UNIVERSITAT POLITÈCNICA DE CATALUNYA

DOCTORAL THESIS

Unfitted finite element methods for explicit boundary representations

Author:
Pere Antoni MARTORELL POL

Supervisors:
Prof. Santiago BADIA
Dr. Francesc VERDUGO

A thesis submitted in fulfilment of the requirements for the degree of Doctor of Philosophy

in the

Doctorat en Enginyeria Civil

Escola Tècnica Superior d'Enginyeria de Camins, Canals i Ports de Barcelona

Barcelona, January 2024

To Ester, my family and friends.

"Everything must be made as simple as possible. But not simpler."

Albert Einstein

Abstract

This thesis covers the development of large-scale numerical methods for the simulations of partial differential equations on arbitrarily complex geometries. The target application of this thesis is the structural simulation of buildings and civil infrastructures, in which lightweight and aesthetical demands usually increase the complexity of their geometries. In these applications, empirical experimentation is often not feasible during the design loop. Thus, we rely on numerical simulations to predict the performance of these shapes under realistic loads, e.g., wind loads that increase the complexity with the sophistication of the geometry. Current simulation tools are based on unstructured body-fitted meshes. The generation of unstructured meshes is time-consuming and involves human intervention. Furthermore, body-fitted methods cannot efficiently exploit modern high-performance computing resources.

The main goal of this thesis is to design novel simulation tools for rapid, accurate, and automated solutions of partial differential equations on geometries described by computer-aided design. Thus, we aim for a framework that combines (1) an automated pipeline from computer-aided design to finite element analysis, (2) a novel space-time formulation for moving geometries, and (3) a scalable implementation for high-performance computing resources. Our developments are accessible through open-source software within the Gridap ecosystem and FEMPAR packages (written in Julia and Fortran, respectively).

The contributions of this thesis increase the functionality of state-of-the-art unfitted (or immersed or embedded) finite element methods. These methods utilize structured background meshes to solve partial differential equations on complex domains. In the literature, these domains are implicitly represented by level sets. To address this limitation, (1) we developed a robust algorithm that solves problems on domains described by linear boundary representations. This algorithm is based on robust polyhedra clipping algorithms. We have tested the algorithm against all the analysis-ready geometries (STL files) in the Thingi10k dataset (almost 5,000). We have extended this algorithm to high-order boundary representations. In this extension, we utilize Bernstein-Bézier basis and multi-variate root-finding algorithms. We validate the resulting method with analytical benchmarks and real-world geometries from computer-aided design files.

Then, (2) we formulated an unfitted space-time finite element framework for moving explicit geometries. In this formulation, we utilized space-only meshes, circumventing the need for 4D geometrical algorithms. In turn, we developed a transfer method for evaluating the initial values at each time slab. The results matched with analytical analysis and external numerical experiments. Furthermore, we have demonstrated the applicability of fluid problems on rotating complex (2D and 3D) geometries.

Finally, (3) we proposed the acceleration of the methods of this thesis through highly scalable algorithms. These algorithms tackle the bottlenecks of parallelization of the intersection algorithms. We have demonstrated the scalability of these algorithms over one billion cells and 12,000 cores. Furthermore, we can combine these algorithms with adaptive mesh refinement techniques to reduce the computational cost further. These tools provide the means to significantly accelerate the design-to-simulation pipeline while increasing the fidelity of the results.

Resum

Aquesta tesi cobreix el desenvolupament de mètodes numèrics a gran escala per a la simulació d'equacions amb derivades parcials en geometries arbitràriament complexes. L'aplicació d'aquesta tesi està orientada a la simulació estructural d'edificis i infraestructures civils, en les quals les demandes estètiques i de lleugeresa augmenten la complexitat de les geometries. En aquestes aplicacions, l'experimentació empírica no és factible durant el procés de disseny. Per tant, es confia en simulacions numèriques per a predir el comportament d'aquestes estructures sotmeses a càrregues realistes, e.g., efectes del vent que es compliquen amb la sofisticació de la geometria. Les eines de simulació actuals es basen en malles no estructurades adaptades a la geometria. La creació de malles no estructurades és un procés demandant que requereix la intervenció humana. A més, els mètodes de malles adaptades a la geometria no poden explotar els recursos computacionals d'alt rendiment moderns de manera eficient.

