account and it is thus usually applicable to moderately thick plates, the latter assumes that those forces are negligible and hence it is only applicable to thin plates in general. On the other hand, the Reissner-Mindlin model requires to introduce some additional degrees of freedom to represent rotations of the normals to the mid-surface, while those normals always stay orthogonal in the model by Kirchhoff-Love. Hence, for thin plates it is usually desireable to employ the Kirchhoff-Love plate theory due to its lower costs. Nevertheless in classical finite element methods the Reissner-Mindlin model is far more common since its implementation does only require C^0 continuous elements, while an implementation of the Kirchhoff-Love model requires C^1 continuous elements that are non-trivial to achieve. On the contrary, the PUM allows us to construct higher order function spaces easily, since the local smoothness of the function spaces V_i transfers directly into the smoothness of the global space V^{PU}. To this end, we employ the Kirchhoff-Love plate theory in this example. The Kirchhoff-Love plate theory was introduced in the context of the PUM in [86], based on the work in [103]. We briefly summarize that theory in the following.
Let us consider a parametrized surface $\boldsymbol{S} : \Omega \to \mathbb{R}^3$ with $\Omega \subset \mathbb{R}^2$. Additionally, let $\boldsymbol{x} \in \Omega$ and

$$\boldsymbol{S}_{\partial d} := \frac{\partial S}{\partial x_d}, \qquad d \in \{1, 2\}$$

be the partial derivative of $\boldsymbol{S}$ with respect to directional coordinate x_d. Then the normal

to the surface at any location $\boldsymbol{x}$ is given by

$$\boldsymbol{n}_{\boldsymbol{S}}(\boldsymbol{x}) := \frac{\boldsymbol{S}_{\partial 1}(\boldsymbol{x}) \times \boldsymbol{S}_{\partial 2}(\boldsymbol{x})}{\|\boldsymbol{S}_{\partial 1}(\boldsymbol{x}) \times \boldsymbol{S}_{\partial 2}(\boldsymbol{x})\|}.$$

Using that normal we can define the *orthogonal projection operator* as

$$\boldsymbol{P}(\boldsymbol{x}) := \mathbb{I}_3 - \boldsymbol{n}_{\boldsymbol{S}}(\boldsymbol{x}) \otimes \boldsymbol{n}_{\boldsymbol{S}}(\boldsymbol{x}).$$

Next we need to introduce some derivatives with respect to the surface $\boldsymbol{S}$ employing *tangential differential calculus*. To this end, the *tangential gradient* of a scalar field $u : \Omega \to \mathbb{R}$ is given by

$$\nabla_{\boldsymbol{S}} u(\boldsymbol{x}) = J_{\boldsymbol{S}}(\boldsymbol{x}) \cdot G_{\boldsymbol{S}}(\boldsymbol{x})^{-1} \cdot \nabla u(\boldsymbol{x}) \tag{4.41}$$

where $J_{\boldsymbol{S}}(\boldsymbol{x}) \in \mathbb{R}^{3\times 2}$ is the jacobian of the surface $\boldsymbol{S}$ and $G_{\boldsymbol{S}}(\boldsymbol{x}) := J_{\boldsymbol{S}}(\boldsymbol{x})^T J_{\boldsymbol{S}}(\boldsymbol{x})$. Similarly the tangential gradient of a vector field $\boldsymbol{v} : \Omega \to \mathbb{R}^3$ can be defined by applying (4.41) to all components of that vector field. The *tangential divergence* of a such a vector field is then given by

$$\operatorname{div}_{\boldsymbol{S}} \boldsymbol{v}(\boldsymbol{x}) = \operatorname{trace}(\nabla_{\boldsymbol{S}} \boldsymbol{v}(\boldsymbol{x})) \tag{4.42}$$

and similarly the divergence of a matrix field $A : \Omega \to \mathbb{R}^{3\times 3}$ by applying (4.42) component-wise to all rows of $A(\boldsymbol{x})$. Finally the so-called *Weingarten map* is given by

$$H(\boldsymbol{x}) := \nabla_{\boldsymbol{S}} \boldsymbol{n}_{\boldsymbol{S}}(\boldsymbol{x}).$$

Let us now consider the original three dimensional shell domain $\underline{\underline{\Omega}}$ that can be described by its mid-surface $\underline{\Omega} := \boldsymbol{S}[\Omega] = \{\boldsymbol{S}(\boldsymbol{x}) \mid \boldsymbol{x} \in \Omega\}$ and a constant *thickness* τ such that any point $\underline{\underline{\boldsymbol{x}}} \in \underline{\underline{\Omega}}$ is given by

$$\underline{\underline{\boldsymbol{x}}} = \boldsymbol{S}(\boldsymbol{x}) + \xi \boldsymbol{n}_{\boldsymbol{S}}$$

for $\boldsymbol{x} \in \Omega$ and $\xi \in (-\frac{\tau}{2}, \frac{\tau}{2})$. Let $\boldsymbol{u} : \Omega \to \mathbb{R}^3$ be the displacement field at the mid-surface. Then, in the Kirchhoff-Love model the *strain tensor* is given by

$$\epsilon(\boldsymbol{u}, \xi) := \epsilon_M(\boldsymbol{u}) + \xi \epsilon_B(\boldsymbol{u})$$

where

$$\epsilon_M(\boldsymbol{u}) := \operatorname{sym}(\nabla_{\boldsymbol{S}} \boldsymbol{u}) \qquad \text{and} \qquad \epsilon_B(\boldsymbol{u}) := \operatorname{sym}(\boldsymbol{n}_{\boldsymbol{S}} \cdot \nabla_{\boldsymbol{S}} \nabla_{\boldsymbol{S}} \boldsymbol{u})$$

with $\operatorname{sym}(A) := \frac{1}{2}(A + A^T)$ are the *membrane* and *bending* part of the strain respectively. If we then assume an isotropic linear elastic material and plane stress, the *in-plane stress tensor* is given by

$$\sigma(\boldsymbol{u}, \xi) := \sigma_M(\boldsymbol{u}) + \xi \sigma_B(\boldsymbol{u})$$

where

$$\sigma_M(\boldsymbol{u}) := \boldsymbol{P}\Big(2\mu\epsilon_M(\boldsymbol{u}) + \lambda \operatorname{trace}\big(\epsilon_M(\boldsymbol{u})\big)\mathbb{I}\Big)\boldsymbol{P} \quad \text{and}$$

$$\sigma_B(\boldsymbol{u}) := \boldsymbol{P}\Big(2\mu\epsilon_B(\boldsymbol{u}) + \lambda \operatorname{trace}\big(\epsilon_B(\boldsymbol{u})\big)\mathbb{I}\Big)\boldsymbol{P}$$

are the *in-plane membrane stress* and *in-plane bending stress* respectively and where $\lambda := \frac{E}{2(1+\nu)}$ and $\mu := \frac{E\nu}{1-\nu^2}$ are modified Lamé parameters depending on the Young's modulus

E and Poisson's ratio ν of the given material. Due to the possibility of an analytical pre-integration through the thickness, we can then define the *effective normal force tensor* by

$$N(\boldsymbol{u}) := \int_{-\frac{\tau}{2}}^{\frac{\tau}{2}} \sigma(\boldsymbol{u}, \xi)\ d\xi = \tau \sigma_M(\boldsymbol{u})$$

and the symmetric *moment tensor* by

$$M(\boldsymbol{u}) := \int_{-\frac{\tau}{2}}^{\frac{\tau}{2}} \xi \sigma(\boldsymbol{u}, \xi)\ d\xi = \frac{\tau^3}{12} \sigma_B(\boldsymbol{u}).$$

Assume that we want to impose only two kinds of boundary conditions: so-called *simply supported edge* conditions on the boundary part $\Gamma_{\mathrm{S}} \subseteq \partial\Omega$ and *free edge* conditions on the rest of the boundary $\Gamma_{\mathrm{F}} := \partial\Omega \setminus \Gamma_{\mathrm{S}}$. Let $\boldsymbol{n}_{\partial\Omega} \in \mathbb{R}^2$ be the normal to the boundary $\partial\Omega$ and $\underline{\boldsymbol{n}}_{\partial\Omega} := J_{\boldsymbol{S}} G_{\boldsymbol{S}}^{-1} \boldsymbol{n}_{\partial\Omega}$ its respective normal in the three-dimensional space. Similarly $\boldsymbol{t}_{\partial\Omega} \in \mathbb{R}^2$ is the tangent to $\partial\Omega$ and $\underline{\boldsymbol{t}}_{\partial\Omega} := J_{\boldsymbol{S}} \boldsymbol{t}_{\partial\Omega}$ its three-dimensional counterpart. Then, with

$$b_n(\boldsymbol{u}) := \underline{\boldsymbol{n}}_{\partial\Omega} \cdot \big(M(\boldsymbol{u}) \underline{\boldsymbol{n}}_{\partial\Omega}\big) \qquad \text{and} \qquad b_t(\boldsymbol{u}) := \underline{\boldsymbol{t}}_{\partial\Omega} \cdot \big(M(\boldsymbol{u}) \underline{\boldsymbol{n}}_{\partial\Omega}\big)$$

being the *bending* and *twisting moment* respectively, the strong form of the shell problem on a single surface $\boldsymbol{S}$ is given by

$$\begin{aligned}
-\operatorname{div}_{\boldsymbol{S}} \big(N(\boldsymbol{u}) + HM(\boldsymbol{u})\big) & & & \\
-\operatorname{div}_{\boldsymbol{S}} \big(\boldsymbol{P} \operatorname{div}_{\boldsymbol{S}} M(\boldsymbol{u})\big) \boldsymbol{n}_{\boldsymbol{S}} - 2H \operatorname{div}_{\boldsymbol{S}} M(\boldsymbol{u}) &= \boldsymbol{f} && \text{in } \Omega \\
\boldsymbol{u} &= 0 && \text{on } \Gamma_{\mathrm{S}} \\
\big(N(\boldsymbol{u}) + HM(\boldsymbol{u})\big) \underline{\boldsymbol{n}}_{\partial\Omega} + H \underline{\boldsymbol{t}}_{\partial\Omega} b_t(\boldsymbol{u}) &= 0 && \text{on } \Gamma_{\mathrm{F}} \\
\underline{\boldsymbol{n}}_{\partial\Omega} \cdot \boldsymbol{P} \operatorname{div}_{\boldsymbol{S}} M(\boldsymbol{u}) + \underline{\boldsymbol{t}}_{\partial\Omega} \cdot \nabla_{\boldsymbol{S}} b_t(\boldsymbol{u}) &= 0 && \text{on } \Gamma_{\mathrm{F}} \\
b_n(\boldsymbol{u}) &= 0 && \text{on } \Gamma_{\mathrm{S}} \cup \Gamma_{\mathrm{F}}
\end{aligned} \tag{4.43}$$

where $\boldsymbol{f}$ is an applied body force per unit area. It is important to note that even though the shell problem is three-dimensional, (4.43) is formulated on Ω only and hence it can be solved entirely in the two-dimensional parameter space of $\boldsymbol{S}$.

In the case of our door model, the complete shell does not consist of a single surface, but is a collection of multiple surfaces that are connected through common edges. Hence, (4.43) has to be solved for many surfaces $\boldsymbol{S}_i : \Omega_i \to \mathbb{R}^3$ simultaneously and additional constraints need to be added where those surfaces are connected along edges in the three-dimensional space. The topological information of the CAD shell is used to determine which surfaces are connected with each other, which edges are involved in the connection and which curves represent those edges in the boundary descriptions of $\partial\Omega_i$ and $\partial\Omega_j$. Then for each edge $E_t := (i, j)$ that connects the surface $\boldsymbol{S}_i$ to the surface $\boldsymbol{S}_j$ via the boundary parts $\Gamma_{i,t} \subseteq \partial\Omega_i$ and $\Gamma_{j,t} \subseteq \partial\Omega_i$ respectively, we have the additional constraints

$$\begin{aligned}
\boldsymbol{u}_i(\boldsymbol{x}_i) &= \boldsymbol{u}_j(\boldsymbol{x}_j) && \text{on } \Gamma_{i,t} \\
\theta_n(\boldsymbol{u}_i, \boldsymbol{x}_i, i) &= -\theta_n(\boldsymbol{u}_j, \boldsymbol{x}_j, j) && \text{on } \Gamma_{i,t}
\end{aligned} \tag{4.44}$$

with

$$\theta_n(\boldsymbol{u}, \boldsymbol{x}, i) := \underline{\boldsymbol{n}}_{\partial\Omega_i} \cdot \Big((\nabla_{\boldsymbol{S}_i} \boldsymbol{u}(\boldsymbol{x})) \boldsymbol{n}_{\boldsymbol{S}_i} \Big)$$

being the *normal rotation* and $\boldsymbol{x}_j \in \Gamma_{j,t}$ being the point where $\boldsymbol{S}_i(\boldsymbol{x}_i) = \boldsymbol{S}_j(\boldsymbol{x}_j)$ holds.[12] Note that for each edge $E_t := (i, j)$ we can formulate a corresponding symmetric edge $\bar{E}_t := (j, i)$ which represents the same constraint from the other side. The efficient and robust treatment of those *interface* constraints is the topic of currently ongoing research at the *Institute for Numerical Simulation* at the *University of Bonn* [67]. Hence, we only give a very brief overview of a proof-of-concept implementation employed for the experiment in this thesis. For the constraints on the displacement field from (4.44) we employ a method based on the conforming Dirichlet treatment as presented in Section 1.4 and for the constraints on the rotations we employ a skew-symmetric Nitsche method. To this end, let us assume that the indices i, j of the surfaces related to an edge E_t are sorted, i.e. $i \leq j$. Then the surface $\boldsymbol{S}_i$ is appointed the *primary* side and $\boldsymbol{S}_j$ the *secondary* side of the interface.
We first consider the rotational interface constraints. To this end, for a vector field w, let

$$[\theta_n(\boldsymbol{w})]_{\Gamma_{i,t}} := \theta_n(\boldsymbol{w}_i, \boldsymbol{x}_i, i) - \theta_n(\boldsymbol{w}_j, \boldsymbol{x}_j, j) \qquad \text{and}$$
$$\{b_n(\boldsymbol{w})\}_{\Gamma_{i,t}} := \frac{1}{2}\big(b_n(\boldsymbol{w}_i, \boldsymbol{x}_i, i) + b_n(\boldsymbol{w}_j, \boldsymbol{x}_j, j)\big)$$

be the *jump of the normal rotation* and the *average of the bending moment* across the interface edge $\Gamma_{i,t}$ respectively. Then the Nitsche terms that are employed in the weak form of the problem are given by

$$\gamma \int_{\underline{\Gamma}_{i,t}} [\theta_n(\boldsymbol{u})]_{\Gamma_{i,t}} [\theta_n(\boldsymbol{v})]_{\Gamma_{i,t}} \, ds - \int_{\underline{\Gamma}_{i,t}} [\theta_n(\boldsymbol{u})]_{\Gamma_{i,t}} \{b_n(\boldsymbol{v})\}_{\Gamma_{i,t}} \, ds + \int_{\underline{\Gamma} i,t} [\theta_n(\boldsymbol{v})]_{\Gamma_{i,t}} \{b_n(\boldsymbol{u})\}_{\Gamma_{i,t}} \, ds$$

with γ being the regularization parameter, $\boldsymbol{u}$ being the trial functions and $\boldsymbol{v}$ the test functions from spaces V_i^{PU} and V_j^{PU} corresponding to each side at the interface.
For the conforming coupling of the displacements, the basic idea is to introduce a global-to-local, inter-surface L^2-projection

$$\tilde{u}_i = (\hat{M}_i^i)^{-1} \hat{M}_j^i \tilde{u}_j =: \Pi_j^i \tilde{u}_j \tag{4.45}$$

that allows us to replace the degrees of freedom in the coefficient vector $\tilde{u}_i$ for V_i^{PU}, that are associated with the interface constraint, by the respective solution represented by the coefficient vector $\tilde{u}_j$ for V_j^{PU}. To construct the projection Π_j^i we employ the local, block-diagonal mass matrix

$$\big((M_i^i)_{n,n}\big)_{r,s} := \big\langle \vartheta_{i,n}^s(\boldsymbol{x}_i), \vartheta_{i,n}^r(\boldsymbol{x}_i) \big\rangle_{L^2(\omega_{i,n} \cap \Gamma_{i,t})}$$

and the inter-surface transfer matrix

$$\big((M_j^i)_{n,m}\big)_{r,s} := \big\langle \phi_{j,m}(\boldsymbol{x}_i) \vartheta_{j,m}^s(\boldsymbol{x}_j), \vartheta_{i,n}^r(\boldsymbol{x}_i) \big\rangle_{L^2(\omega_{i,n} \cap \Gamma_{i,t})}.$$

[12] In fact, due to gaps in the geometry representation, the point $\boldsymbol{x}_j := D_t(\boldsymbol{x}_i)$ is determined by applying a map D_t that tries to find a point $\boldsymbol{x}_j \in \Gamma_{j,t}$ such that $\|\boldsymbol{S}_i(\boldsymbol{x}_i) - \boldsymbol{S}_j(\boldsymbol{x}_j)\|$ is minimal. A more detailed explanation is given around equations (5.1) and (5.2) in Section 5.1 when dealing with 3D geometries.

Note that M_j^j does exactly correspond to the boundary trace operator matrix from (1.36) of the conforming Dirichlet treatment and the right-hand side moment vector from (1.37) is replaced by the transfer $M_i^j \tilde{u}_i$. Hence, the eigenvalue decomposition of M_i^i can be used to construct a basis transformation matrix T_i^i similar to (1.38) that allows to create a splitting $V_i^{\mathrm{PU}} = V_{i,K}^{\mathrm{PU}} \oplus V_{i,I}^{\mathrm{PU}}$ into the subspace $V_{i,I}^{\mathrm{PU}}$ that is determined by the constraint and the subspace $V_{i,K}^{\mathrm{PU}}$ that is left to resolve the interior of the PDE. The matrices $\hat{M}_i^i$ and $\hat{M}_j^i$ from (4.45) are then created by applying T_i^i to M_i^i and M_j^i and setting all rows that belong to the degrees of freedom from $V_{j,K}^{\mathrm{PU}}$ to zero. Similar to the construction for the Dirichlet boundary conditions we can then assemble the equation system in the original basis without taking the displacement interface conditions into account and transform that equation system by linear algebra operations only, into a coupled system. In contrast to the Dirichlet boundary condition case where the prescribed values ended up on the right-hand side of the equation, in the interface conditions introduce additional entries in off-diagonal positions of the stiffness matrix.

Let us now turn our focus on how those equations are discretized. To this end, we first construct covers C_{Ω_i} for the parameter domains Ω_i of each of the surfaces $\boldsymbol{S}_i$ involved in the CAD shell. Due to the CAD description of the surfaces, we have $\boldsymbol{S}_i : \mathcal{P}_{\boldsymbol{S}_i} \to \mathbb{R}^3$ where $\mathcal{P}_{\boldsymbol{S}_i} := (p_1^{\min}, p_1^{\max}) \times (p_2^{\min}, p_2^{\max})$ is a tensor product domain. It is then $\Omega_i \subseteq \mathcal{P}_{\boldsymbol{S}_i}$ and $\partial\Omega_i$ is explicitly given as the trim domain of the surface. An important point to note is that the extends of $\mathcal{P}_{\boldsymbol{S}_i}$ and $\boldsymbol{S}_i[\mathcal{P}_{\boldsymbol{S}_i}]$ are usually vastly different. Hence, if the patches from two different covers $\omega_{i,n} \in C_{\Omega_i}$ and $\omega_{j,m} \in C_{\Omega_j}$ are about the same size in their respective parameter domains, their images in the three dimensional space are usually not, i.e.

$$\operatorname{diam}(\omega_{i,n}) \approx \operatorname{diam}(\omega_{j,m}) \quad \not\Rightarrow \quad \operatorname{diam}(\boldsymbol{S}_i[\omega_{i,n}]) \approx \operatorname{diam}(\boldsymbol{S}_j[\omega_{j,m}]).$$

Hence we need to aim directly for $\operatorname{diam}(\boldsymbol{S}_i[\omega_{i,n}]) \approx \operatorname{diam}(\boldsymbol{S}_j[\omega_{j,m}])$ if we want to create a uniform cover across all surfaces. A simple approximation to achieve that is by assuming a constant relation between $\operatorname{diam}(\mathcal{P}_{\boldsymbol{S}_i})$ and $\operatorname{diam}(\boldsymbol{S}_i[\mathcal{P}_{\boldsymbol{S}_i}])$. To this end, let us define the anisotropic extend of the parameter domain $\boldsymbol{h}_{\mathcal{P}_{\boldsymbol{S}_i}} \in \mathbb{R}^2$ by

$$(\boldsymbol{h}_{\mathcal{P}_{\boldsymbol{S}_i}})_m := p_m^{\max} - p_m^{\min}, \quad m \in \{1, 2\}.$$

Let us then consider the four bounding curves of the surface

$$\begin{aligned}
\boldsymbol{C}_{i,1}^{\min} &: (p_1^{\min}, p_1^{\max}) \to \mathbb{R}^3, \quad t \mapsto \boldsymbol{S}_i(t, p_2^{\min}) \\
\boldsymbol{C}_{i,1}^{\max} &: (p_1^{\min}, p_1^{\max}) \to \mathbb{R}^3, \quad t \mapsto \boldsymbol{S}_i(t, p_2^{\max}) \\
\boldsymbol{C}_{i,2}^{\min} &: (p_2^{\min}, p_2^{\max}) \to \mathbb{R}^3, \quad t \mapsto \boldsymbol{S}_i(p_1^{\min}, t) \\
\boldsymbol{C}_{i,2}^{\max} &: (p_2^{\min}, p_2^{\max}) \to \mathbb{R}^3, \quad t \mapsto \boldsymbol{S}_i(p_1^{\max}, t)
\end{aligned}$$

and let their respective lengths be given exemplary for $\boldsymbol{C}_{i,1}^{\min}(t)$ by

$$L_{i,1}^{\min} := \int_{p_1^{\min}}^{p_1^{\max}} |(\boldsymbol{C}_{i,1}^{\min})'(t)| \, dt. \tag{4.46}$$

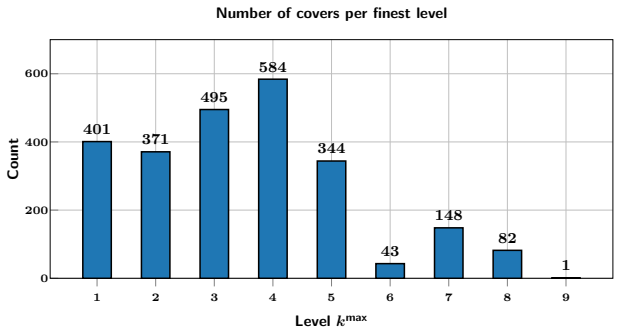

Figure 4.32: Histogram of the levels $k_i^{\max}$ which the covers C_{Ω_i} per surface $\boldsymbol{S}_i$ of the inner shell of the door model have been refined to.

