Parallel Solution of Linear Systems Arising in Domain Decomposition Methods

Bachelor thesis in the field of study “Computational Engineering” by Magnus Dierking
Date of submission: 07.09.2022

1. Review: Prof. Dr. rer. nat. Sebastian Schöps
2. Review: Maximilian Nolte, M.Sc.
Darmstadt

TECHNISCHE UNIVERSITÄT DARMSTADT

Computational Electromagnetics Group
Darmstadt Centre for Computational Engineering

Erklärung zur Abschlussarbeit gemäß §22 Abs. 7 APB TU Darmstadt

Hiermit versichere ich, Magnus Dierking, die vorliegende Bachelorarbeit gemäß §22 Abs. 7 APB der TU Darmstadt ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Falle eines Plagiats (§38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

Darmstadt, 07.09.2022

M. Dierking

Abstract

This bachelor thesis addresses the implementation of the dual-primal isogeometric tearing and interconnecting algorithm into the **B**oundary **E**lement **M**ethod **B**ased **E**ngineering **L**ibrary framework, short Bembel. The first part reviews theoretical concepts in order to solve a boundary value problem using a domain decomposition approach. The global problem is split up into smaller boundary value problems on independent subdomains and continuity constraints are introduced using Lagrange multipliers.
Afterwards, we review the implementation, paying special attention to local and global matrix structures that are exploited to divide computations among parallel threads.
Numerical experiments with various geometries and settings are performed in order to validate the implementation. The acquired data shows convergence rates predicted by preceding theoretical work of other authors, while also taking advantage of multithreading.

Contents

1. Introduction

1.1. Motivation

Partial differential equations (PDEs) are essential in our modern understanding of physical processes such as thermo-, fluid- or electrodynamics. Naturally, the study of numerical methods to find an approximate solution for those PDEs is a central aspect of computer aided science and engineering. Among those techniques are the *finite element methods* (FEM), which use a space discretization to approximate an unknown function describing the desired solution [27, 35]. Complex structures require finer discretizations, which is why many practically relevant applications lead to large sparse linear systems [32].

Solving these systems can require enormous computational resources and the performance of available machines often is a limiting factor [7]. As clock speed began to stagnate, increasing performance became almost entirely due to increasing the number of physical cores per processor [26, Sec. 1]. Hence, numerical methods that take better advantage of parallelism are emerging. Among the most widely used approaches are the so-called *domain decomposition methods* [9, 32]. Also referred to as *substructuring* methods, these techniques solve the problem on the original domain by subdividing it into multiple sub-problems on subdomains of the original structure. In the process, additional coupling constraints have to be introduced between those subdomains. The application of domain decomposition to FEM has led to powerful solvers for large-scale problems called *finite element tearing and interconnecting* (FETI) methods [9, 16, 18].

Constructing digital models of domains as parts of geometries relies on the technologies of *computer aided design* (CAD), where an established standard is the usage of certain functions called *basis splines* (B-splines) and *non-uniform rational basis splines* (NURBS) in the mathematical foundation [23, 30, 34].
Introduced by Hughes, Cottrell and Bazilevs [21] in 2005, *isogeometric analysis* (IGA) integrates FEM and CAD into a single, unified process by employing the same functions that define the geometry as a basis for numerical simulations. As NURBS provide an exact representation of the computational domain, the expensive step of mesh generation can be avoided [21, Sec. 1].

Since the modeling of complex geometries is often achieved by combining multiple NURBS geometries, the article [24] by Kleiss, Pechstein, Jüttler and Tomar proposes the combination of a FETI method and IGA into the ***I**sog**E**ometric **T**earing and **I**nterconnecting* (IETI) method. To be precise, the *dual-primal* FETI (FETI-DP) method is chosen, resulting in IETI-DP. This method can then take advantage of the exact geometry of IGA, while also employing the already well-developed solvers of FETI-DP.

The C++ code framework of Bembel, developed by Dölz, Harbrecht, Kurz, Multerer, Schöps and Wolf [10] already provides *OpenMP* parallelization [29] with geometry evaluation processes. In 2021, an operator based on an *isogeometric mortar method* [5] was integrated into the framework by Nolte [28], which can be used to impose continuity constraints between decoupled subdomains in a domain decomposition context.

The aim of this bachelor thesis is to implement the IETI-DP algorithm proposed by [24] into Bembel and combine it with the implemented mortar method. The idea is to provide a solver that takes advantage of the parallelism of domain decomposition by distributing subdomain-wise computations among multiple threads. Special emphasis is placed on the matrix structures that are exploited during the implementation in order to realize the parallel solution of the linear system resulting from the discretization.

1.2. Structure of the Document

After the second chapter establishes a theoretical basis for the analytical problem and how to solve PDE's numerically, a discretized problem can be formulated. In the following Chapter 3, the algebraic steps to deduce a dual problem are reproduced, which leads to the proposed IETI-DP algorithm. Chapter 4 then summarizes the most important facets regarding the implementation of this algorithm into the Bembel framework. In Chapter 5, the implementation is used to analyze geometries of increasing complexity. In this step, convergence rates and the runtime advantages of the parallel approach are investigated. The closing Chapter 6 ends this thesis with a summary and an outlook with possible future extensions and fixes.

2. Foundations

In this chapter, we introduce the theoretical background for this thesis. After defining the needed function spaces and differential operators for the analysis, a given problem is discretized using a finite element method and the IGA setting is introduced. A domain decomposition approach then yields a linear system of decoupled subproblems. Two methods for coupling the resulting subdomains are presented and the ensuing problem's structure is stated. In the notation used, vectors and matrices are written in bold with small and capital letters respectively, e.g. $\mathbf{b}$ and $\mathbf{A}$.

2.1. Sobolev Spaces

In this thesis, we are considering domains with special conditions regarding the smoothness of their boundary. Those domains are referred to as *Lipschitz domains* and are used without further definition, see [27, Def. 3.1] for details. Let $\Omega \subset \mathbb{R}^d$, $d = 2, 3$ be an open Lipschitz domain.

Definition 2.1 (Hilbert Space, [15, A. 3])**.** A vector space X equipped with an inner product $(\cdot,\cdot)_X$ and an induced norm given by

$$||u||_X \coloneqq \sqrt{(u,u)_X}\ , \quad u \in X \tag{2.1}$$

is a *Hilbert Space* if it is complete with respect to $||\cdot||_X$.

Using this definition, the space $L_2(\Omega)$ of functions u that are square-integrable on the domain Ω can be equipped with the inner product

$$(u,v)_{L_2(\Omega)} = \int_\Omega u(\boldsymbol{x})v(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}\ , \tag{2.2}$$

for all functions $u, v\colon \Omega \to \mathbb{R}$ with an induced norm

$$||u||_{L_2(\Omega)} = \sqrt{(u,u)_{L_2(\Omega)}}\ . \tag{2.3}$$

With this, $L_2(\Omega)$ is a Hilbert space.

Definition 2.2 (Weak Derivative, [4, Def. 1.1])**.** The function $u \in L_2(\Omega)$ has the weak derivative $v = \partial^\alpha u$, if $v \in L_2(\Omega)$ and

$$(\Phi, v)_{L_2(\Omega)} = (-1)^{|\alpha|}(\partial^\alpha \Phi, u)_{L_2(\Omega)}\ , \quad \forall \Phi \in C_0^\infty(\Omega)\ . \tag{2.4}$$

Here, $C_0^\infty(\Omega)$ denotes the space of smooth functions over Ω with a compact support strictly included in Ω. The concept of weak derivatives can be transferred to other differential operators, e.g. $v \in L_2(\Omega)$ is the divergence of the d-dimensional function $\boldsymbol{\nu} \in (L_2(\Omega))^d$ in a weak sense, if $(\Phi, v)_{L_2(\Omega)} = (\Phi, \nabla\cdot\boldsymbol{\nu})_{L_2(\Omega)} = -(\nabla\Phi, \boldsymbol{\nu})_{L_2(\Omega)}$ for all functions $\Phi \in C_0^\infty(\Omega)$. In this case, the right-hand side is obtained by applying the divergence theorem.

From now on, a multi-index notation is used. A multi-index is a tuple $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_d) \in \mathbb{N}^d$. We define $|\boldsymbol{\alpha}| = \sum_{1 \leq j \leq d} \alpha_j$ and for any sufficiently differentiable function $D^{\boldsymbol{\alpha}} f \coloneqq \partial_{x_1}^{\alpha_1} \ldots \partial_{x_d}^{\alpha_d} f$.

Definition 2.3 (Sobolev Spaces, [25, p. 73ff])**.** For integers $m \geq 0$ the Sobolev space $H^m(\Omega)$ is a Hilbert space, which denotes all functions $u \in L_2(\Omega)$ that possess weak derivatives for all $|\boldsymbol{\alpha}| \leq m$. In $H^m(\Omega)$ with $m \geq 1$, an inner product with corresponding norm is induced via

$$(u, v)_{H^m(\Omega)} \coloneqq (u, v)_{H^{m-1}(\Omega)} + \sum_{|\boldsymbol{\alpha}|=m} (D^{\boldsymbol{\alpha}} u, D^{\boldsymbol{\alpha}} v)_{L_2(\Omega)} \, , \tag{2.5}$$

with

$$||u||^2_{H^m(\Omega)} \coloneqq ||u||^2_{H^{m-1}(\Omega)} + \sum_{|\boldsymbol{\alpha}|=m} ||D^{\boldsymbol{\alpha}} u||^2_{L_2(\Omega)} \, , \tag{2.6}$$

and $H^0(\Omega) = L_2(\Omega)$.

With this definition $H^0(\Omega) \supset H^1(\Omega) \supset H^2(\Omega) \ldots$ applies. A PDE with given boundary constraints can be formulated as a minimization problem, a so-called *variational problem*. The derivation is not part of this work, please refer to [4, Sec. 2] for details. Solving this variational problem in the Sobolev spaces defined above yields a PDE's *weak solutions* [4, Def. 2.8].

2.2. Surface Partial Differential Equations

In this thesis, we are dealing with PDEs that are stated on a curved surface Γ embedded in $\mathbb{R}^3$. We therefore have to introduce differential operators that incorporate the curvature.
Determine quantities on such a surface, for example the distance between points, can be achieved by doing measurements on tangent vectors. In order to express a direction, one has to take inner products with these tangent vectors. The euclidean inner product is not suitable in this case, as it does not take the surface's curvature into account. As it is derived by [22, Sec. 1.4], the needed information in a point $\boldsymbol{x} \in \Gamma$ is encoded by a smoothly-varying positive-definite bilinear form, the so-called *Riemannian metric*

$$g_{\boldsymbol{x}}(\mathbf{a}, \mathbf{b}) \coloneqq \mathbf{a}^T \mathbf{G}(\boldsymbol{x}) \mathbf{b} \, , \quad \mathbf{a}, \mathbf{b} \in \mathbb{R}^2 \, , \tag{2.7}$$

where $\mathbf{G} \colon \mathbb{R}^2 \to \mathbb{R}^{2\times 2}$ is referred to as the *first fundamental form*. Using this, one can also state differential operators that are defined on curved surfaces.

Definition 2.4 (Surface Differential Operators, [27, Sec. 3.4])**.** Suppose $\boldsymbol{x} \in \Gamma$ can be parameterized using $(u_1, u_2) \in \mathbb{R}^2$ via

$$\boldsymbol{x} = \big(x_1(u_1, u_2), x_2(u_1, u_2), x_3(u_1, u_2)\big)^T \, . \tag{2.8}$$

Using the entries g^{ij} of the inverse of $\mathbf{G}$, the surface gradient $\mathbf{grad}_\Gamma p$ of a function $p \in H^1(\Gamma)$ is defined as

$$\mathbf{grad}_\Gamma p = \sum_{i=1}^{2} \sum_{j=1}^{2} g^{ij} \frac{\partial p}{\partial u_i} \frac{\partial \boldsymbol{x}}{\partial u_j} \, , \tag{2.9}$$

while the divergence of a function $\boldsymbol{\nu} = (\nu_1, \nu_2) \in (H^1(\Gamma))^2$ can be expressed via

$$\operatorname{div}_\Gamma \boldsymbol{\nu} = \frac{1}{\sqrt{\det(\boldsymbol{G})}} \Big\{ \frac{\partial}{\partial u_1} (\sqrt{\det(\boldsymbol{G})}\nu_1) + \frac{\partial}{\partial u_2} (\sqrt{\det(\boldsymbol{G})}\nu_2) \Big\} , \tag{2.10}$$

where $\sqrt{\det(\boldsymbol{G})}$ can be understood as a volume factor for the infinitesimal volumes.

At this point, we can define the *Laplace-Beltrami operator* $\Delta_\Gamma = \operatorname{div}_\Gamma \mathbf{grad}_\Gamma$ and state the boundary value problem.

Problem 2.5 (Laplace-Beltrami on Open Surfaces, [12, Sec. 3.1])**.** For a given $f \in L^2(\Gamma)$ and a sufficiently smooth function g find $u \in H^1(\Gamma)$, such that

$$-\Delta_\Gamma u = f \ , \quad \text{on } \Gamma \ , \tag{2.11}$$

$$u = g \ , \quad \text{at } \partial\Gamma \ . \tag{2.12}$$

The variational formulation of this partial differential equation reads:
With the positive bilinear form a and the linear form l

$$a(u, v) \coloneqq \int_\Gamma \mathbf{grad}_\Gamma u \cdot \mathbf{grad}_\Gamma v \, \mathrm{d}\sigma \ , \tag{2.13}$$

$$l(v) \coloneqq \int_\Gamma f v \, \mathrm{d}\sigma \ , \tag{2.14}$$

find $u \in H^1(\Gamma)$, such that

$$a(u, v) = l(v) \ , \quad \forall v \in H^1(\Gamma) \ . \tag{2.15}$$

Solving such problems numerically on a computer requires a formulation that can be represented by a finite amount of numerical values. This requires a discretization of the continuous problem.

2.3. Finite Element Methods

Galerkin Discretization

For the numerical solution of an elliptical Dirichlet boundary value problem such as Problem 2.5, [4, Sec. 2.4] states that one can find an approximation of the solution in a finite-dimensional, discrete subspace S_h of $H^1(\Gamma)$.
Let $\{V_{h,1}, \ldots, V_{h,\eta}\}$ be the η-dimensional basis of S_h. The function u_h is a solution in S_h, if

$$a(u_h, V_{h,i}) = (f, V_{h,i}) \ , \quad \forall V_{h,i} \in S_h \ . \tag{2.16}$$

In this context, S_h is referred to as the space of *test functions*. Writing the desired solution with the same basis as S_h in the form of

$$u_h = \sum_{k=1}^{\eta} \phi_k V_{h,k} \ , \tag{2.17}$$

and substituting in equation (2.16) yields the linear problem

$$\sum_{k=1}^{\eta} a(V_{h,k}, V_{h,i})\phi_k = (f, V_{h,i}) \ , \quad \forall V_{h,i} \in S_h \ . \tag{2.18}$$

In the second context of (2.17), S_h is denominated the space of *ansatz functions*. Using $\alpha_{i,k} = a(V_{h,k}, V_{h,i})$ as coefficients of a matrix $\mathbf{A} \in \mathbb{R}^{\eta \times \eta}$, $\beta_i = (f, V_{h,i})$ as entries of a vector $\mathbf{b} \in \mathbb{R}^{\eta}$ and $\mathbf{\Phi} = (\Phi_1, \ldots, \Phi_\eta)^T$, the above equations form a system

$$\mathbf{A}\boldsymbol{\phi} = \mathbf{b} \ . \tag{2.19}$$

After solving this linear system, one can obtain the approximation u_h to the solution u of the continuous problem using (2.17). [4, Sec. 2.4.2] derives that the accuracy of u_h substantially depends on the chosen space $S_h \subset H^1(\Gamma)$.

2.4. Isogeometric Analysis

In practice, the domain Γ is often available in the form of a CAD representation. The fundamental idea of isogeometric analysis is to use basis functions that are able to exactly model the underlying geometry in a CAD context to also serve as the basis for the solution space of a finite element analysis [21]. The used functions are defined in the following way.

B-Spline Basis

Definition 2.6 (B-Spline Basis, [30, Sec. 2])**.** Let $= \{\xi_1, \ldots, \xi_{n+p+1}\}$ with $n, p \in \mathbb{N}$ and $\xi_k \in \mathbb{R}$ be a non-decreasing ($\xi_k \leq \xi_{k+1}$) sequence for every $k \in [1, \ldots, n+p+1]$. The vector is termed *knot vector* and the ξ_k *knots*. For the remainder of this thesis it is assumed that the knot vector has the form

$$= \{\underbrace{\xi_1 = \ldots = \xi_{p+1}}_{=0} < \xi_{p+2} \leq \ldots < \underbrace{\xi_{n+1} = \ldots = \xi_{n+p+1}}_{=1}\} \subset [0,1] \ , \tag{2.20}$$

which we, in accordance to a variety of literature on the subject, refer to as an *open* knot vector. Furthermore, we also assume that the multiplicity of knots ξ_j with $j = p+2, \ldots, n$ is at most p. Having specified a knot vector and a degree p, one can compute a set of n B-spline basis functions.
A B-spline basis function $N_{i,\Xi}^{(p)}$ of degree p, related to the knot vector Ξ is defined recursively for $p > 0$ as

$$N_{i,\Xi}^{(p)}(x) \coloneqq \frac{x - \xi_i}{\xi_{i+p} - \xi_i} N_{i,\Xi}^{(p-1)}(x) + \frac{\xi_{i+p+1} - x}{\xi_{i+p+1} - \xi_{i+1}} N_{i+1,\Xi}^{(p-1)}(x) \ , \tag{2.21}$$

and if $p = 0$ as

$$N_{i,\Xi}^{(0)}(x) \coloneqq \begin{cases} 1 \ , & \text{if } \xi_i \leq x < \xi_{i+1} \ , \\ 0 \ , & \text{otherwise} \ , \end{cases} \tag{2.22}$$

for all i with $1 \leq i < n+1$.
Note that fractions with zero denominators are considered to be zero [30, p. 51].

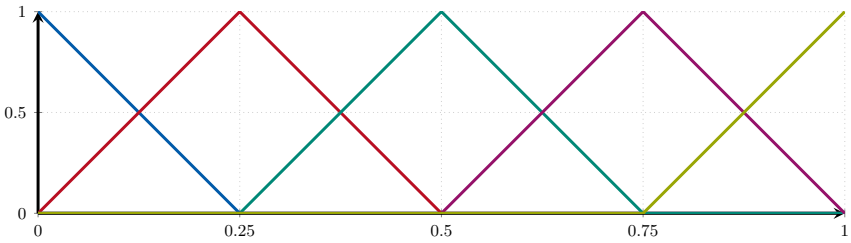
Refinement in the form of adding basis functions on the unit interval is achieved by knot insertion.