L'objectiu principal d'aquesta tesi és el disseny de noves eines per a simular ràpidament, acurada i automàtica equacions amb derivades parcials sobre geometries definides per disseny assistit per ordinador. Per aquest motiu, es cerca un sistema que combina (1) un procés automàtic des del disseny assistit per ordinador fins a l'anàlisi d'elements finits, (2) una nova formulació espai-temps per a geometries en moviment, i (3) una implementació escalable per explotar recursos computacionals d'alt rendiment. Els desenvolupaments estan accessibles mitjançant programari de codi obert dins de l'ecosistema Gridap i del paquet FEMPAR (escrits en Julia i Fortran, respectivament).

Les contribucions d'aquesta tesi incrementen la funcionalitat dels mètodes d'elements finits embeguts ("immersed", "embeddedd" o "unfitted" en anglès) respecte a l'estat de l'art. Aquests mètodes utilitzen malles de fons estructurades per resoldre equacions amb derivades parcials en dominis complexos. En la literatura, aquests dominis es representen implícitament per conjunts de nivell. Per a abordar aquesta limitació, (1) s'ha desenvolupat un algorisme robust que resol problemes en dominis descrits per representacions de contorn lineals. Aquest algorisme es basa en algorismes robustos per retallar poliedres. S'ha testejat l'algorisme amb totes les geometries (fitxers STL) analitzables del recurs Thingi10k (gairebé 5.000). S'ha ampliat l'algorisme per a representacions de contorn d'alt ordre. En aquesta extensió, es fan servir les bases de Bernstein-Bézier i algorismes de cerca d'arrels multivariables. S'ha validat el mètode mitjançant referències analítiques i geometries reals creades amb eines de disseny assistit per ordinador.

A continuació, (2) s'ha formulat un marc d'elements finits embegut i espai-temps per a geometries explícites que es mouen. En aquesta formulació, s'utilitzen malles espacials per a evitar la necessitat d'algorismes geomètrics en 4D. En canvi, s'ha desenvolupat un mètode de transferència de valors inicials a les llesques temporals. Els resultats coincideixen amb l'anàlisi matemàtica i amb els experiments numèrics externs. Així mateix, s'ha demostrat l'aplicabilitat a problemes de fluids amb geometries complexes (2D i 3D) en rotació.

Finalment, (3) s'ha proposat l'acceleració dels mètodes d'aquesta tesi a través d'algorismes altament escalables. Aquests algorismes aborden colls d'ampolla de la paral·lelització dels algorismes d'intersecció. S'ha demostrat l'escalabilitat d'aquests algorismes amb més de mil milions de cel·les i 12.000 processadors. Per afegiment, es poden combinar aquests algorismes amb tècniques d'adaptació de malla per reduir, encara més, el cost computacional. Aquestes eines proporcionen els mitjans per a accelerar el procés de disseny a simulació tot incrementant la fidelitat dels resultats.

Acknowledgements

I would like to gratefully acknowledge the guidance, advice, and support of my supervisors, Santiago Badia and Francesc Verdugo. I am very grateful for your patience, availability, and commitment despite the geographical distance during these years. I have been fortunate to work with all the researchers at the Large Scale Scientific Computing group of CIMNE. Thank you Àlex, for your motivation and for introducing me to the research group. I would like to thank Alberto for selflessly sharing his deep knowledge and Javier for his kind support and valuable advice.

I would like to thank my office mates Eric, Jesús, and Marc for the valuable advice, teachings, and warm working environment. Every one of them has been an essential example throughout the entire journey. I also thank the other members of the research group: Daniel, Jerrad, Manuel, and Víctor, for the fruitful discussions and advice.

I would like to acknowledge the support from Universitat Politecnica de Catalunya and Santander Bank for the PhD fellowship (FPI-UPC-2019). I would like to extend my gratitude to the European Council, the Spanish Ministry, and the Australian Research Council for the financial support through ExaQUte, AMBBOS, and Discovery projects, respectively. I want to thank the CIMNE and UPC staff for their administrative support and availability. Thank you, Silvia A., for your kind help.

I want to thank the members of the JIPI organization committee for their support in cross-cutting research issues. I hope it will continue again. I also want to thank all the people who warmly welcomed me in Melbourne during my stay at Monash University, especially Angelika, John, and Ricardo. I am grateful to Alex, Connor, Hridya, Kishore, Wei, Jiahao, and Martin for making a memorable stay through interesting discussions, amazing trips, and social events. I would like to thank Asseem for his kind hospitality from the very first day.