Then we define $\boldsymbol{h}_{\boldsymbol{S}_i} \in \mathbb{R}^2$ with

$$(\boldsymbol{h}_{\boldsymbol{S}_i})_m := \max\{L_{i,m}^{\min}, L_{i,m}^{\max}\}, \quad m \in \{1, 2\}$$

as the extend of the surface in the world space. Note that (4.46) can be given explicitly for elementary surfaces like planes, cylindrical, conical, toroidal or spherical surfaces and for NURBS surfaces it is sufficient to approximate that integral very roughly. With the extends in the parameter space and the three-dimensional space we can define an extend ratio $\boldsymbol{\beta}_i \in \mathbb{R}^2$ for each parameter direction m by

$$(\boldsymbol{\beta}_i)_m := (\boldsymbol{h}_{\boldsymbol{S}_i})_m / (\boldsymbol{h}_{\mathcal{P}_{\boldsymbol{S}_i}})_m. \tag{4.47}$$

Let us then assume that $\boldsymbol{h}_{\Omega_i} \in \mathbb{R}^2$ is the anisotropic extend of the bounding box of Ω_i and $\hat{h} \in \mathbb{R}$ is the desired patch diameter in the three-dimensional domain that we want to achieve across all surfaces. We can then define

$$k_{i,m}^{\min} := \max\left\{0, \operatorname{round}\left(\log_2\left((\boldsymbol{h}_{\Omega_i})_m (\boldsymbol{\beta}_i)_m \hat{h}^{-1}\right)\right)\right\} \tag{4.48}$$

as the minimum level so that patches on that level have an extend in direction m comparable to the desired extend $\hat{h}$. To resolve the anisotropy, we can then stretch the bounding box of Ω_i that is used to initialize the root node during the construction of the cover C_{Ω_i}, anisotropically corresponding to the aspect ratio $(\boldsymbol{\beta}_i)_1/(\boldsymbol{\beta}_i)_2$. Then, we refine each cover uniformly to their respective level $k_i^{\max} := \max\{k_{i,1}^{\min}, k_{i,2}^{\min}, k_*^{\min}\}$ where $k_*^{\min} \in \mathbb{N}_0$ is a global, coarsest level we want all covers to be at least refined to. For the door model we did use $k_*^{\min} = 1$ to make sure that no cover consists of a single patch, which would limit the freedom to represent the interface-constraints from neighboring faces significantly. Additionally a desired patch size of $\hat{h} = 3$ mm has been employed, which resulted in a total of 148 898 patches across the covers on all faces. A histogram of the levels that are employed per cover is depicted in Figure 4.32. It is important to note that over half of the covers (1 267 of 2 469) have only been refined to very coarse levels $k_i^{\max} \leq 3$. This emphasis the required robustness of the overall geometry operations to support even very coarse

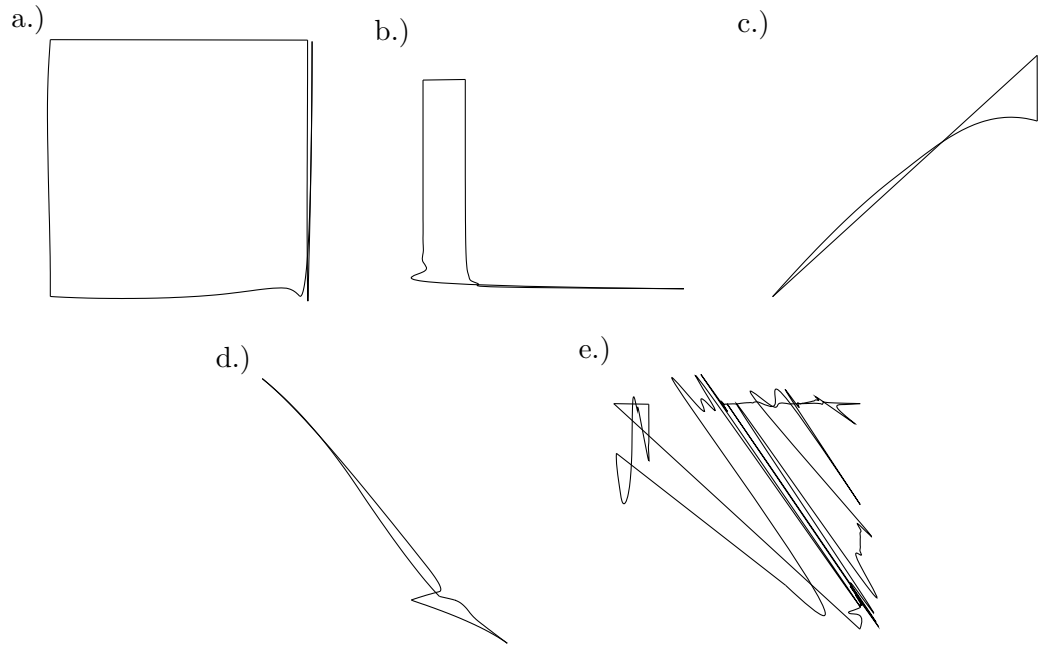

Figure 4.33: Boundaries $\partial\Omega_i$ of the five significantly self-intersecting trim domains included in the door model.

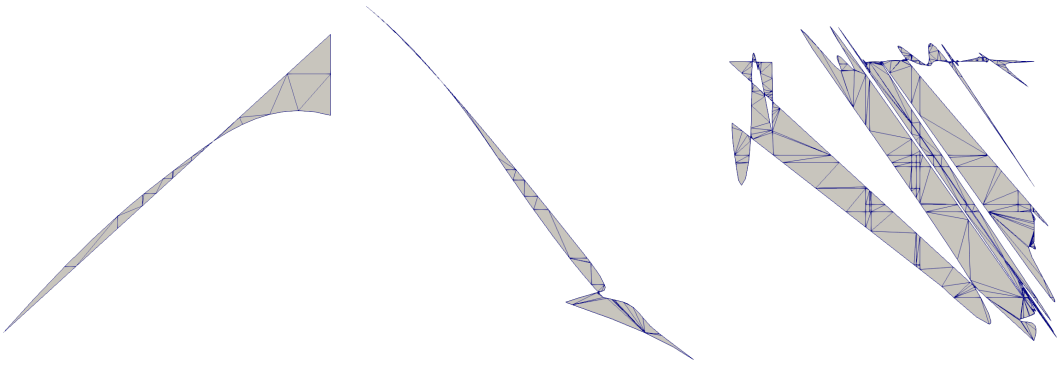

Figure 4.34: Resulting integration cells for the trim-domains c.), d.) and e.) from Figure 4.33.

levels without enforcing additional refinements of patches only to workaround special geometric constellations. It should additionally be noted that the CAD file contains five surfaces where the trim domains have significant[13] self-intersections in the descriptions of the boundaries $\partial\Omega_i$ (compare Figure 4.33). It is very likely that those trim domains were not intended to be self-intersecting originally, but the self-intersections arose due to robustness issues during the surface-surface intersection computations and especially during the creation of the approximate curves in the parameter domain. Nevertheless the methods presented throughout this chapter are *fault-tolerant* with respect to such self-intersections and hence valid integration cells were created for all those trim domains (compare Figure 4.34).
A generic steel material with Young's modulus $E = 200\,000$ MPa, Poisson's ratio $\nu = 0.3$ and density $\rho = 7\,900$ kg $\cdot$ m^{-3} was used. A shell thickness $\tau = 1$ mm was employed. Simply supported boundary conditions were applied on all edges that form the outermost boundary of the complete shell structure. A gravitational acceleration of 9.81 m $\cdot$ s^{-2} was acting in direction $(0, -1, 0)$ on the complete shell via the body force $\boldsymbol{f}$. Local polynomial spaces of order $p = 2$ were used on all patches $\omega_{i,n}$ and quadratic splines were used for the weight functions $\mathcal{W}_{i,n}$ during the construction of the Shepard partition of unity functions $\varphi_{i,n}$. Results of the final displacement field $\boldsymbol{u}$ are depicted in Figure 4.35 and the von Mises stress at the top-surface ($\xi = \frac{\tau}{2}$), mid-surface ($\xi = 0$) and bottom-surface ($\xi = -\frac{\tau}{2}$) is shown in Figure 4.36.

[13] Many more trim domains contain self-intersections that we classify as *insignificant*. Those are usually self-intersections of parameter curves very close to vertices of the CAD file. All those self-intersections are within the tolerance assigned to the corresponding vertex by the CAD system and should thus be considered equal to the intersection (or rather connection) at the vertex itself.

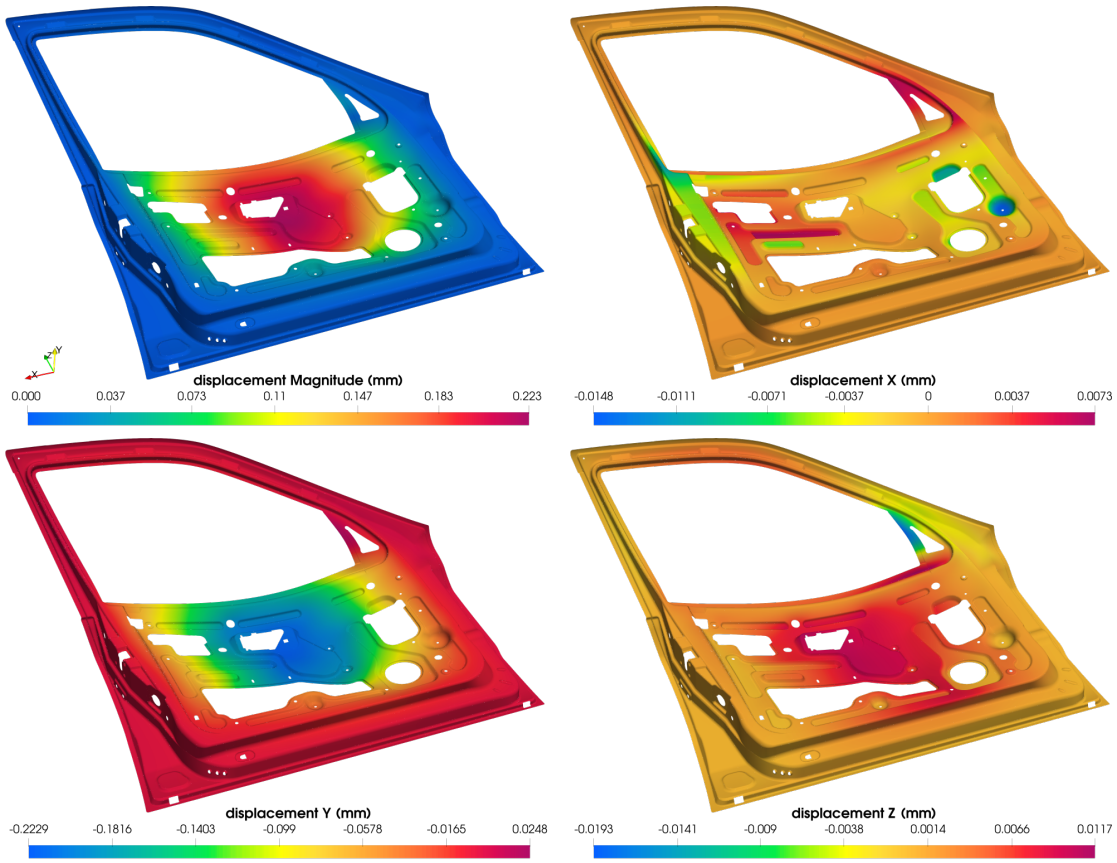

Figure 4.35: Color plots of th displacement field on the mid-surface. The magnitude of the displacement and the separate displacement components in x, y and z direction are depicted. The original geometry is warped by the displacement field, employing a scaling factor of 475.

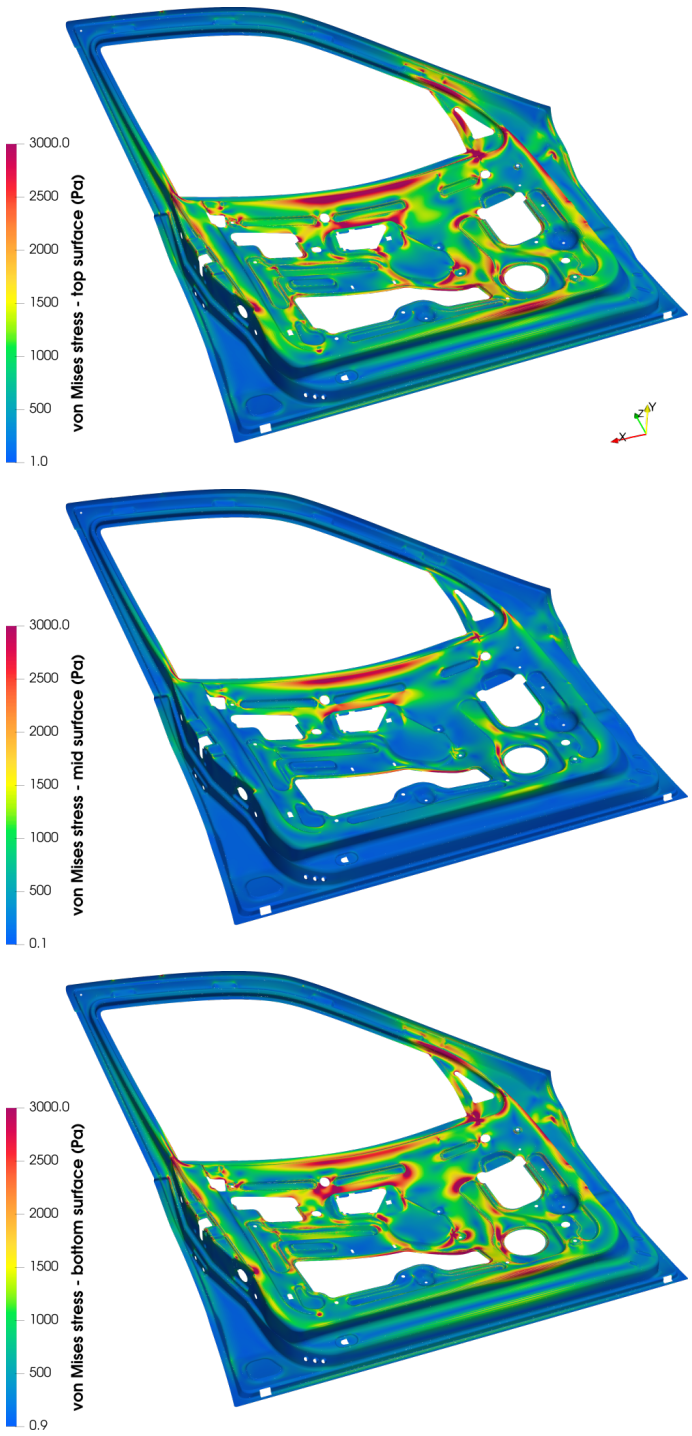

Figure 4.36: Von Mises stress at the top-, mid- and bottom-surface of the shell (from top to bottom). The original geometry is warped by the displacement field, employing a scaling factor of 475.

5 Approximated Geometry (3D)

While in Chapter 4 we were concerned with two-dimensional problems or BREP shell models, we now turn to three-dimensional problems where $\Omega \subset \mathbb{R}^3$ is represented by a solid BREP model as described in Chapter 3. To this end we need to solve the same problems as in two dimensions. Namely, we need to compute the intersections $\mathcal{D}_i^m \cap \Omega$ between the integration domains $\mathcal{D}_i^m$ and the simulation domain Ω and then decompose those intersections into smaller geometric entities where known Gaussian quadrature rules are available. Unfortunately, the problem of computing $\mathcal{D}_i^m \cap \Omega$ turns out to be a lot harder in three dimensions, than it was in two dimensions. In 2D, the domain was described by curves which we required to intersect with axis-aligned line segments. This could either result in curve parts in the case of partial overlaps or in individual points otherwise. Since the 3D domain is described by its boundary surfaces, we need to compute the intersection of parametrized surfaces and axis-aligned plane parts (the boundary faces of the axis-aligned boxes $\mathcal{D}_i^m$). The results of those intersections can be surface parts, curves, or individual points. Especially intersections resulting in curves confront us with the hardest problems. In Section 3.1 we already discussed the difficulties with representing the surface-surface intersection curves for arbitrary surfaces. But even if one of the surfaces is an axis-aligned plane part, the intersection curve with a bi-cubic surface patch would be of degree 18 [113] and intersection curves are usually still approximated only [140]. Besides the issues of curve approximation itself it is usually hard to guarantee that all intersection curves have been detected, especially when considering that the intersection curves can form loops that do not touch the boundary of any of the parametrized surfaces [19,47,61,74,101,114,115,124]. Then, even if all intersection curves could be found, the surface belonging to the domain Ω is potentially *trimmed* and the intersection curves are usually computed for the untrimmed surface first. This means to decompose the surface into the surface parts that are included in the box $\mathcal{D}_i^m$ and are required to represent the intersection $\mathcal{D}_i^m \cap \Omega$ as a BREP itself, we need to compute the intersection points of the intersection curves and the trim curves in each surface [75]. Hence we additionally need a robust and efficient curve-curve intersection algorithm.

Even though these problems have been tackled a lot in the past (the references given above are only a small selection of literature on that topic), the underlying complexity

of those problems remains persistent. This means that a robust implementation of the algorithms to compute the intersection of an arbitrary BREP solid and an axis-aligned box is still quite involved. Additionally the number of tasks (intersection computation, curve approximation, loop detection, trimming, computation of the resulting BREP topology) entail an amount of work per intersection computation that can hardly be reduced. Most of those algorithms have been developed to be applied during solid modelling, i.e. when a CAD engineer designs the model in question. These processes usually only require a small number of intersection computations, and the engineer performing those tasks can wait a few seconds for a single intersection computation to be performed. But the performance requirements become quite different when we need to perform thousands or millions of intersections when generating the integration cells in the PUM setting, since the number of integration cells $\mathcal{D}_i^m$ that need to be intersected with Ω depend linearly on the number of patches ω_i that intersect the boundary $\partial\Omega$.[1]
That is why we propose to **not** use the exact, curved geometry representation of Ω for three-dimensional solid geometries. Instead we create a linear approximation $\tilde{\Omega}$ where the boundary $\partial\tilde{\Omega}$ is represented by triangles only. We then compute the intersections of $\mathcal{D}_i^m$ with that approximated geometry $\tilde{\Omega}$ to generate the integration cells during the Galerkin discretization step in our PUM. Since $\partial\tilde{\Omega}$ consists of triangles only, a robust implementation to compute those intersections is a lot easier to achieve and can be executed a lot faster compared to when using the original curved domain Ω. In the two-dimensional Example 4.2 in Section 4.4 it was shown that optimal convergence rates of our PUM can be achieved even when the boundary of our domain Ω is approximated by linear segments, as long as we choose an appropriate approximation $\partial\tilde{\Omega}$ depending on the size of our patches ω_i and their respective local approximation spaces V_i. Driven by those results we compute a sequence of geometry approximations $\tilde{\Omega}_0, \ldots, \tilde{\Omega}_{k_{\max}}$ that matches the approximation power of the global function spaces V_k^{PU} on each level k of the covers C_Ω^k.
The remainder of this chapter is structured as follows: In Section 5.1 we describe how we compute an initial approximation $\partial\tilde{\Omega}$ of a curved BREP solid Ω represented by its boundary $\partial\Omega$ and in Section 5.2 we then describe how to compute the intersections $\mathcal{D}_i^m \cap \tilde{\Omega}$ and decompose those domains into tetrahedra to perform the numerical integration. In Section 5.3 we finally discuss how we can estimate the approximation error of $\tilde{\Omega}$ compared to the original Ω. We then put the estimated domain approximation error into relation to the employed cover C_Ω^k and a PUM space V_k^{PU} on levels k to propose a geometry refinement scheme that allows the overall method to converge with optimal rates.

5.1 Triangular Approximation of Curved BREPs

Let us start this section by discussing the properties the linear approximations $\tilde{\Omega}$ should have. In general we could use *any* kind of approximation of the domain that allows us to perform the subsequent operations (namely intersection and decomposition into integration cells) more performant and robust than on the original domain Ω. To make

[1] A naive implementation where the CAD kernel Open CASCADE [1] has been used to compute the necessary intersections could not provide the robustness, nor the performance required for our use case.

these operations as easy and fast as possible we propose that the approximation consists only of linear entities so that we can compute intersections fast and reliable. Specifically this means that all faces in the approximated geometry should be planes and all edges should be line segments. For simplicity and consistency we further restrict ourselves to use only triangles and no other plane polygons to approximate surfaces. To further simplify the subsequent operations we impose some additional restrictions: We want the approximation of a shell to be a single closed triangular mesh (often denoted as *watertightness*). I.e. every edge has exactly two neighboring triangles. This property makes it easy to partition the space unambiguously into points that are *inside* the closed shell and the ones *outside* of it. Additionally we want the triangular approximation to be non-self-intersecting. This is required by the algorithm that we apply to decompose $\mathcal{D}_i^m \cap \tilde{\Omega}$ into tetrahedra to form our final integration cells.[2]

Besides those hard requirements we have one main goal when constructing the geometry approximations: We want it to consist of as few triangles as possible such that the required accuracy can be achieved. This is because the number of triangles in the geometry approximation has a big influence on the number of integration cells that need to be generated at the boundary of the domain and hence is of significant importance for the overall performance of the assemble step in the PUM. On the other hand, it is very important to note that the "quality" (in the sense of the minimal angle of all triangles) is **not** an important measure for our geometry approximation. This is in contrast to most meshes used for finite element simulations and the reason for it is that we use the triangles for integration only and our basis functions are created completely independent of those triangles. Hence, the quality demands on our geometry approximation are more comparable to the ones when generating *tessellations* for visualization purposes than the ones for surface meshes generated for finite element simulations. Nevertheless, experience has shown that a mesh consisting of triangles with larger minimal angles often allows to get a better approximation of a curved surface with a lower number of triangles than a mesh with many sharp triangles. Thus, whenever we have the opportunity to reduce the number of triangles with small angles without introducing new triangles, we will try to do so.

Surface mesh generation has been studied a lot in the past, with some of the most notable application fields being finite element analysis (either for shell problems or as initial step to finally form a volume mesh) and visualization. The method presented in the following falls into the category of *parametric surface meshing*, where first a triangulation of the trim domain in the parameter space is created which is then mapped into the three-dimensional space by simply applying the surface parametrization to each vertex of the triangulation. One of the earliest such methods was proposed in [119]. Such an approach has two major benefits that we want to take advantage of in our use case: Due to having a direct association of each triangle in the three-dimensional world to the triangle in the parameter space we can easily evaluate error measures against the actual surface. The

[2]Note that the restrictions are not inherently given by the geometry approximation approach itself but by the algorithms we chose to solve the subsequent intersection and decomposition problems as described in Section 5.2. By choosing different algorithms for those problems we might be able to lift the restrictions on the approximation in the future.