Definition 2.7 (Refinement, [6, Sec. 2.1.3])**.** Using the same degree p as before, introducing an extended knot vector with an additional knot in the form of

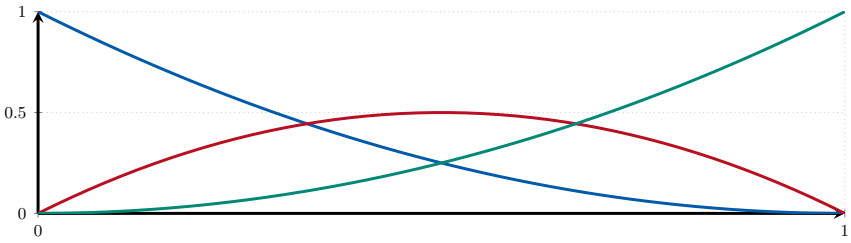
$$\bar{\Xi} \coloneqq \Xi \cup \{\bar{\xi}\} \ , \tag{2.23}$$

and assuming that $\xi_{p+1} < \bar{\xi} < \xi_{n+1}$ and $\bar{\xi} \notin \Xi$ results in a new set of $n+1$ B-spline basis functions. Non-empty intervals $[\xi_k, \xi_{k+1}]$ are called *elements*.

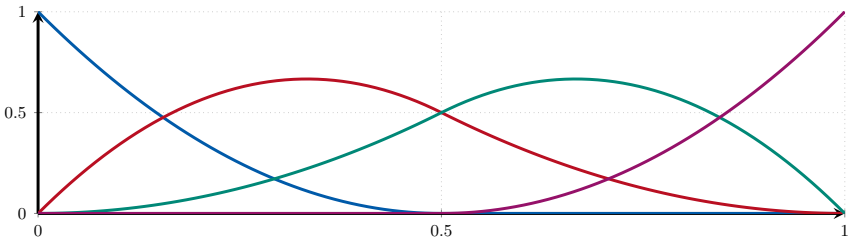
The effect can be retraced by considering Figure 2.1b and Figure 2.1c.



(a) $p = 1, \ \ \Xi = \{0, 0, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}, 1, 1\}$.



(b) $p = 2, \ \ \Xi = \{0, 0, 0, 1, 1, 1\}$.



(c) $p = 2, \ \ \Xi = \{0, 0, 0, \frac{1}{2}, , 1, 1, 1\}$.

Figure 2.1.: B-spline bases for $p = 1, 2$ and different open knot vectors.

For any arbitrary element, the sum of all B-spline basis functions always yields a constant function with value one [30, P. 2.4]. This property is called *partition of unity*.

Definition 2.8 (B-Spline Space, [6, Sec. 3.2])**.** The B-spline space $\mathcal{S}_{p,\Xi}$ is the space spanned by the B-splines

$$\mathcal{S}_p(\Xi) \coloneqq \left\{ \sum_{i=1}^{n} c_i N_{i,\Xi}^{(p)}(x) \ , \quad c_i \in \mathbb{R} \right\} , \tag{2.24}$$

where n denotes the dimension of the space.

The one-dimensional definitions above can be extended to a multivariate setting by using a so-called tensor-product approach. In this thesis, we focus on the bivariate setting.

Definition 2.9 (Tensor-Product B-Splines, [6, Sec. 3.5])**.** Given two knot vectors

$$\Xi \coloneqq [\xi_1 \leq \ldots \leq \xi_{n_1+p_1+1}] \,, \quad \Upsilon \coloneqq [\upsilon_1 \leq \ldots \leq \upsilon_{n_2+p_2+1}] \,, \tag{2.25}$$

one can construct a two-dimensional tensor-product B-spline surface via

$$f(x_1, x_2) \coloneqq \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} c_{i,j} N_{i,\Xi}^{(p_1)}(x_1) N_{j,\Upsilon}^{(p_2)}(x_2) \,, \quad c_{i,j} \in \mathbb{R} \,, \tag{2.26}$$

which can be seen as a surface in $\mathbb{R}^3$ for $(x_1, x_2) \in [\xi_{p_1+1}, \xi_{n_1+1}] \times [\upsilon_{p_2+1}, \upsilon_{n_2+1}]$.

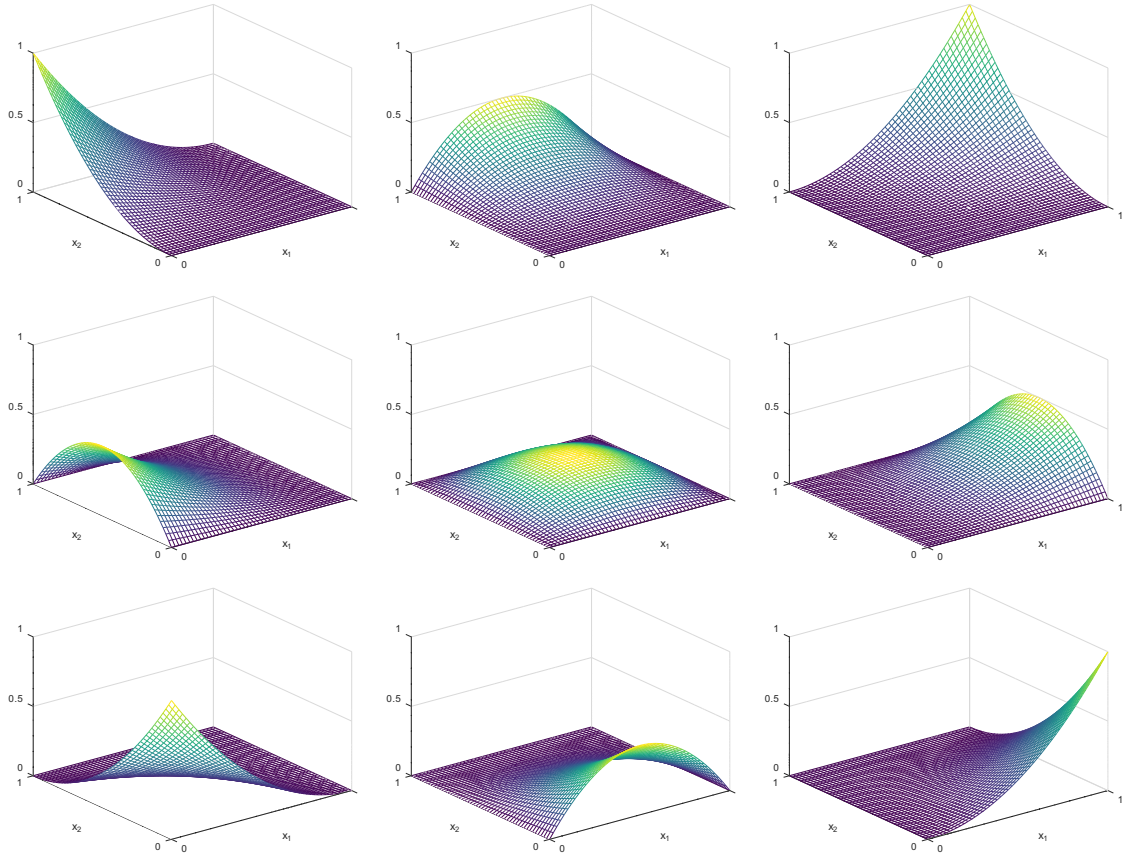


Figure 2.2.: Exemplary bivariate tensor-product B-spline basis functions.

For instance, choosing the B-spline basis of Figure 2.1b for both dimensions yields the nine tensor-product basis functions shown in Figure 2.2. Note that in general the (x_1, x_2) domain of this mapping is a rectangle. We limit this work to always using the same basis for both dimensions, yielding $n_1 = n_2 = n$ and n^2 basis functions. This results in a quadratic domain, which is written as $\square \coloneqq (0, 1)^2$ for the rest of this thesis.
The tensor-product construction (2.26) defines the *tensor-product B-spline space*

$$S_{p_1,p_2}(\Xi, \Upsilon) \,. \tag{2.27}$$

As noted before, we use $p_1 = p_2 = p$ and $\Xi = \Upsilon$ and therefore denote the space on $\square$ with

$$\mathbb{S}_{p,\Xi}(\square) \coloneqq S_{p,p}(\Xi, \Xi) \ . \tag{2.28}$$

In the two-dimensional case, the non-empty sets $[\xi_{p+1}, \xi_{n+1}] \times [\xi_{p+1}, \xi_{n+1}]$ are also referred to as elements.
In IGA, an extension of B-splines in the form of NURBS is utilized as an ansatz space for a finite element analysis. Some additional information can be found in Appendix A.
In classical FEM, there exists a concept of using the same basis functions that are used to approximate the solution to also approximate the geometry, which is called *isoparametric concept* [8, Sec. 3.1]. IGA chooses NURBS, which are the standard in most modern CAD systems [6, Sec. 3.6] as basis functions. They allow an exact representation of the geometry while also possessing desirable properties for approximating solutions, effecively reversing the isoparametric concept [8, Sec. 3.1]. In our case, because of the chosen discretization of Section 2.3, the NURBS are additionally used as test functions.

2.5. Domain Decomposition

In practice, most geometries are not efficiently representable by a single NURBS construct [8, Sec. 2.3]. This is crucial when the geometry's topology differs from the cube topology of $\square$, but also varying materials or physical models on different parts of the domain can be problematic [8, Sec. 2.3]. This leads to the fact that, especially when dealing with complex structures, the combination of multiple surfaces leads to much more flexibility when constructing a geometry [34, Sec. 4].
This offers a natural domain decomposition approach for solving a problem on such geometries, which has great potential for parallel computing.

Multi-patch Geometry Mappings

Definition 2.10 (Patch, [36, Def. 3.6])**.** We define a *patch* Γ to be the image of $\square$ under a mapping $\boldsymbol{F}\colon \square \to \Gamma \subset \mathbb{R}^3$. Let $\Omega \in \mathbb{R}^3$ be a Lipschitz domain and $\Gamma = \partial\Omega$. In this context, $\square$ is referred to as the *parameter domain* and Γ as the *physical domain*. Now assume that the physical domain Γ is represented by N single-patch geometry-mappings $\{\Gamma_i\}_{0 \le i < N}$, $N \in \mathbb{R}$, given by a family of mappings

$$\{\boldsymbol{F}_i\colon \square \to \Gamma_i\}_{0 \le i < N} \ , \tag{2.29}$$

called *parametrisation*.

One possible approach for such mappings are the NURBS in the form

$$\boldsymbol{F}(\square) \coloneqq \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} \mathbf{C}_{i,j} \frac{w_{i,j} N_{i,\Xi}^{(p_1)}(x_1) N_{j,\Upsilon}^{(p_2)}(x_2)}{\sum_{s=1}^{n_1} \sum_{r=1}^{n_2} w_{s,r} N_{s,\Xi}^{(p_1)}(x_1) N_{r,\Upsilon}^{(p_2)}(x_2)} \ , \quad w_{i,j} > 0, \ \mathbf{C}_{i,j} \in \mathbb{R}^d \ . \tag{2.30}$$

Bembel uses the information of the mapping and derives the Jacobian in order to compute the integrals for the variational formulation in Section 2.5 in the parameter domain $\square$ [11, Sec. 3.2]. This integrates the information needed by the operators in Section 2.4 into the bilinear form. The space of test- and ansatz functions is therefore defined on $\square$ with the B-splines and all the calculations are executed there [36, Def. 3.9].

We require the images of $\square$ of all $\boldsymbol{F}_i$ to be disjoint, meaning that $\Gamma_i \cap \Gamma_j = \emptyset$ for $i \neq j$. The mappings in equation (2.29) are utilized in the transformations

$$\iota_i(f) = f \circ \boldsymbol{F}_i \ , \tag{2.31}$$

which enables us to define patch-wise B-spline spaces

$$\mathbb{S}_{p,\Xi_i}(\Gamma_i) \coloneqq \{f \colon \iota_i(f) \in \mathbb{S}_{p,\Xi_i}(\square)\} \ . \tag{2.32}$$

Collecting those spaces for all patches leads to the patch-wise defined B-spline space

$$V_{h,p} \coloneqq \prod_{0 \leq i < N} \mathbb{S}_{p,\Xi_i}(\Gamma_i) \ , \tag{2.33}$$

where Ξ_i denotes the knot vector of the patch Γ_i. By definition, this space is discontinuous across patch boundaries. In this work, we consider patch-wise *uniform* knot vectors, meaning that $\xi_{k+1} - \xi_k = h_i$ for every $k \in [p+1, \dots, n]$, where h_i is referred to as the *element width* in the reference domain. Using Definition 2.7, we define a patch-wise *refinement level* $m_i \geq 0$, which denotes that the associated knot vector possesses 2^{m_i} elements. The concept is outlined in Figure 2.3.

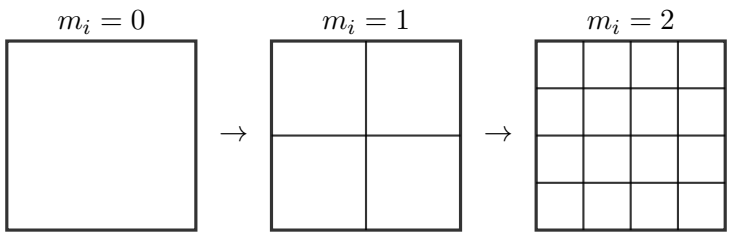


Figure 2.3.: Elements of the parameter domain of the patch Γ_i with increasing refinement level.

From now on, we denote the set of all double-indices (i, j) of basis functions by

$$\mathcal{R} = \big\{(i,j) \colon i \in \{1, \dots, n\}, j \in \{1, \dots, n\}\big\} \ , \tag{2.34}$$

and collapse the double index into one multi-index $\mathbf{k}$. A subset containing all double indices related to the patch Γ_i is written as $\mathcal{R}^{(i)}$. Furthermore, patch indices are also used as superscripts on matrices, while subscripts are applied to vectors or coefficients.

2.5.1. The Stiffness Matrix

The space defined above enables the approach in Definition 2.3 to be used with the variational formulation of Problem 2.5. Using the B-spline basis functions $\varphi_{\mathbf{k},i}, \varphi_{\mathbf{l},i} \in \mathbb{S}_{p,\Xi_i}(\Gamma_i)$, representing the desired solution on $\square$ via

$$u_{h,i} = \sum_{\mathbf{k} \in \mathcal{R}^{(i)}} u_{\mathbf{k},i} \varphi_{\mathbf{k},i} \ , \tag{2.35}$$

and using

$$\begin{aligned} \mathbf{A}^{(i)} &\coloneqq \big(a(\varphi_{\mathbf{k},i}, \varphi_{\mathbf{l},i})\big) \ , \\ \mathbf{f}_i &\coloneqq \big(l(\varphi_{\mathbf{k},i})\big)^T, \quad \forall \mathbf{k}, \mathbf{l} \in \mathcal{R}^{(i)} \ , \end{aligned} \tag{2.36}$$

a linear System

$$\mathbf{A}^{(i)}\mathbf{u}_i = \mathbf{f}_i \ , \tag{2.37}$$

as in (2.19) is obtained on every patch $i = 0, \dots, N-1$ of the domain. The vector $\mathbf{u}_i$ therefore contains all the coefficients $u_{\mathbf{k},i}$ for the chosen tensor-product B-spline space that is used as an ansatz space for the desired solution in the parameter domain of patch Γ_i. These entries are also referred to as *degrees of freedom* (DOFs). In this context, $\mathbf{f}$ is called the *load vector* and $\mathbf{A}$ the *stiffness matrix*.
Collecting all the local stiffness matrices and load vectors as well as concatenating a global solution vector $\mathbf{u}$ for the space $V_{h,p}$ out of the single-patch $\mathbf{u}_i$ vectors results in

$$\mathbf{A} = \begin{pmatrix} \mathbf{A}^{(0)} & & \mathbf{0} \\ & \ddots & \\ \mathbf{0} & & \mathbf{A}^{(N-1)} \end{pmatrix} \begin{pmatrix} \mathbf{u}_0 \\ \vdots \\ \mathbf{u}_{N-1} \end{pmatrix} = \begin{pmatrix} \mathbf{f}_0 \\ \vdots \\ \mathbf{f}_{N-1} \end{pmatrix} . \tag{2.38}$$

With the global stiffness matrix $\mathbf{A}$ acquired above, there are no continuity constraints between the patches themselves as well as between the patches and the global boundary. The kernel of $\mathbf{A}$ is therefore non-trivial, as it includes all patch-wise constant functions. The system (2.38) does not have a unique solution.

2.5.2. The Jump Operator

Since the spaces in (2.33) are in general discontinuous across the interfaces between the subdomains, continuity conditions have to be imposed separately. In order to do that, the authors of [24] propose the following operator.
We define an interface $\overline{e}_{i,j} \coloneqq \overline{\Gamma}_i \cap \overline{\Gamma}_j$ between two patches Γ_i and Γ_j with $i \neq j$ as an *interface edge*, if it contains more than a point. If it only consists of one point it is called an *interface vertex*.
Collecting all the interface-tupels of all interfaces that are non-empty and eliminating duplicates by only keeping those in which the first index is the smaller one yields the index set

$$\mathcal{C} = \{(i,j) \in \mathcal{C}_\Gamma \colon i < j\} \ , \tag{2.39}$$

where each interface and vertex is only represented once. For each pair $(i,j) \in \mathcal{C}$, we say that Γ_i is the *primary domain* and Γ_j is the *secondary domain*.
With this set, one can collect all the indices of the basis functions in Γ_i whose support intersects the interface $\overline{e}_{i,j}$ in

$$\mathcal{B}(i,j) = \{\mathbf{k} \in \mathcal{R}^{(i)} \colon \operatorname{supp} \varphi_{\mathbf{k},i} \cap \overline{e}_{i,j} \neq \emptyset\} \ , \tag{2.40}$$

as well as all the indices of the basis functions whose support intersects the global boundary $\partial\Gamma$ in

$$\mathcal{D}(i) = \{\mathbf{k} \in \mathcal{R}^{(i)} \colon \operatorname{supp} \varphi_{\mathbf{k},i} \cap \overline{e}_{i,D} \neq \emptyset\} \ , \tag{2.41}$$

with $\overline{e}_{i,D} = \partial\Gamma_i \cap \partial\Gamma$.
When using this operator, we only consider the setting of *fully matching* subdomains, meaning that the following two conditions must be fulfilled:

(i) The edge $\overline{e}_{i,j}$ is the image of an entire edge of the respective parameter domains.

(ii) For each multi-index $\mathbf{k} \in \mathcal{B}(i,j)$, there must be a unique multi-index $\mathbf{l} \in \mathcal{B}(j,i)$, such that

$$\varphi_{\mathbf{k},i}\Big|_{\overline{e}_{i,j}} = \varphi_{\mathbf{l},j}\Big|_{\overline{e}_{i,j}} , \tag{2.42}$$

meaning that for a basis function on the edge of patch Γ_i there must be a unique associated basis function on the patch Γ_j that is connected to Γ_i via this edge.