I want to thank all the friends from my hometown, university, and Cranfield who have always been there: Jaume, Joan Pere, Joan Toni, Pau, Robert, and the whole crew, Aura, Carles, David, Edu, Eloi, Jordi C. A., Jordi C. J., Juan F., Leo, Manel, Marc, Robert, Sergi, Ulises, and Xavi, Damien, Loïc and Xavier. Thank you for all the adventures and happy moments. I have learned a lot from every one of you.

I am very grateful to my partner's relatives for considering me part of their family: Jordi, Llúcia, Laura, Miguel, and our joyful godson Sergi. Thank you, uncle Andreu, for your inspiration and scientific advice. I want to thank my brother for all we have shared and for his inspiration in continuously learning. Thank you, Joana and wonderful little Guida, for being part of my family. I thank my parents for teaching me the value of patience, perseverance, knowledge, and love. Thank you for your blind trust and support. I want to express special gratitude to my beloved Ester for her love, patience, and inspiration. You have enriched my character in many dimensions. Thank you for being at my side, regardless of the circumstances.

Contents

Abstract vii

Resum ix

Acknowledgements xi

List of Figures xvii

List of Tables xix

List of Abbreviations xxiii

1 Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Thesis objectives . . . 2
- 1.3 Document structure . . . 3
- 1.4 Research publications . . . 5
- 1.5 Conference talks . . . 5
- 1.6 Research stays . . . 6

2 Unfitted discretizations for linear explicit BREPs 7
- 2.1 Introduction . . . 7
- 2.2 Unfitted finite element discretisations . . . 11
- 2.3 Intersection algorithms . . . 14
  - 2.3.1 Polyhedra and polygonal surface representations . . . 15
  - 2.3.2 Half-space representations . . . 17
  - 2.3.3 Clipping a polyhedron with a plane . . . 18
  - 2.3.4 Intersecting a polyhedron with a surface . . . 19
  - 2.3.5 Robust computation of signed distances . . . 25
  - 2.3.6 Global intersection algorithm . . . 28
- 2.4 Numerical experiments . . . 31
  - 2.4.1 Objectives . . . 31
  - 2.4.2 Experimental setup . . . 32
  - 2.4.3 Batch processing the STL models of the Thingi10K data-set . . . 32
  - 2.4.4 Robustness test . . . 35
  - 2.4.5 Finite Element convergence test . . . 37

2.5 Conclusions and future work . . . 39

3 High-order unfitted discretizations for explicit BREPs 43
3.1 Introduction . . . 43
3.2 Unfitted finite element method . . . 47
3.2.1 Unfitted finite element formulations . . . 47
3.2.2 Geometrical ingredients for unfitted finite elements . . . 49
3.2.3 Integration methods for cut cells . . . 50
3.3 Intersection algorithm . . . 52
3.3.1 Intersection points . . . 53
Curve-plane intersection . . . 53
Surface-line intersection . . . 53
Critical points of the zero isosurface of a distance function . . . 54
3.3.2 Nonlinear trimming surface . . . 54
3.3.3 Connection algorithm . . . 59
3.3.4 Surface partition . . . 62
3.3.5 Surface parametrization . . . 63
3.3.6 Cell intersection . . . 64
3.3.7 Global algorithm . . . 66
3.4 Numerical experiments . . . 68
3.4.1 Experimental setup . . . 68
3.4.2 Approximation and parametrization analysis . . . 68
3.4.3 Robustness experiments . . . 69
3.4.4 Unfitted FE experiments . . . 70
3.5 Conclusions and future work . . . 73

4 Space-time unfitted FEM for moving explicit BREPs 75
4.1 Introduction . . . 75
4.2 Space time unfitted finite element method . . . 77
4.2.1 Geometry description for moving domains . . . 77
4.2.2 Space-time finite element spaces . . . 79
4.2.3 Extension of the deformation map . . . 80
4.2.4 Extended active mesh . . . 82
4.3 Variational formulation on a model problem . . . 82
4.3.1 Weak formulation . . . 83
4.3.2 Inter-slab integration . . . 85
4.4 Intersection algorithm for time slab transfer . . . 87
4.5 Numerical experiments . . . 88
4.5.1 Objectives . . . 88
4.5.2 Environment setup . . . 88
4.5.3 Space-time convergence tests . . . 89
4.5.4 Moving domains examples . . . 91