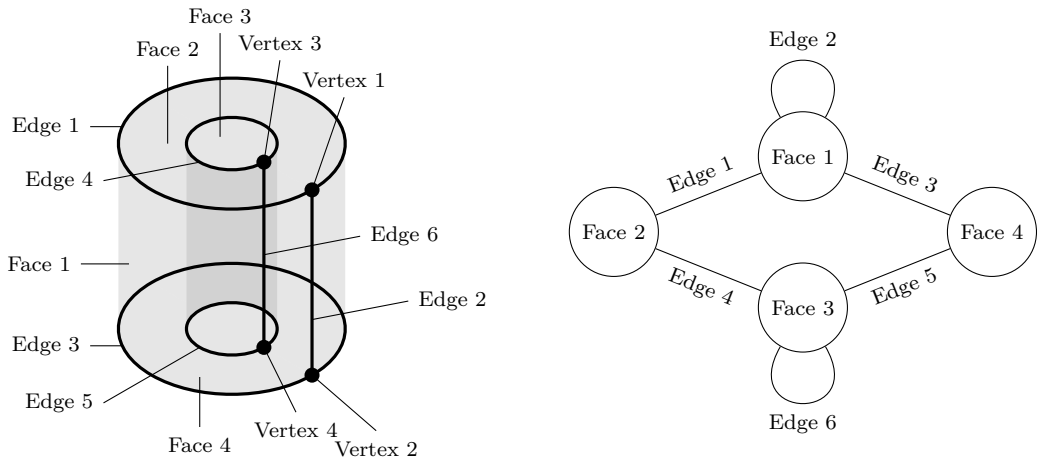

Figure 5.1: Solid BREP model of a tube domain represented by a single shell, consisting of four faces, six edges and four vertices. *Left:* Geometric representation of all topological entities. *Right:* Face-Edge connection graph.

kind of error norms we want to evaluate will be discussed in Section 5.3. Additionally the usage of an iterative triangulation algorithm in the parameter space allows us to efficiently refine the geometry approximation whenever the employed error estimator indicates that a refinement is necessary. We employ an approach very similar to what has been proposed in [71] and many subsequent papers, e.g. [5, 78]. All of the approaches that we are aware of have in common that the first step is to create linear approximations of the edges. The number and distribution of linear segments to be used is usually driven by an error estimate (commonly based on the *curvature* of the edge) and an error threshold based on the final approximation accuracy to be achieved. But in our use case the desired final approximation accuracy is in general unknown *a priori*, since it depends on the approximation power of the employed function spaces V_k^{PU} which we want to be controllable via an *a posteriori* error estimator that drives the h and p refinement of V_k^{PU}. Hence we want to start with an approximation that is *as coarse as possible* and then refine it only based on the currently employed V_k^{PU} when necessary. To this end, we create triangulations of the trim domains $\hat{\mathcal{P}}_{\boldsymbol{S}_i} \subseteq \mathcal{P}_{\boldsymbol{S}_i}$ sequentially for each surface $\boldsymbol{S}_i : \mathcal{P}_{\boldsymbol{S}_i} \to \mathbb{R}^3$ that forms the boundary of a BREP solid. Within each triangulation we approximate all curves by a single line segment only. Hence, we only insert points into the triangulation that correspond to topological vertex entities in the BREP. An exception is made for rings that consist of a single or only two curve parts. In this case we add one or two additional points along the curves to make sure that each ring is approximated by at least three points and can thus encapsulate a valid area. The triangulations are created by an incremental constrained Delaunay triangulation algorithm just like it was done in the two-dimensional case described in Section 4.2.2. Note that just like in the two-dimensional case it can happen that the linear constraints in the triangulation that approximate trim curves can intersect, even though the curves they approximate do not intersect. We resolve those cases by the same sub-division approach

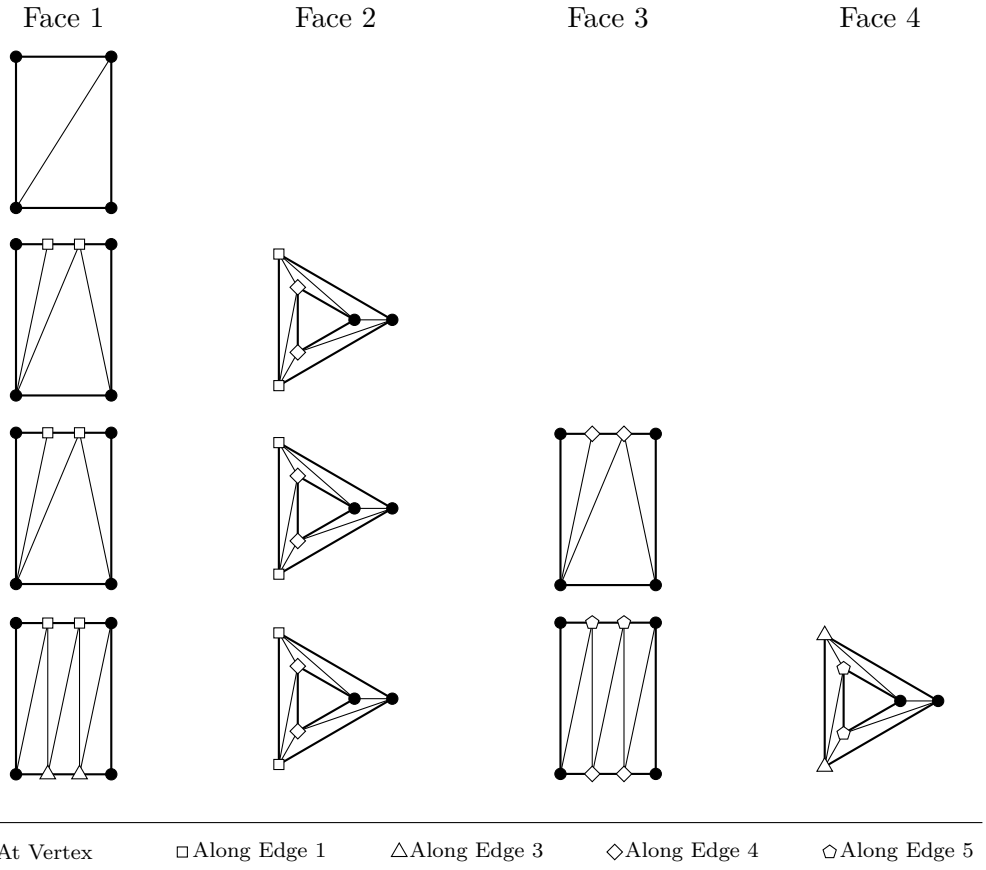

Figure 5.2: Subsequent triangulation steps of the four faces from the tube BREP depicted in Figure 5.1 (top row to bottom row).
Row 1: The triangulation of Face 1 is created. In consists of a single ring with four edges. Only the endpoints of the edges that correspond to vertices of the BREP are added. Note that due to Face 1 having a periodic parametrization, both vertices on the top edge in the triangulation correspond to Vertex 1 of the BREP and the vertices on the bottom edge correspond to Vertex 2.
Row 2: The triangulation of Face 2 is created. It has two rings, each consisting of a single edge. Two additional vertices along each of the edges are added. Edge 1 is shared between Face 1 and Face 2, hence the vertices are transferred to Face 1 and added into its triangulation.
Row 3: The triangulation of Face 3 is created. Additional vertices along Edge 4 are immediately added into the triangulation.
Row 4: The triangulation of Face 4 is created. Similar to Face 2, additional vertices are added along Edge 3 and Edge 5. The vertices are transferred to the connected Faces 1 and 3 and added into the respective triangulations.

as in the two-dimensional case (compare Figure 4.7). The points that are added during this sub-division approach (just like the additional points we add for rings consisting of less than three curve parts) need to be treated specially. Since they are located along edges of the BREP they have to be added to the triangulation of the neighboring face that is connected via that edge. Let us assume we have to add a point located at the parameter $t \in \mathcal{P}_{\boldsymbol{C}_n}$ along the curve $\boldsymbol{C}_n$ that is part of the boundary $\partial\hat{\mathcal{P}}_{\boldsymbol{S}_i}$. Let us further assume that the face corresponding to the surface $\boldsymbol{S}_i$ is connected to another face represented by a surface $\boldsymbol{S}_j$ along an edge that is represented by the curve $\boldsymbol{C}_n$ in $\mathcal{P}_{\boldsymbol{S}_i}$ and the curve $\boldsymbol{C}_m$ in $\mathcal{P}_{\boldsymbol{S}_j}$. We first transfer the point to be added into the three-dimensional space by evaluating the curve and then the surface to get $\boldsymbol{x}_t := (\boldsymbol{S}_i \circ \boldsymbol{C}_n)(t)$. Then we can find the parameter location $\boldsymbol{\xi}_t \in \mathcal{P}_{\boldsymbol{S}_j}$ of the point on $\boldsymbol{S}_j$ that is closest to $\boldsymbol{x}_t$ by solving

$$\boldsymbol{\xi}_t := \underset{\boldsymbol{\xi} \in \mathcal{P}_{\boldsymbol{S}_j}}{\arg\min} \|\boldsymbol{S}_j(\boldsymbol{\xi}) - \boldsymbol{x}_t\|. \tag{5.1}$$

In an ideal world it should be $\|\boldsymbol{S}_j(\boldsymbol{\xi}_t) - \boldsymbol{x}_t\| = 0$ but due to gaps in the BREP representation, it is usually only $\|\boldsymbol{S}_j(\boldsymbol{\xi}_t) - \boldsymbol{x}_t\| \leq \epsilon$ where ϵ is some tolerance assigned to the edge in the BREP. After that we can solve a similar problem

$$s_t := \underset{s \in \mathcal{P}_{\boldsymbol{C}_m}}{\arg\min} \|\boldsymbol{C}_m(s) - \boldsymbol{\xi}_t\| \tag{5.2}$$

to find the parameter $s_t \in \mathcal{P}_{\boldsymbol{C}_m}$ along the curve $\boldsymbol{C}_m$. Again we can only expect to get points where $\|\boldsymbol{C}_m(s_t) - \boldsymbol{\xi}_t\| \leq \epsilon$ due to the non-matching representations of $(\boldsymbol{S}_i \circ \boldsymbol{C}_n)$ and $(\boldsymbol{S}_j \circ \boldsymbol{C}_m)$.[3] In the past, many algorithm have been presented to solve (5.1) and (5.2), e.g. [17, 33, 81]. A method that gave sufficient performance and robustness to solve the two-dimension point to curve projection problem was presented in [18]. After solving (5.1) and (5.2) we add the point $\boldsymbol{C}_m(s_t)$ into the triangulation of $\hat{\mathcal{P}}_{\boldsymbol{S}_j}$ if it has been created already (i.e. if $j \leq i$). Otherwise we remember that point and will add it immediately when creating the triangulation for surface $\boldsymbol{S}_j$.

After all triangulations for the trimmed parameter domains of all involved faces have been generated, we can now create a closed, conforming, non-self-intersecting three-dimensional surface mesh for each BREP shell. To create the vertices of the surface mesh we simply need to apply the surface maps $\boldsymbol{S}_i$ to the vertices in the triangulation of the corresponding trim domains $\hat{\mathcal{P}}_{\boldsymbol{S}_i}$. Since each triangulation point on an edge is present in two faces and triangulation points that belong to BREP vertices can be present on even more faces, we need to take some care to not create duplicated vertices in the surface mesh. Hence, we transfer every point just once by simply using only the point and the surface map $\boldsymbol{S}_i$ with the lowest index i for the transfer.[4] The topology of the surface mesh (i.e. the connections between the three-dimensional vertices) is then simply the same as the topology of the two-dimensional triangulations. At this point we have a closed, conforming surface mesh

[3] Alternatively, a slightly better transfer could be achieved by combining (5.1) and (5.2) to solve directly for $s_t := \arg\min_{s \in \mathcal{P}_{\boldsymbol{C}_m}} \|(\boldsymbol{S}_j \circ \boldsymbol{C}_m)(s) - \boldsymbol{x}_t\|$ but since $(\boldsymbol{S}_j \circ \boldsymbol{C}_m)$ is only implicitly represented this problem is potentially a lot harder to solve robustly.

[4] Alternatively we could compute the average of all three-dimensional points, but we did not see any benefit of doing this in practice.

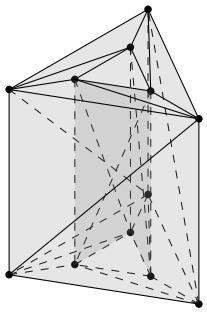

Figure 5.3: Initial triangular approximation of the tube BREP from Figure 5.1. It consists of 12 vertices, and 24 triangles.

but due to the triangles only approximating the curved surface, it can happen that some triangles are intersecting other ones. As discussed in the beginning of this section this is not a property that we tolerate and hence we need to resolve those self-intersections. The idea is very simply: In a first step we detect all pairs of intersecting triangles and collect them into a set. In the second step we refine all those triangles by inserting additional points into the corresponding triangulations and update the three-dimensional surface mesh accordingly. These steps are repeated until there are no more intersecting triangles. The detection of intersecting triangles can be speed-up by constructing an R-tree [57] with the bounding boxes of the triangles. When refining triangles we apply some heuristics to determine where to add the new points into the triangulations which will be described in more detail in Section 5.3.2.
The overall approach is depicted exemplary in Figure 5.2 for a solid BREP model of a tube domain Ω as shown in Figure 5.1. The resulting approximation $\tilde{\Omega}$ can be seen in Figure 5.3.

Remark 5.1. It is important to note that the parametrizations of the surfaces $\boldsymbol{S}_i$ are not area-preserving in general, i.e. for $\mathcal{A} \subseteq \mathcal{P}_{\boldsymbol{S}_i}$ we have

$$\text{area}(\mathcal{A}) \neq \text{area}(\boldsymbol{S}_i[\mathcal{A}]).$$

In particular that means that curves along the surface are not arc-length parametrized, which usually leads to the parametrization being *anisotropic* in the sense that curves parallel to the different axes in the parameter space get stretched by different factors when applying the surface parametrization. As a consequence of this anisotropy, a triangulation that fulfills the Delaunay criterion in the parameter space will most likely not fulfill that criterion when mapped to the world space. In contrast it is quite common for the mapped triangulations to be highly distorted and to contain very sharp triangles. Unlike surface meshes that are used for finite element simulations, such sharp triangles do not impose fatal problems when employed during the assemble step in the PUM. Nevertheless sharp triangles impose robustness issues to the following algorithms described in Section 5.2 due to limited floating point precision. Even though these robustness issues can be overcome by using *exact predicates* like those given in [120], applying those predicates to sharp triangles does usually increase their runtime cost. Hence there is still a benefit in at least

reducing the number of sharp triangles in our method. To this end, we do not create the triangulations directly in the parameter space $\mathcal{P}_{\boldsymbol{S}_i}$ of the surface $\boldsymbol{S}_i$. Instead we employ a transformation $\Phi_{\boldsymbol{S}_i} : \mathcal{P}_{\boldsymbol{S}_i} \to \mathbb{R}^2$ that transforms any point before we insert it into our triangulation. Hence, the triangulation is built in the image of $\Phi_{\boldsymbol{S}_i}$ instead of directly in $\mathcal{P}_{\boldsymbol{S}_i}$. We use a simple affine transformation

$$\Phi_{\boldsymbol{S}_i}(\boldsymbol{x}) := \begin{pmatrix} (\boldsymbol{\beta}_{\boldsymbol{S}_i})_1 & 0 \\ 0 & (\boldsymbol{\beta}_{\boldsymbol{S}_i})_2 \end{pmatrix} \boldsymbol{x}$$

with $\boldsymbol{\beta}_{\boldsymbol{S}_i}$ as given in (4.47). This transformation is designed to only resolve the anisotropy of the parametrization. Such a transformation does resolve most sharp triangles that we encountered in practice, is easy to compute, cheap to apply and has the additional benefit that its inverse $\Phi^{-1}_{\boldsymbol{S}_i}$ is trivial to form. The inverse is important when new points need to be added during the triangulation step, which then need to be mapped into the world space to form the actual surface mesh.

5.2 Intersections & Decomposition into Integration Cells

After constructing the domain approximation according the previous section, the boundary $\partial\tilde{\Omega}$ is only represented by triangles and hence the intersections $\tilde{\Omega} \cap \mathcal{D}_i^m$ should be a lot easier to compute. Since $\partial\mathcal{D}_i^m$ only consists of plane segments, the intersection $\tilde{\Omega} \cap \mathcal{D}_i^m$ is just the intersection of two polyhedral domains. The evaluation of such intersections has been well studied in the past. One of the first algorithms to intersect two polyhedra described by plane polygonal faces in the context of CAD has been presented in [77]. In [59] an algorithm has been presented where the boundary of both polyhedra consist of triangular faces only. Algorithms where both polyhedra are only represented by triangles are of very high interest since those representations are very common in computer graphics and some engineering fields as additive manufacturing. Hence a lot of literature exists on the topic that mainly deals with improvements of the efficiency and robustness of these intersection problems [39, 65]. In general all those algorithms utilize the fact that the boundary of the intersection $A \cap B$ of two polyhedra A and B is a combination of parts of the boundaries ∂A and ∂B. Ignoring degenerate intersection cases this can be written as

$$\partial(A \cap B) = (\partial A \cap \bar{B}) \cup (\bar{A} \cap \partial B). \tag{5.3}$$

Most algorithms decompose the boundary of both objects into disjoint parts that are either completely within the other object or completely outside of it. Then the boundary that describes the intersection can be constructed by selecting only those boundary parts that are within the other one by applying an appropriate classification algorithm. But in our case $\mathcal{D}_i^m$ is actually an axis-aligned bounding box and we can use this property to accelerate the intersection computation. To this end, we use two different algorithms to compute the two parts $\partial\tilde{\Omega} \cap \bar{\mathcal{D}}_i^m$ and $\bar{\tilde{\Omega}} \cap \partial\mathcal{D}_i^m$ of (5.3). In the following we briefly summarize the implemented approach.
In the first step to compute $\partial\tilde{\Omega} \cap \bar{\mathcal{D}}_i^m$ we employ the Sutherland-Hodgman clipping algorithm [129] to clip all triangles that form $\partial\tilde{\Omega}$ with the six axis-aligned planes that form the

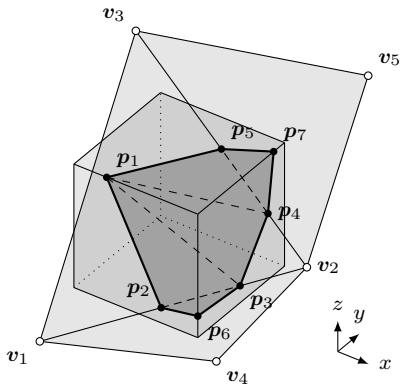

Figure 5.4: Three triangles $\mathcal{T}_1$, $\mathcal{T}_2$ and $\mathcal{T}_3$ defined by the vertices $(\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3)$, $(\boldsymbol{v}_1, \boldsymbol{v}_4, \boldsymbol{v}_2)$ and $(\boldsymbol{v}_2, \boldsymbol{v}_5, \boldsymbol{v}_3)$, respectively, are clipped by an axis-aligned bounding box $\mathcal{B}$. $\mathcal{T}_1 \cap \mathcal{B}$ results in a pentagon $(\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3, \boldsymbol{p}_4, \boldsymbol{p}_5)$, $\mathcal{T}_2 \cap \mathcal{B}$ results in a single triangle $(\boldsymbol{p}_2, \boldsymbol{p}_6, \boldsymbol{p}_3)$ and $\mathcal{T}_2 \cap \mathcal{B}$ in another triangle $(\boldsymbol{p}_4, \boldsymbol{p}_7, \boldsymbol{p}_5)$.

boundary of $\mathcal{D}_i^m$. Each clipped triangle results in a convex, plane polygon that consists of at most nine vertices which we can triangulate easily[5] and hence we do immediately retrieve a valid triangle representation of $\partial\tilde{\Omega} \cap \bar{\mathcal{D}}_i^m$ (compare Figure 5.4). In this step the R-Tree that we have constructed over all triangles to detect self-intersections in the end of Section 5.1 is used to significantly speed up the clipping. The hierarchical tree data structure can be used to quickly filter triangles that are either outside of $\mathcal{D}_i^m$ (which are discarded) or completely inside of it (which are transferred unchanged to the result). Only triangles that intersect the boundary of $\mathcal{D}_i^m$ need to be clipped.

For the second step, to compute $\bar{\tilde{\Omega}} \cap \partial\mathcal{D}_i^m$, we should first note that all new vertices (and edges) created during the clipping algorithm that are not already part of the mesh of $\partial\tilde{\Omega}$ are located on the boundary of $\mathcal{D}_i^m$. Since the mesh of $\partial\tilde{\Omega}$ is closed, these edges and vertices form connected lines on the six plane-segments of $\partial\mathcal{D}_i^m$ where both endpoints are on the boundary of the plane-segment or they form a closed ring (compare the first row in Figure 5.5). In fact these lines are exactly the intersection curves that form $\partial\bar{\tilde{\Omega}} \cap \partial\mathcal{D}_i^m$. The lines can then be used to partition each of the plane-segments into separate polygons (potentially with holes) by an algorithm that was proposed in [75]. Each of the polygons will then either be completely within $\tilde{\Omega}$ or outside of it. To distinguish which polygon is inside we will create a point within each polygon[6] and check whether that point is located within $\tilde{\Omega}$[7] (compare the second row in Figure 5.5). We discard all polygons that are not

[5]The triangulation can be performed by *fan triangulation* where one vertex is connected with all others, or we can simply use a table lookup for all seven possible cases (since the polygon can consist of between three and up to nine vertices).

[6]We can find such a point by computing the centroid of the bounding box of the polygon and then shot a ray through that point. By computing all intersection points with that ray and applying an even-odd rule we can find a segment of the ray that is within the polygon. We select the midpoint of that segment.

[7]We check this by shooting a (half-)ray through the point and computing all intersections of $\partial\tilde{\Omega}$ and the ray. Then we apply an even-odd rule on the number of intersections before the point.

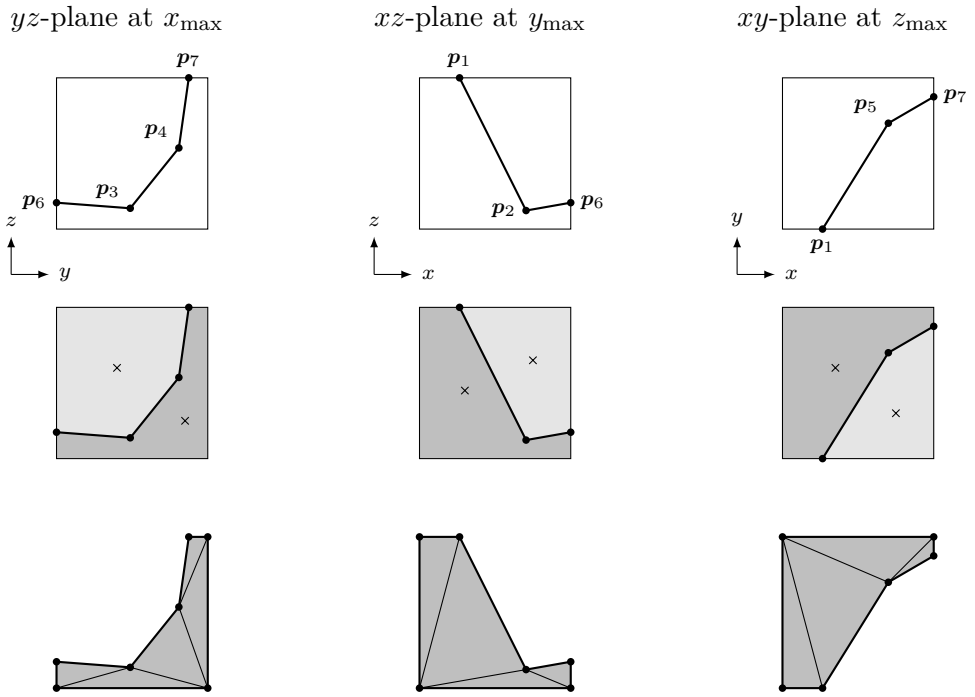

Figure 5.5: Decomposition of the axis-aligned plane segments that form $\partial\mathcal{D}_i^m$ after $\partial\tilde{\Omega}$ has been clipped by $\mathcal{D}_i^m$ as seen in Figure 5.4. Only the plane segments that intersect a triangle are shown.
Top: The vertices created during the clipping form polylines on the plane segments.
Middle: The polylines are used to split each of the plane segments into disjoint polygons. A point in the interior of each polygon is created (crosses).
Bottom: Polygons outside of $\tilde{\Omega}$ have been discarded and the other ones have been triangulated.

within $\tilde{\Omega}$ and create a triangulation for all the other polygons. Note that these polygons are not necessarily convex and can even have holes. One possibility to triangulate them is by applying the constrained Delaunay triangulation algorithm from Section 4.2 (compare the third row in Figure 5.5). The final triangular, conforming and closed mesh[8] that forms the boundary of the intersection $\tilde{\Omega} \cap \mathcal{D}_i^m$ is then formed by collecting all triangles from the clipping and the ones on the plane-segments (compare Figure 5.6).
Now we are left with the task of decomposing the intersections $\tilde{\Omega} \cap \mathcal{D}_i^m$ into integration cells. Since we have a triangle representation of its boundary, the most natural choice is to decompose the volume into tetrahedra. To this end we employ a 3D constrained tetrahedralization provided by Tetgen [123]. It is worth mentioning that in contrast to the triangular decomposition for the 2D case presented in Section 4.2 our description of the boundary in 3D is linear already. Hence there is no need to worry about any self-

[8] In fact the result can consist of multiple (volumetric) disjoint closed meshes.