This fully matching setting in combination with the tensor-product structure of the basis functions can be exploited to collect the DOFs associated with an index set. The only information one has to know is which side of the parameter domain defines the interface.

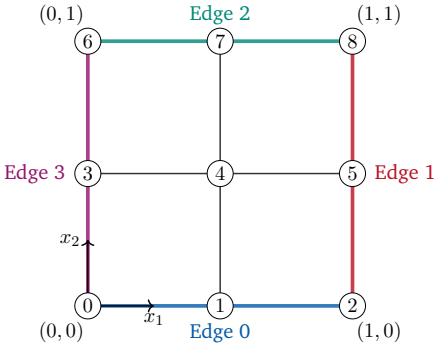


Figure 2.4.: Numbering of DOFs and edges for a single patch in the parameter domain $\square$.

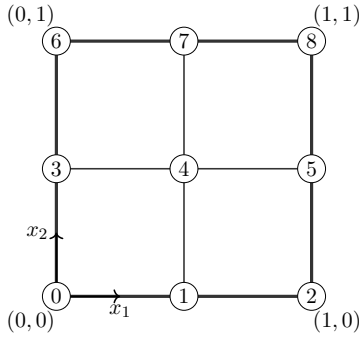
In order to clarify the numbering, consider the parameter domain $\square$ with the B-spline basis of Figure 2.1b. As stated before, this thesis only takes tensor-product surfaces with equal degree and knot vectors into account, leading to nine DOFs per patch in this example case. As can be seen in Figure 2.4, the DOFs are numbered canonically. Respect that the indicated DOFs are the same ones that were shown in Figure 2.2. The edges of a patch are enumerated counter-clockwise, signalized by the coloring in Figure 2.4.
The total number of continuity constraints resulting from (2.40) and (2.41) is designated as $\mathcal{J}$ and the total number of DOFs from all patches as $\mathcal{N}$. Using this, the *jump operator* $\mathbf{B} \in \{-1, 0, 1\}^{\mathcal{J} \times \mathcal{N}}$ is defined as

$$\mathbf{B} \colon V_{h,p} \to \mathbb{R}^{\mathcal{J}} , \tag{2.43}$$

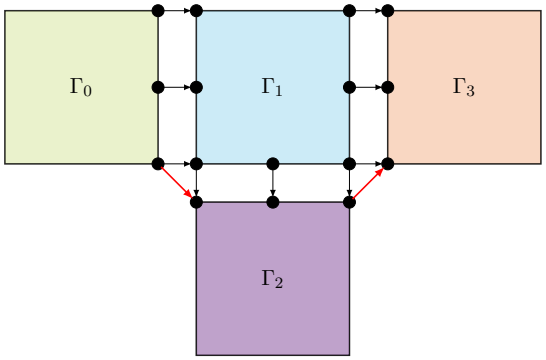
$$(\mathbf{B}u)_{\mathbf{k},(i,j)} = u_{\mathbf{k},i} - u_{\mathbf{l},j} , \tag{2.44}$$

$$(\mathbf{B}u)_{\mathbf{k},(i,\mathcal{D})} = u_{\mathbf{k},i} . \tag{2.45}$$

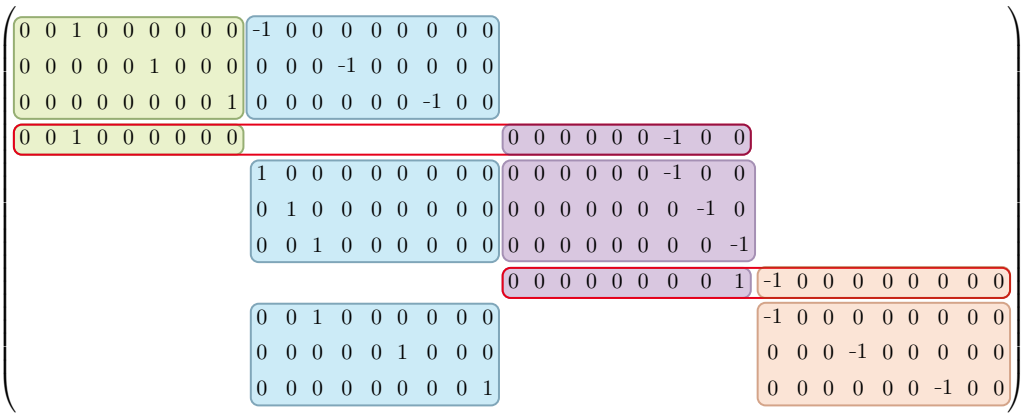
This results in a matrix for the linear operator $\mathbf{B}$, where a row with exactly one 1 and one -1 indicates a coupling condition between two patches and a row with a single 1 a condition for the global boundary.



(a) Numbering scheme on every patch for reference.



(b) Exemplary multi-patch geometry.



(c) Resulting structure of the jump operator.

Figure 2.5.: Assembly of a jump operator without boundary conditions for an exemplary geometry.

In order to outline the jump operators global structure, consider a simple setting of four patches as in Figure 2.5b without incorporating the conditions on the global boundary. For this example, also assume that the patches aren't twisted against each other, meaning the lower left DOF is always the one with the smallest index on the patch. With three B-spline basis functions in every direction on every patch, this yields $3^2 \cdot 4 = 36$ DOFs. As can be seen in the exemplary geometry, three edge couplings with three DOFs each, as well as two additional vertex couplings need to be taken into account. This leads to $\mathcal{J} = 3 \cdot 3 + 2 \cdot 1 = 11$ constraints and a 11×36 jump operator. Again, outside of this example additional conditions for the global boundary $\partial\Gamma$ are indispensable.
The resulting matrix for the jump operator takes on the structure in Figure 2.5c, with two colored blocks aligned horizontally indicating one interface coupling. The block with positive entries accounts for the DOFs of the primary domain, while the block with negative entries does so for the secondary one. The red markings highlight the couplings of subdomain vertices. All entries outside of the colored blocks are zero.
For C^0-continuity and the incorporation of Dirichlet boundary conditions, the resulting System has the form

$$\mathbf{Bu} = \mathbf{b} \,, \tag{2.46}$$

where the entries of the vector $\mathbf{b} \in \mathbb{R}^{\mathcal{J}}$ are zero when corresponding to an interface condition and equal to a coefficient when corresponding to a boundary condition.

Regarding Dirichlet boundary conditions, the work of [24] assumes that the prescribed Dirichlet data g_D is such that there exists a $\hat{g} \in V_{h,p} \colon g_D = \hat{g}\big|_{\partial\Gamma}$. Therefore, the subsets of continuous functions

$$V_{0,h,p} = \{v \in V_{h,p} \colon v\big|_{\partial\Gamma} = 0\} \ , \tag{2.47}$$

$$V_{g,h,p} = \hat{g} + V_{h,p} = \{v \in V_{h,p} \colon v\big|_{\partial\Gamma} = g_D\} \ . \tag{2.48}$$

in $V_{h,p}$ are defined. In the non-homogeneous case, the authors of [24] assume that the given function on the boundary can be approximate with the used B-spline basis via

$$g_D\big|_{\bar{e}_{i,\mathcal{D}}} = \sum_{\mathbf{k} \in \mathcal{D}(i)} g_{\mathbf{k},i} \varphi_{\mathbf{k}}\big|_{\bar{e}_{i,\mathcal{D}}} \ , \tag{2.49}$$

with real-valued coefficients $g_{\mathbf{k},i}$. This can be incorporated in the jump operator by requiring

$$\mathbf{u}_{\mathbf{k},i} = g_{\mathbf{k},i} \ , \tag{2.50}$$

on all edges associated with the global boundary. Adding these coefficients into the vector $\mathbf{b}$ virtually corresponds to coupling the DOFs with a virtual neighbour subdomain. If only homogeneous Dirichlet conditions are considered, $\mathbf{b}$ is a zero vector.

2.5.3. Mortar Method

The jump operator defined above imposes C^0-continuity by matching the DOF between two patches of the boundary along their respective edges or vertices. An alternative approach for coupling the subdomains is offered by the *mortar method*, which was originally published for FEM by Maday et al. in 1989. In 2021, the isogeometric mortar method based on the work of Brivadis et al. [5] was implemented into the Bembel framework by Nolte [28].
The method imposes continuity along interfaces in a weak sense by using an m-dimensional space $M_{h,l}$ of discrete Lagrange multipliers. This enables the formulation of the *mortar linear form*

$$b(v_h, \mu_h) \coloneqq \sum_{l=1}^{L} \int_{\bar{e}_l} [v_h] \lambda_h \mathrm{d}s, \quad v_h \in V_{h,p} \ , \ \lambda_h \in M_h \ , \tag{2.51}$$

with $[v_h]|_{\bar{e}_l} = v_h|_{\Gamma_{i(l)}} - v_h|_{\Gamma_{j(l)}}$ being the basis function's jump from primal to secondary patch on the l-th edge. This leads to the adapted variational formulation

$$a(u_h, v_h) + b(v_h, \lambda_h) = l(v_h) \qquad \forall v_h \in V_{h,p} \ , \tag{2.52}$$

$$b(u_h, \mu_h) \qquad\qquad = \langle g \ , \mu_h \rangle \quad \forall \mu_h \in M_h \ . \tag{2.53}$$

If $\lambda_h \in M_h$ is represented using basis functions via

$$\lambda_h = \sum_{i=1}^{m} \lambda_{h,i} N_i \ , \tag{2.54}$$

and with the spaces from (2.33), the constraints can be expressed in a matrix $\mathbf{B} \in \mathbb{R}^{m \times \mathcal{N}}$

$$\mathbf{B} \coloneqq \big(b(N_j, \varphi_i)\big)_{1 \leq j \leq m, 1 \leq i \leq \mathcal{N}} \ . \tag{2.55}$$

The corresponding right-hand side $\mathbf{b} \in \mathbb{R}^m$ for this linear operator is constructed via

$$\mathbf{b} \coloneqq (\langle g, \mu_1 \rangle, \dots, \langle g, \mu_m \rangle)^T \ , \tag{2.56}$$

which again is a zero vector in the case of homogeneous Dirichlet data. The resulting mortar operator can be used as an alternative to the jump operator. From now on, we assume this operator as given. For additional information on the analytical background and the implementation, please refer to the given sources.

2.6. Saddle Point Formulation

After assembling the matrices above, the discretized problem can be formulated as a saddle point Problem [24, Sec. 4.2]. Assume the given load vector

$$\mathbf{f} \coloneqq \begin{pmatrix} \mathbf{f}_0^T & \dots & \mathbf{f}_{N-1}^T \end{pmatrix}^T, \tag{2.57}$$

the stiffness matrix $\mathbf{A}$ from Section 2.5 and one of the coupling operators $\mathbf{B}$ with corresponding right-hand side $\mathbf{b}$ defined above. From now on, when a specific coupling operator is meant, the subscripts J and M are used.
For the jump operator this yields the following.

Problem 2.11 (Saddle Point Problem with Jump Operator)**.** Using this, find $u \in V_{g,h,p}$ with the vector representation $\mathbf{u}$ and *Lagrange multipliers* $\lambda \in \mathbb{R}^{\mathcal{J}}$, such that

$$\begin{pmatrix} \mathbf{A} & \mathbf{B}_\mathrm{J}^T \\ \mathbf{B}_\mathrm{J} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \mathbf{u} \\ \lambda \end{pmatrix} = \begin{pmatrix} \mathbf{f} \\ \mathbf{b}_\mathrm{J} \end{pmatrix}. \tag{2.58}$$

In the case of homogeneous boundary conditions, $V_{g,h,p}$ is replaced by $u \in V_{0,h,p}$ and $\mathbf{b}_\mathrm{J}$ is a zero vector.

Instead using the provided mortar operator to enforce the constraints, the following problem applies.

Problem 2.12 (Saddle Point Problem with Mortar Operator)**.** Using this, find $u \in V_{h,p}$ with the vector representation $\mathbf{u}$ and Lagrange multipliers $\lambda \in \mathbb{R}^m$, such that

$$\begin{pmatrix} \mathbf{A} & \mathbf{B}_\mathrm{M}^T \\ \mathbf{B}_\mathrm{M} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \mathbf{u} \\ \lambda \end{pmatrix} = \begin{pmatrix} \mathbf{f} \\ \mathbf{b}_\mathrm{M} \end{pmatrix}. \tag{2.59}$$

In the case of homogeneous boundary conditions, $\mathbf{b}_\mathrm{M}$ is a zero vector.

3. Dual-Primal IETI

In this chapter, we reproduce the linear algebra operations done by [24] in order to state an algorithm for the IETI-DP method and introduce the setting for this thesis. Information on the global boundary as well as continuity constraints are added and the matrices in Section 2.5 then undergo restructuring. In the process, boundary data is directly incorporated into the stiffness matrix with the aim of sustaining a solvable dual system. The size of this system is determined by the number of constraints of the used coupling operator, which is significantly smaller than the initial saddle point problem. The solution of the dual system can then be used to find the DOFs of the original problem.
Part of the restructuring and part of the solving for the DOFs happens independently for each subdomain, which can be exploited later by parallelizing parts of the algorithm. In the process, a preconditioner for the dual system is also stated. This whole chapter therefore sticks closely to the introduction stated by [24].

3.1. Primal and Remaining Degrees of Freedom

As stated in Section 2.4, we only consider open knot vectors. In combination with the partition of unity property, this leads to the fact that at every vertex of the parameter domain $\square$, only one basis function has the value 1, while the support of all the others vanishes. This can be validated with (2.21) or with Figure 2.1. In Figure 2.5 this would be DOF $0, 2, 6$ and 8 according to the defined scheme. Since these DOFs can be distinguished from the others, they are designated as *primal* DOFs and all others are referred to as *remaining* DOFs. To indicate this partition, the subscripts P and R are used respectively.
All previously defined vectors and matrices apart from λ and $\mathbf{b}$ can be sorted according to this classification. This is achieved by collecting all the primal DOFs that are associated with a common point in the physical domain and fixing a global numbering system. The solution vector

$$\widetilde{\mathbf{u}} = \begin{pmatrix} \widetilde{\mathbf{u}}_{\mathrm{P}}^T & \widetilde{\mathbf{u}}_{\mathrm{R},0}^T & \dots & \widetilde{\mathbf{u}}_{\mathrm{R},N-1}^T \end{pmatrix}^T , \tag{3.1}$$

is constructed by collecting the primal DOFs in the vector $\widetilde{\mathbf{u}}_{\mathrm{P}}$ and then concatenating it with all the patch-wise remaining DOFs. When using the jump operator, the desired solution is associated with a subspace of $V_{h,p}$, which is defined as

$$\widetilde{W}_{h,p} = \{u \in V_{h,p} \colon u \text{ is continuous at all vertices}\} . \tag{3.2}$$

The mortar operator imposes the continuity in a weak sense and therefore seeks a solution in $V_{h,p}$. Using the same procedure for the respective constraint operator, one can obtain the operator $\widetilde{\mathbf{B}}$ with the structure

$$\widetilde{\mathbf{B}} = \begin{pmatrix} \widetilde{\mathbf{B}}_{\mathrm{P}} & \widetilde{\mathbf{B}}_{\mathrm{R}}^{(0)} & \dots & \widetilde{\mathbf{B}}_{\mathrm{R}}^{(N-1)} \end{pmatrix} . \tag{3.3}$$

Regarding the stiffness matrix, this classification can be applied to each local subdomain stiffness matrix. Assembling the global stiffness matrix from the local contributions yields the structure

$$\widetilde{\mathbf{A}} = \begin{pmatrix} \widetilde{\mathbf{A}}_{\mathrm{PP}} & \widetilde{\mathbf{A}}_{\mathrm{PR}} \\ \widetilde{\mathbf{A}}_{\mathrm{PR}} & \widetilde{\mathbf{A}}_{\mathrm{RR}} \end{pmatrix}, \tag{3.4}$$

with the corresponding global load vector

$$\widetilde{\mathbf{f}} = \begin{pmatrix} \widetilde{\mathbf{f}}_{\mathrm{P}} \\ \widetilde{\mathbf{f}}_{\mathrm{R}} \end{pmatrix}. \tag{3.5}$$

It is important to note that only for $\widetilde{\mathbf{A}}_{RR}$ the block-diagonal form shown in (2.38) is preserved. As it is done at the end of Section 2.5, the problem can be formulated as a saddle point problem:

Problem 3.1 (Dual-Primal Formulation)**.** Find $u \in \widetilde{W}_{h,p}$, represented by $\tilde{\mathbf{u}}$ as in (3.1) and Lagrange multipliers $\lambda \in \mathbb{R}^{\mathcal{J}}$, such that

$$\begin{pmatrix} \widetilde{\mathbf{A}} & \widetilde{\mathbf{B}}_{\mathrm{J}}^{T} \\ \widetilde{\mathbf{B}}_{\mathrm{J}} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \widetilde{\mathbf{u}} \\ \lambda \end{pmatrix} = \begin{pmatrix} \widetilde{\mathbf{f}} \\ \mathbf{b}_{\mathrm{J}} \end{pmatrix}. \tag{3.6}$$

When using the operator $\mathbf{B}_{\mathrm{M}}$ with right-hand side $\mathbf{b}_{\mathrm{M}}$, the space $\widetilde{W}_{h,p}$ is replaced by $V_{h,p}$ and $\lambda \in \mathbb{R}^{m}$.

At this point, the global stiffness matrix is still singular, as we only incorporated the continuity between the patches via the constraint operators, effectively only rearranging the stiffness matrix.

3.2. Essential Boundary Conditions

In order to obtain a regular matrix, essential boundary conditions have to be incorporated. To achieve that, a further distinction of the primal DOFs defined above has to be undertaken.

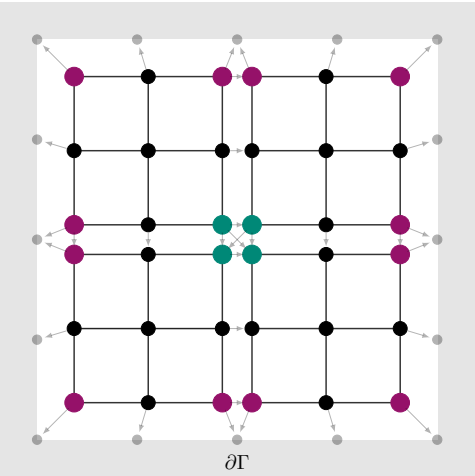


Figure 3.1.: Categorization of DOFs for a four-patch geometry.