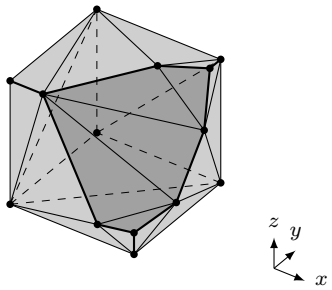

Figure 5.6: Mesh representation of $\partial(\tilde{\Omega} \cap \mathcal{D}_i^m)$ consisting of vertices from the mesh representation of $\partial\tilde{\Omega}$, the corner points that form $\mathcal{D}_i^m$ and intersection points created during the clipping algorithm.

intersections after curving the boundaries. Finally to integrate over the tetrahedra we can employ well-known Gaussian quadrature rules. One of the most recent sources for such quadrature rules that provides rules to integrate polynomials up to a degree of 20 can be found in [64].

5.3 Estimation of Approximation Errors & Local Refinement

Now that we have shown how to construct an approximation $\tilde{\Omega}$ of the original domain Ω and how to use such an approximation to efficiently create integration cells for the PUM, we want to discuss the influence that the domain approximation has on the overall error when carrying out simulations over $\tilde{\Omega}$ that where originally formulated over Ω. To this end, we do develop a complete error theory with thorough error estimates. We rather motivate simple estimates that are easily accessible, straightforward to implement, efficient to evaluate and of practical significance. In a subsequent step we show how we can implement refinement strategies for the domain approximation $\tilde{\Omega}$ that are driven by the developed error estimates.

5.3.1 Error Estimation

Let us assume that the error $\|u - u^{\mathrm{PU}}\|$, where $u \in H^1(\Omega)$ is the actual solution to our PDE and $u^{\mathrm{PU}} \in V^{\mathrm{PU}}(\tilde{\Omega})$ is the approximate solution we retrieve from the PUM, is subject to similar behavior as described by the *lemmas of Strang*. Then the error bound for $\|u - u^{\mathrm{PU}}\|$ consists of two components: the one commonly denoted the *approximation error*, that results from the discretization of the *function space* and the one denoted the *consistency error*, that results from the discretization of the *equation* to be solved. The approximation error is resolved via *h*-, *p*-, *hp*-refinement or in general by improving the approximation power of the local function spaces V_i and error bounds were established in Theorem 1.1. In the following we are more concerned with the consistency error. That error is well known to include errors that arise due to insufficient numerical integration or

floating point precision in general. But in addition to that the consistency error is related to the discretization of the simulation domain Ω. Let us assume that the consistency error is in general related to a term

$$|a(\cdot,\cdot) - a_h(\cdot,\cdot)| \tag{5.4}$$

where $a_h(\cdot,\cdot)$ is the discretized variant of the original continuous bilinear form $a(\cdot,\cdot)$ from the given variational problem. Let us assume that $a(\cdot,\cdot)$ is mainly described by an integral

$$\int_\Omega f(\boldsymbol{x})\ d\boldsymbol{x} \tag{5.5}$$

where surface integrals have been omitted for simplicity and f is just a placeholder for the actual integrand corresponding to the given PDE. In the discretized form of $a_h(\cdot,\cdot)$, this translates to

$$\int_{\tilde{\Omega}} f(\boldsymbol{x})\ d\boldsymbol{x} \tag{5.6}$$

where the integral over Ω has been replaced by an integral over $\tilde{\Omega}$.[9] Let us assume that the integral in (5.6) can be computed exactly.[10] Let us then introduce $\tilde{\Omega}^{=} := \Omega \cap \tilde{\Omega}$ as the region where both domains agree, $\tilde{\Omega}^{-} := \Omega \setminus \tilde{\Omega}$ as the region where our approximated domain is to small and $\tilde{\Omega}^{+} := \tilde{\Omega} \setminus \Omega$ as the region where our approximated domain is to large. Then we can decompose the domains into

$$\Omega = \tilde{\Omega}^{=} \cup \tilde{\Omega}^{-} \quad \text{and} \quad \tilde{\Omega} = \tilde{\Omega}^{=} \cup \tilde{\Omega}^{+}.$$

The error introduced by the domain approximation that we get from applying (5.5) and (5.6) to (5.4) can then be decomposed similarly, so we get

$$\begin{aligned}
&\Big| \int_\Omega f(\boldsymbol{x})\ d\boldsymbol{x} - \int_{\tilde{\Omega}} f(\boldsymbol{x})\ d\boldsymbol{x} \Big| \\
&\quad = \Big| \Big(\int_{\tilde{\Omega}^{=}} f(\boldsymbol{x})\ d\boldsymbol{x} + \int_{\tilde{\Omega}^{-}} f(\boldsymbol{x})\ d\boldsymbol{x} \Big) - \Big(\int_{\tilde{\Omega}^{=}} f(\boldsymbol{x})\ d\boldsymbol{x} + \int_{\tilde{\Omega}^{+}} f(\boldsymbol{x})\ d\boldsymbol{x} \Big) \Big| \\
&\quad = \Big| \int_{\tilde{\Omega}^{-}} f(\boldsymbol{x})\ d\boldsymbol{x} - \int_{\tilde{\Omega}^{+}} f(\boldsymbol{x})\ d\boldsymbol{x} \Big| \\
&\quad \leq \Big| \int_{\tilde{\Omega}^{-}} f(\boldsymbol{x})\ d\boldsymbol{x} \Big| + \Big| \int_{\tilde{\Omega}^{+}} f(\boldsymbol{x})\ d\boldsymbol{x} \Big|
\end{aligned} \tag{5.7}$$

were in the last step the triangle inequality has been applied. Finally we ignore the actual integrand f and simply assume $f(x) = 1$ to define our estimate for the domain approximation error as

$$\tau := \int_{\tilde{\Omega}^{-}} 1\ d\boldsymbol{x} + \int_{\tilde{\Omega}^{+}} 1\ d\boldsymbol{x}. \tag{5.8}$$

Let us now take a look at how we can compute τ. Considering the surface $\boldsymbol{S}_i$, let $\mathcal{T}_{i,n} \subset \mathcal{P}_i$ be one of the triangles in the parameter space that is defined by the vertices $\boldsymbol{v}_{i,n}^m \in \mathcal{P}_i$

[9] Here we assume that the integrand f is the same for $a(\cdot,\cdot)$ as well as $a_h(\cdot,\cdot)$ and that f is defined for all x in both Ω and $\tilde{\Omega}$.

[10] In practice that integral is only approximated by applying a discrete quadrature rule, but this kind of discretization error is not subject of the current discussion.

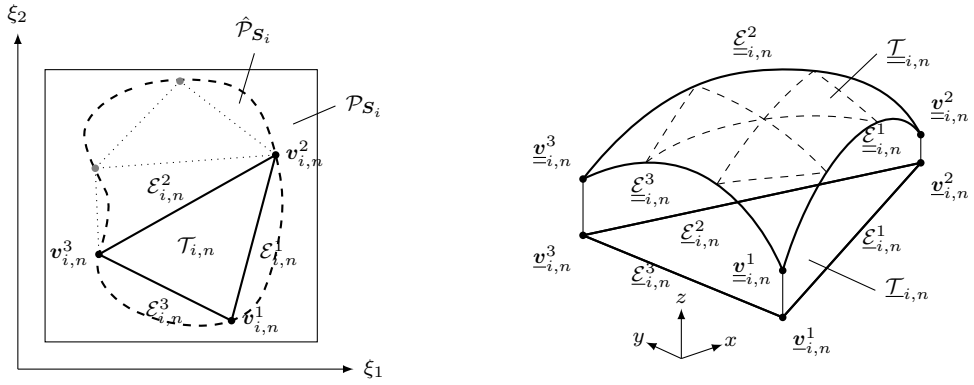

Figure 5.7: *Left:* Parameter space of the a surface $\boldsymbol{S}_i$ with the triangle $\mathcal{T}_{i,n}$, its edges $\mathcal{E}^m_{i,n}$ and vertices $\boldsymbol{v}^m_{i,n}$ from the triangulation of the trim domain $\hat{\mathcal{P}}_{\boldsymbol{S}_i} \subset \mathcal{P}_{\boldsymbol{S}_i}$.
Right: The triangle $\underline{\mathcal{T}}_{i,n}$ from the surface mesh in the world space that corresponds to $\mathcal{T}_{i,n}$ and the curved triangle $\underline{\underline{\mathcal{T}}}_{i,n}$.

for $m \in \{1,2,3\}$. Then let $\underline{\mathcal{T}}_{i,n} \subset \partial\tilde{\Omega}$ be the three-dimensional counterpart of that triangle, defined by the vertices $\underline{\boldsymbol{v}}^m_{i,n} \in \mathbb{R}^3$. Additionally let $\underline{\underline{\mathcal{T}}}_{i,n} := \boldsymbol{S}_i[\mathcal{T}_{i,n}]$ be the curved counterpart of the triangle on the surface and $\underline{\underline{\boldsymbol{v}}}^m_{i,n} := \boldsymbol{S}_i(\boldsymbol{v}^m_{i,n})$ its vertices. Finally let the edges of the triangle in the parameter space be denoted as $\mathcal{E}^m_{i,n} \in \mathcal{P}_i$ for $m \in \{1,2,3\}$, the respective linear edges in the three-dimensional space $\underline{\mathcal{E}}^m_{i,n}$ and the curved edges on the surface $\underline{\underline{\mathcal{E}}}^m_{i,n}$ (compare Figure 5.7). Note that we would expect $\underline{\boldsymbol{v}}^m_{i,n}$ and $\underline{\underline{\boldsymbol{v}}}^m_{i,n}$ for a given m to be the same point, but the vertex $\boldsymbol{v}^m_{i,n}$ can be located on the boundary of the trim domain $\partial\mathcal{P}_i$ and thus belong to the representation of a CAD edge. Hence, it is possible that $\underline{\boldsymbol{v}}^m_{i,n}$ has been created for a neighboring surface $\boldsymbol{S}_j$ and we can only expect $\underline{\boldsymbol{v}}^m_{i,n} \approx \underline{\underline{\boldsymbol{v}}}^m_{i,n}$. Now, to compute τ, we want to compute the volume that is enclosed between $\underline{\mathcal{T}}_{i,n}$ and $\underline{\underline{\mathcal{T}}}_{i,n}$ for all triangles. To this end, we first define parametric representations for the edges $\mathcal{E}^m_{i,n}$ and $\underline{\mathcal{E}}^m_{i,n}$. Let

$$\boldsymbol{E}^m_{i,n}(t) := \boldsymbol{v}^m_{i,n} + t(\boldsymbol{v}^{(m \bmod 3)+1}_{i,n} - \boldsymbol{v}^m_{i,n})$$
$$\underline{\boldsymbol{E}}^m_{i,n}(t) := \underline{\boldsymbol{v}}^m_{i,n} + t(\underline{\boldsymbol{v}}^{(m \bmod 3)+1}_{i,n} - \underline{\boldsymbol{v}}^m_{i,n})$$

with $t \in [0,1]$ be the parametrizations of $\mathcal{E}^m_{i,n}$ and $\underline{\mathcal{E}}^m_{i,n}$ respectively. A parametrization of $\underline{\underline{\mathcal{E}}}^m_{i,n}$ can then be defined by the composition $(\boldsymbol{S}_i \circ \boldsymbol{E}^m_{i,n})(t)$. Finally we can define three surfaces that close the sides between each edge pair $(\underline{\mathcal{E}}^m_{i,n}, \underline{\underline{\mathcal{E}}}^m_{i,n})$ via a simple *ruled surface* construction given by

$$\boldsymbol{F}^m_{i,n}(r,s) := (1-s)\underline{\boldsymbol{E}}^m_{i,n}(r) + s\underline{\underline{\boldsymbol{E}}}^m_{i,n}(r)$$

with $r,s \in [0,1]$. Let $\mathcal{F}^m_{i,n} := \{\boldsymbol{F}^m_{i,n}(r,s) \mid r,s \in [0,1]\}$, then we can define $\mathcal{V}_{i,n}$ to be the set of points that are enclosed by the faces $\underline{\mathcal{T}}_{i,n}$, $\underline{\underline{\mathcal{T}}}_{i,n}$ and $\mathcal{F}^m_{i,n}$, with $m \in \{1,2,3\}$, i.e. the wedge like object depicted on the right in Figure 5.7. Now, due to the *divergence theorem*

we can rewrite the volume integral of a vector field $\boldsymbol{\phi} : \mathbb{R}^3 \to \mathbb{R}^3$ into surface integrals, so we get

$$\int_{\mathcal{V}_{i,n}} \operatorname{div} \boldsymbol{\phi} \; d\boldsymbol{x} = \int_{\underline{\mathcal{T}}_{i,n}} \boldsymbol{\phi} \cdot \boldsymbol{n} \; ds + \int_{\underline{\underline{\mathcal{T}}}_{i,n}} \boldsymbol{\phi} \cdot \boldsymbol{n} \; ds + \sum_{m=1}^{3} \int_{\mathcal{F}_{i,n}^m} \boldsymbol{\phi} \cdot \boldsymbol{n} \; ds$$

with $\boldsymbol{n}$ being the outwards pointing normal vectors to each of the respective surfaces. Choosing any vector field where $\operatorname{div} \boldsymbol{\phi}(\boldsymbol{x}) = 1$, e.g. $\boldsymbol{\phi}(\boldsymbol{x}) := (0, 0, x_3)^T$, finally gives us a simple equation to compute the volume of $\mathcal{V}_{i,n}$ by

$$\int_{\mathcal{V}_{i,n}} 1 \; d\boldsymbol{x} = \int_{\underline{\mathcal{T}}_{i,n}} x_3 n_3 \; ds + \int_{\underline{\underline{\mathcal{T}}}_{i,n}} x_3 n_3 \; ds + \sum_{m=1}^{3} \int_{\mathcal{F}_{i,n}^m} x_3 n_3 \; ds =: \tilde{\tau}_{i,n}. \tag{5.9}$$

To compute the surface integrals in (5.9) we can employ simple Gauss quadrature rules. The parametrizations of the surfaces $\underline{\mathcal{T}}_{i,n}$ and $\underline{\underline{\mathcal{T}}}_{i,n}$ can be used to perform the integration in a two-dimensional reference triangle domain. The parametrizations $\boldsymbol{F}_{i,n}^m$ allow to perform the integration on each $\mathcal{F}_{i,n}^m$ in two-dimensional reference rectangle domains. We can then define an estimate for the total domain approximation error by

$$\tilde{\tau} := \sum_i \sum_n \tilde{\tau}_{i,n}. \tag{5.10}$$

Note that $\tilde{\tau}$ is just an approximation for τ from (5.8). One reason for this is that the triangles $\mathcal{T}_{i,n}$ do only form a linear approximation of the trim domain $\hat{\mathcal{P}}_i$ that transfers into an insufficient representation of the CAD edges on the boundary of each $\underline{\underline{\mathcal{T}}}_{i,n}$. Hence, if we consider an edge $\underline{\mathcal{E}}_{i,n}^m$ at the boundary of the triangulation and the same edge $\underline{\mathcal{E}}_{j,k}^m$ from the approximation of a neighboring surface $\boldsymbol{S}_j$, their respective curved edges $\underline{\underline{\mathcal{E}}}_{i,n}^m$ and $\underline{\underline{\mathcal{E}}}_{j,k}^m$ do usually not overlap and hence there is a gap or overlap in the global volume computation, even if we assume that the original geometry is perfectly watertight. On the other hand, that inaccuracy in the error estimation is going to get reduced when the triangulation in the parameter domain $\hat{\mathcal{P}}_i$ is refined along the edges. Another reason for inaccuracies in $\tilde{\tau}$ comes from the preconditions that are required for (5.9) to be accurate. One of those preconditions is that the surface of $\mathcal{V}_{i,n}$ is not allowed to be self-intersecting. But such cases can arise where surfaces $\boldsymbol{S}_i$ are neither convex, nor concave everywhere and hence $\underline{\underline{\mathcal{T}}}_{i,n}$ can run through $\underline{\mathcal{T}}_{i,n}$, which results in some parts of the actual volume canceling out each other or $\tilde{\tau}_{i,n}$ can even become negative.[11] To this end we propose a second way to estimate the error. Let $\underline{\boldsymbol{x}} \in \underline{\mathcal{T}}_{i,n}$ be a point in the parameter triangle and $\boldsymbol{\xi}_{\underline{\boldsymbol{x}}}$ its corresponding point in the parameter triangle $\mathcal{T}_{i,n}$. Then we define

$$\hat{\tau}_{i,n} := \int_{\underline{\mathcal{T}}_{i,n}} \| \underline{\boldsymbol{x}} - \boldsymbol{S}_i(\boldsymbol{\xi}_{\underline{\boldsymbol{x}}}) \| \; d\underline{\boldsymbol{x}} \tag{5.11}$$

as the local approximation error. There are some benefits of using (5.11) instead of (5.9) to measure local errors. For one thing, the computation of $\hat{\tau}_{i,n}$ does only require to evaluate

[11] Hence, in practice, we do actually use the absolute values $|\tilde{\tau}_{i,n}|$ for the local error estimate and in (5.10).

a single surface integral on a triangle and is hence cheaper to compute than $\tilde{\tau}_{i,n}$. Secondly the integrand in (5.11) is always positive. Hence, $\hat{\tau}_{i,n}$ is always positive and there is no canceling out effect in the case where $\underline{\mathcal{T}}_{i,n}$ and $\underline{\underline{\mathcal{T}}}_{i,n}$ intersect. On the other hand, (5.11) is only remotely related to the actual volume of $\mathcal{V}_{i,n}$, but it does indeed give a good estimate if the vector $\underline{\boldsymbol{x}} - \boldsymbol{S}_i(\boldsymbol{\xi}_{\underline{\boldsymbol{x}}})$ is (almost) orthogonal to $\underline{\mathcal{T}}_{i,n}$ and when $\underline{\underline{\mathcal{T}}}_{i,n}$ does not vary to much. Both of those properties are more likely to be fulfilled the finer the triangulation in the parameter space is. Hence, for ever finer domain approximations, we expect the difference between $\hat{\tau} := \sum_i \sum_n \hat{\tau}_{i,n}$ and $\tilde{\tau}$ to vanish, and both should eventually approach τ.

5.3.2 Refining the Domain Approximation

We now discuss a simple way to refine the domain approximation $\tilde{\Omega}$ based on the error estimates introduced in the last section. To this end, let us assume that we are at an iteration k where we do have a triangular description of $\partial\tilde{\Omega}_k$ and we have computed local errors $\tilde{\tau}_{k,i,n}$ or $\hat{\tau}_{k,i,n}$ for all triangles involved in the representation of $\partial\tilde{\Omega}_k$. We drop the distinction between $\tilde{\tau}_{k,i,n}$ and $\hat{\tau}_{k,i,n}$ in the following and simply refer to the estimated errors as $\tau_{k,i,n}$ and the respective global error as τ_k. Let us then assume that a desired domain approximation error $\bar{\tau}_{k+1} < \tau_k$ for the next iteration is given. The goal is to create a new domain approximation $\partial\tilde{\Omega}_{k+1}$ such that $\tau_{k+1} \approx \bar{\tau}_{k+1}$. To this end, we propose an algorithm that works in two steps: In the first step we collect a set of triangles that should be refined. In the second step those triangles are refined and the errors for the new triangles are computed. Those steps are repeated until the new global approximation error τ_{k+1} is less than the desired error $\bar{\tau}_{k+1}$. To this end, let us define

$$r^* := \tau_k - \bar{\tau}_{k+1} \tag{5.12}$$

as the target error reduction. Additionally let $\mathcal{T}_{\hat{n}}$ and $\tau_{\hat{n}}$ with $\hat{n} = 1, \dots, \hat{N}$ be the triangles and their respective errors without the distinction which surfaces $\boldsymbol{S}_i$ they belong to, hence $\hat{N}$ being the total number of triangles over all surfaces. Then, let us assume that the triangles are ordered by their errors in descending order, i.e. $\tau_{\hat{n}} \geq \tau_{\hat{n}+1}$ for all $\hat{n} = 1, \dots, \hat{N} - 1$. Let us then introduce $\beta \in (0, 1)$ as the *conservative error reduction factor* and $\sigma \in (0, 1)$ with $\sigma \leq \beta$ as the *optimistic error reduction factor*.[12] We then start to mark the triangles $\mathcal{T}_{\hat{n}}$ as to be refined until the estimated total error reduction after refining m triangles given by

$$r_m := \sum_{\hat{n}=1}^{m} \beta\tau_{\hat{n}} = r_{m-1} + \beta\tau_m \tag{5.13}$$

reaches r^*, i.e. $r_m \geq r^*$ or when we reach a triangle at index m such that

$$\tau_m < \sigma\tau_1. \tag{5.14}$$

The condition from (5.14) tries to avoid refining triangles with errors that are less than the errors we expect to be present on the new triangles that result from the refinement

[12] Throughout this thesis we used $\beta := 0.5$ and $\sigma := 0.25$.

of the triangle with the largest error. The idea behind that condition is that it should be beneficial to refine those new triangles in the next iteration first. Hence the algorithm above will eventually establish a state where the errors should be distributed evenly among all triangles. When either (5.13) or (5.14) are fulfilled or when we have reached $m = \hat{N}$ the collection step is finished and we go over to the refinement step.
The refinement step is simple to implement. For each triangle to refine we compute new points to be inserted into the parameter triangulations of its respective faces. Since we employed an incremental constrained Delaunay triangulation, adding new points into the triangulation is directly supported. We usually refine a triangle by adding three new points at the centers of its edges so that the refinement of a single triangle results in four new triangles. Hence, such a refinement strategy is often denoted the *1-to-4* refinement. An exception to this is when we have to deal with *sharp* triangles. For sharp triangles we do not add new points at the centers of all edges, but only a single point at the center of the longest edge. This is supposed to mitigate cases where two new points very close to each other would be added. Since all new points that we add are along edges of triangles, we need to take special care of constrained edges. Those edges represent parts of the originally curved description of the trim domain $\hat{\mathcal{P}}_i$. Hence, new points that are added along those edges are not added at the center of the linear edge from the triangle, but they are added at the parametric center of the corresponding curve part that represents $\partial\hat{\mathcal{P}}_i$. Additionally those points need to be added to the triangulation of the adjacent surface $\boldsymbol{S}_j$. To this end, they are transferred according to (5.1) and (5.2) and then added to the target triangulation. Special care has to be taken that only a single point is added for any refined edge between two triangles. This is true for two triangles that belong to the same surface as well as for two triangles that belong to different surfaces. Then, after all points have been added to the triangulations, we recompute the local error estimate $\tau_{k+1,i,n}$ according to (5.9) or (5.11) for all modified triangles. This does include triangles that have been newly created after a refinement, as well as triangles that have been modified due to flips being performed during the Delaunay insertion algorithm. Then, we can reevaluate the total domain approximation error τ_{k+1} and check whether we have reached our goal $\tau_{k+1} < \bar{\tau}_{k+1}$. If the goal has not yet been reached, we compute the remaining target error reduction according to (5.12) and continue with another refinement collection step.

Remark 5.2. A simpler version of the algorithm above could be formulated by keeping a heap like data structure of the triangles with their respective errors. Then, we always pop the triangle with the highest error, refine it immediately and update the heap with the newly inserted triangles and their errors.[13] We would stop as soon as we have reached our target error $\bar{\tau}_{k+1}$. But the benefit of formulating the algorithm in two steps as given above is that it is easier to adapt to common domain decomposition approaches for the parallelization in the PUM as e.g. given in [54]. Here the patches of a cover are distributed among multiple parallel entities (usually processes) and each process performs operations only local to its patches. Hence each process does only need to have the fully resolved

[13] The heap would be more difficult to update, since not only the refinement triangle needs to be removed and the new ones inserted, but triangles that got modified due to edge flips in the Delaunay insertion of a new point need to be handled correctly as well.

description of the domain approximation $\partial\tilde{\Omega}$ that intersects the patches it owns. To this end, the basic idea to adapt the algorithm described above to a parallel PUM is that each process would perform the collection step of the triangles to be refined only locally. Then the triangles to be refined are communicated to the neighboring processes, and the refinement is then applied locally by each process, synchronizing the total approximation error after each iteration. This has not yet been implement and is a topic for future work.

Let us now shortly discuss a few strategies how to make choices for the desired domain approximation errors $\bar{\tau}_{k+1}$. The basic idea is that according to the Strang lemmas, the total error is bounded by a combination of the approximation error and the consistency error. Since we have a solid theory for the convergence of the approximation error, we want to reduce the consistency error by the same rate so that the consistency error does never dominate the total error and hence slows down the overall convergence of the method. Based on that idea there are some ways to choose $\bar{\tau}_{k+1}$. One way is to simply assume that Theorem 1.1 and its h-version (1.4) hold from the beginning of the refinement sequence, so we can furthermore assume that there is a given convergence rate that depends on the global polynomial degree employed in the local function spaces. Considering the L^2-error, we expect the error to get reduced by a factor of $2^{-(p+1)}$, with p being the global polynomial degree, when refining the cover and the respective PUM space from a level k to a level $k+1$ by uniform h-refinement.[14] Hence, a very simply strategy is to make sure the domain approximation error gets reduced by the same factor for each level of refinement and the desired approximation error on a refined level $k+1$ is given by

$$\bar{\tau}_{k+1} := \tau_k \frac{1}{2^{(p+1)}}. \tag{5.15}$$

If we additionally assume that our domain refinement algorithm does not overrefine too much, i.e. $\tau_{k+1} \approx \bar{\tau}_{k+1}$ holds, we can adjust (5.15) to

$$\bar{\tau}_{k,p} := \tau_0 \left(\frac{1}{2^{(p+1)}}\right)^k \tag{5.16}$$

which allows us to directly give the desired approximation error for any level k based only on the initial domain approximation error τ_0 and the employed polynomial degree p. Some alternative strategies will be examined in the next section. A strategy where we try to estimate the actual approximation error on each level and then select a desired domain approximation error in relation to that estimate is studied in Example 5.2. An alternative strategy that is similar to (5.16), but tries to compensate for patches that are outside of the domain is studied in Example 5.3.

Remark 5.3. If the final level $k_{\max}$ is known beforehand we can use (5.16) to determine a final desired domain approximation error immediately. Hence, we can first refine the domain up to the accuracy required for the final level and then build the covers and PUM spaces via the hierarchical tree construction algorithm for that static domain, i.e. all covers $C_{\tilde{\Omega}}^k$ with $k = 0, \ldots, k_{\max}$ will be built according to the same domain approximation

[14] This would only hold under additional assumptions, like none of the children of a patch are outside of the domain or get eliminated by the cover post-processing step.

$\tilde{\Omega}_{k_{\max}}$. An alternative approach is to refine the domain approximation synchronously to the refinement of the cover, i.e. at a given level k, the covers $C^k_{\tilde{\Omega}}$ will be built according to the approximated domain $\tilde{\Omega}_k$. Such an approach is always required when the final level $k_{\max}$ is not known a priori, e.g. when the refinement process and stopping criterion is based on an error estimate for a solution u_k on the currently finest level (such a case will be demonstrated in Example 5.2). Additionally it can be beneficial to assemble the stiffness matrices and other operators required for a multilevel solver with respect to coarser domain approximations that correspond to the level of the cover instead of to the finest level since those coarser domain approximations result in a significantly lower amount of integration cells to be used on the coarser levels. But the adjustment of the domain approximation after each cover refinement imposes an additional difficulty. Let us assume we are on a level k with a cover $C^k_{\tilde{\Omega}}$, built for a domain $\tilde{\Omega}_k$ and then we refine the domain according to (5.16) or (5.15) for the level $k+1$. The cover $C^k_{\tilde{\Omega}}$ will not be valid anymore for the domain $\tilde{\Omega}_{k+1}$ since some tree cells $\mathcal{C}_{i,k}$ did not intersect $C^k_{\tilde{\Omega}}$ and thus were not stretched into patches $\omega_{i,k}$, but those cells would now intersect $\tilde{\Omega}_{k+1}$ and hence we are missing some patches in $C^k_{\tilde{\Omega}}$. Similarly, patches that were intersecting $C^k_{\tilde{\Omega}}$ could not intersect $\tilde{\Omega}_{k+1}$ anymore and should thus be removed from the cover. But not only is the cover $C^k_{\tilde{\Omega}}$ invalid for $\tilde{\Omega}_{k+1}$. Using that cover to construct a new cover $C^{k+1}_{\tilde{\Omega}}$ according to our hierarchical cover construction algorithm 1.1 would be incorrect as well, since we do only consider patches (or their respective tree cells) for refinement that were not outside of the domain on the previous level. Hence, we propose to update the cover $C^k_{\tilde{\Omega}}$ to be valid according to $\tilde{\Omega}_{k+1}$ before performing the refinement step. This can be done by a tree traversal algorithm from the root patch. For each patch we check whether its children do intersect $C^k_{\tilde{\Omega}}$ but not $C^{k+1}_{\tilde{\Omega}}$ or vice versa. If such cases are found we insert new patches into the respective covers or delete patches with all their children when they became outside patches. Note that we do not traverse the children of patches that were completely inside (or outside) of $\tilde{\Omega}_k$ and that stayed completely inside (or outside) of $\tilde{\Omega}_{k+1}$. Hence only patches that intersect either $\partial\tilde{\Omega}_k$ or $\partial\tilde{\Omega}_{k+1}$ need to be visited. After the update of the tree structure, we can create the new cover $C^{k+1}_{\tilde{\Omega}}$ by Algorithm 1.1 as usual and continue from there on.

5.4 Numerical Experiments

In the following we present the results of numerical experiments, conducted to show that the method presented in this chapter yields a converging refinement scheme. All the experiments are carried out for three-dimensional domains Ω where at least a part of the boundary is described by non-linear surfaces and hence we employ a sequence of domain approximations $\tilde{\Omega}_k$. We start by measuring error norms as given in (2.4) for a uniformly h-refined cover and show that we are indeed able to achieve good overall convergence towards smooth solutions when employing the domain approximation refinement as presented throughout this chapter. Then we proceed to show results for non-smooth solutions and adaptively refined covers. Finally we employ an example on a industrial CAD geom-

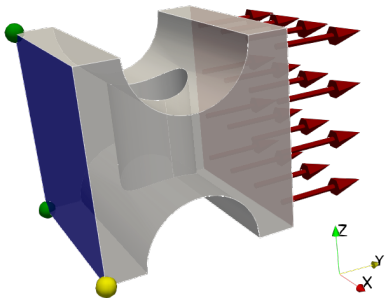

Figure 5.8: Geometry and boundary conditions for Example 5.1. The geometry is a cube $[0, 10]^3$ with two half-cylindrical cuts and one cylindrical hole. A force is applied on the face $\Gamma_N := \{\boldsymbol{x} \in \partial\Omega \mid x_2 = 10\}$ (red) and the displacement in y-direction is fixed on the face $\Gamma_D := \{\boldsymbol{x} \in \partial\Omega \mid x_2 = 0\}$ (blue). Additionally the x-displacement is fixed in the points $(0, 0, 0)^T$ and $(0, 0, 10)^T$ (green) and the z-displacement is fixed in the point $(10, 0, 0)^T$ (yellow).

etry described nearly entirely by curved NURBS surfaces and compare our results to the results presented in a reference paper where a classical FEM has been used to obtain the solution.

Example 5.1 (Cut-Cube with hole)**.** This first experiment can be considered as the three-dimensional equivalent of the two-dimensional Experiment 4.1. To this end, we again consider the equations of linear elasticity as given in (4.32) and compare sequences of solutions on uniformly refined covers for different polynomial degrees to a numerically computed reference solution. The domain Ω in this example is a cube $[0, 10]^3$ with two half-cylindrical cuts and one cylindrical hole. Let $\Gamma_D := \{\boldsymbol{x} \in \partial\Omega \mid x_2 = 0\}$ be the face at minimal y-coordinates and $\Gamma_N := \{\boldsymbol{x} \in \partial\Omega \mid x_2 = 10\}$ the opposite face at maximum y-coordinates. Then we employ the boundary conditions

$$\begin{aligned} u_2 &= 0 && \text{on } \Gamma_D, \\ \sigma(\boldsymbol{u}) \cdot \boldsymbol{n} &= (0, 0.01, 0)^T && \text{on } \Gamma_N, \\ \sigma(\boldsymbol{u}) \cdot \boldsymbol{n} &= 0 && \text{on } \partial\Omega \setminus (\Gamma_D \cup \Gamma_N) \end{aligned}$$

and we additionally fix the displacement in x-direction (i.e. $u_1 = 0$) in the points $(0, 0, 0)^T$ and $(0, 0, 10)^T$ and the displacement in z-direction (i.e. $u_3 = 0$) in the point $(10, 0, 0)^T$ (compare Figure 5.8). An artificial material with Young's modulus $E = 10$ and a Poisson's ratio $\nu = 0.3$ is used. An exemplary solution to that problem is depicted in Figure 5.9.
It is important to note that the domain is described by faces that are either planar or cylindrical surfaces that cut **into** the original cube. Hence the overall domain can be considered to have a "concave" kind of shape. This means that given a triangle that is part of the boundary description of an approximation $\tilde{\Omega}$, that triangle is always completely outside of the original domain Ω. Thus, no triangle intersects the surface it is approximating and it is safe to use (5.9) to compute the domain approximation error estimates $\tilde{\tau}_{i,n}$. Hence, whenever we use τ_k to refer to the domain approximation error on a level k in this

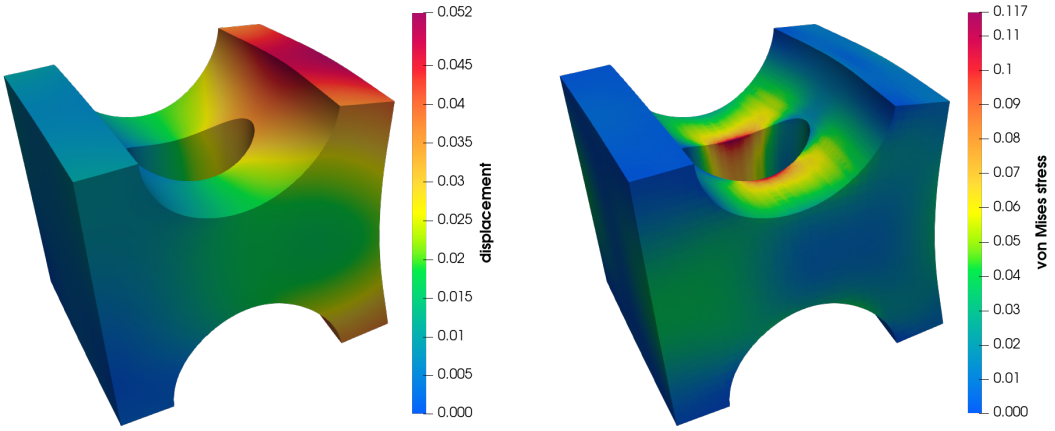

Figure 5.9: Magnitude of the displacement field (left) and von Mises stress (right) of the solution computed on level $k = 5$ employing polynomials of degree $p = 2$. The domain has been warped by the displacement field scaled by a factor of 28.

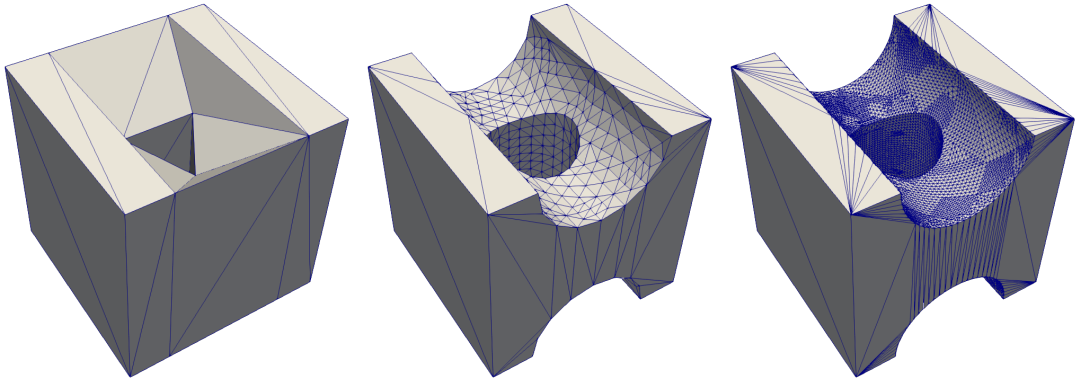

Figure 5.10: Approximations of the domain Ω for different levels k employing polynomials of degree $p = 1$. Initial approximation with estimated approximation error $\tau_0 \approx 156.46$ (left), approximation for level $k = 3$ with $\bar{\tau}_{3,1} \approx 2.44$ and $\tau_{3,1} \approx 2.32$ (center) and approximation for level $k = 5$ with $\bar{\tau}_{5,1} \approx 0.153$ and $\tau_{5,1} \approx 0.151$ (right).

example, we mean the estimated errors computed according to (5.9) and (5.10). Let us then consider that we create domain approximations $\tilde{\Omega}_k$ for a cover on level k subject to the desired domain approximation error according to (5.16). Then the concave property of the domain ensures that subsequent refinements of the approximated domain are always included within the coarser domain approximation, i.e.

$$\tilde{\Omega}_0 \supseteq \tilde{\Omega}_1 \supseteq \cdots \supseteq \tilde{\Omega}_k \tag{5.17}$$

holds for any $k \in \mathbb{N}_0$. Approximations for some exemplary levels are depicted in Figure 5.10. Due to the nesting property of the domain approximations, we compute all errors on a very fine reference domain $\tilde{\Omega}_{\text{ref}}$ that forms a subset of all the other domains $\tilde{\Omega}_k$. The final estimated domain approximation error of the employed reference domain $\tilde{\Omega}_{\text{ref}}$ is given by $\tau_{\text{ref}} \approx 1.161 \cdot 10^{-4}$ where for comparison the initial domain approximation error is given by $\tau_0 \approx 156.46$. We compute the reference solution $\boldsymbol{u}^*$ on that domain where we employ a polynomial degree $p_{\text{ref}} = 4$ on a uniformly refined cover up to level $k_{\text{ref}} = 6$ which results in a total of $N_{\text{ref}} = 132\,528$ patches and $\text{dof}_{\text{ref}} = 13\,915\,440$ degrees of freedom.

Let us first take a look at the convergence behavior subject to h-refinement for PUM spaces with different polynomial degrees p on static domain approximations $\tilde{\Omega}_q$ where $q \in \mathbb{N}$ is fixed for all levels k of the employed cover. The domain approximations $\tilde{\Omega}_q$ are created according to desired target domain approximation errors

$$\bar{\tau}_q := \tau_0 \left(\frac{1}{4}\right)^q \tag{5.18}$$

which is equal to the level dependent domain approximation strategy given by (5.16) for linear polynomials. We first conduct the experiments for polynomial degrees $p = 1, 2, 3$, each on a range of domain approximations $\tilde{\Omega}_q$ with $q = 3, \ldots, 10$. Results are shown in Figure 5.11. As first result, we want to point out that for $p = 1$ we can achieve close to optimal rates up until the final level $k = 7$ for a domain approximation using $q = 7$. Since (5.18) resembles the level dependent target domain approximation error (5.16) for linear polynomials, this indicates that this level dependent estimate is indeed sufficient. Additionally, we can see that coarser domain approximations impose higher errors on the final level. While for $q = 6$ the difference on the final level is not yet significant (but present), $q = 5$ does already give a visually distinguishable final error in the plot and the error for $q = 3$ is about a magnitude higher than the one for $q = 7$. These results imply that the level dependent estimate (5.16) is also not too pessimistic and will thus not impose too fine domain approximations that would lead to the creation of a large amount of unnecessary integration cells. In Figure 5.12 the results are depicted for the domains $\tilde{\Omega}_q$ with $q = 4, 7, 10$ and different polynomial degrees. Those plots clearly show that insufficient static domain approximations can lead to states where further refinement of the PUM space does not reduce the overall error of the retrieved solutions (similar to what has been observed in the two-dimensional Example 4.2). Only the very fine domain approximation with $q = 10$ allows to retrieve close to optimal rates in the L^2- and H^1-norms for all polynomial degrees up to the finest levels. The non-optimal behavior in

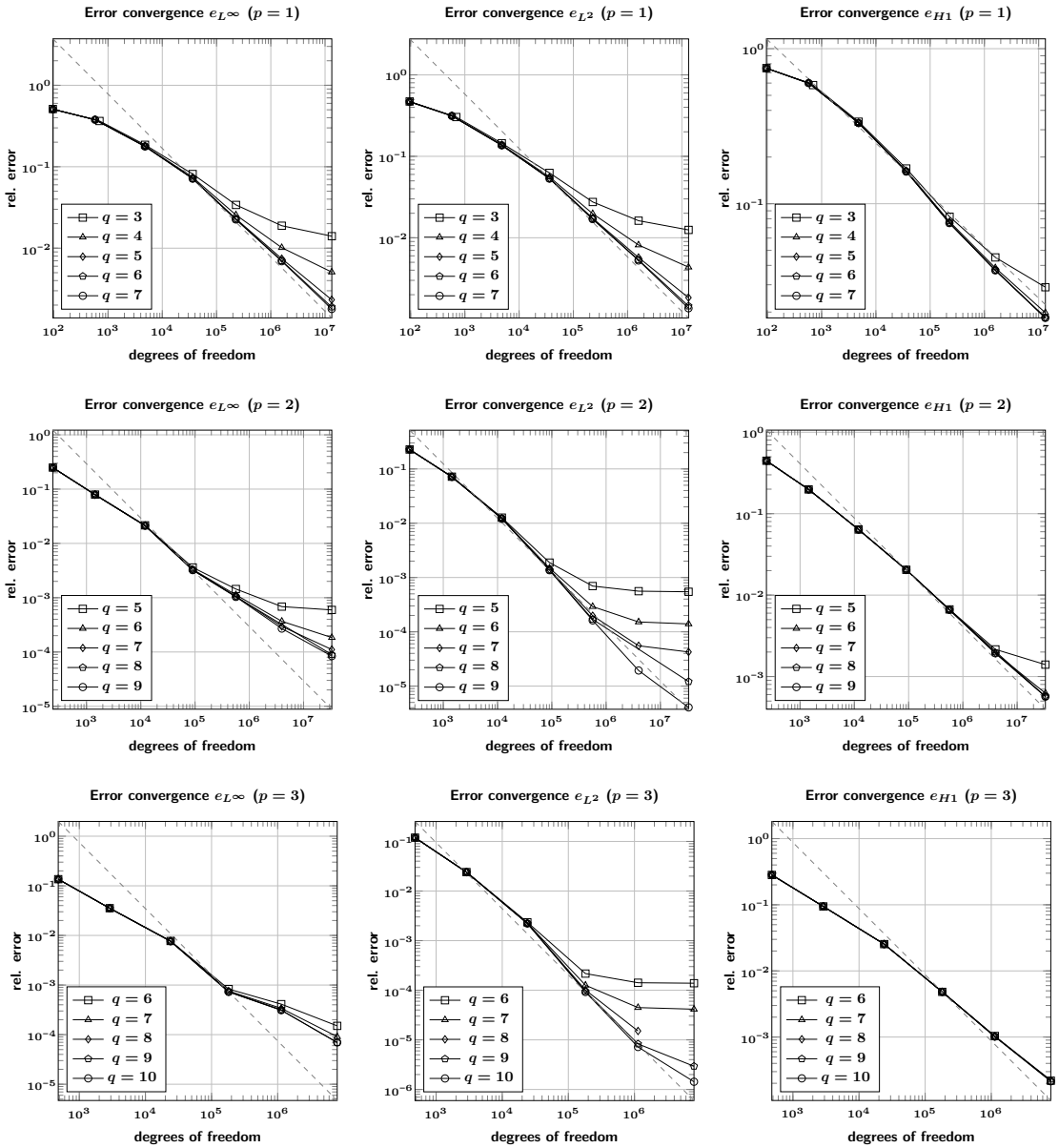

Figure 5.11: Error convergence for Example 5.1 (cut-cube) using polynomials of degrees $p = 1, 2, 3$ (top to bottom). Results on different static geometries $\tilde{\Omega}_q$ for $q = 3, \ldots, 10$ are shown. For $p = 1$ and $p = 2$ the results for levels $k = 1, \ldots, 7$ are shown and for $p = 3$ results on levels $k = 1, \ldots, 6$ are shown. Optimal convergence rates are depicted by dashed lines.