The primal DOFs that are associated with the global boundary $\partial\Gamma$ are labeled *essential* primal DOFs and those in the interior of our domain Ω *floating* primal DOFs. This is indicated again by using subscripts, namely d and f.
An example of this categorization can be examined in Figure 3.1, where the same B-spline basis as in the example for the jump operator in Section 2.5 is used. Essential primal DOFs are colored purple, floating DOFs green and remaining DOFs black. Using this to again rearrange matrices and vectors, we get a sorted variant of $\widetilde{\mathbf{u}}$ in the form of

$$\widetilde{\mathbf{u}} = \begin{pmatrix} \widetilde{\mathbf{u}}_\mathrm{d}^T & \widetilde{\mathbf{u}}_\mathrm{f}^T & \widetilde{\mathbf{u}}_{\mathrm{R},0}^T & \dots & \widetilde{\mathbf{u}}_{\mathrm{R},N-1}^T \end{pmatrix}^T , \tag{3.7}$$

with corresponding matrices

$$\widetilde{\mathbf{A}} = \begin{pmatrix} \widetilde{\mathbf{A}}_\mathrm{dd} & \widetilde{\mathbf{A}}_\mathrm{df} & \widetilde{\mathbf{A}}_\mathrm{dR} \\ \widetilde{\mathbf{A}}_\mathrm{fd} & \widetilde{\mathbf{A}}_\mathrm{ff} & \widetilde{\mathbf{A}}_\mathrm{fR} \\ \widetilde{\mathbf{A}}_\mathrm{Rd} & \widetilde{\mathbf{A}}_\mathrm{Rf} & \widetilde{\mathbf{A}}_\mathrm{RR} \end{pmatrix} , \tag{3.8}$$

$$\widetilde{\mathbf{B}} = \begin{pmatrix} \widetilde{\mathbf{B}}_\mathrm{d} & \widetilde{\mathbf{B}}_\mathrm{f} & \widetilde{\mathbf{B}}_\mathrm{R}^{(0)} & \dots & \widetilde{\mathbf{B}}_\mathrm{R}^{(N-1)} \end{pmatrix} , \tag{3.9}$$

and right-hand side

$$\widetilde{\mathbf{f}} = \begin{pmatrix} \widetilde{\mathbf{f}}_\mathrm{d}^T & \widetilde{\mathbf{f}}_\mathrm{f}^T & \widetilde{\mathbf{f}}_{\mathrm{R},0}^T & \dots & \widetilde{\mathbf{f}}_{\mathrm{R},N-1}^T \end{pmatrix}^T . \tag{3.10}$$

Since the solution for the essential primal DOFs is known from (2.50), the corresponding parts can be transferred to the right side. With that, Problem 3.1 is equivalent to the following problem:

Problem 3.2 (Dual-Primal Formulation with Essential Boundary Conditions)**.** Find $u \in \widetilde{W}_{h,p}$, represented by $\widetilde{\mathbf{u}}$ and Lagrange multipliers $\lambda \in \mathbb{R}^J$, such that

$$\begin{pmatrix} \overline{\mathbf{A}} & \overline{\mathbf{B}}_\mathrm{J}^T \\ \overline{\mathbf{B}}_\mathrm{J} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \widetilde{\mathbf{u}} \\ \lambda \end{pmatrix} = \begin{pmatrix} \bar{\mathbf{f}} \\ \bar{\mathbf{b}}_\mathrm{J} \end{pmatrix} , \tag{3.11}$$

where

$$\overline{\mathbf{A}} = \begin{pmatrix} \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \widetilde{\mathbf{A}}_\mathrm{ff} & \widetilde{\mathbf{A}}_\mathrm{fR} \\ \mathbf{0} & \widetilde{\mathbf{A}}_\mathrm{Rf} & \widetilde{\mathbf{A}}_\mathrm{RR} \end{pmatrix} , \tag{3.12}$$

$$\bar{\mathbf{f}} = \begin{pmatrix} \widetilde{\mathbf{g}}_\mathrm{d} \\ \tilde{\mathbf{f}}_\mathrm{f} - \widetilde{\mathbf{A}}_\mathrm{fd}\widetilde{\mathbf{g}}_\mathrm{d} \\ \tilde{\mathbf{f}}_\mathrm{R} - \widetilde{\mathbf{A}}_\mathrm{Rd}\widetilde{\mathbf{g}}_\mathrm{d} \end{pmatrix} , \tag{3.13}$$

$$\overline{\mathbf{B}}_\mathrm{J} = \begin{pmatrix} \mathbf{0} & \widetilde{\mathbf{B}}_\mathrm{J,f} & \widetilde{\mathbf{B}}_\mathrm{J,R} \end{pmatrix} , \tag{3.14}$$

$$\overline{\mathbf{b}}_\mathrm{J} = \mathbf{b}_\mathrm{J} - \widetilde{\mathbf{B}}_\mathrm{J,d}\widetilde{\mathbf{g}}_\mathrm{d} , \tag{3.15}$$

with $\mathbf{I}$ being the identity matrix. The vector $\widetilde{\mathbf{g}}_\mathrm{d}$ contains the values of the analytical function g_D evaluated at the essential primal DOFs. Again, when using the operator $\mathbf{B}_\mathrm{M}$ with right-hand side $\mathbf{b}_\mathrm{M}$, the space $\widetilde{W}_{h,p}$ is replaced by $V_{h,p}$ and $\lambda \in \mathbb{R}^m$.

The constraints between the patches eliminate the patch-wise constant functions from the kernel, while the incorporation of global boundary conditions eliminates the global constant functions. The matrix $\overline{\mathbf{A}}$ is therefore invertible. The matrices above can be separated into blocks that correspond to primal and remaining DOFs respectively, yielding

$$\overline{\mathbf{A}}_{\mathrm{PP}} = \begin{pmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \widetilde{\mathbf{A}}_{\mathrm{ff}} \end{pmatrix}, \qquad \overline{\mathbf{A}}_{\mathrm{PR}} = \begin{pmatrix} \mathbf{0} \\ \widetilde{\mathbf{A}}_{\mathrm{fR}} \end{pmatrix}, \tag{3.16}$$

$$\overline{\mathbf{A}}_{\mathrm{RP}} = \begin{pmatrix} \mathbf{0} & \widetilde{\mathbf{A}}_{\mathrm{Rf}} \end{pmatrix}, \qquad \overline{\mathbf{A}}_{\mathrm{RR}} = \widetilde{\mathbf{A}}_{\mathrm{RR}}\ , \tag{3.17}$$

$$\bar{\mathbf{f}}_{\mathrm{P}} = \begin{pmatrix} \widetilde{\mathbf{g}}_{\mathrm{d}} \\ \widetilde{\mathbf{f}}_{\mathrm{f}} - \widetilde{\mathbf{A}}_{\mathrm{fd}} \widetilde{\mathbf{g}}_{\mathrm{d}} \end{pmatrix}, \qquad \bar{\mathbf{f}}_{\mathrm{R}} = \widetilde{\mathbf{f}}_{\mathrm{R}} - \widetilde{\mathbf{A}}_{\mathrm{Rd}} \widetilde{\mathbf{g}}_{\mathrm{d}}\ , \tag{3.18}$$

$$\overline{\mathbf{B}}_{\mathrm{P}} = \begin{pmatrix} \mathbf{0} & \widetilde{\mathbf{B}}_{\mathrm{f}} \end{pmatrix}, \qquad \overline{\mathbf{B}}_{\mathrm{R}} = \widetilde{\mathbf{B}}_{\mathrm{R}}\ . \tag{3.19}$$

3.3. Dual Problem

Inverting the matrix (3.12) and applying it to the first line in (3.11) on both sides yields

$$\widetilde{\mathbf{u}} = \overline{\mathbf{A}}^{-1}(\bar{\mathbf{f}} - \overline{\mathbf{B}}^T \lambda)\ . \tag{3.20}$$

Problem 3.3 (Dual Problem)**.** With this, one can reformulate the second line of (3.11) and obtain the dual problem

$$\mathbf{F}\lambda = \mathbf{d}\ , \tag{3.21}$$

with the symmetric and positive definite matrix $\mathbf{F} = \overline{\mathbf{B}\mathbf{A}}^{-1}\overline{\mathbf{B}}^T$ and the vector $\mathbf{d} = \overline{\mathbf{B}\mathbf{A}}^{-1}\bar{\mathbf{f}} - \overline{\mathbf{b}}$.

The authors of [24] state that the system of Problem 3.3 can be solved using a *Conjugate Gradient* (CG) algorithm [32, Sec. 6.7]. To explicitly compute the needed inverse, the block-matrices defined in Problem 3.2 can be utilized with the block factorization

$$\overline{\mathbf{A}}^{-1} = \begin{pmatrix} \mathbf{I} & \mathbf{0} \\ -\overline{\mathbf{A}}_{\mathrm{RR}}^{-1}\overline{\mathbf{A}}_{\mathrm{RP}} & \mathbf{I} \end{pmatrix} \begin{pmatrix} \overline{\mathbf{S}}_{\mathrm{PP}}^{-1} & \mathbf{0} \\ \mathbf{0} & \overline{\mathbf{A}}_{\mathrm{RR}}^{-1} \end{pmatrix} \begin{pmatrix} \mathbf{I} & -\overline{\mathbf{A}}_{\mathrm{PR}}\overline{\mathbf{A}}_{\mathrm{R}}^{-1} \\ \mathbf{0} & \mathbf{I} \end{pmatrix}, \tag{3.22}$$

where

$$\overline{\mathbf{S}}_{\mathrm{PP}} = \overline{\mathbf{A}}_{\mathrm{PP}} - \overline{\mathbf{A}}_{\mathrm{PR}}\overline{\mathbf{A}}_{\mathrm{RR}}^{-1}\overline{\mathbf{A}}_{\mathrm{RP}}\ . \tag{3.23}$$

One can assemble this global matrix $\overline{\mathbf{S}}_{PP}$ from local contributions

$$\overline{\mathbf{S}}_{PP}^{(i)} = \overline{\mathbf{A}}_{\mathrm{PP}}^{(i)} - \overline{\mathbf{A}}_{\mathrm{PR}}^{(i)}(\overline{\mathbf{A}}_{\mathrm{RR}}^{(i)})^{-1}\overline{\mathbf{A}}_{\mathrm{RP}}^{(i)}\ . \tag{3.24}$$

To construct $\mathbf{u}$ from the solution of the dual problem, one can first calculate

$$\widetilde{\mathbf{u}}_{\mathrm{P}} = \overline{\mathbf{S}}_{\mathrm{PP}}^{-1}\big(\bar{\mathbf{f}}_{\mathrm{P}} - \overline{\mathbf{B}}_{\mathrm{P}}^T\lambda - \overline{\mathbf{A}}_{\mathrm{PR}}\overline{\mathbf{A}}_{\mathrm{RR}}^{-1}(\bar{\mathbf{f}}_{\mathrm{R}} - \overline{\mathbf{B}}_{\mathrm{R}}^T\lambda)\big)\ , \tag{3.25}$$

and then compute the remaining local solutions via

$$\widetilde{\mathbf{u}}_{\mathrm{R},i} = (\overline{\mathbf{A}}_{\mathrm{RR}}^{(i)})^{-1}\big(\bar{\mathbf{f}}_{\mathrm{R},i} - (\overline{\mathbf{B}}_{\mathrm{R}}^{(i)})^T\lambda - \overline{\mathbf{A}}_{\mathrm{RP}}^{(i)}\widetilde{\mathbf{u}}_{\mathrm{P}}\big)\ . \tag{3.26}$$

3.4. Preconditioner

In order to improve the convergence of dual system when using the jump operator, a scaled Dirichlet preconditioner that was introduced in [17] and extended to the dual-primal formulation in [16, Sec. 2.3] is used by the authors of [24]. The usage of this preconditioner also requires a *Preconditioned Conjugate Gradient* algorithm [32, Sec. 9].
For this, another categorization of the local DOFs is required. DOFs on the boundary $\partial\Gamma_i$ of a patch Γ_i are indicated with the subscript B and the ones in the interior with subscript I. Listing the DOFs in the interior first, one obtains the structure

$$\mathbf{A}^{(i)} = \begin{pmatrix} \mathbf{A}_{\mathrm{II}}^{(i)} & \mathbf{A}_{\mathrm{IB}}^{(i)} \\ \mathbf{A}_{\mathrm{BI}}^{(i)} & \mathbf{A}_{\mathrm{BB}}^{(i)} \end{pmatrix}. \tag{3.27}$$

The dual-primal Dirichlet preconditioner is defined as

$$\mathbf{M}^{-1} = \sum_{i=1}^{N} \mathbf{D}^{(i)} \mathbf{B}_{\mathrm{J}}^{(i)} \begin{pmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{S}_{\mathrm{BB}}^{(i)} \end{pmatrix} (\mathbf{B}_{\mathrm{J}}^{(i)})^T \mathbf{D}^{(i)} \ , \tag{3.28}$$

where

$$\mathbf{S}_{\mathrm{BB}}^{(i)} = \mathbf{A}_{\mathrm{BB}}^{(i)} - \mathbf{A}_{\mathrm{Bl}}^{(i)} (\mathbf{A}_{\mathrm{ll}}^{(i)})^{-1} \mathbf{A}_{\mathrm{lB}}^{(i)} \ , \tag{3.29}$$

and $\mathbf{B}_{\mathrm{J}}^{(i)}$ denotes the constraints associated with patch Γ_i. The entry at position (k, k) of the matrix $\mathbf{D}^{(i)} \in \mathbb{R}^{\mathcal{J} \times \mathcal{J}}$ is $\frac{1}{mult(k)}$ with

- $mult(k) = 1$, if λ_k corresponds to an essential boundary condition,
- $mult(k) = 2$, if λ_k corresponds to a coupling condition that does not involve a subdomain vertex,
- $mult(k) \geq 2$, if λ_k corresponds to a coupling condition that involves a subdomain vertex.

3.5. IETI-DP Algorithm

With the partitioning explained above, [24] proposes the following algorithm.

Algorithm (IETI-DP)**.** *Let* $\mathbf{A}$ *be the global stiffness matrix and* $\mathbf{f}$ *the global load vector as shown in* (2.38).

for $i \leftarrow 1$ **to** N **do** /* in parallel */
  Assemble $\mathbf{A}^{(i)}$ *from* $\mathbf{A}$ *and* $\mathbf{f}_i$ *from* $\mathbf{f}$ *;*
  Partition $\mathbf{A}^{(i)}$ *and* $\mathbf{f}_i$ *as in Section 3.2 and calculate* $\overline{S}_{PP}^{(i)}$*;*
  Partition $\mathbf{A}^{(i)}$ *as in* (3.27) *and calculate* $S_{BB}^{(i)}$*;*
end
Assemble global dual problem $\mathbf{F}\lambda = \mathbf{d}$ *as in section 3.3;*
Solve dual problem by PCG with preconditioner defined in 3.4;
Calculate primal solution $\widetilde{\mathbf{u}}_P$ *as in* (3.25)*;*
for $i \leftarrow 1$ **to** N **do** /* in parallel */
  calculate local solutions $\widetilde{\mathbf{u}}_{\mathrm{R},i}$ *as in* (3.26)*;*
end
assemble global `solution` *vector from primal and local solutions;*
return `solution`*;*

The aim of this thesis is to implement the given IETI-DP algorithm in the existing C++ framework of Bembel provided by [10].

4. Implementation

This chapter highlights the most important aspects of the implementation of IETI-DP into the Bembel framework. The used data structure for storing patch-wise information is specified and a class for computing the jump operator as in Section 2.5.2 is stated. Based on that, classes for parallel matrix assembly and the computation of the solution are presented and certain implementation details are highlighted. Pseudo-code is used for algorithms outside the given sketch in Section 3.5. Regarding the actual IETI-DP algorithm, special emphasis is placed on the local and global matrix structures that are obtained with Problem 3.2 and exploited for parallelism.

4.1. BlockMatrix Struct

As the first loop of the proposed algorithm in Section 3.5 computes various local information on every patch Γ_i in parallel, a data structure to store the resulting matrices and vectors is needed.

Field	Description	Type
`essentials`	Total number of essential primal DOFs on the patch.	`int`
`offset_ess`	Global index of the patch's first essential primal DOF in the global partition.	`int`
`offset_float`	Global index of the patch's first floating primal DOF in the global partition.	`int`
`BR_loc`	Constraints for the patch's remaing DOFs as in (3.9).	`Eigen::SparseMatrix<double>`
`A_RP`	Local $\overline{\mathbf{A}}_{\mathrm{RP}}$ as in (3.17).	`Eigen::SparseMatrix<double>`
`A_PR`	Local $\overline{\mathbf{A}}_{\mathrm{PR}}$ as in (3.16).	`Eigen::SparseMatrix<double>`
`permutations`	Local dual-primal and essential-floating partition.	`Eigen::PermutationMatrix`
`f_bar_i`	Entries of the local load vector as in (3.18).	`Eigen::VectorXd`
`triplets_M`	Local contribution to the dirichlet preconditioner.	`std::vector<Triplet>`
`triplets_F_i`	Local contribution to the dual system matrix.	`std::vector<Triplet>`

Table 4.1.: Fields of the used `BlockMatrix` struct with `typedef Triplet = Eigen::Triplet<double>`.

We use a simple struct called `BlockMatrix` with the fields listed in Table 4.1. Every patch Γ_i with $i = 0, \dots, N-1$ needs its own `BlockMatrix`-struct and every thread of the parallelization will have to work independently on

one of them. For that, all the structs are stored in a `std::vector<BlockMatrix>`. The resulting scheme can be retraced in Figure 4.1

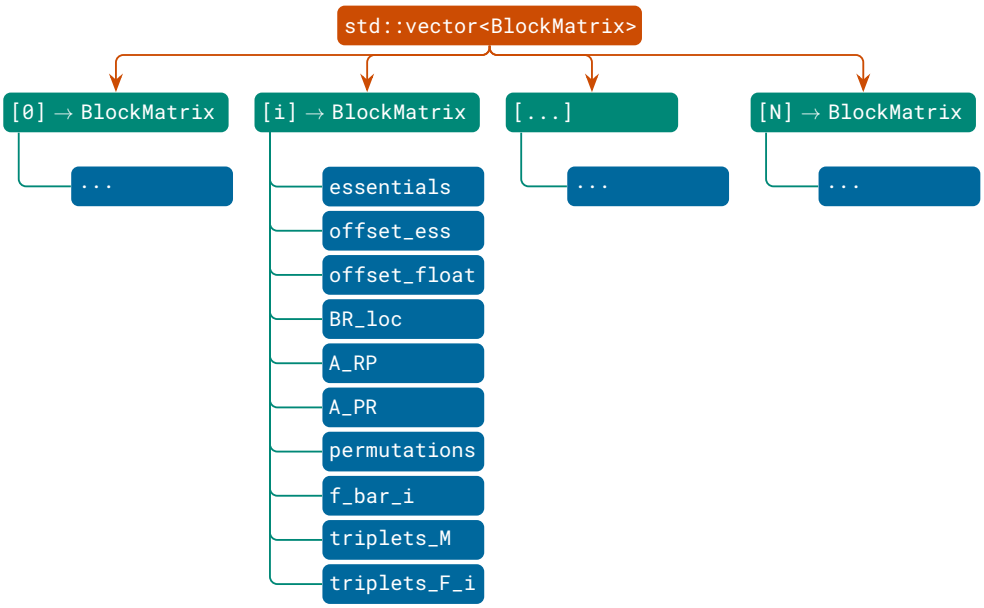


Figure 4.1.: Schematic of the used data-structure.

4.2. Dirichlet Data

In order to impose inhomogeneous Dirichlet conditions, we assume a given vector $\hat{\mathbf{g}}$, containing the coefficients in (2.50) at the indices corresponding to DOFs affected by a boundary constraint. A class `DirichletDataIETI` then computes an approximation to the vector $\widetilde{\mathbf{g}}_\Delta$ by sorting $\hat{\mathbf{g}}$ as in Section 3.2 and extracting as many elements as there are essential primal DOFs. The implications of this approach are discussed in Chapter 5. Regarding homogeneous Dirichlet boundary conditions, $\hat{\mathbf{g}}$ and $\widetilde{\mathbf{g}}_\Delta$ are obtained by initializing them as zero vectors.
Later on, the aglorithm needs to determine whether a DOF is essential primal or floating primal. As we have inspect all edges to find the ones on the boundary, a `std::vector<bool>` called `lookup` is computed alongside, in which we encode which DOFs are global primal. All entries are initialized with the boolean value 1 and updated to 0 if the corresponding DOF is associated with the global boundary. This is the case if the `edge[1]` contains the value -1. By doing this for every patch, all interior DOFs and DOFs on an interface between patches are still associated with the value 1. However, because the index of every primal DOF can be computed solely with the number of DOFs on the patch, `lookup` is then only evaluated for these primal indices.

4.3. Jump Operator Class

Concerning the local contributions to the dual problem, one first has to evaluate global information in order to assemble the jump operator stated in Section 2.5 as well as the matrix $\mathbf{D}$ in Section 3.4. Both need to consider the global information regarding the coupling of the patches via constraints. Therefore, a class

named `JumpOperator` is implemented to handle the assembly and manage those global operators.
The Bembel framework provides us with the method `patchTopologyInfo()` of the `ElementTree` class, which returns a `std::vector<std::array<int, 4>>` that encodes every edge case in an array of four integers we call `edge`:

`edge[0]` → patchnumber of primary patch

`edge[1]` → patchnumber of secondary patch

`edge[2]` → edge case on primary patch

`edge[3]` → edge case on secondary patch

Note that the resulting vector does not contain any duplicates.
Additionally, the function `GlueRoutines::getEdgeDofIndices()` returns the global index of all DOFs associated with an edge. Furthermore, the `ElementTree` class uses a tree structure consisting of `ElementTreeNode` instances to provide information on neighbourhood relations between patches. Each `ElementTreeNode` instance stores a patch number and pointers to the edge-wise adjacent patches.

Algorithm (Compute Jump Operator)**.**

```
   Data: ElementTree element_tree
   Result: Eigen::SparseMatrix B, Eigen::SparseMatrix D
 1 extract edges from element_tree;
 2 std::vector<Triplet> trip_rem, trip_prim, trip_prim_bndry;
 3 for each edge do
 4     if edge[1] == -1 then
 5         compute DOFs using GlueRoutines::getEdgeDofIndices;
 6         for each DOF do
 7             if DOF is primal then
 8                 if not in trip_prim_bndry then
 9                     trip_prim_bndry.push_back();
10                 end
11             else
12                 trip_rem.push_back();
13             end
14         end
15     else
16         for each remaining DOF do
17             trip_rem.push_back();
18         end
19         check constraints clockwise;
20         check constraints counter-clockwise;
21     end
22 end
23 concatenate triplet vectors;
24 B.setFromTriplets();
25 D.setFromTriplets();
```

Using these routines, the algorithm above for computing the jump operator and the matrix **D** is implemented. Here and at various other sequences of the implementation, a `std::vector<Triplet>` is used to store data for `Eigen::Sparse` matrices. This is done to increase performance, as suggested by [19]. When collecting the triplets for the jump operators non-zero entries, the routine has to inspect the already collected entries in order to prevent duplicates. With the aim to significantly decrease the number of triplets to inspect, three separate vectors are used. Since there are no duplicates in the edge vector, only primal DOFs can be identified multiple times by `getEdgeDofIndices()`. Therefore, one triplet vector is used to exclusively store the information regarding remaining DOFs. In almost any case, the majority of constraints is generated by these DOFs and this way no additional checking for duplicates is necessary for them.
Finding vertex conditions between patches that are not otherwise linked via an edge requires to iterate over neighbouring patches. This is done in clockwise and counter-clockwise direction for every edge that connects two patches. In the process, the number of connected patches needed for the multiplicities in the matrix **D** introduced in Section 3.4 can be tracked and then calculated after each search. The principle is exactly the same for both routines, only indices and switch cases differ between the procedures.

Algorithm (Check Constraints (Counter)-Clockwise)**.**

```
   Data: std::array<int, 4> edge
 1 determine primal_dof on starting patch;
 2 int next_patch = edge[1];
 3 int incoming_edge = edge[3];
 4 int multiplicity = 1;
 5 do
 6 |   ++multiplicity;
 7 |   determine primal_dof_2 on next_patch;
 8 |   if not in trip_prim then
 9 |   |   trip_prim.push_back();
10 |   update next_patch;
11 |   update incoming_edge;
12 while next patch != starting patch;
13 set multiplicities in D;
```

The variable `primal_dof` stores the index of the DOF on the starting patch for which the constraints are computed. Another variable `incoming_edge` encodes via which edge the algorithm stepped onto the patch it is currently on by using the edge-cases depicted in Figure 2.4. This information is also needed in order to find the next patch `next_patch` to scan, as well as to determine which DOF `primal_dof_2` is linked to `primal_dof` in order to obtain the correct constraint.

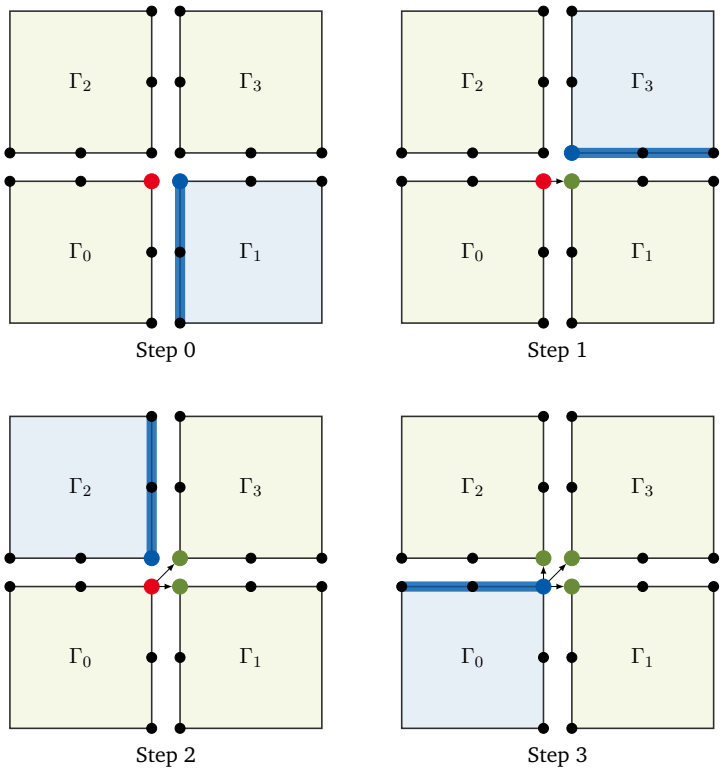


Figure 4.2.: Searching for constraints in counter-clockwise direction. With the last depicted step, the termination condition is reached.

Considering Figure 4.2, where a counter-clockwise search starting at the patch Γ_0 is performed, the procedure can be retraced. The `primal_dof` is marked in red, the `incoming_edge`, `primal_dof_2` as well as the `next_patch` in blue and detected constraints in green.
For the primal DOFs, one vector stores all the constraints with the boundary, while the other one stores constraints between primal DOFs. The constraints between patches can only be detected by the clockwise and counter-clockwise search routines above and both aren't executed at all when an edge is associated with the global boundary. This means the search routines can only inspect the vector with constraints between patches for duplicates, while the part of the algorithm for boundary contraints does so in a separate vector. At the end, the three triplet vectors are concatenated and the global operators are computed using `setFromTriplets()`.

4.4. Block Matrix Operator Class

In pursuance of storing the data structure presented in Section 4.1 and to execute the patch-wise calculations, we define the `BlockMatrixOperator` class. The core of the class is the `assembleLocalMatrices()` routine, which uses the `AnsatzSpace`, the global load vector $\mathbf{f}$, the vector $\widetilde{\mathbf{g}}_d$ and the `lookup` vector generated in Section 4.2 to compute all fields for every `BlockMatrix` struct. During the calculations, the structures of the matrices derived in Section 3 are exploited in order to realize large parts of the procedure in parallel.

Computing the Partitions

Before constructing the triplets and matrices, the algorithm computes all necessary indices and permutations. The partitions regarding primal and dual as well as interior and boundary DOFs in the fully matching setting are identical for every patch and therefore only need to be computed once. Both are directly linked to the number of DOFs on every patch and can therefore be computed by simply collecting indices with a single `for` loop.
In contrast, the partition in essential and floating primal DOFs is in general unique for every patch of the geometry. Because of that, these partitions are collected in parallel on every patch, storing each of them using a `Eigen::PermutationMatrix` in the field `permutations` of the corresponding `BlockMatrix` struct. In the process, the `lookup` vector is used to determine which DOFs are essential primal in the way described in Section 4.2.
Afterwards, two additional `Eigen::PermutationMatrix` named `perm_local` and `perm_global` are computed. The first one realises a permutation matrix that sorts all patches according to their individual dual-primal and essential-floating partition simultaneously, while the second one then sorts the resulting matrix or vector globally as in Section 3.2.

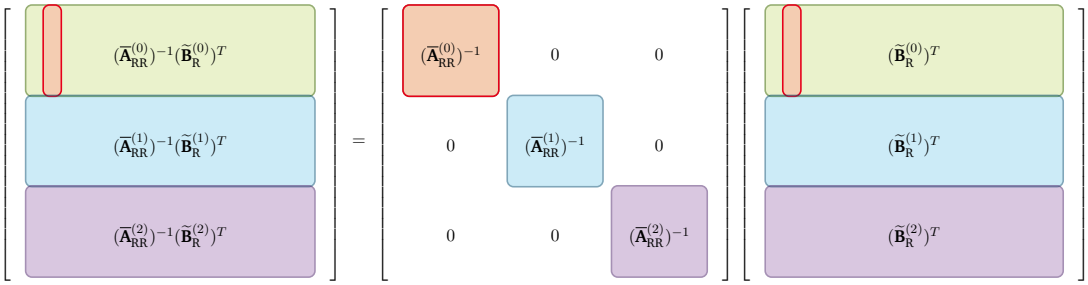
Local Contributions to the Dual Problem

Explicitly computing the dual problem's system matrix using the block factorization (3.3) yields

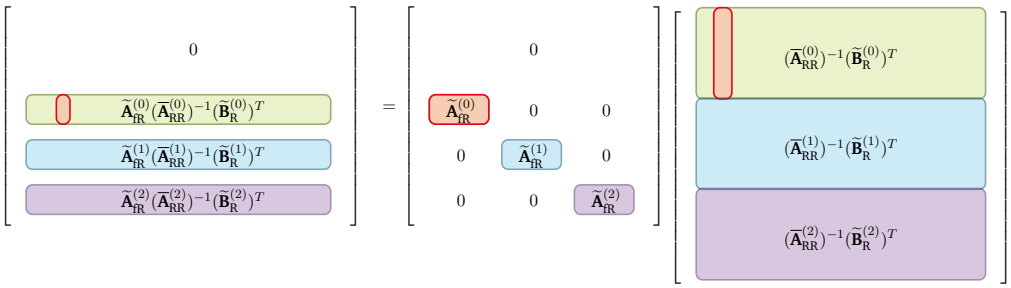
$$\mathbf{F} = \overline{\mathbf{B}} \begin{pmatrix} \overline{\mathbf{S}}_{\mathrm{PP}}^{-1} \left[\overline{\mathbf{B}}_{\mathrm{P}}^{T} - \overline{\mathbf{A}}_{\mathrm{PR}} \overline{\mathbf{A}}_{\mathrm{RR}}^{-1} \overline{\mathbf{B}}_{\mathrm{R}}^{T} \right] \\ -\overline{\mathbf{A}}_{\mathrm{RR}}^{-1} \overline{\mathbf{A}}_{\mathrm{RP}} \overline{\mathbf{S}}_{\mathrm{PP}}^{-1} \left[\overline{\mathbf{B}}_{\mathrm{P}}^{T} - \overline{\mathbf{A}}_{\mathrm{PR}} \overline{\mathbf{A}}_{\mathrm{RR}}^{-1} \overline{\mathbf{B}}_{\mathrm{R}}^{T} \right] + \overline{\mathbf{A}}_{\mathrm{RR}}^{-1} \overline{\mathbf{B}}_{\mathrm{R}}^{T} \end{pmatrix}. \tag{4.1}$$

For the term in brackets forming $\overline{\mathbf{A}}^{-1}\overline{\mathbf{B}}^{T}$, the structures of the matrices can be exploited in order to directly assemble the triplets for the global matrix independently on every patch. Every following step is therefore computed in parallel with each thread handling a single patch and the corresponding `BlockMatrix` struct.
At first, the local stiffness matrix and the complementary part of the load vector are extracted from the global system (2.38). The local permutation matrix stored in `permutations` is applied to sort both of them according to the patches unique dual-primal and essential-floating partition. This sorted stiffness matrix is then split up into the four independent matrices $\widetilde{\mathbf{A}}_{\mathrm{PP}}^{(i)}$, $\widetilde{\mathbf{A}}_{\mathrm{PR}}^{(i)}$, $\widetilde{\mathbf{A}}_{\mathrm{RP}}^{(i)}$ and $\widetilde{\mathbf{A}}_{\mathrm{RR}}^{(i)}$ as in Section 3.2. Since $\widetilde{\mathbf{A}}_{\mathrm{PP}}^{(i)}$ is always a four-by-four matrix, a dense `Eigen::MatrixXd` object is used, while the other matrices are stored using `Eigen::Sparse<double>` instances.
The entries of $\widetilde{\mathbf{A}}_{\mathrm{PP}}^{(i)}$, $\widetilde{\mathbf{A}}_{\mathrm{PR}}^{(i)}$ and $\widetilde{\mathbf{A}}_{\mathrm{RP}}^{(i)}$ that correspond to bilinear forms involving essential primal DOFs are then set to zero and one respectively. In the process, the local load vector is updated in order to obtain the structures of Problem 3.2 locally, yielding $\overline{\mathbf{A}}_{\mathrm{PP}}^{(i)}$, $\overline{\mathbf{A}}_{\mathrm{PR}}^{(i)}$ and $\overline{\mathbf{A}}_{\mathrm{RP}}^{(i)}$.
$\overline{\mathbf{A}}_{\mathrm{RR}}$ is then factorized using Eigen's built-in sparse LU decomposition. Using this factorization and the matrix $\overline{\mathbf{A}}_{\mathrm{RP}}$ as the right-hand side, the four-by-four matrix $\overline{\mathbf{S}}_{PP}^{(i)}$ as in (3.23) can be computed and inverted. The global $\overline{\mathbf{S}}_{PP}$ is a block diagonal matrix with as many ones as there are essential primal DOFs on the diagonal, followed by irregular sized blocks whose dimensions match the number of floating primal DOFs on each patch. Instead sorting this matrix and all the other global matrices and vectors in the sortation realized by the `perm_local` matrix leads to a patch-wise structure for the term in brackets in (4.1). This structure first lists all local primal DOFs sorted with respect to the essential-floating partition and then all the remaining DOFs patch-wise. In conclusion, computing the triplets for the matrix product $\overline{\mathbf{A}}^{-1}\overline{\mathbf{B}}^{T}$ in parallel is achieved by first assembling it using the sortation realized by the `perm_local` matrix and then sorting the assembled matrix using `perm_global`.

This is realized by first sorting the jump- or mortar operator using `perm_local`. Each thread then extracts the matrices $(\overline{\mathbf{B}}_{\mathrm{R}}^{(i)})^T$ and $(\overline{\mathbf{B}}_{\mathrm{P}}^{(i)})^T$ corresponding to its patch. The local contribution to the product $\overline{\mathbf{A}}_{\mathrm{PR}}\overline{\mathbf{A}}_{\mathrm{RR}}^{-1}\overline{\mathbf{B}}_{\mathrm{R}}^T$ is then computed in two steps in order to store the interim result marked orange in (4.1).
Note that $(\widetilde{\mathbf{B}}_{\mathrm{R}}^{(i)})^T = (\overline{\mathbf{B}}_{\mathrm{R}}^{(i)})^T$ is a sparse matrix and only the columns corresponding to constraints that apply to the current patch have non-zero values. Because of this, the dimension of $\overline{\mathbf{A}}_{\mathrm{RR}}$ grows faster than the number of these constraints. Especially when considering a high refinement level, the number of systems solved with the LU-factorization of $\overline{\mathbf{A}}_{\mathrm{RR}}$ is therefore smaller than the number of columns in $(\widetilde{\mathbf{B}}_{\mathrm{R}}^{(i)})^T$ or $\overline{\mathbf{A}}_{\mathrm{RR}}^{-1}$. As $(\widetilde{\mathbf{B}}_{\mathrm{R}}^{(i)})^T$ is stored using the column-major storage scheme, the built-in method `col()` can be used to read and write the column marked in red in Figure 4.3a. Using these informations, one can iterate over the columns of $(\widetilde{\mathbf{B}}_{\mathrm{R}}^{(i)})^T$ containing non-zero entries and use each of them as an input vector for the LU-decompositions `solve()` method. The result can then be written back into the same row, yielding the matrix product depicted in Figure 4.3a.



(a) Column-wise solution via LU-decomposition.



(b) Matrix product leads to entry.

Figure 4.3.: Steps for computing the term $\overline{\mathbf{A}}_{\mathrm{PR}}\overline{\mathbf{A}}_{\mathrm{RR}}^{-1}\overline{\mathbf{B}}_{\mathrm{R}}^T$.