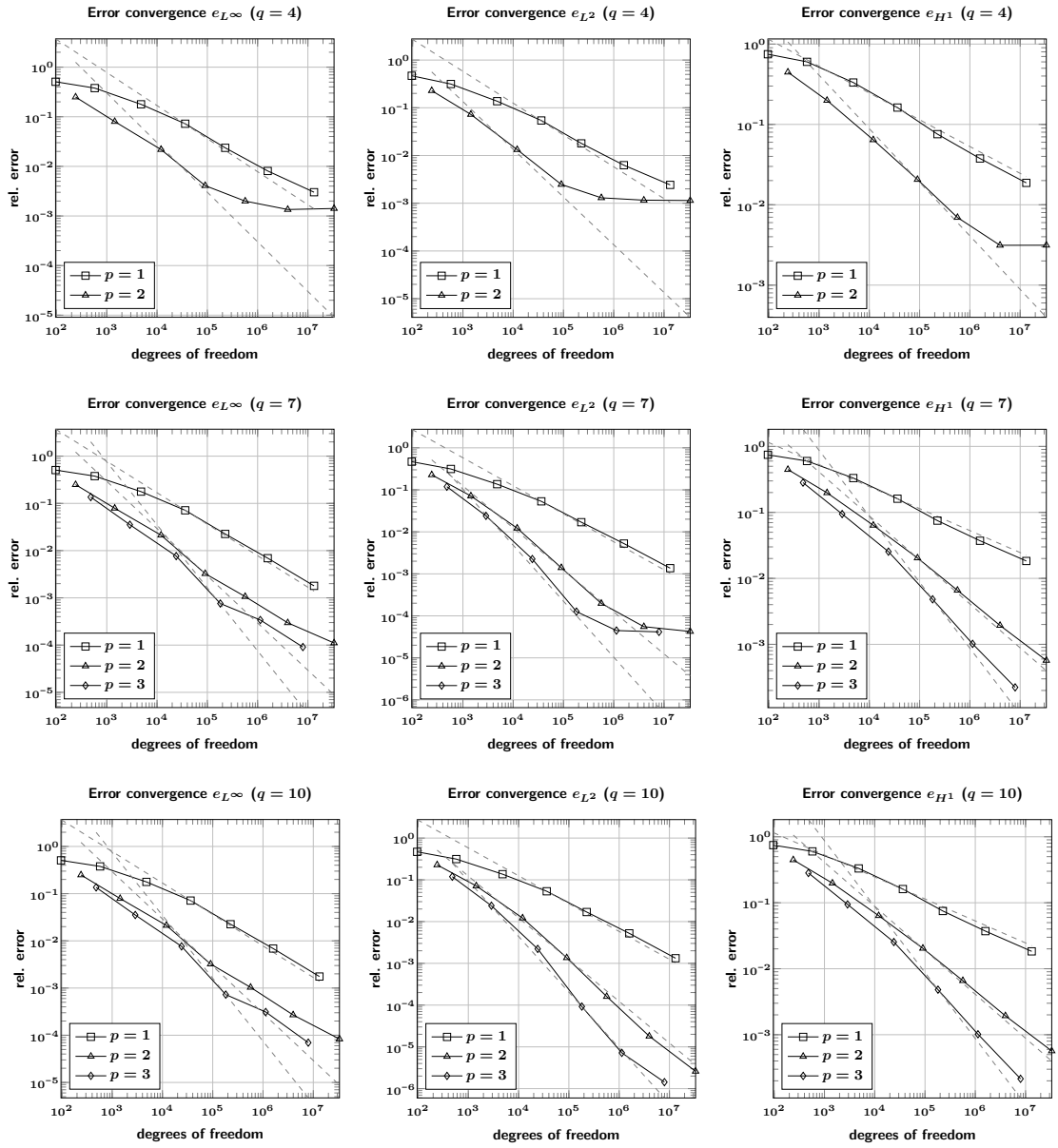

Figure 5.12: Error convergence for Example 5.1 (cut-cube) on different static geometries $\tilde{\Omega}_q$, for $q = 4, 7, 10$ (top to bottom). Each graph contains plots for different polynomial degrees p. Optimal convergence rates are depicted by dashed lines.

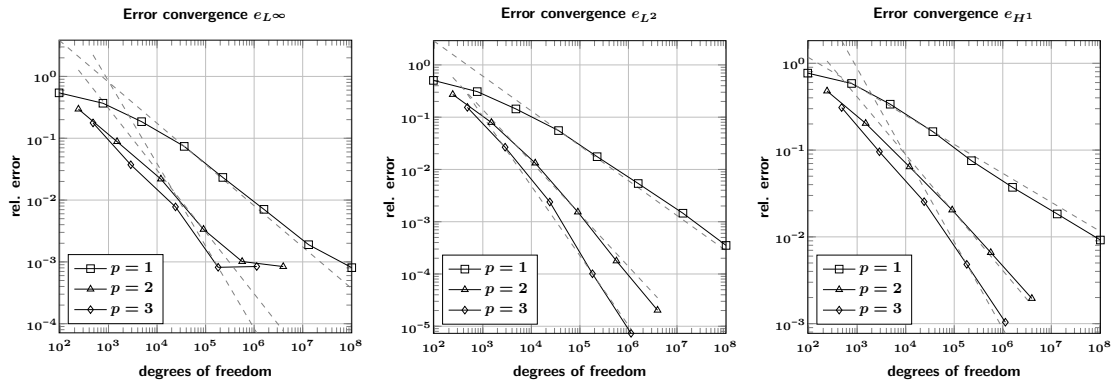

Figure 5.13: Error convergence for Example 5.1 (cut-cube) for polynomial degrees $p = 1, 2, 3$ on dynamic domain approximations $\tilde{\Omega}_k$ on each level k. Optimal convergence rates are depicted by dashed lines.

the L^∞ norm can be explained by the inherent difficulty to compute errors in that norm. In particular, we do only compute the errors on $\tilde{\Omega}_{\text{ref}}$ here and hence the computation does not include any integration points that are located on the boundary of $\tilde{\Omega}_q$ where the approximate solutions $\boldsymbol{u}_k^{\text{PU}}$ live on. Additionally, for the computations we always resolve all discontinuities (i.e. especially the patches) of the PUM space that the reference solution is computed on and the PUM space on the level k that the solution $\boldsymbol{u}_k^{\text{PU}}$ is computed on. Hence, we use a different set of quadrature points to compute the errors for the solutions on each level k which makes an accurate measurement of convergence even more difficult.[15] Finally we conduct the experiment where we use domain approximations $\tilde{\Omega}_k$ that are refined according to the target domain approximation error per level k according to (5.16). The results are depicted in Figure 5.13. Those results show that the simultaneous refinement of the PUM space and the domain approximation yields a method that converges with close to optimal rates in the L^2- and H^1-norms.[16]

Example 5.2 (Flange)**.** In this example we examine the practically more relevant geometry of a mechanical flange[17]. We again consider the equations of linear elasticity given by (4.32) and a generic steal material with a Young's modulus $E = 200$ GPa and a Poisson's ratio $\nu = 0.3$. We then conduct experiments with two different sets of boundary conditions (compare Figure 5.14). In both cases we fix the displacement on the two left bolt holes via a homogenous Dirichlet boundary condition $\boldsymbol{u} = 0$[18]. Then, in the first

[15]The result for $\|\boldsymbol{u}^*\|_{L^\infty}$ is only consistent up to three or four digits on average over the different sets of quadrature points used to compute the errors for $\boldsymbol{u}_k^{\text{PU}}$ on different levels k.

[16]The error computations in the L^∞-norm are subject to the same problems as described above.

[17]The flange model has been taken from `https://grabcad.com/library/flange-175` by author "Sanddip Padhariya" (`https://grabcad.com/sanddip.padhariya-1`). *Accessed: 2022-09-29.*

[18]The Dirichlet boundary conditions are not enforced by the conforming approach that was presented in Section 1.4. Instead we apply a weak enforcement by *Nitsche's method* as presented in [55]. This is beneficial in the current case where due to the piecewise linear approximation of the boundary, the conforming method would need to fix all basis functions to represent the boundary condition. In contrast

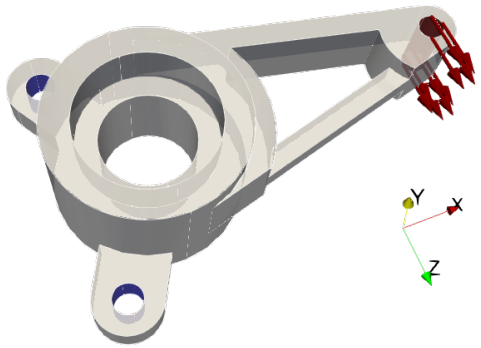

Figure 5.14: Geometry and boundary conditions for Example 5.2. The displacement is fixed at the two bolt holes on the left (blue). On the bolt hole on right (red) either a force in direction $\boldsymbol{d} = (0, 0, 1)$ is applied or a prescribed displacement of $\beta\boldsymbol{d}$ is given for all points at that bolt hole.

set of boundary conditions, we apply a force on the right bolt hole. That force is given by a Neumann boundary condition $\sigma(\boldsymbol{u})\boldsymbol{n} = \boldsymbol{d}\frac{L}{A}$, where $L = 1$ kN is the force acting in direction $\boldsymbol{d} = (0, 0, 1)^T$ on the surface of the right bolt hole that has a surface area of A. Alternatively, we apply a Dirichlet boundary condition on the right bolt hole with a prescribed displacement of $\boldsymbol{u} = \beta\boldsymbol{d}$ with $\beta = 0.0072$ mm.
Since the interior and exterior edges and corners of the geometry have not been equipped with fillets, we expect the solution to be subject to high stress concentrations or even singularities at those locations (compare Figure 5.15). In the case of such non-smooth solutions we cannot expect a uniform refinement scheme to yield optimal convergence rates. Hence we employ adaptively refined covers and PUM spaces in this example. To this end, the adaptive refinement is controlled by an *a posteriori sub-domain error estimator* that was proposed for our PUM in [56]. Here we compute the solution on a given cover and PUM space and then solve *local* problems on each patch, employing local function spaces V_i with an increased polynomial degree $p + q$ (throughout this thesis an increment of $q = 1$ is used). From those local solutions we can then retrieve local error estimates η_i in the energy norm on each patch that are used to control the h-refinement of the cover and PUM space. To this end, we first uniformly refine the cover to an initial level k_{init} were we solve the problem to obtain a solution $\boldsymbol{u}_m^{\text{PU}}$ for the first iteration $m = 0$. The error of that solution is then estimated per patch and the local estimate is used to control the refinement of the cover. Then we continue to compute a global solution for the next iteration $m + 1$ on the refined cover. Note that our hierarchical cover construction algorithm 1.1 does only create a new level when at least one leaf cell/patch is h-refined, thus we do usually not create a new level in every iteration m and hence it is $k_{\max}^m \leq m$,

Nitsche's method has more freedom to balance the correct representation of the boundary conditions and the solution of the PDE in the interior.

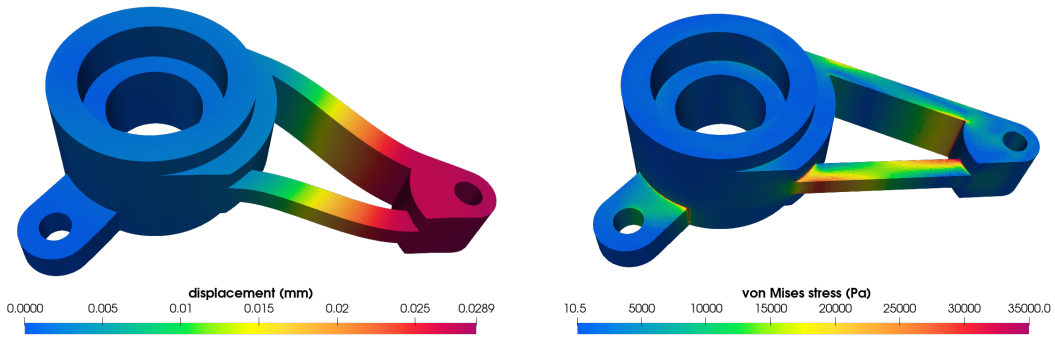

Figure 5.15: Magnitude of the displacement field (left) and von Mises stress (right) of the solution in iteration $m = 18$ employing polynomials of degree $p = 2$. The domain has been warped by the displacement field scaled by a factor of 1000.

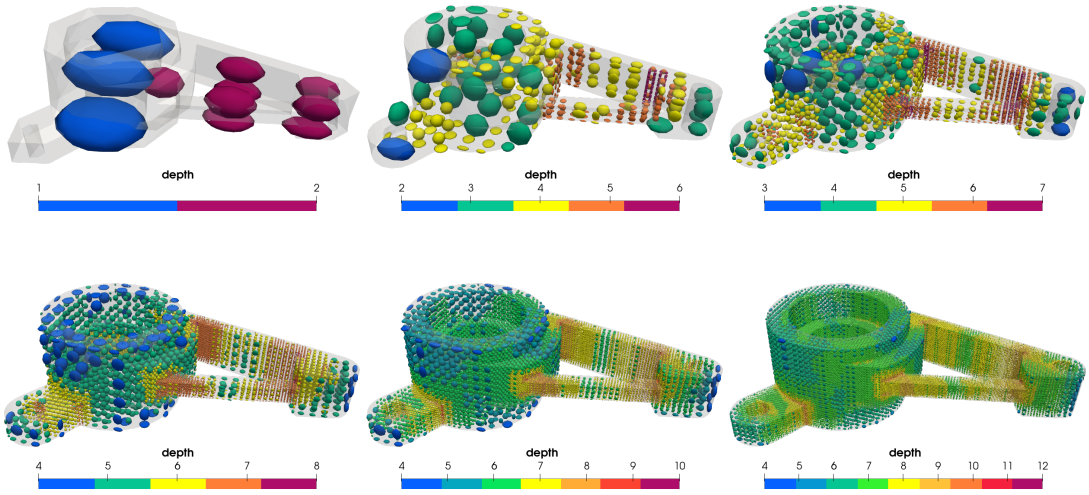

Figure 5.16: Patches represented by spheres around the centers of the respective domains ω_i. The spheres have been scaled anisotropically by half the radius of the patches' domains. The color indicates the depth of the associated cell in the cover tree. Depicted are the patches of the refinement iterations $m = 1$, 5, 9, 13, 17 and 21 (top left to bottom right). In the final iteration $m = 21$ the finest level of the cover is $k^m_{\max} = 12$.

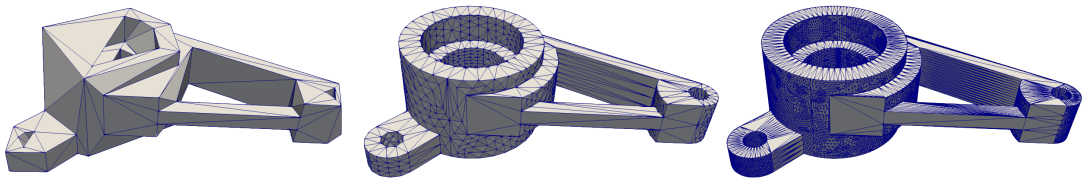

Figure 5.17: Approximations of the domain Ω in different refinement iterations m employing polynomials of degree $p = 1$. Initial approximation with estimated approximation error $\tau_0 \approx 1.44 \cdot 10^5$ (left), approximation in iteration $m = 5$ with $\tau_5 \approx 2.47 \cdot 10^3$ (center) and approximation in iteration $m = 13$ with $\tau_{13} \approx 2.15 \cdot 10^2$ (right).

where $k^m_{\max}$ is the finest cover level in iteration m. Since we expect the errors in the energy norm to be higher at locations of stress concentrations or singularities, the refinement will yield covers that are adaptively refined to those locations (compare Figure 5.16). We use a stretch factor $\alpha = 1.2$ when creating the patches ω_i from the local tree cells $\mathcal{C}_i$ and hence we can allow a maximum depth difference of $L_{\max} = 2$ according to (1.31) to get sufficiently adaptive covers.
It is important to note that the sub-domain error estimator does not alter the geometry when estimating the local errors. Hence it does only give an estimate for the *approximation error* and does not include any error components due to the domain approximation. Since the overall idea of the geometry refinement is to couple the convergence of the approximation error with the convergence of the consistency error we hence use the error estimates to determine the desired target domain approximation errors $\bar{\tau}$. Up to the initial level k_{init}, where we do not have any error estimates yet, we employ our usual equation based on the optimal convergence rates for a polynomial degree p, i.e.

$$\bar{\tau}_{\text{init}} = \tau_0 \left(\frac{1}{2^{p+1}} \right)^{k_{\text{init}}}$$

is the initial target domain approximation error that is applied before solving for the first solution $\boldsymbol{u}_0^{\text{PU}}$. Then, let $\eta_m^{L_2}$ be the global estimate of the approximation error in the L^2-norm according to the sub-domain error estimator for a solution $\boldsymbol{u}_m^{\text{PU}}$ in iteration m. We then use

$$\bar{\tau}_{m+1} = \tau_{\text{init}} \frac{\eta_m^{L_2}}{\max\limits_{n=0,\dots,m} \{\eta_n^{L_2}\}} \tag{5.19}$$

as the desired target domain approximation error for the next iteration. The fraction in that equation resembles the overall factor by which the approximation error has been improved up to iteration m. In the denominator we take the maximum error estimate of all previous iterations $n = 0, \dots, m$ instead of simply the estimated error $\eta_0^{L_2}$ in the first iteration, since we cannot always expect $\eta_0^{L_2}$ to be the largest error estimate. The reason for that is that on very coarse covers, the estimate itself might not be very precise

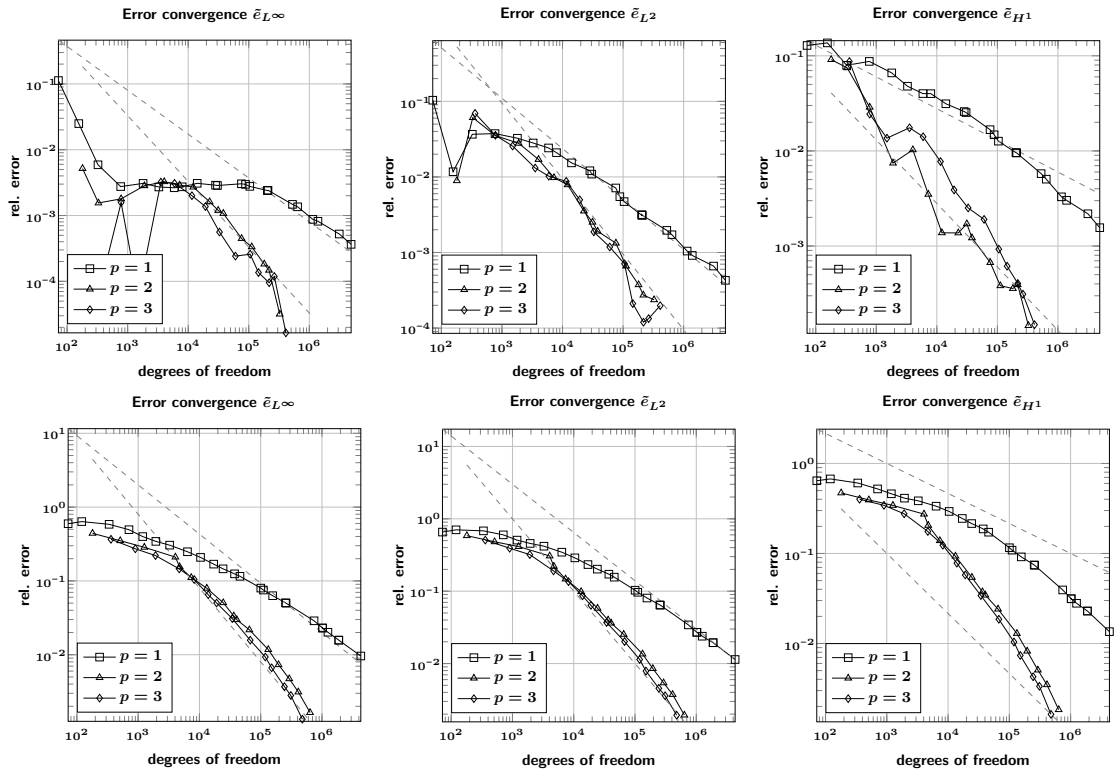

Figure 5.18: Error convergence according to (5.20) for Example 5.2 (flange). Optimal convergence rates are depicted by dashed lines (for polynomial degrees $p = 1$ and $p = 2$ only). The top row depicts results for the case where a fixed displacement is prescribed on the right bolt hole. The bottom row depicts results where a force is prescribed at that bolt hole.

and often underestimates the actual error. Hence, it is common for the error estimates to increase on the first few iterations before they start to decrease monotonically.
Let us now turn our focus on how we measure errors in this example. In contrast to the geometry used in Example 5.1, the flange geometry has both, surfaces that are concave and surfaces that are convex. Hence, we have to assume that

$$\tilde{\Omega}_{m+1} \not\subseteq \tilde{\Omega}_m, \quad \tilde{\Omega}_{m+1} \not\supseteq \tilde{\Omega}_m$$

for all domain approximation $\tilde{\Omega}_{m+1}$ that result from the refinement of a domain approximation $\tilde{\Omega}_m$. This does especially mean that is not easy to find a common sub-domain where we can evaluate the errors according to (2.4) on. To this end, we employ a simplified error measure

$$\tilde{e}_{L^2} := \frac{\left| \|u^*\|_{L^2(\tilde{\Omega}^*)} - \|u_m^{\mathrm{PU}}\|_{L^2(\tilde{\Omega}_m)} \right|}{\|u^*\|_{L^2(\tilde{\Omega}^*)}} \tag{5.20}$$

that does only evaluate norms of solutions on the respective domains they have been

computed on. Error measures utilizing the L^∞ and H^1 norms are created analogously. For the reference solution $\boldsymbol{u}^*$ we then simply use the finest solution in the final iteration of the refinement process. The convergence results for both sets of boundary conditions as mentioned in the beginning are depicted in Figure 5.18. Let us first discuss the results for the case where we apply a fixed displacement on the right bolt hole. The first thing to mention is that the error in the L^∞-norm first decreases very fast, plateaus for a short time and then starts do decrease with a close to constant rate. This initially fast convergence can be explained by the fact that the maximum displacement (on the final level we measure $\sim$7.226 μm) is very close to the prescribed displacement ($\sim$7.2 μm) and hence we get a very accurate result, right from the start. Looking at the errors in the L^2- and H^1-norm, we observe that we get good convergence for polynomial degrees $p = 1$ and $p = 2$, with rates close to what would be expected for errors measured by the original formulas given in (2.4). Especially in the H^1 norm we see convergence with even better rates. This is due to the fact that the employed error measures (5.20) are less strict than the original ones given in (2.4) and additionally the improving convergence on fine levels can be explained due to the numerical reference solution not being sufficiently accurate itself. Also note that the convergence employing polynomials of degree $p = 3$ does not yield better results than when employing polynomials with $p = 2$. This is not surprising, since the solution is known to be singular at the re-entrant edges and hence cannot be expected to be of high regularity. Overall, we can observe very similar convergence behavior for the case where force boundary conditions are applied on the right bolt hole (compare second row of Figure 5.18). The convergence in the L^∞-norm does not show the initially fast convergence and the effect of observed high convergence rate in the H^1-norm is even stronger here. Similarly, we cannot see any improved convergence behavior when employing polynomials of degree $p = 3$. We want to note that it might be possible to achieve better convergence for higher polynomial degrees when allowing more adaptivity by increasing the allowed maximum depth difference $L_{\max}$ or by applying a more sophisticated error estimator and refinement strategy that performs both, adaptive h- as well as adaptive p-refinement as has been presented in [106]. We would expect such an refinement strategy to only increase the polynomial degrees on patches further away from the singular locations where the solution exhibits smoother behavior locally.
Nevertheless, the reduced convergence for higher polynomial degrees allows us to demonstrate another beneficial behavior of the geometry refinement that is bound to the error estimator as given in (5.19), compared to the one only determined by the employed polynomial degree as given in (5.16). Since the error estimator observes similar error reductions for both polynomials of degree $p = 2$ as well as $p = 3$, this results in similar geometry refinement in both cases (compare Figure 5.19). Hence, in the case of $p = 3$ we do not unnecessarily overrefine the geometry and create a lot of integration cells when the approximation error would not convergence fast enough anyways.