The resulting columns containing non-zeros are then multiplied by the local $\overline{\mathbf{A}}_{\mathrm{PR}}$, yielding a matrix as in Figure 4.3b. On a patch, only the marked strip with as many rows as there are floating primal DOFs on the associated patch is computed. This matrix is then subtracted from the rows of the local $(\overline{\mathbf{B}}_{\mathrm{P}}^{(i)})^T$ that correspond to the local floating primal DOFs. After that, the inverse of $\overline{\mathbf{S}}_{PP}^{(i)}$ is used to compute the end result for the first line marked blue in (4.1). This result can then be reused to compute the second row of the matrix. For both, a helper routine is used to extract the triplets of the local sparse matrices and equip them with an offset for their row and column value. Using the offset fields of the struct, this leads to triplets that can later be used to directly assemble the global matrix $\overline{\mathbf{A}}^{-1}\overline{\mathbf{B}}^T$ after applying the global permutation matrix `perm_global`.
Using the same approach as above, part of the dual problem's right-hand side

$$\mathbf{d} = \overline{\mathbf{B}} \begin{pmatrix} \overline{\mathbf{S}}_{\mathrm{PP}}^{-1}\left[\overline{\mathbf{f}}_{\mathrm{P}} - \overline{\mathbf{A}}_{\mathrm{PR}}\overline{\mathbf{A}}_{\mathrm{RR}}^{-1}\overline{\mathbf{f}}_{\mathrm{R}}\right] \\ -\overline{\mathbf{A}}_{\mathrm{RR}}^{-1}\overline{\mathbf{A}}_{\mathrm{RP}}\overline{\mathbf{S}}_{\mathrm{PP}}^{-1}\left[\overline{\mathbf{f}}_{\mathrm{P}} - \overline{\mathbf{A}}_{\mathrm{PR}}\overline{\mathbf{A}}_{\mathrm{RR}}^{-1}\overline{\mathbf{f}}_{\mathrm{R}}\right] + \overline{\mathbf{A}}_{\mathrm{RR}}^{-1}\overline{\mathbf{f}}_{\mathrm{R}} \end{pmatrix} - \overline{\mathbf{b}}\,, \tag{4.2}$$

can be computed using the same steps as for the system matrix. This time however, the calculations result in vectors, which means only a single linear system has to be solved with the LU factorizations each time. The number of DOFs per patch is known in advance, therefore the term in brackets in (4.2) is directly assembled in a single vector. As before, the result is sorted according to the patch-wise dual-primal and essential-floating partition of `perm_local`.
It is important to note that the current implementation is not able to handle geometries with all-floating patches, meaning patches without any essential primal DOFs. If this is the case, no information is added to the corresponding stiffness matrix, leaving it in the state of Section 2.5.1. A block in matrix (2.38) corresponding to an all-floating patch is therefore not invertible, invalidating all further derivation of Section 3.2. Every patch in the implementation presented with this thesis needs at least one essential primal DOF, as the kernel of each local stiffness matrix has dimension one.

4.5. IETI-DP Routine

Solving the Dual Problem

After using the classes defined in the sections before, we need to assemble the global matrices for the dual Problem 3.3 from the local information of the `BlockMatrix` structs. All the local vectors of triplets can be directly inserted into the global matrix without changing any of their data. To avoid copying all of them into a global vector for the `setfromTriplets()` routine, a custom `forward_iterator` is implemented for the `BlockMatrixOperator`. This iterator additionally stores a patch index and a pointer to a `std::vector<BlockMatrix>` as in Figure 4.1. When incrementing the iterator past the last address of a patch's `triplets_F_i` vector, it is determined whether the patch index corresponds to the number of entries in the `std::vector<BlockMatrix>`. If this is not the case, the patch index is increased by one and the pointer is set to the first address of the next patch's `triplets_F_i` vector. By doing this and applying the global permuation matrix, the global matrix $\overline{\mathbf{A}}^{-1}\overline{\mathbf{B}}^{T}$ can be assembled from the `BlockMatrix` structs with a single line of code. The vector containing local contributions to $\mathbf{d}$ is then also sorted globally. When using the jump operator, the vector $\mathbf{b}$ is obtained as noted in [24, p. 205] via

$$\mathbf{B}_J\hat{\mathbf{g}}\,, \tag{4.3}$$

and $\overline{\mathbf{b}}$ is then computed via (3.15). With the operator $\overline{\mathbf{B}}$, the dual Problem 3.3 is set up and solved using Eigen's implementation of a CG algorithm for sparse systems.

Preconditioner

The contributions to the preconditioner matrix $\mathbf{M}^{-1}$ are computed locally as shown in Section 3.4 and saved as a list of triplets for each patch. Implementing another iterator for the `BlockMatrixOperator`, the preconditioner is assembled in the same way as the matrix product $\overline{\mathbf{A}}^{-1}\overline{\mathbf{B}}^{T}$ for the system matrix $\mathbf{F}$ of the dual problem. In this case, no sortations need to be considered. The `setfromTriplets()` method has a third optional argument in the form of a functor, which is used when duplicates are met during the assembly. With a simple lambda function, the needed summation in (3.28) is implemented.

Assembling the Solution

The last part of the IETI-DP routine computes the DOFs of the original Problem 3.2 using the solution λ of the dual problem. Contrary to the algorithm proposed by [24], this thesis pursues a completely parallel approach for assembling the vector $\widetilde{\mathbf{u}}$. This is again achieved by adhering to the sortation implied by `perm_local`.
The number of essential DOFs on a patch stored in the `essentials` field is used to compute the matrix-vector $\overline{\mathbf{B}}_{\mathrm{P}}^{T}\lambda$ product in (3.25) patch-wise. This is achieved by extracting the corresponing columns using Eigen's `middleCols()` method. All other matrices and vectors needed for $\widetilde{\mathbf{u}}_{\mathrm{P}}$ as in (3.25) are stored in the `BlockMatrix` struct. After aquiring the solution for a patch's primal DOFs, the remaining DOFs (3.26) are computed within the same parallel loop. The resulting global vector of DOFs is therefore sorted with the local partition. This is reversed in a last step by applying the transpose of the corresponding global permutation matrix, yielding the solution vector $\mathbf{u}$.

4.6. Mortaring and Local Refinement

Already indicated by [24, Rem. 5.1], the jump operator can be replaced by the mortar operator of Section 2.5.3, yielding Problem 2.12. The implementation by [28] also provides a routine to assemble the corresponding right-hand side $\mathbf{b}_{\mathrm{M}}$. The resulting vector is used instead of (4.3) to assemble the dual problem.
The mortar operator is not limited to the fully matching setting we introduced with the jump operator, which opens the possibility for patch-wise refinement. In other words, distinct patches can now posses different refinement levels. In order to realize this, the struct illustrated in Figure 4.1 has to store information regarding the patchs refinement and code of the `BlockMatrixOperator` and has to be altered slightly. Where we before computed the dual-primal and interior-boundary partition once, we now have to do so for every patch. In addition to that, important information like the number of DOF on every patch and the index of the first remaining DOF in the global indexing are not as trivial as before.
Those information are computed in an additional `for` loop, which is executed at the begining of the `assembleGlobalMatrices()` routine. By doing this, the dual-primal and interior-boundary partitions can be computed before the essential-floating partition within the same parallel loop.

5. Results

In the following, the implementation of IETI-DP is tested using three exemplary geometries of increasing complexity, which are constructed with the Octave [13] nurbs package [33]. The examples are chosen in a way that an analytical solution is known, which enables us to compute a reference solution u analytically and then compare it to our numerical results u_h using the relative L^2-error

$$\epsilon_{\text{rel}} = \frac{||u_h - u||_{L^2(\Gamma)}}{||u||_{L^2(\Gamma)}} \,. \tag{5.1}$$

However, because the solution is represented by the vector $\mathbf{u}$, we need the L^2-projection of the reference solution $\mathbf{u}_{\text{ref}}$ onto the same space as $\mathbf{u}$. The projection onto the $\mathcal{N}$-dimensional space $V_{h,p}$ is achieved as in [28] by using the mass matrix

$$\mathbf{M} \coloneqq \Big(\int_\Gamma \varphi_i \varphi_j \mathrm{d}\sigma \Big)_{1 \le i,j \le \mathcal{N}} \,, \quad \varphi_i, \varphi_j \in V_{h,p} \,, \tag{5.2}$$

with a right-hand side computed with the reference solution

$$\mathbf{f} \coloneqq \Big(\int_\Gamma \varphi_i u \mathrm{d}\sigma \Big)_{1 \le i \le \mathcal{N}} \,, \quad \varphi_i \in V_{h,p} \,, \tag{5.3}$$

which leads to a linear system

$$\mathbf{M}\mathbf{u}_{\text{ref}} = \mathbf{f} \,. \tag{5.4}$$

Solving this system yields a vector $\mathbf{u}_{\text{ref}}$ that can be compared to the solution vector $\mathbf{u}$ of the IETI-DP routine. We therefore compute

$$\epsilon_{\text{rel}} = \sqrt{\frac{(\mathbf{u} - \mathbf{u}_{\text{ref}})^T \mathbf{M} (\mathbf{u} - \mathbf{u}_{\text{ref}})}{\mathbf{u}_{\text{ref}}^T \mathbf{M} \mathbf{u}_{\text{ref}}}} \,. \tag{5.5}$$

Additionally, because two variants of coupling matrices are available, we again indicate everything related to the mortar operator by the subscript M and for the jump operator by J. If not stated otherwise, the solution of every following setting is computed using first- to fifth-order B-splines as basis functions and a fixed global refinement level m on every patch. This setting therefore also implies a global element width h. The refinement level is then incremented multiple times for every considered order. With the theoretical work by Bazilevs, da Veiga, Cottrell, Hughes and Sangalli [3], we expect higher refinement levels to yield lower relative errors and that the convergence rate of this error increases with the used degree of B-spline basis functions. Additionally, the results by [28, Sec. 5.1] are used as a reference, where the problem 2.12 was solved for various geometries using a direct method.
At the date of submission of this thesis, the Eigen library does not support the implemented preconditioner. Subsequently, all data is generated without preconditioning of the dual system.

5.1. Quad Patch Geometry

Homogeneous Dirichlet Data

In a first experiment, we investigate the validity of the given implementation with a simple example. We choose a square

$$\Gamma \coloneqq \left\{ \boldsymbol{x} \in \mathbb{R}^3 \colon 0 \leq x_1, x_2 \leq 2, x_3 = 0 \right\} , \tag{5.6}$$

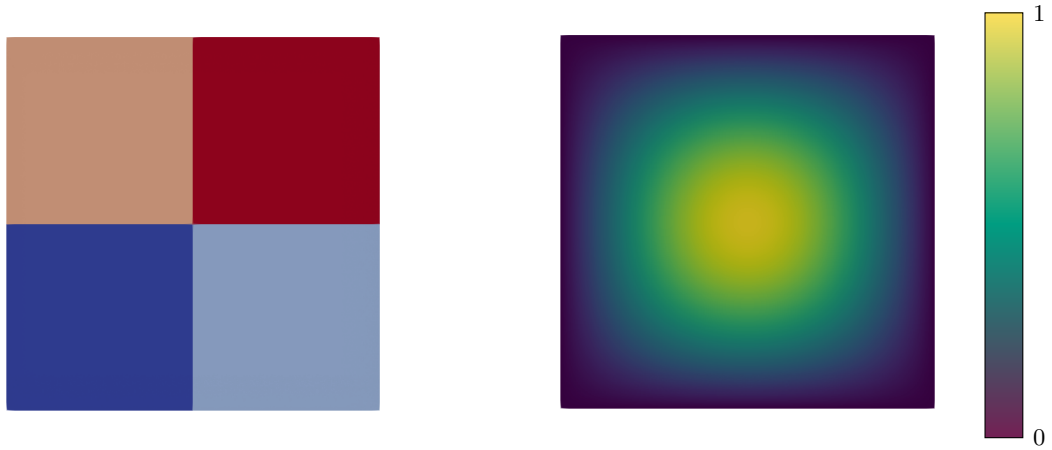
that is made up of four shifted unit square patches. Since this example geometry has no curvature, Problem 2.5 is equivalent to solving Poisson's equation in two dimensions. We choose an exemplary solution $u(\boldsymbol{x})$ that can easily be treated analytically with

$$u(\boldsymbol{x}) = \sin(\pi \frac{x_1}{2}) \sin(\pi \frac{x_2}{2}) . \tag{5.7}$$

The right-hand side is then be computed via

$$-\Delta_\Gamma u(\boldsymbol{x}) = -\Delta u(\boldsymbol{x}) = \frac{1}{2}\pi^2 \sin(\pi \frac{x_1}{2}) \sin(\pi \frac{x_2}{2}) . \tag{5.8}$$

This way, a setting with homogeneous Dirichlet conditions is achieved. A plot of the reference solution and the subdivision of the domain Γ into patches is depicted in Figure 5.1.



(a) Patches of the quad patch geometry.

(b) Reference solution (5.7) on the geometry.

Figure 5.1.: Quad patch geometry with reference solution.

Using the same ansatz space, the reference solution is obtained as in (5.4) and a relative error is computed via (5.5). The following plots display the results using the jump operator to enforce the constraints between patches and for the boundary.

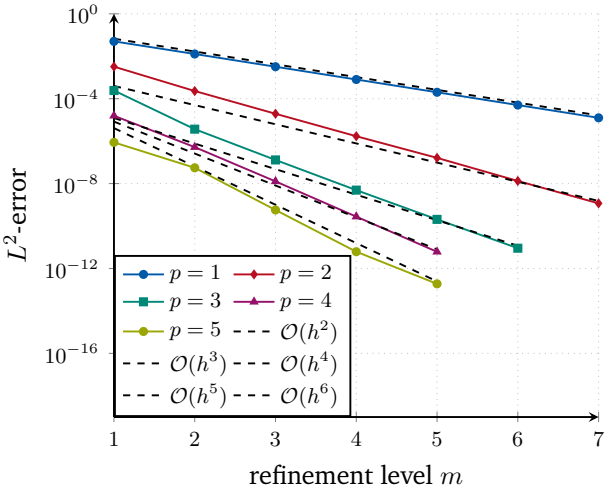


Figure 5.2.: Convergence rate of Laplace-Beltrami problem without curvature and homogeneous Dirichlet data on a four-patch geometry. Continuity constraints between patches and with the boundary are imposed by the jump operator.

All data can be examined in Appendix B. The convergence rates observed via Figure 5.2 meet our expectations, as they resemble the results by [3] and [28].
The refinement levels not shown in the plot either lead to the machine in use to run out of memory or the results do not continue the indicated convergence rates. We assume the condition number of the unpreconditioned dual system matrix to be the reason for this behavior. The results by [24, Sec. 6] showed a fast increase of this number with further refinement and [31] showed that the condition number of the underlying stiffness matrix increases with the polynomial degree of the used basis functions.
The relative error was computed for the jump operator as well as mortaring, but the results in this homogeneous setting do not differ for most cases, at least up to the sixth significant figure. Only the cases that do not show the described convergence and one case with a significantly deviant iteration number are exceptions. In general, the computations using the mortar operator needed significantly more iterations for solving the dual problem. The concrete numbers can be retraced in Table B.1. This indicates worse condition numbers for the dual system matrix obtained with the mortar operator implemented by [28] in the fully matching setting.

5.2. Quarter Sphere Geometry

Inhomogeneous Dirichlet Data

Continuing with a more complex case, curvature is added to a six-patch geometry to achieve a quarter of a sphere

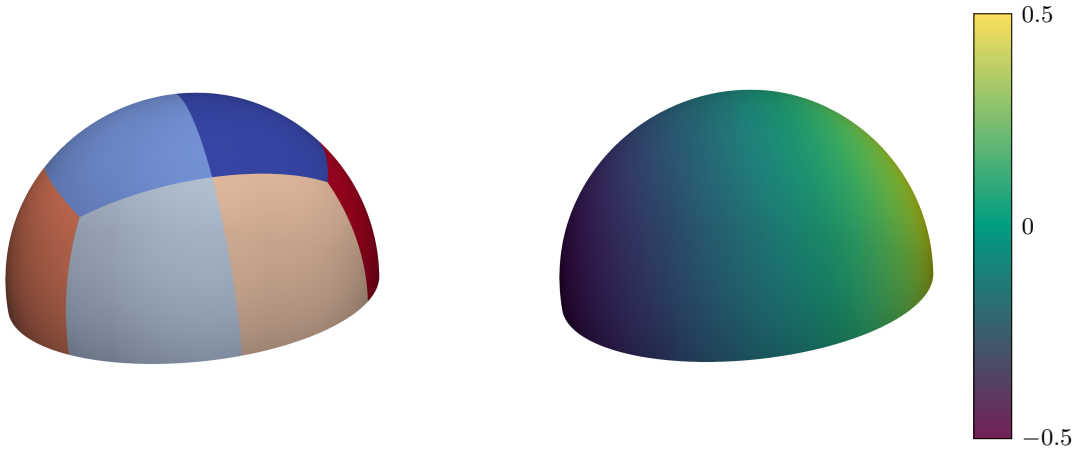
$$\Gamma := \left\{ \boldsymbol{x} \in \mathbb{R}^3 \colon ||\boldsymbol{x}||_2 = 1,\, z \geq 0,\, y \leq 0 \right\} . \tag{5.9}$$

With [2, Sec. 3.3], we know that the Laplace-Beltrami operator on the unit sphere has the spherical harmonics as eigenfunctions. Therefore, a possible analytical solution of Problem 2.5 has the form

$$u(\boldsymbol{x}) = Y_{1,1}(\boldsymbol{x}) := \sqrt{\frac{3}{4\pi}} x_1 \, , \tag{5.10}$$

if inhomogeneous Dirichlet boundary conditions are used. The corresponding right-hand side is again computed analytically, yielding

$$-\Delta_\Gamma u(\boldsymbol{x}) = \sqrt{\frac{3}{\pi}} x_1 = f(\boldsymbol{x}) \ . \tag{5.11}$$



(a) Patches of the quarter sphere geometry.

(b) Reference solution (5.10) on the geometry.

Figure 5.3.: Quarter sphere geometry with reference solution.