Example 5.3 (Landing Gear's upper panel)**.** In this final example we validate the PUM

[19] Renderings based on CAD models taken from `https://grabcad.com/library/main-landing-gear-blg-tandem-bogie-airbus-a380-1` by author "Mohammed Ismail A" (`https://grabcad.com/mohammed.ismail.a-2`). *Accessed: 2022-09-29.*

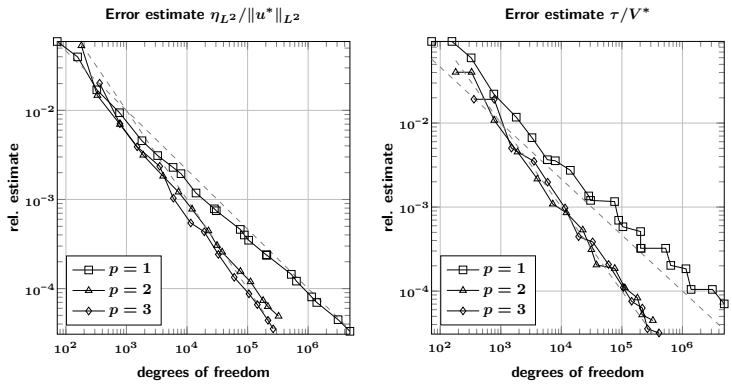

Figure 5.19: *Left:* Relative total estimated approximation error by the employed error estimator in the L^2-norm.
Right: The corresponding domain approximation errors according to (5.19) relative to the reference volume V^* of the complete geometry.

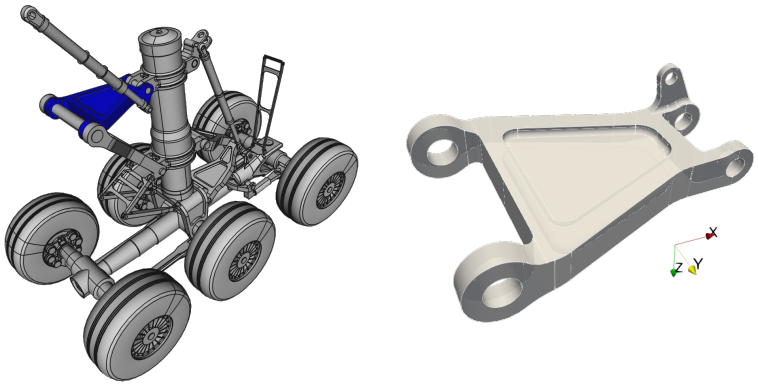

Figure 5.20: *Left:* Main landing gear of an Airbus A380 with the *upper panel* highlighted in blue. *Right:* Only the upper panel in its conventional design.[19]

results by comparing them to external solutions that have been obtain by a classical FEM. To this end, we consider a solid mechanical problem on the *upper panel*, that is a part of the main landing gear of an Airbus A380 (compare Figure 5.20). We thereby reproduce the results that have been presented in [85]. In that work, alternative geometries for the upper panel have been constructed by employing a *generative design* technique, with the goal to create parts that are lighter than the conventional design but do still withstand all applied loading conditions. The two main results from that generative design are denoted *iteration 1* and *iteration 2* which are depicted in Figure 5.21.[20] It is important to note that both geometries are nearly entirely described by NURBS surfaces. Again, we consider the

[20]The geometries of the upper panel models have been taken from `https://grabcad.com/library/landing-gear-s-torsion-link-1` by author "Ravi Maurya" (`https://grabcad.com/ravi.maurya-3`). *Accessed: 2022-09-29.*

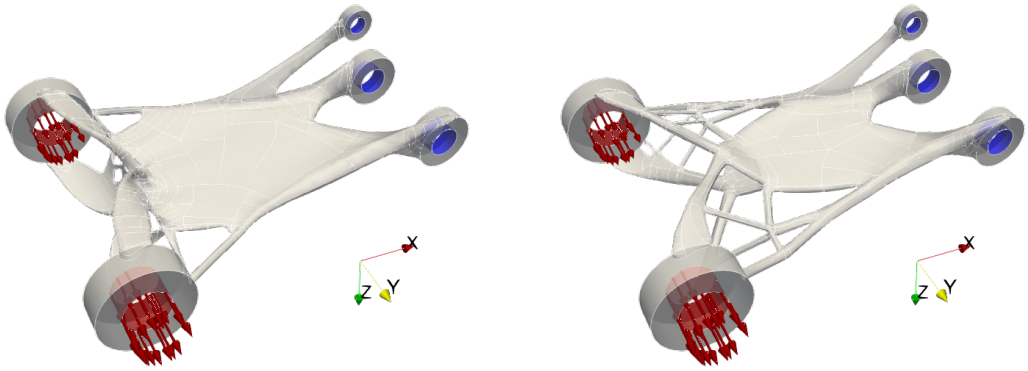

Figure 5.21: Geometries and boundary conditions for Example 5.3. The geometries are variants of the upper panel from the landing gear of an Airbus A380 created by generative design. The geometry on the left is denoted *iteration 1* and the one on the right is donated *iteration 2* of the design. The displacement is fixed at the three bolt holes on the right (blue) and a force in axial direction $\boldsymbol{d} = (0, 1, 0)^T$ is applied on the bolt holes on the left (red).

equations of linear elasticity (4.32). As boundary conditions we apply a fixed displacement $\boldsymbol{u} = 0$ on the three smaller bolt holes on the right. On the larger bolt holes on the left, we apply a Neumann boundary condition $\sigma(\boldsymbol{u})\boldsymbol{n} = \boldsymbol{d}\frac{L}{A}$, where a force of $L = 4.48$ kN is acting in axial direction $\boldsymbol{d} = (0, 1, 0)^T$ on the holes with a combined surface area of A. The material is the Titanium alloy *Ti-6Al-4V* with a Young's modulus $E = 113\,800$ MPa and a Poisson's ratio $\nu = 0.33$.
To control the geometry refinement, we employ yet another equation for the target domain approximation error, given by

$$\bar{\tau}_{k,p} = \tau_0 \left(\frac{1}{2^{p+1}} \right)^{\log_8(N_k)} \tag{5.21}$$

where N_k is the number of patches on the level k. In contrast to the original formulation in (5.16), this modified version does not assume the optimal reduction on all levels. Instead the reduction is based on the actual number of patches that end up on the specific level of the cover. The idea behind that modification is to compensate for cases where on the first few levels many of the patches end up being outside of the domain and hence the number of patches does not increase by a factor of eight per level. Note that in a case where no patches are outside of the domain (e.g. in the case of a cube domain), equations (5.16) and (5.21) are equal. Since in the considered geometries we cannot easily make any assumptions about concave or convex properties of the surfaces, we employ the simpler, parametrization based domain approximation error estimation from (5.11) that behaves better in cases where surfaces intersect the triangles that are used to approximate them. The initial domain approximation error for iteration 1 of the design is given by $\tau_0 \approx 1.226 \cdot 10^7$ and for iteration 2 it is $\tau_0 \approx 1.139 \cdot 10^7$.
Let us now compare our results to the solution from [85], which we will use as our reference

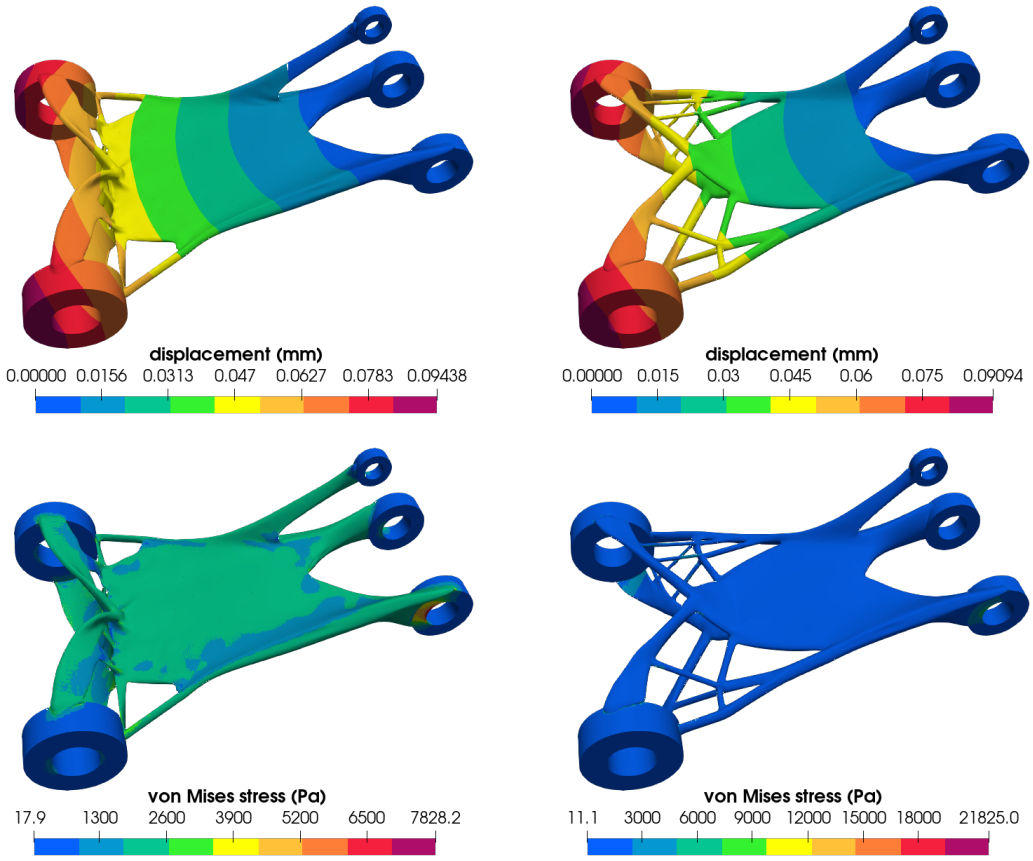

Figure 5.22: Magnitude of the displacement field (upper row) and von Mises stress (lower row) of the solution on level $k = 8$ using polynomials of degree $p = 1$.

solution. In that paper a mesh with a target element size of ∼5 mm was used. For iteration 1 of the design this resulted in 1 097 621 elements and 1 604 103 nodes (due to it being a vector valued problem, we assume each node to correspond to three degrees of freedom, hence ∼4 812 309 degrees of freedom in total). The maximum displacement is given as 0.094308 mm. In comparison we used a PUM space with linear polynomials on level $k = 8$ which resulted in 159 489 patches with a diameter of ∼6.3515 mm and a total of 1 913 868 degrees of freedom. The final domain approximation error for that cover is $\tau_{8,1} \approx 3.801 \cdot 10^3$. The maximum displacement over all integration points that we measured with that PUM discretization is 0.094380 mm, which is in very good agreement to the reference results. Employing a PUM space with quadratic polynomials allowed us to get very similar results with far less patches and degrees of freedom. Here we did refine the cover to level $k = 5$ which resulted in just 773 patches with a diameter of ∼50.812 mm and 23 190 degrees of freedom and a final domain approximation error $\tau_{5,2} \approx 1.439 \cdot 10^4$. The maximum displacement resulting from that discretization is 0.0945098 mm, which is still in very good agreement to both, the fine PUM solution as well as the reference

FEM solution. For iteration 2 of the design, the reference used a mesh that consists of 1 092 078 elements and 1 579 567 nodes (∼4 738 701 degrees of freedom). The maximum displacement is given as 0.089805 mm. With a PUM employing linear polynomials on level $k = 8$ we ended up with 158 830 patches and 1 905 960 degrees of freedom and a domain approximation error $\tau_{8,1} \approx 3.465 \cdot 10^3$. Since the bounding boxes of the two geometries are identical, the patch diameter is the same as for the first iteration. Here the maximum displacement is given by 0.090937 mm, which is again in good agreement to the reference results. Similarly to the case in iteration 1, we can get comparable results when employing quadratic polynomials on a coarser level with $k = 5$. Here we end up with 778 patches and 23 340 degrees of freedom, a domain approximation error $\tau_{5,2} \approx 1.271 \cdot 10^4$ and measure a maximum displacement of 0.0899502 mm. Results for both iterations are depicted in Figure 5.22. The color ranges have been adjusted to resemble the figures in the reference paper.

Concluding Remarks

In this thesis we demonstrated that the flat-top partition of unity method is indeed a capable meshfree method that can be used to solve partial differential equations on real-world, industrial-grade CAD geometries. The first stage to attain that goal was achieved by introducing a new post-processing step to the cover construction of the PUM. By eliminating all patches that violate the strict flat-top property, we guarantee that we retrieve a globally stable basis, whenever we impose locally stable bases on each patch. A stable polynomial basis for boundary patches, that keeps its full approximation power, was obtained by a simple shrinking step. Singularities, that arise in the derivatives of PU functions due to the elimination of patches, were moved away from the boundary such that they do not impose any practical problems to the subsequent integration steps. The robustness of the post-processing step was demonstrated in two and three space-dimensions by numerical experiments. Furthermore, for the construction of a multilevel solver, we showed that a simple combination of a global-to-local and local-to-local transfer can be built to overcome issues introduced by the patch elimination.
To implement a reliable numerical quadrature, we developed two different approaches in two and three space-dimensions. In two-dimensions we presented a method that keeps an exact representation of the geometry on all levels of our PUM construction. First, we solved the problem of reliable intersection operations by employing a monotone decomposition of all curves that form the boundary of the simulation domain. The properties resulting from the monotone decomposition were used, such that all intersection points of the axis-aligned integration domains and the boundary of the simulation domain can be computed robust and efficiently. The employment of the Grainer-Hormann clipping algorithm then allowed to compute the intersection domains of the axis-aligned boxes with the simulation domain reliably, even when the input geometry contains self-intersections in its boundary description. The resulting curved polygon domains where then first decomposed into linear triangles. The usage of an iterative Delaunay triangulation algorithm was chosen such that potential self-intersections of the input curves can be resolved in this step. It was demonstrated, that the insertion of the original boundaries' inflection points into the local triangulations allows to derive an important property of all curve parts included in the local triangulations: The curve parts are contained within a triangle determined by their endpoints and the tangents at those endpoints. With this property we developed an efficient and robust test to detect invalid curved triangles and showed how this test can be used to reliably resolve those invalid cases. Finally, we presented how to use a transfinite-interpolation for the numerical quadrature on each curved triangle.

In three space-dimensions we presented an approach where the boundary of the input geometry is first approximated by a triangular surface mesh. The linear description of the approximation allowed us to implement the subsequent intersection and decomposition operations robust and efficiently. To overcome errors introduced by the domain approximation, we presented a way to estimate the domain approximation error. Then, we established a relation between the domain approximation error and the error induced by the PUM discretization. It has was shown, that an approach where the domain approximation error is reduced by a refinement strategy, such that it matches the PUM approximation error, is capable of obtaining close to optimal convergence rates, for polynomials up to a degree of $p = 3$.

The work presented here lays the groundwork for a robust geometry treatment in the PUM and makes it finally accessible to industrial use cases. Furthermore, we think that the presented methods for the two-dimensional integration can be applied to any immersed or embedded boundary method that uses axis-aligned rectangle cells for its background mesh. A runtime and robustness comparison with currently established methods would be an interesting topic for future work.

The parallelization of the presented methods was not discussed in this thesis. Nevertheless, we think the presented methods are good candidates to be applied in a parallel PUM subject to a domain decomposition approach as presented in [54]. In fact, the patch-wise distribution of work to parallel processes does immediately lead to a parallelization of the complete integration step. Hence, in most of the experiments presented throughout this thesis, the assemble step was already executed on parallel computers with multiple processes. The load estimate for the space-filling curve based load distribution was thereby adjusted to incorporate the increased amount of work for boundary patches, but a more thorough analysis of the balancing problem is still an open question. One aspect that is not trivial to parallelize is the patch elimination step during the cover post-processing. Here, the presented algorithm is inherently sequential, since the elimination of one patch can influence whether a subsequently visited patch needs to be eliminated or not. Nevertheless, we think a parallelization can be achieved. Here, all processes eliminate local patches that are not located at process boundaries first. Then, they mark patches at process boundaries that are candidates for elimination and communicate those patches with their neighboring processes. The process with the lowest rank then decides which patches to eliminate and communicates that information back to the neighboring processes. The communication step might need to be iterated until all in-valid patches have been eliminated. The resulting cover will probably end up being different, than what would have been achieved by a sequential algorithm, since the order in which patches are eliminated, will depend on the number of processes. Details of such a parallel implementation are left for future work.

The integration process for two-dimensional domains could be subject to some improvements as well. E.g. an alternative code path could be introduced to handle cases where the intersected integration domains $\mathcal{D}_i^m \cap \Omega$ are bounded by only four curves. In such a case the triangulation step could be skipped and the integration could be carried out on a rectangular reference domain. The required transformation $\boldsymbol{\Psi}$ could thereby be constructed similar to Coons patches [23]. It is an open question whether the bi-monotone and constant sign curvature properties of the involved curve parts have any implications

on whether the Coons map will be a valid diffeomorphism or not. Furthermore, due to the skipped triangulation step, a new method to detect self-intersecting input curves would be required. A possible answer to those questions might be found in [96] where sufficient and necessary conditions for the regularity of a Coons map are given.
The generation of integration cells for three-dimensional domains could also be further improved. First of all, due to the requirement of a watertight surface mesh for the domain approximation, our current approach can only tolerate flaws in the geometries to a limited extend. The main reason why this requirement was imposed, is that we need a robust point containment test. Thus, we might be able to remove that restriction by applying a containment test that can deal with gaps and self-intersections. A possible candidate for such an algorithm has been proposed in [63]. Lifting the restriction would then allow us to generate the approximations of all faces from the CAD solid independent of each other, since they would not need to match along edges anymore. Resulting gaps would not impose a problem anymore, and self-intersections of the approximated surfaces can be detected and resolved during the Delaunay tetrahedralization step. Another benefit of an independent approximation of all surfaces is that we could choose different approximation strategies for each of those surfaces. This is particularly interesting for faces described by elementary surface types like e.g. cylinders, cones and spheres. For these types of surface, closed-form solutions for the intersection curves with a plane exist. Hence, we would not need to approximate those surfaces at all. The intersections $\mathcal{D}_i^m \cap \Omega$ and the integration cell creation could then be considering the exact geometry for those surfaces. Only after the intersections have been computed, we would need to create a local triangulation of the curved surface before we can tetrahedralize the volume. This would result in a method that is similar to what has been demonstrated in Example 4.2 for the linear elastic hole case in two-dimensions, where the curving after the triangulation was disabled. But a final improvement could be to incorporate the original surface representation when creating the quadrature points. Similar to the two-dimensional, case this would involve curving the resulting tetrahedral cells at triangles that are related to the curved boundary. Transfinite interpolations for tetrahedra can be found in [20, 93]. It is an open question whether the approach to detect invalid triangles presented in this thesis can be extended to 3D.

Bibliography

[1] *Open CASCADE Technology*. `https://dev.opencascade.org`. Accessed: 2022-07-11.

[2] R. Abgrall, C. Dobrzynski, and A. Froehly, *A method for computing curved 2D and 3D meshes via the linear elasticity analogy: preliminary results*, Research Report RR-8061, INRIA, Sept. 2012.

[3] C. Albrecht, C. Klaar, J. E. Pask, M. A. Schweitzer, N. Sukumar, and A. Ziegenhagel, *Orbital-enriched flat-top partition of unity method for the schrödinger eigenproblem*, Computer Methods in Applied Mechanics and Engineering, 342 (2018), pp. 224 – 239.

[4] G. Allen, *Geometric modeling problems in industrial CAD/CAM/CAE*, in Proceedings of the 10th SIAM Conference on Geometric Design and Computing, San Antonio, TX, USA, 2007, pp. 109–126.

[5] R. Aubry, S. Dey, E. Mestreau, B. Karamete, and D. Gayman, *A robust conforming nurbs tessellation for industrial applications based on a mesh generation approach*, Computer-Aided Design, 63 (2015), pp. 26–38.

[6] I. Babuška and B. Q. Guo, *The h-p version of the finite element method for domains with curved boundaries*, SIAM Journal on Numerical Analysis, 25 (1988), pp. 837–861.

[7] I. Babuška, G. Caloz, and J. E. Osborn, *Special Finite Element Methods for a Class of Second Order Elliptic Problems with Rough Coefficients*, SIAM J. Numer. Anal., 31 (1994), pp. 945–981.

[8] I. Babuška and J. M. Melenk, *The Partition of Unity Finite Element Method: Basic Theory and Applications*, Comput. Meth. Appl. Mech. Engrg., 139 (1996), pp. 289–314. Special Issue on Meshless Methods.

[9] ——, *The Partition of Unity Method*, Int. J. Numer. Meth. Engrg., 40 (1997), pp. 727–758.

[10] J. W. Barrett and C. M. Elliott, *A Finite-element Method for Solving Elliptic Equations with Neumann Data on a Curved Boundary Using Unfitted Meshes*, IMA Journal of Numerical Analysis, 4 (1984), pp. 309–325.

[11] T. Belytschko, Y. Y. Lu, and L. Gu, *Element-free galerkin methods*, International Journal for Numerical Methods in Engineering, 37 (1994), pp. 229–256.

[12] T. Belytschko, N. Moës, S. Usui, and C. Parimi, *Arbitrary discontinuities in finite elements*, International Journal for Numerical Methods in Engineering, 50 (2001), pp. 993–1013.

[13] M. Birner, *Global-local Enrichments in the Partition of Unity Method*, Master's thesis, Institute for Numerical Simulation, University of Bonn, June 2018.

[14] J.-D. Boissonnat, O. Devillers, M. Teillaud, and M. Yvinec, *Triangulations in cgal (extended abstract)*, in Proceedings of the Sixteenth Annual Symposium on Computational Geometry, SCG '00, New York, NY, USA, 2000, Association for Computing Machinery, pp. 11–18.

[15] A. Bowyer, *Computing Dirichlet tessellations*, The Computer Journal, 24 (1981), pp. 162–166.

[16] E. Burman, S. Claus, P. Hansbo, M. G. Larson, and A. Massing, *Cutfem: Discretizing geometry and partial differential equations*, International Journal for Numerical Methods in Engineering, 104 (2015), pp. 472–501.

[17] X.-D. Chen, G. Xu, J.-H. Yong, G. Wang, and J.-C. Paul, *Computing the minimum distance between a point and a clamped b-spline surface*, Graphical Models, 71 (2009), pp. 107–112.

[18] X.-D. Chen, J.-H. Yong, G. Wang, J.-C. Paul, and G. Xu, *Computing the minimum distance between a point and a nurbs curve*, Computer-Aided Design, 40 (2008), pp. 1051–1054.

[19] K.-P. Cheng, *Using plane vector fields to obtain all the intersection curves of two general surfaces*, in Theory and Practice of Geometric Modeling, W. Straßer and H.-P. Seidel, eds., Springer, 1989, pp. 187–204.

[20] P. Chenin, *Quelques problèmes d'interpolation à plusieurs variables*, theses, Institut National Polytechnique de Grenoble - INPG ; Université Joseph-Fourier - Grenoble I, June 1974. Université : Université scientifique et médicale de Grenoble et Institut National Polytechnique.

[21] L. P. Chew, *Guaranteed-quality mesh generation for curved surfaces*, in Proceedings of the Ninth Annual Symposium on Computational Geometry, SCG '93, New York, NY, USA, 1993, Association for Computing Machinery, pp. 274–280.

[22] G. E. Collins and R. Loos, *Polynomial real root isolation by differentiation*, in Proceedings of the Third ACM Symposium on Symbolic and Algebraic Computation, SYMSAC '76, New York, NY, USA, 1976, Association for Computing Machinery, pp. 15–25.

[23] S. A. Coons, *Surfaces for computer-aided design of space forms*, tech. rep., USA, 1967.

[24] J. A. Cottrell, T. J. R. Hughes, and Y. Bazilevs, *Isogeometric Analysis: Toward Integration of CAD and FEA*, Wiley Publishing, first ed., 2009.

[25] B. Curless and M. Levoy, *A volumetric method for building complex models from range images*, in Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '96, New York, NY, USA, 1996, Association for Computing Machinery, pp. 303–312.

[26] L. Dasch, *Efficient Numerical Integration in Partition of Unity Methods on CAD Geometries*, Master's thesis, Institute for Numerical Simulation, University of Bonn, May 2016.

[27] L. De Floriani and E. Puppo, *An on-line algorithm for constrained delaunay triangulation*, CVGIP: Graphical Model and Image Processing, 54 (1992), pp. 290–300.

[28] O. Devillers, S. Pion, and M. Teillaud, *Walking in a triangulation*, in Proceedings of the Seventeenth Annual Symposium on Computational Geometry, SCG '01, New York, NY, USA, 2001, Association for Computing Machinery, pp. 106–114.

[29] S. Dey, R. M. O'Bara, and M. S. Shephard, *Curvilinear mesh generation in 3d*, in Proceedings of the eighth International Meshing Roundtable, John Wiley & Sons, 1999, pp. 407–417.

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

[31] A. Düster, J. Parvizian, Z. Yang, and E. Rank, *The finite cell method for three-dimensional problems of solid mechanics*, Computer Methods in Applied Mechanics and Engineering, 197 (2008), pp. 3768–3782.

[32] S. Duczek and U. Gabbert, *Efficient integration method for fictitious domain approaches*, Computational Mechanics, 56 (2015), pp. 725–738.

[33] E. Dyllong and W. Luther, *Distance calculation between a point and a nurbs surface*, in Proceedings of the 4th International Conference on Curves and Surfaces, P.-J. Laurent, P. Sablonnire, and L. L. Schumaker, eds., vol. 1 of Innovations in Applied Mathematics, Saint-Malo, France, 2000, pp. 55–62.

[34] M. Eigel, E. George, and M. Kirkilionis, *The partition of unity meshfree method for solving transport-reaction equations on complex domains: Implementation and applications in the life sciences*, in Meshfree Methods for Partial Differential Equations IV, M. Griebel and M. A. Schweitzer, eds., Springer, 2008, pp. 69–93.

[35] M. Eigel, E. George, and M. Kirkilionis, *A mesh-free partition of unity method for diffusion equations on complex domains*, IMA Journal of Numerical Analysis, 30 (2009), pp. 629–653.

[36] R. Farouki and V. Rajan, *On the numerical condition of polynomials in bernstein form*, Computer Aided Geometric Design, 4 (1987), pp. 191–216.

[37] ——, *Algorithms for polynomials in bernstein form*, Computer Aided Geometric Design, 5 (1988), pp. 1–26.

[38] R. T. Farouki and T. Sakkalis, *Real rational curves are not 'unit speed'*, Computer Aided Geometric Design, 8 (1991), pp. 151–157.

[39] F. Feito, C. Ogayar, R. Segura, and M. Rivero, *Fast and accurate evaluation of regularized boolean operations on triangulated solids*, Computer-Aided Design, 45 (2013), pp. 705–716.

[40] D. R. Forsey and R. H. Bartels, *Hierarchical b-spline refinement*, in Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '88, New York, NY, USA, 1988, Association for Computing Machinery, pp. 205–212.

[41] M. Fortunato and P.-O. Persson, *High-order unstructured curved mesh generation using the winslow equations*, Journal of Computational Physics, 307 (2016), pp. 1–14.

[42] E. L. Foster, K. Hormann, and R. T. Popa, *Clipping simple polygons with degenerate intersections*, Computers & Graphics: X, 2 (2019), p. 100007.

[43] Y. J. García R, M. A. López, and S. T. Leutenegger, *A greedy algorithm for bulk loading r-trees*, in Proceedings of the 6th ACM International Symposium on Advances in Geographic Information Systems, GIS '98, New York, NY, USA, 1998, Association for Computing Machinery, pp. 163–164.

[44] A. Ghasemi, L. K. Taylor, and I. Newman, James C., *Massively parallel curved spectral/finite element mesh generation of industrial cad geometries in two and three dimensions*, in Proceedings of the ASME 2016 Fluids Engineering Division Summer Meeting, vol. 1B, Washington, DC, USA, July 2016, ASME. V01BT26A002.

[45] R. A. Gingold and J. J. Monaghan, *Smoothed particle hydrodynamics: theory and application to non-spherical stars*, Monthly Notices of the Royal Astronomical Society, 181 (1977), pp. 375–389.

[46] W. J. Gordon and C. A. Hall, *Construction of curvilinear co-ordinate systems and applications to mesh generation*, International Journal for Numerical Methods in Engineering, 7 (1973), pp. 461–477.

[47] T. A. Grandine and F. W. Klein, *A new approach to the surface intersection problem*, Computer Aided Geometric Design, 14 (1997), pp. 111–134.

[48] A. Gray, *Modern Differential Geometry of Curves and Surfaces with Mathematica*, Taylor & Francis Inc, New York, NY, USA, 3rd ed., 2006.

[49] G. Greiner and K. Hormann, *Efficient clipping of arbitrary polygons.*, ACM Trans. Graph., 17 (1998), pp. 71–83.

[50] M. Griebel, P. Oswald, and M. A. Schweitzer, *A Particle-Partition of Unity Method—Part VI: A p-robust Multilevel Solver*, in Meshfree Methods for Partial Differential Equations II, M. Griebel and M. A. Schweitzer, eds., vol. 43 of Lecture Notes in Computational Science and Engineering, Springer, 2005, pp. 71–92.

[51] M. Griebel and M. A. Schweitzer, *A Particle-Partition of Unity Method for the solution of Elliptic, Parabolic and Hyperbolic PDE*, SIAM J. Sci. Comput., 22 (2000), pp. 853–890.

[52] ——, *A Particle-Partition of Unity Method—Part II: Efficient Cover Construction and Reliable Integration*, SIAM J. Sci. Comp., 23 (2002), pp. 1655–1682.

[53] ——, *A Particle-Partition of Unity Method—Part III: A Multilevel Solver*, SIAM J. Sci. Comp., 24 (2002), pp. 377–409.

[54] ——, *A Particle-Partition of Unity Method—Part IV: Parallelization*, in Meshfree Methods for Partial Differential Equations, M. Griebel and M. A. Schweitzer, eds., vol. 26 of Lecture Notes in Computational Science and Engineering, Springer, 2002, pp. 161–192.

[55] ——, *A Particle-Partition of Unity Method—Part V: Boundary Conditions*, in Geometric Analysis and Nonlinear Partial Differential Equations, S. Hildebrandt and H. Karcher, eds., Springer, 2002, pp. 517–540.

[56] ——, *A Particle-Partition of Unity Method—Part VII: Adaptivity*, in Meshfree Methods for Partial Differential Equations III, M. Griebel and M. A. Schweitzer, eds., vol. 57 of Lecture Notes in Computational Science and Engineering, Springer, 2006.

[57] A. Guttman, *R-trees: A dynamic index structure for spatial searching*, in Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data, SIGMOD '84, New York, NY, USA, 1984, Association for Computing Machinery, pp. 47–57.

[58] Y. Hu, T. Schneider, X. Gao, Q. Zhou, A. Jacobson, D. Zorin, and D. Panozzo, *Triwild: Robust triangulation with curve constraints*, ACM Trans. Graph., 38 (2019), pp. 52:1–52:15.

[59] P. M. Hubbard, *Constructive solid geometry for triangulated polyhedra*, tech. rep., USA, 1990.

[60] T. Hughes, J. Cottrell, and Y. Bazilevs, *Isogeometric analysis: Cad, finite elements, nurbs, exact geometry and mesh refinement*, Computer Methods in Applied Mechanics and Engineering, 194 (2005), pp. 4135–4195.

[61] S. Hur, M. jae Oh, and T. wan Kim, *Approximation of surface-to-surface intersection curves within a prescribed error bound satisfying g2 continuity*, Computer-Aided Design, 41 (2009), pp. 37–46.

[62] *Industrial automation systems and integration — Product data representation and exchange — Part 42: Integrated generic resource: Geometric and topological representation*, standard, International Organization for Standardization, Geneva, CH, Nov. 2019.

[63] A. Jacobson, L. Kavan, and O. Sorkine-Hornung, *Robust inside-outside segmentation using generalized winding numbers*, ACM Trans. Graph., 32 (2013).

[64] J. Jaśkowiec and N. Sukumar, *High-order cubature rules for tetrahedra*, International Journal for Numerical Methods in Engineering, 121 (2020), pp. 2418–2436.

[65] X. Jiang, Q. Peng, X. Cheng, N. Dai, C. Cheng, and D. Li, *Efficient booleans algorithms for triangulated meshes of geometric modeling*, Computer-Aided Design and Applications, 13 (2016), pp. 1–12.

[66] P. Jiménez Recio, *Numerical solution of variational inequalities with the Partition of Unity Method*, Master's thesis, Institute for Numerical Simulation, University of Bonn, July 2019.

[67] ——. Personal communication, 2022.

[68] D.-H. Kim and M.-J. Kim, *An extension of polygon clipping to resolve degenerate cases*, Computer-Aided Design & Applications, 3 (2006), pp. 447–456.

[69] D.-S. Kim, T. Jang, H. Shin, and J. Park, *Rational bézier form of hodographs of rational bézier curves and surfaces*, Computer-Aided Design, 33 (2001), pp. 321–330.

[70] C. Klaar, *A Partition of Unity Method for Quantum Mechanical Material Calculations*, Master's thesis, Institute for Numerical Simulation, University of Bonn, 2016.

[71] R. Klein and W. Strasser, *Mesh Generation from Boundary Models with Parametric Face Representation*, in Third Symposium on Solid Modeling and Applications, ACM Press, 1995, pp. 431–440.

[72] P. M. Knupp, *Winslow smoothing on two-dimensional unstructured meshes*, in Proceedings of the Seventh International Meshing Roundtable, Park City, UT, 1998, pp. 449–457.

[73] O. Kovaljov, N. Larkin, W. Fomin, and N. Yanenko, *The solution of non-homogeneous thermal problems and the stefan single-phase problem in arbitrary domains*, Computer Methods in Applied Mechanics and Engineering, 22 (1980), pp. 259–271.

[74] G. Kriezis, N. Patrikalakis, and F.-E. Wolter, *Topological and differential-equation methods for surface intersections*, Computer-Aided Design, 24 (1992), pp. 41–55.

[75] S. Krishnan, A. Narkhede, and D. Manocha, *Boole: A system to compute boolean combinations of sculptured solids*, tech. rep., 1995.

[76] L. Kudela, N. Zander, T. Bog, S. Kollmannsberger, and E. Rank, *Efficient and accurate numerical quadrature for immersed boundary methods*, Advanced Modeling and Simulation in Engineering Sciences, 2 (2015), p. 10.

[77] D. H. Laidlaw, W. B. Trumbore, and J. F. Hughes, *Constructive solid geometry for polyhedral objects*, in Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '86, New York, NY, USA, 1986, Association for Computing Machinery, pp. 161–170.

[78] P. Laug, *Some aspects of parametric surface meshing*, Finite Elements in Analysis and Design, 46 (2010), pp. 216–226. Mesh Generation - Applications and Adaptation.

[79] C. Lawson, *Software for C1 surface interpolation*, in Mathematical Software II, Academic Press, New York, 1977, pp. 161–194.

[80] S. Leutenegger, M. Lopez, and J. Edgington, *Str: a simple and efficient algorithm for r-tree packing*, in Proceedings 13th International Conference on Data Engineering, 1997, pp. 497–506.

[81] H. Liao, P. K. Vaitheeswaran, T. Song, and G. Subbarayan, *Algebraic point projection for immersed boundary analysis on low degree nurbs curves and surfaces*, Algorithms, 13 (2020).

[82] W. K. Liu, Y. Chen, S. Jun, J. S. Chen, T. Belytschko, C. Pan, R. A. Uras, and C. T. Chang, *Overview and applications of the reproducing kernel particle methods*, Archives of Computational Methods in Engineering, 3 (1996), pp. 3–80.

[83] W. K. Liu, S. Jun, and Y. F. Zhang, *Reproducing kernel particle methods*, International Journal for Numerical Methods in Fluids, 20 (1995), pp. 1081–1106.

[84] L. B. Lucy, *A numerical approach to the testing of the fission hypothesis.*, The Astronomical Journal, 82 (1977), pp. 1013–1024.

[85] R. M, M. R. Dayaram, Raghu.B.S, R. Raj, and S. Mathew, *Design optimization of aircraft landing gear's torsion link using generative design*, International Journal for Science and Advance Research In Technology, 6 (2020), pp. 248–255.

[86] J. Michels, *A tangential differential calculus based PUM for Kirchhoff shells*, Master's thesis, Institute for Numerical Simulation, University of Bonn, Mar. 2021.

[87] B. Nayroles, G. Touzot, and P. Villon, *Generalizing the finite element method: Diffuse approximation and diffuse elements*, Computational Mechanics, 10 (1992), pp. 307–318.

[88] A. Neumaier, *Rundungsfehleranalyse einiger verfahren zur summation endlicher summen*, ZAMM - Journal of Applied Mathematics and Mechanics / Zeitschrift für Angewandte Mathematik und Mechanik, 54 (1974), pp. 39–51.

[89] J. Nitsche, *Über ein Variationsprinzip zur Lösung von Dirichlet-Problemen bei Verwendung von Teilräumen, die keinen Randbedingungen unterworfen sind*, Abh. Math. Sem. Univ. Hamburg, 36 (1971), pp. 9–15.

[90] J. Parvizian, A. Düster, and E. Rank, *Finite cell method: h- and p-extension for embedded domain problems in Solid Mechanics*, Computational Mechanics, 41 (2007), pp. 121–133.

[91] N. M. Patrikalakis, *Surface-to-surface intersections*, IEEE Computer Graphics and Applications, 13 (1993), pp. 89–95.

[92] N. M. Patrikalakis and T. Maekawa, *Handbook of Computer Aided Geometric Design*, North Holland, 2002, ch. 25: Intersection Problems.

[93] A. Perronnet, *Interpolation transfinie sur le triangle, le tétraèdre et le pentaèdre. application à la création de maillages et à la condition de dirichlet*, Comptes Rendus de l'Académie des Sciences - Series I - Mathematics, 326 (1998), pp. 117–122.

[94] L. Piegl and W. Tiller, *The NURBS Book*, Springer-Verlag, New York, NY, USA, second ed., 1997.

[95] Z. Qin, M. D. McCool, and C. Kaplan, *Precise vector textures for real-time 3d rendering*, in Proceedings of the 2008 Symposium on Interactive 3D Graphics and Games, I3D '08, New York, NY, USA, 2008, Association for Computing Machinery, pp. 199–206.

[96] M. Randrianarivony and G. Brunnett, *Necessary and sufficient conditions for the regularity of a planar Coons map*. Preprint SFB393/04-07, May 2004.

[97] G. Renner and V. Weiss, *Exact and approximate computation of b-spline curves on surfaces*, Computer-Aided Design, 36 (2004), pp. 351–362.

[98] A. Requicha and H. Voelcker, *Boolean operations in solid modeling: Boundary evaluation and merging algorithms*, Proceedings of the IEEE, 73 (1985), pp. 30–44.

[99] A. Rockwood, K. Heaton, and T. Davis, *Real-time rendering of trimmed surfaces*, in Proceedings of the 16th Annual Conference on Computer Graphics and

Interactive Techniques, SIGGRAPH '89, New York, NY, USA, 1989, Association for Computing Machinery, pp. 107–116.

[100] J. Ruppert, *A delaunay refinement algorithm for quality 2-dimensional mesh generation*, Journal of Algorithms, 18 (1995), pp. 548–585.

[101] M. Sarfraz, M. Irshad, F. Sarfraz, and M. Z. Hussain, *A novel approach for surface to surface intersection approximation*, in 2013 17th International Conference on Information Visualisation, 2013, pp. 482–487.

[102] V. K. Saul'ev, *Solution of certain boundary-value problems on high-speed computers by the fictitious-domain method*, Sibirsk. Mat. Zh., 4 (1963), pp. 912–925. In Russian.

[103] D. Schöllhammer and T. P. Fries, *Kirchhoff–love shell theory based on tangential differential calculus*, Computational Mechanics, 64 (2019), pp. 113–131.

[104] A. Schollmeyer and B. Fröhlich, *Direct trimming of nurbs surfaces on the gpu*, ACM Trans. Graph., 28 (2009).

[105] M. A. Schweitzer, *A Particle-Partition of Unity Method—Part VIII: Hierarchical Enrichment*, in Meshfree Methods for Partial Differential Equations IV, M. Griebel and M. A. Schweitzer, eds., vol. 65 of Lecture Notes in Computational Science and Engineering, Springer, 2008, pp. 277–299.

[106] ——, *Meshfree and Generalized Finite Element Methods*, habilitation, Institute for Numerical Simulation, University of Bonn, 2008.

[107] ——, *An Algebraic Treatment of Essential Boundary Conditions in the Particle–Partition of Unity Method*, SIAM Journal on Scientific Computing, 31 (2009), pp. 1581–1602.

[108] ——, *Stable enrichment and local preconditioning in the particle–partition of unity method*, Numer. Math., 118 (2011), pp. 137–170.

[109] ——, *Variational mass lumping in the partition of unity method*, SIAM Journal on Scientific Computing, 35 (2013), pp. A1073–A1097.

[110] M. A. Schweitzer and M. Randrianarivony, *Treatment of General Domains in two Space Dimensions in a Partition of Unity Method*, in Meshfree Methods for Partial Differential Equations V, M. Griebel and M. A. Schweitzer, eds., vol. 79 of Lecture Notes in Computational Science and Engineering, Springer, 2010, pp. 27–50.

[111] M. A. Schweitzer and A. Ziegenhagel, *Embedding Enriched Partition of Unity Approximations in Finite Element Simulations*, in Meshfree Methods for Partial Differential Equations VIII, M. Griebel and M. A. Schweitzer, eds., vol. 115 of Lecture Notes in Science and Engineering, Springer International Publishing, 2017, pp. 195–204.

[112] ——, *Multilevel preconditioners for embedded enriched partition of unity approximations*, Advanced Modeling and Simulation in Engineering Sciences, 5 (2018), p. 13.

[113] T. Sederberg, D. Anderson, and R. Goldman, *Implicit representation of parametric curves and surfaces*, Computer Vision, Graphics, and Image Processing, 28 (1984), pp. 72–84.

[114] T. Sederberg, H. Christiansen, and S. Katz, *Improved test for closed loops in surface intersections*, Computer-Aided Design, 21 (1989), pp. 505–508.

[115] T. W. Sederberg and R. J. Meyers, *Loop detection in surface patch intersections*, Computer Aided Geometric Design, 5 (1988), pp. 161–171.

[116] T. W. Sederberg and X. Wang, *Short communication: Rational hodographs*, Comput. Aided Geom. Des., 4 (1987), pp. 333–335.

[117] T. W. Sederberg, J. Zheng, A. Bakenov, and A. Nasri, *T-splines and t-nurccs*, ACM Trans. Graph., 22 (2003), pp. 477–484.

[118] R. Sevilla, S. Fernández-Méndez, and A. Huerta, *Nurbs-enhanced finite element method (nefem)*, International Journal for Numerical Methods in Engineering, 76 (2008), pp. 56–83.

[119] X. Sheng and B. Hirsch, *Triangulation of trimmed surfaces in parametric space*, Computer-Aided Design, 24 (1992), pp. 437–444.

[120] J. R. Shewchuk, *Robust adaptive floating-point geometric predicates*, in Proceedings of the Twelfth Annual Symposium on Computational Geometry, SCG '96, New York, NY, USA, 1996, Association for Computing Machinery, pp. 141–150.

[121] J. R. Shewchuk, *Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator*, in Applied Computational Geometry: Towards Geometric Engineering, M. C. Lin and D. Manocha, eds., vol. 1148 of Lecture Notes in Computer Science, Springer-Verlag, May 1996, pp. 203–222. From the First ACM Workshop on Applied Computational Geometry.

[122] J. R. Shewchuk and B. C. Brown, *Fast segment insertion and incremental construction of constrained delaunay triangulations*, Computational Geometry, 48 (2015), pp. 554–574.

[123] H. Si, *Tetgen, a delaunay-based quality tetrahedral mesh generator*, ACM Trans. Math. Softw., 41 (2015).

[124] P. Sinha, E. Klassen, and K. K. Wang, *Exploiting topological and geometric properties for selective subdivision*, in Proceedings of the First Annual Symposium on Computational Geometry, SCG '85, New York, NY, USA, 1985, Association for Computing Machinery, pp. 39–45.

[125] S. Sloan, *A fast algorithm for constructing delaunay triangulations in the plane*, Advances in Engineering Software (1978), 9 (1987), pp. 34–55.

[126] M. R. Spencer, *Polynomial Real Root Finding in Bernstein Form*, PhD thesis, Department of Civil Engineering, Brigham Young University, 1994.

[127] T. Strouboulis, I. Babuška, and K. Copps, *The design and analysis of the generalized finite element method*, Computer Methods in Applied Mechanics and Engineering, 181 (2000), pp. 43–69.

[128] T. Strouboulis, K. Copps, and I. Babuška, *The generalized finite element method*, Computer Methods in Applied Mechanics and Engineering, 190 (2001), pp. 4081–4193.

[129] I. E. Sutherland and G. W. Hodgman, *Reentrant polygon clipping*, Commun. ACM, 17 (1974), pp. 32–42.

[130] L. M. H. Troska, *Nonlinear Solvers in PUMA*, Master's thesis, Institute for Numerical Simulation, University of Bonn, Mar. 2019.

[131] G. Turk and M. Levoy, *Zippered polygon meshes from range images*, in Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '94, New York, NY, USA, 1994, Association for Computing Machinery, pp. 311–318.

[132] M. Vigo Anglada, *An Improved Incremental algorithm for constructing restricted Delaunay triangulations*, Computers & Graphics, 21 (1997), pp. 215–223.

[133] M. Walter and A. Fournier, *Approximate arc length parametrization*, in Proceedings of the 9th Brazilian Symposium on Computer Graphics and Image Processing, Caxambu, Minas Gerais, Brazil, 29 Oct.–1 Nov. 1996, pp. 143–150.

[134] D. F. Watson, *Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes**, The Computer Journal, 24 (1981), pp. 167–172.

[135] H. Xiao and Z. Gimbutas, *A numerical algorithm for the construction of efficient quadrature rules in two and higher dimensions*, Computers & Mathematics with Applications, 59 (2010), pp. 663–676.

[136] Z. Q. Xie, R. Sevilla, O. Hassan, and K. Morgan, *The generation of arbitrary order curved meshes for 3d finite element analysis*, Computational Mechanics, 51 (2013), pp. 361–374.

[137] J. Yang and S. Han, *Repairing cad model errors based on the design history*, Computer-Aided Design, 38 (2006), pp. 627–640.

[138] J. Yang, S. Han, and S. Park, *A method for verification of computer-aided design model errors*, Journal of Engineering Design, 16 (2005), pp. 337–352.

[139] Y.-J. YANG, S. CAO, J.-H. YONG, H. ZHANG, J.-C. PAUL, J.-G. SUN, AND H.-J. GU, *Approximate computation of curves on B-spline surfaces*, Computer-Aided Design, 40 (2008), pp. 223–234.

[140] W. ZHAO, L. XUEYI, AND L. XIAOMIN, *Analytical algorithm for nurbs surface-plane intersections*, in 2010 International Conference on Mechanic Automation and Control Engineering, 2010, pp. 6153–6157.