The subdivision of the quarter sphere into six patches is depicted in Figure 5.3a and Figure 5.3b shows the analytical reference solution on the geometry.
Regarding the boundary data, we use the reference solution $\mathbf{u}_{\text{ref}}$ obtained via (5.4) with the approach depicted in Section 4.2. The vector $\hat{\mathbf{g}}$ is computed by copying the entries of the reference solution at the indices of DOFs associated with the boundary into a new vector. All other entries are initialzed as zero.
It is important to note that this does not agree with the vectors presumed by [24] and merely corresponds to an approximation of the boundary data. Additionally, the vector $\mathbf{u}_{\text{ref}}$ does not adhere to the matching of DOFs along edges enforced by the jump operator, because the mass matrix does not incorporate this continuity between patches. Theoretically, this is problematic when considering essential primal DOFs that are additionally linked via a condition resulting from an interface edge. However, the right-hand side is obtained via (4.3), which leads to the difference between the essential primal DOFs being set accordingly.
Moreover, the right-hand side for the mortar operator (2.53) is provided by [28]. Regarding homogeneous Dirichlet boundary conditions, the said vector is a zero vector and therefore agrees with Section 4.2. On the contrary, this leads to divergent boundary data when considering an inhomogeneous setting with the mortar and jump operator. For this reason we expect a more significant deviation between the results of problem 2.12 and Problem 2.11 when inhomogeneous Dirichlet boundary conditions as in Section 4.2 are used. However, the resulting convergence rates should not be fundamentally different.

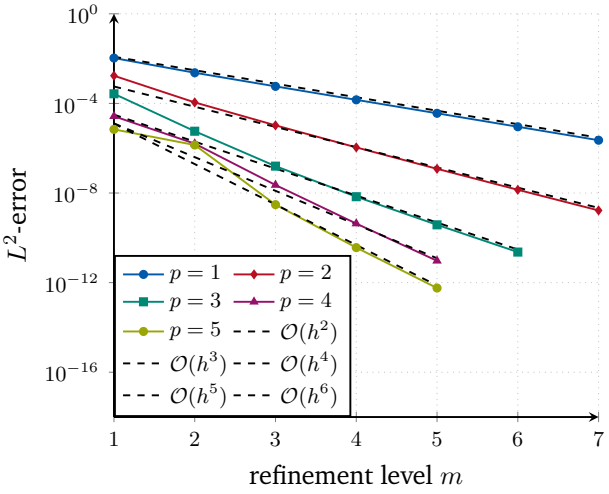


Figure 5.4.: Convergence rate of Laplace-Beltrami problem with curvature and inhomogeneous Dirichlet data on the quarter sphere geometry. Continuity constraints between patches and with the boundary are imposed by the jump operator.

The resulting convergence rates in Figure 5.4 resemble the rates of the homogeneous example before. This indicates that the incorporation of the essential boundary conditions in Section 3.2 is correctly implemented with the approach depicted in Section 4.4. The used Dirichlet data leads to the expected deviation between the two operators. However, the dataset in Table B.2 shows that the convergence rates are indeed similar.

Parallelization

In order to examine the parallelism of the algorithm each refinement step is executed twice, once with the loops over all patches in `assembleLocalMatrices()` and `IETI` in parallel and once sequential. It is necessary not to completely turn of multithreading in the second case because Eigen and Bembel themselves provide support for parallelism, what would therefore distort the result. For the jump operator, the runtime of the IETI-DP routine is measured and used to compute the ratio of parallel runtime t_{mult} to the single-threaded one t_{single}. The utilized machine uses an AMD Ryzen™ 5800H chip [1] with 8 cores.
Since we specifically want to investigate the parallel runtime advantage of the IETI-DP routine depicted in this thesis, we also measured the runtime for the global stiffness matrix assembly inside the `IETI` class separately and subtracted it from both parallel and sequential runtime.

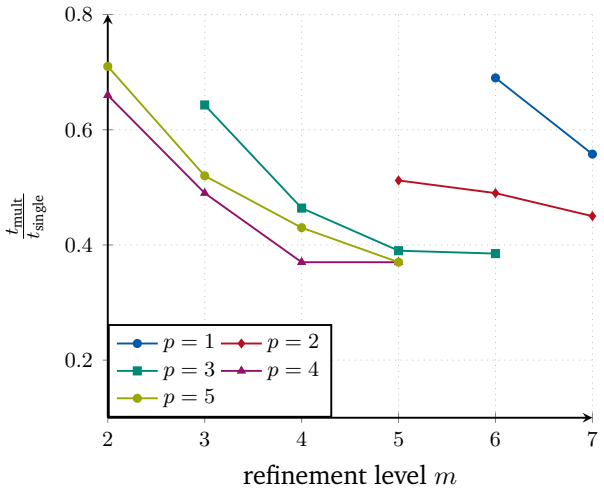


Figure 5.5.: Ratio of parallel to sequential runtime of Laplace-Beltrami problem with curvature and inhomogeneous Dirichlet data on quarter sphere geometry.

The resulting data is displayed in Figure 5.5 and shows a significant runtime advantage of the implementation when multithreading is enabled. This confirms the highly parallelizable nature of the approach presented by the authors of [24]. For the excluded cases, the total runtime is either smaller than 1 ms or the resulting ratio varies too distinctly between multiple runs to achieve a dependable average.

Patch-Wise Refinement

With the knowledge of Section 4.6, the mortar operator can be used in a patch-wise refined setting. This experiment solely aims to confirm the functionality of this approach, which is why we use the same geometry as before and add additional refinement steps to certain patches.

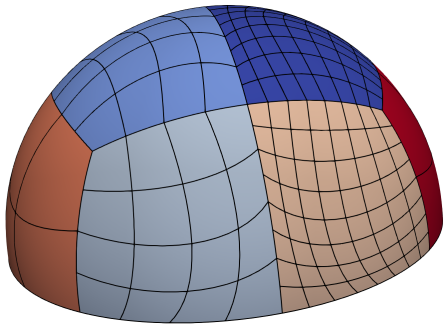


Figure 5.6.: Elements of the quarter sphere geometry with exemplary patch-wise refinement.

As it is shown Figure 5.6, a pair of two patches of the quarter sphere geometry shares one of three possible refinements m_a, $m_\mathrm{b} = m_\mathrm{a} + 1$ and $m_\mathrm{c} = m_\mathrm{a} + 2$ with the resulting element widths h_a, h_b and h_c. It is reported by [24, Sec. 6] that this setting leads to a larger condition number for the dual system matrix. We therefore limit this setting to first- and second-order polynomials, as the experiments before revealed that higher orders stop converging at lower refinement levels. The results are shown in Figure 5.7, where the number on the abscissa now describes the lowest refinement used on the geometry.

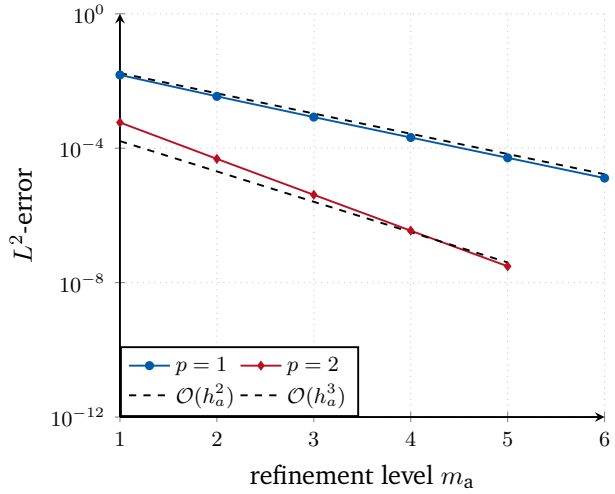
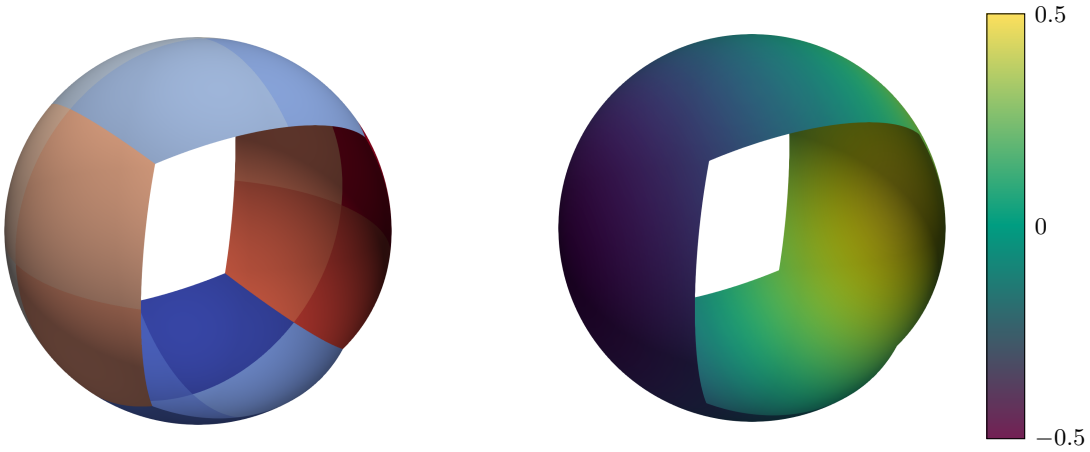


Figure 5.7.: Convergence rate of Laplace-Beltrami problem with curvature and inhomogeneous Dirichlet data on the quarter sphere geometry using patch-wise refinement. Continuity constraints between patches and with the boundary are imposed by the mortar operator.

The indicated convergence rates are the same as in the settings before. This meets our expectations, because we incremented the refinement level of every patch by one in every step and used B-splines of the same order on all patches. These observations confirm the functionality of the updates to IETI-DP described in Section 4.6 in a setting that is not fully matching.

5.3. Sphere Geometry

Lastly, a sphere geometry with two holes is examined in order to further test the scalability of the implementation. The geometry consists of 16 patches and is depicted in Figure 5.8. Because this geometry is again part of a unit-sphere, the spherical harmonic (5.10) with right-hand side (5.11) is still valid if inhomogeneous Dirichlet conditions are used. For this, we use the same approach as for the quarter sphere geometry, this time plotting the results obtained with the mortar operator.



(a) Patches of the sphere geometry.

(b) Reference solution (5.10) on the geometry.

Figure 5.8.: Sphere geometry with reference solution.

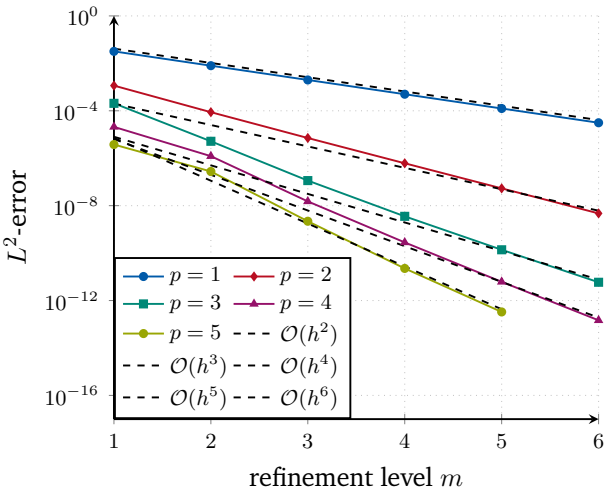


Figure 5.9.: Convergence rate of Laplace-Beltrami problem with curvature and inhomogeneous Dirichlet data on the sphere geometry. Continuity constraints between patches and with the boundary are imposed by the mortar operator.

With the mortar operator, the convergence rates depicted in Figure 5.9 agree with the results we obtained with the settings before. The jump operator produces an outlier regarding the iterations to solve the dual problem and the relative error. The data can be observed in Table B.3.
As a last experiment, we solve the given problem with fixed polynomial degree and refinement, but with an increasing number of available threads. Although the machine in use supports hyper-threading with up to 16 threads, [14] strongly advises against its use. This is why the number of threads is limited to the advised number of available cores, namely eight.

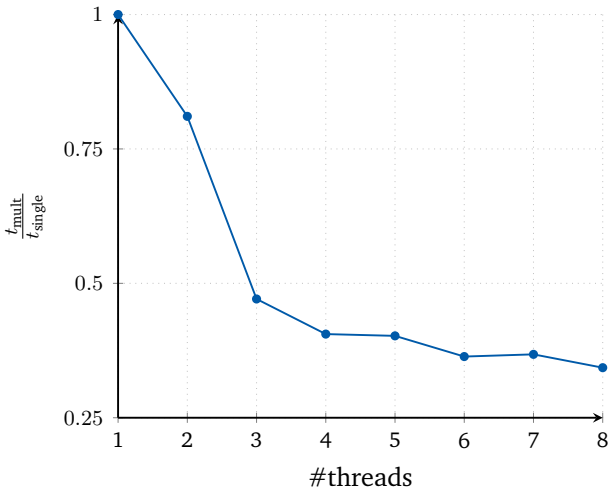


Figure 5.10.: Ratio of parallel to sequential runtime of Laplace-Beltrami problem on the sphere geometry with increasing number of available threads.

We choose a setting with third-order B-splines and refinement level five. As before, the ratio of parallel to sequential runtime is computed. The results presented in Figure 5.10 further validate the parallelism of the implemented algorithm.

6. Final Remarks

In this final chapter, the results of this bachelor thesis are summarized and we propose some suggestions to follow up this work.

6.1. Summary

We discussed the analysis for solving PDEs stated on curved surfaces using a finite element method. A multi-patch NURBS representation of this surface led to an isogeometric domain decomposition approach, resulting in a saddle point problem in the form of Problem 2.11. In Chapter 2 we then retraced the dual-primal and essential-floating partition of the resulting matrices and vectors in order to state the IETI-DP algorithm.

With Chapter 4, we presented an implementation of the jump operator introduced by [24] that minimizes the computational cost of checking for duplicates when assembling the constraints for C^0-continuity between the subdomains in a domain decomposition context. This led to a sparse matrix that can be utilized as a coupling operator in a fully matching setting for Problem 2.11.

Regarding the actual IETI-DP algorithm, we introduced a data structure in Section 4.1 that is suitable for parallelization of the algorithm. Using this, we implemented the IETI-DP algorithm into the Bembel framework [10]. The provided routine uses sparse matrices whenever useful and apart from two cheap sparse matrix multiplications, assembles the dual problem completely in parallel from local contributions and without unnecessary copying. With the solution vector of the dual problem, the DOFs of the original problem are assembled completely in parallel as well, only requiring sorting via a single sparse matrix-vector multiplication. Problem 2.11 is thereby solved without ever assembling the full system. The largest linear subsystem the IETI-DP algorithm solves is either bound by the number of constraints or the maximum number of remaining DOFs on a single patch.

We were also able to replace the jump operator by the mortar operator implemented by [28], which enabled patch-wise refined settings.

The algorithms functionality was then verified in settings with curvature, inhomogeneous Dirichlet boundary conditions and patch-wise refinement. With both operators, the observed convergence rates met expectations based on papers such as [3] and the preceding work by [28]. We also examined the runtime advantage of the implemented algorithm resulting from multithreading, which showed a significant effect even for the relatively small examples considered in this work.

6.2. Further Work

In this section, limitations that the submitted implementation and the method itself showed during testing are discussed. Additionally, fixes as well as future extensions that went beyond the scope of this bachelor thesis are proposed and some already developed ideas are sketched.

- At the date of submission, the current version of the implementation has to factorize the local matrix $\overline{\mathbf{A}}_{\mathrm{RR}}$ twice, once during the computations in `assembleLocalMatrices()` described in Section 4.4 and a second time during the assembly of the DOFs during execution of the `IETI` routine of Section 4.5.
- The marked entry in Table B.1 stands out with a significantly higher number of iterations when using the jump operator. This was the only case for which the relative errors of the operators differed when using homogeneous Dirichlet data. Regarding inhomogeneous conditions, the outlier in Table B.3 additionally led to a clearly deviating relative error. As opposed to the jump operator, the number of iterations and the relative error of the mortar operator is plausible in both cases.
- Another problem arises when considering geometries with patches that are only connected to the global boundary via vertex conditions. These are not considered by the `patchTopologyInfo()` routine and the corresponding entries in the `lookup` vector are not set correctly in the process shown in Section 4.2. This can be fixed in the future by using an approach similar to the one for the vertex conditions of the jump operator in Section 4.3. After evaluating all the edges on the global boundary, the routine would need to execute the clockwise and counter-clockwise search and afterwards add a boundary condition for every found DOF if the starting DOF `primal_dof` is affected by a boundary constraint.
- Additional measurements are required for the runtime-ratios in Table B.2 and Table B.5 in order to make them scientifically valid. The submissed version of this thesis took an average of three measurements.
- A possible approach for all-floating patches is proposed by [20], where additional conditions are introduced into the dual problem of Section 3.3 using a pseudo-inverse of the local stiffness matrix and a matrix representing its null space. This yields a saddle point problem for the dual problem.
- The class `DirichletDataIETI` needs to be extended in order to compute the Dirichlet boundary data presumed by [24]. The vector $\widetilde{\mathbf{g}}_{\mathrm{d}}$ can be obtained by determining the corner of the domain $\square$ that corresponds to each essential primal DOF. Bembel provides the method `map2surface()` of the `SuperSpace` class, which can be used to evaluate the function g_D at the point this corner is mapped to in the physical domain. Regarding the coefficients $g_{\mathbf{k},i}$ in (2.50), a separate routine would need to solve an approximation problem on the boundary using the B-spline basis on that patch.
- The computation of the dual system matrix $\mathbf{F}$ in (4.1) can be achieved completely in parallel by also sorting the matrix $\overline{\mathbf{B}}$ with the sortation implied by `perm_local`. However, this would lead to a rearranged version of $\mathbf{F}$, which has implications on the following computations.

A. Non-Uniform-Rational-B-Splines

As a polynomial curve, the introduced B-spline cannot represent conical shapes. These are needed in a variety of common geometries and should therefore be representable in the geometry mapping.
In order to overcome this, one has to work with rational functions [6, Sec. 3.6]. Hence, the rational extension of B-splines, in the form of NURBS are introduced. They allow to represent conic segments while maintaining most of the B-splines properties. For additional information on these properties, please refer to [30, Chap. 4].
Given an open knot vector, we can define n B-splines according to Definition 2.6. A strictly positive weight $w_i \in \mathbb{R}^{>0}$ is assigned with each of them, collecting those weights in a vector $W \coloneqq (w_1, \dots, w_n)$.

Definition A.1 (Non-Uniform Rational B-Splines, [6, Def. 9])**.** The n NURBS basis functions $R^{(p)}_{i,\Xi,W}$, related to the knot sequence Ξ and the weight-vector W, are given by

$$R^{(p)}_{i,\Xi,W}(x) \coloneqq \frac{w_i N^{(p)}_{i,\Xi}(x)}{\sum_{j=1}^{n} w_j N^{(p)}_{j,\Xi}(x)} \,. \tag{A.1}$$

As with the B-splines in Definition 2.9 with two given knot vectors, one can construct tensor-product NURBS surfaces

$$\boldsymbol{F}(\square) \coloneqq \sum_{i=1}^{n_1}\sum_{j=1}^{n_2} \mathbf{C}_{i,j} \frac{w_{i,j} N^{(p_1)}_{i,\Xi}(x_1) N^{(p_2)}_{j,\Upsilon}(x_2)}{\sum_{s=1}^{n_1}\sum_{r=1}^{n_2} w_{s,r} N^{(p_1)}_{s,\Xi}(x_1) N^{(p_2)}_{r,\Upsilon}(x_2)} \,, \quad w_{i,j} > 0,\ \mathbf{C}_{i,j} \in \mathbb{R}^d. \tag{A.2}$$

A step-by-step exemplary construction of a NURBS geometry can be found in [8, Sec. 2.4].

B. Data

	$p = 1$						
#DOF	36	100	324	1156	4356	16900	66564
$\#\lambda_{\mathrm{J}}$	34	58	106	202	394	778	1546
$\#iter_{\mathrm{J}}$	1	5	11	27	43	61	87
$\#\lambda_{\mathrm{M}}$	12	36	84	180	372	756	1524
$\#iter_{\mathrm{M}}$	1	6	14	38	107	196	270
ϵ_{rel}	0.05036	0.01277	3.207e-3	8.028e-4	2.008e-4	5.012e-5	1.255e-5
	$p = 2$						
#DOF	64	144	400	1296	4624	17424	67600
$\#\lambda_{J}$	46	70	118	214	406	790	1556
$\#iter_{\mathrm{J}}$	2	6	11	27	812*	53	76
$\#\lambda_{M}$	24	48	96	192	384	768	1536
$\#iter_{\mathrm{M}}$	3	7	19	46	121	251	367
ϵ_{rel}	3.234e-3	2.300e-4	1.950e-5	1.706e-6	1.5035e-7*	1.327e-8	1.172e-9
	$p = 3$						
#DOF	100	196	484	1444	4900	17956	
$\#\lambda_{J}$	58	82	130	226	418	802	
$\#iter_{\mathrm{J}}$	4	6	12	26	41	60	
$\#\lambda_{M}$	36	60	108	204	396	780	
$\#iter_{\mathrm{M}}$	4	10	24	57	148	376	
ϵ_{rel}	2.430e-4	3.672e-6	1.301e-7	4.913e-9	2.058e-10	9.015e-12	
	$p = 4$						
#DOF	144	256	576	14	5184		
$\#\lambda_{J}$	70	94	142	238	430		
$\#iter_{\mathrm{J}}$	9	9	15	28	46		
$\#\lambda_{M}$	48	72	120	216	408		
$\#iter_{\mathrm{M}}$	9	16	32	77	193		
ϵ_{rel}	1.574e-5	5.225e-7	1.269e-8	2.775e-10	6.109e-12		
	$p = 5$						
#DOF	196	324	676	1764	5476		
$\#\lambda_{J}$	82	106	154	250	442		
$\#iter_{\mathrm{J}}$	21	19	21	37	61		
$\#\lambda_{M}$	60	84	132	228	420		
$\#iter_{\mathrm{M}}$	14	26	50	104	262		
ϵ_{rel}	8.762e-7	5.540e-8	5.705e-10	6.165e-12	1.890e-13		

Table B.1.: Data for the quad patch geometry with homogeneous Dirichlet data. Each column corresponds to the used refinement level, starting with one. In the case marked with *, the relative error of the jump operator is 8.251% larger than the one depicted here.

	$p = 1$						
#DOF	54	150	486	1734	6534	25350	99846
$\#\lambda_{\mathrm{J}}$	48	80	144	272	528	1040	2064
$\#iter_{\mathrm{J}}$	7	20	43	65	93	138	201
$\#\lambda_{\mathrm{M}}$	16	48	112	240	496	1008	2032
$\#iter_{\mathrm{M}}$	4	19	63	154	262	380	531
$\epsilon_{\mathrm{rel,J}}$	0.0106	2.340e-3	5.7619e-4	1.4441e-4	3.6219e-5	9.074e-6	2.271e-6
$\epsilon_{\mathrm{rel,M}}$	0.0170	3.944e-3	9.6209e-4	2.3848e-4	5.9452e-5	1.485e-5	3.711e-6
$\frac{t_{\mathrm{mult}}}{t_{\mathrm{single}}}$						0.69	0.56
	$p = 2$						
#DOF	96	216	600	1944	6936	26136	101400
$\#\lambda_{J}$	64	96	160	288	544	1056	2080
$\#iter_{\mathrm{J}}$	12	23	38	51	72	101	149
$\#\lambda_{M}$	32	64	128	256	512	1024	2048
$\#iter_{\mathrm{M}}$	12	29	78	198	330	502	681
$\epsilon_{\mathrm{rel,J}}$	1.720e-3	1.109e-4	1.048e-5	1.087e-6	1.202e-7	1.394e-8	1.671e-9
$\epsilon_{\mathrm{rel,M}}$	1.306e-3	0.986e-4	8.149e-6	7.093e-7	6.253e-8	5.526e-9	4.886e-10
$\frac{t_{\mathrm{mult}}}{t_{\mathrm{single}}}$					0.512	0.490	0.450
	$p = 3$						
#DOF	150	294	726	2166	7350	26934	
$\#\lambda_{J}$	80	112	176	304	560	1072	
$\#iter_{\mathrm{J}}$	18	28	44	59	75	97	
$\#\lambda_{M}$	48	80	144	272	528	1040	
$\#iter_{\mathrm{M}}$	18	42	101	260	487	766	
$\epsilon_{\mathrm{rel,J}}$	2.688e-4	5.740e-6	1.585e-7	6.836e-9	3.800e-10	2.311e-11	
$\epsilon_{\mathrm{rel,M}}$	1.852e-4	5.478e-6	1.310e-7	3.864e-9	1.421e-10	5.909e-12	
$\frac{t_{\mathrm{mult}}}{t_{\mathrm{single}}}$			0.643	0.464	0.390	0.385	
	$p = 4$						
#DOF	216	384	864	2400	7776		
$\#\lambda_{J}$	96	128	192	320	576		
$\#iter_{\mathrm{J}}$	25	37	53	66	86		
$\#\lambda_{M}$	64	96	160	288	544		
$\#iter_{\mathrm{M}}$	30	61	145	363	681		
$\epsilon_{\mathrm{rel,J}}$	2.671e-5	1.600e-6	2.274e-8	4.325e-10	9.400e-12		
$\epsilon_{\mathrm{rel,M}}$	2.432e-5	1.297e-6	1.808e-8	3.630e-10	7.913e-12		
$\frac{t_{\mathrm{mult}}}{t_{\mathrm{single}}}$		0.66	0.49	0.37	0.37		
	$p = 5$						
#DOF	294	486	1014	2646	8214		
$\#\lambda_{J}$	112	144	208	336	592		
$\#iter_{\mathrm{J}}$	40	288	71	88	110		
$\#\lambda_{M}$	80	112	176	304	560		
$\#iter_{\mathrm{M}}$	50	100	213	521	985		
$\epsilon_{\mathrm{rel,J}}$	6.923e-6	1.392e-6	3.027e-9	3.657e-11	5.740e-13		
$\epsilon_{\mathrm{rel,M}}$	3.837e-6	2.947e-7	2.091e-9	2.112e-11	2.703e-13		
$\frac{t_{\mathrm{mult}}}{t_{\mathrm{single}}}$		0.71	0.52	0.43	0.37		

Table B.2.: Data for the quarter sphere geometry with inhomogeneous Dirichlet data. Each column corresponds to the used refinement level, starting with one.

	$p = 1$					
#DOF	144	400	1296	4624	17424	67600
$\#\lambda_J$	136	216	376	696	1336	2616
$\#iter_{\mathrm{J}}$	8	25	54	89	133	189
$\#\lambda_M$	40	120	280	600	1240	2520
$\#iter_{\mathrm{M}}$	5	25	80	192	345	535
$\epsilon_{\mathrm{rel,J}}$	0.018	4.968e-3	1.281e-3	3.243e-4	8.153e-5	2.043e-5
$\epsilon_{\mathrm{rel,M}}$	0.032	7.970e-3	1.994e-3	4.995e-4	1.250e-4	3.128e-5
	$p = 2$					
#DOF	256	576	1600	5184	18496	69696
$\#\lambda_J$	176	256	416	736	1376	2656
$\#iter_{\mathrm{J}}$	14	29	43	61	90	129
$\#\lambda_M$	80	160	320	640	1280	2560
$\#iter_{\mathrm{M}}$	13	33	99	235	431	642
$\epsilon_{\mathrm{rel,J}}$	1.146e-3	1.377e-4	1.538e-5	1.799e-6	2.175e-7	2.674e-8
$\epsilon_{\mathrm{rel,M}}$	1.152e-3	8.731e-5	7.122e-6	6.124e-7	5.361e-8	4.721e-9
	$p = 3$					
#DOF	400	784	1936	5776	19600	71824
$\#\lambda_J$	216	296	456	776	1416	2696
$\#iter_{\mathrm{J}}$	21	33	51	70	94	125
$\#\lambda_M$	200	200	360	680	1320	2600
$\#iter_{\mathrm{M}}$	52	52	128	318	641	982
$\epsilon_{\mathrm{rel,J}}$	3.381e-4	8.875e-6	2.341e-7	1.174e-8	7.330e-10	4.707e-11
$\epsilon_{\mathrm{rel,M}}$	2.063e-4	5.179e-6	1.140e-7	3.550e-9	1.380e-10	5.893e-12
	$p = 4$					
#DOF	576	1024	2304	6400	20736	73984
$\#\lambda_J$	256	336	496	816	1456	2736
$\#iter_{\mathrm{J}}$	29	42	62	79	101	133
$\#\lambda_M$	160	240	400	720	1360	2640
$\#iter_{\mathrm{M}}$	35	76	177	425	890	1419
$\epsilon_{\mathrm{rel,J}}$	3.035e-5	2.244e-6	2.855e-8	4.041e-10	1.111e-11	4.233e-13
$\epsilon_{\mathrm{rel,M}}$	2.097e-5	1.221e-6	1.512e-8	2.822e-10	6.145e-12	1.468e-13
	$p = 5$					
#DOF	784	1296	2704	7056	21904	
$\#\lambda_J$	296	376	536	756	1496	
$\#iter_{\mathrm{J}}$	44	752	78	1712*	122	
$\#\lambda_M$	200	280	440	760	1400	
$\#iter_{\mathrm{M}}$	64	125	256	601	1316	
$\epsilon_{\mathrm{rel,J}}$	5.463e-6	3.111e-7	6.588e-9	2.037e-4*	1.570e-12	
$\epsilon_{\mathrm{rel,M}}$	3.774e-6	2.762e-7	2.202e-9	2.257e-11	3.303e-13	

Table B.3.: Data for the sphere geometry with inhomogeneous Dirichlet data. Each column corresponds to the used refinement level, starting with one.

	$p=1$					
#DOF	230	790	2918	11206	43910	173830
$\#\lambda_M$	44	104	224	464	944	1904
$\#iter_{\mathrm{M}}$	26	84	165	247	366	507
$\epsilon_{\mathrm{rel,M}}$	0.01515	3.495e-3	8.433e-4	2.081e-4	5.180e-5	1.293e-5
	$p=2$					
#DOF	304	920	3160	11672	44824	
$\#\lambda_M$	60	120	240	480	960	
$\#iter_{\mathrm{M}}$	39	108	223	328	494	
$\epsilon_{\mathrm{rel,M}}$	5.830e-4	4.820e-5	4.100e-6	3.517e-7	3.063e-8	

Table B.4.: Data for the quarter sphere geometry with inhomogeneous Dirichlet data and patch-wise refinement. Each column corresponds to the used refinement level, starting with one.

	$p=3$							
#threads	1	2	3	4	5	6	7	8
$\frac{t_{\mathrm{mult}}}{t_{\mathrm{single}}}$	1	0.8106	0.4709	0.4056	0.4023	0.3638	0.3679	0.3431

Table B.5.: Ratio of parallel to sequential runtime for the sphere geometry with inhomogeneous Dirichlet data and refinement level five.

Bibliography

[1] *AMD Ryzen™ 7 5800H*. URL: https://www.amd.com/en/product/10821.

[2] K. E. Atkinson. *Spherical harmonics and approximations on the unit sphere: An introduction*. Vol. 2044. Lecture notes in mathematics. Berlin: Springer, 2012.

[3] Y. Bazilevs, L. Beirão da Veiga, J. A. Cottrell, T. J. R. Hughes, and G. Sangalli. "Isogeometric Analysis: Approximation, Stability and error estimates for h-refined meshes". In: *Mathematical Models and Methods in Applied Sciences* 16.07 (2006), pp. 1031–1090.

[4] D. Braess. *Finite Elemente: Theorie, schnelle Löser und Anwendungen in der Elastizitätstheorie*. 5. Aufl. 2013. Masterclass. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013.

[5] E. Brivadis, A. Buffa, B. Wohlmuth, and L. Wunderlich. "Isogeometric mortar methods". In: *Computer Methods in Applied Mechanics and Engineering* 284 (2015), pp. 292–319.

[6] A. Buffa and G. Sangalli, eds. *IsoGeometric analysis: A new paradigm in the numerical approximation of PDEs : Cetraro, Italy 2012*. Vol. 2161. Lecture notes in mathematics CIME Foundation subseries. Cham: Springer, 2016.

[7] B. Butrylo, F. Musy, L. Nicolas, R. Perrussel, R. Scorretti, and C. Vollaire. "A survey of parallel solvers for the finite element method in computational electromagnetics". In: *COMPEL - The international journal for computation and mathematics in electrical and electronic engineering* 23.2 (2004), pp. 531–546.

[8] J. A. Cottrell. *Isogeometric analysis: Toward integration of CAD and FEA*. Chichester, West Sussex, U.K and Hoboken, NJ: Wiley, 2009.

[9] V. Dolean, P. Jolivet, and F. Nataf. *An introduction to domain decomposition methods: Algorithms, theory, and parallel implementation*. Vol. 144. Other titles in applied mathematics. Philadelphia: SIAM Society for Industrial and Applied Mathematics, 2015.

[10] J. Dölz, H. Harbrecht, S. Kurz, M. Multerer, S. Schöps, and F. Wolf. "Bembel: The fast isogeometric boundary element C++ library for Laplace, Helmholtz, and electric wave equation". In: (2021).

[11] J. Dölz, H. Harbrecht, S. Kurz, S. Schöps, and F. Wolf. "A Fast Isogeometric BEM for the Three Dimensional Laplace- and Helmholtz Problems". In: *Computer Methods in Applied Mechanics and Engineering* 330 (2018), pp. 83–101.

[12] G. Dziuk and C. M. Elliott. "Finite element methods for surface PDEs". In: *Acta Numerica* 22 (2013), pp. 289–396.

[13] J. W. Eaton. *GNU Octave Manual*. Network Theory Limited, 2002.

[14] *Eigen and multi-threading*. URL: https://eigen.tuxfamily.org/dox/TopicMultiThreading.html.

[15] A. Ern and J.-L. Guermond. *Finite Elements I: Approximation and Interpolation*. Vol. volume 72. Springer eBook Collection. Cham: Springer, 2021.

[16] C. Farhat, M. Lesoinne, P. LeTallec, K. Pierson, and D. Rixen. “FETI-DP: a dual-primal unified FETI method?part I: A faster alternative to the two-level FETI method”. In: *International Journal for Numerical Methods in Engineering* 50.7 (2001), pp. 1523–1544.

[17] C. Farhat, J. Mandel, and F. X. Roux. “Optimal convergence properties of the FETI domain decomposition method”. In: *Computer Methods in Applied Mechanics and Engineering* 115.3-4 (1994), pp. 365–385.

[18] C. Farhat and F.-X. Roux. “A method of finite element tearing and interconnecting and its parallel solution algorithm”. In: *International Journal for Numerical Methods in Engineering* 32.6 (1991), pp. 1205–1227.

[19] G. Guennebaud, B. Jacob, et al. *Eigen v3*. 2010. URL: `http://eigen.tuxfamily.org`.

[20] T. Hirschler, R. Bouclier, D. Dureisseix, A. Duval, T. Elguedj, and J. Morlier. “A dual domain decomposition algorithm for the analysis of non-conforming isogeometric Kirchhoff–Love shells”. In: *Computer Methods in Applied Mechanics and Engineering* 357 (2019).

[21] T. Hughes, J. A. Cottrell, and Y. Bazilevs. “Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement”. In: *Computer Methods in Applied Mechanics and Engineering* 194.39-41 (2005), pp. 4135–4195.

[22] J. Jost. *Riemannian Geometry and Geometric Analysis*. 7th ed. 2017. SpringerLink Bücher. Cham: Springer, 2017.

[23] B. Jüttler and B. Simeon, eds. *Isogeometric Analysis and Applications 2014*. Vol. 107. Lecture Notes in Computational Science and Engineering. Cham: Springer International Publishing, 2015.

[24] S. K. Kleiss, C. Pechstein, B. Jüttler, and S. Tomar. “IETI - Isogeometric Tearing and Interconnecting”. In: *Computer Methods in Applied Mechanics and Engineering* 247-248.11 (2012), pp. 201–215.

[25] W. MacLean and W. McLean. *Strongly elliptic systems and boundary integral equations*. 1. publ. Cambridge: Cambridge University Press, 2000.

[26] F. Magoulès, F.-X. Roux, and G. Houzeaux, eds. *Parallel Scientific Computing*. Hoboken, NJ, USA: John Wiley & Sons, Inc, 2015.

[27] P. Monk. *Finite element methods for Maxwell’s equations*. Reprinted. Oxford science publications. Oxford: Clarendon Press, 2006.

[28] M. Nolte. “Mortaring for the Isogeometric Boundary Element Method”. MA thesis. Technische Universität Darmstadt, 2021.

[29] OpenMP Architecture Review Board. *OpenMP Application Program Interface Version 3.0*. 2008.

[30] L. Piegl. *The NURBS Book*. Second Edition. Monographs in Visual Communication. Berlin and Heidelberg: Springer, 1997.

[31] T. K. Rusch. “Isogeometric Analysis on Multiple Patches for Aerospace Applications”. 2018.

[32] Y. Saad. *Iterative methods for sparse linear systems*. 2nd ed. Vol. 82. Other titles in applied mathematics. Philadelphia, Pa.: Society for Industrial and Applied Mathematics (SIAM 3600 Market Street Floor 6 Philadelphia PA 19104), 2003.

[33] M. Spink, D. Claxton, C. de Falco, and R. Vazquez. *nurbs package*. URL: `https://octave.sourceforge.io/nurbs`.

[34] N. Vukašinović and J. Duhovnik. *Advanced CAD Modeling: Explicit, Parametric, Free-Form CAD and Re-engineering*. SpringerLink Bücher. Cham: Springer International Publishing, 2019.

[35] J. Whiteley. *Finite Element Methods: A Practical Guide*. Springer eBook Collection Engineering. Cham: Springer, 2017.

[36] F. Wolf. *Analysis and Implementation of Isogeometric Boundary Elements for Electromagnetism*. Darmstadt: Universitäts- und Landesbibliothek Darmstadt, 2020.