4.6 Conclusions and future work 94

5 Distributed unfitted discretizations for explicit BREPs 95
5.1 Introduction 95
5.2 Distributed unfitted finite element method 97
5.3 Distributed intersection algorithm 99
5.3.1 Local intersection 99
5.3.2 Local classification 100
5.3.3 Global distributed algorithm 100
5.4 Numerical experiments 103
5.4.1 Experimental setup 103
5.4.2 Parallel scalability 103
5.5 Conclusions and future work 105

6 Conclusions and future work 107
6.1 Conclusions 107
6.2 Future work 108

Bibliography 111

List of Figures

2.1 Example of an embedded non-convex domain in 2D. 13
2.2 Example of Algorithm 1 that converts (a) a closed polyhedra into (b) an open surface polyhedra. 17
2.3 Illustration of Algorithm 2. 20
2.4 Example of the application of Algorithm 4 to decomposes a cell polyhedron K and a non-convex surface S into convex parts. 24
2.5 Illustration to explain Algorithm 5 for a 2D example. 25
2.6 Illustration of Algorithm 6, which intersects a convex volume P by a convex surface S. 26
2.7 Simple 2D example to justify the choice of open and closed half-spaces in Algorithm 4 and 10. 29
2.8 Illustration of Algorithm 6, i.e., $(S, \mathbf{H}) \cap K$, when calling line 7 of Algorithm 10. 30
2.9 Illustration of Algorithm 10, which, given an STL $\mathcal{B}$ and a background mesh $\mathcal{T}$ intersects each background cell $K \in \mathcal{T}$ with $\mathcal{B}$. 31
2.10 Selection of 11 STL models from the Thingi10K database processed in the numerical examples 33
2.11 Generated volume sub-triangulation for the STL model with id 441708. . 34
2.12 Volume and surface error distributions. 35
2.13 Results of the robustness test. 36
2.14 FE approximation computed with AgFEM on top of three of the STL models analysed in the experiments. 38
2.15 Results of the FE convergence test. 38
2.16 AgFEM approximations of two physical problems on both sides of the *Arc de Triomph* STL. 40

3.1 Example of the embedded nonlinear domain in 2D. 48
3.2 Representation of the surface $\mathcal{B} \doteq \partial\Omega$, its intersection $\mathcal{B}_K^{\text{cut}} \doteq \mathcal{B} \cap K$ for a background cell $K \in \mathcal{T}$, and the domain interior of the cell $K^{\text{cut}} \doteq K \cap \Omega$. 50
3.3 Definition of intersection points. 54
3.4 Refinement steps of Algorithm 11. 55
3.5 Representation of clipping algorithm $F \cap K$. 59
3.6 Representation of Algorithm 14 for strictly increasing curves. 61
3.7 Example of iterative approximation of a curve. 64
3.8 Representation of the steps that generate $(\partial K)^{\text{cut}}$. 65

3.9 CAD geometry and the clipping steps for a background cell. 67
3.10 Surface and volume errors of the approximation and the reparametrization of a sphere. 69
3.11 Demonstration of robustnes concerning the relative position of $\mathcal{B}$ and $\mathcal{T}$. 70
3.12 Convergence tests in AgFEM. 71
3.13 Realistic examples on CAD geometries. 72

4.1 Representation of the space-time domain Q embedded within an artificial space-time domain Q_{art}. 78
4.2 Representation of the deformation map $\boldsymbol{\varphi}_h^n$. 79
4.3 Representation of the deformation map $\boldsymbol{D}$ and the extended map $\boldsymbol{\varphi}_h^n$ in the time slab $J^n = (t^n, t^{n+1})$. 81
4.4 Mesh sequence for solution transfer between time slabs. 86
4.5 Mesh sequence for solution transfer between time slabs. 86
4.6 Scaling of the condition numbers of the mass and stiffness matrices in the initial time slab. Convergence of the space-time DG norm error ($e = u - u_h$) in two and three-dimensional space domains. 90
4.7 Convergence of the $L^2(\Omega^n)$ and $H^1(\Omega^n)$ norms of the error ($e = u - u_h$) at the final time $t = T$ in two and three-dimensional space domains. . . 90
4.8 Background spatial mesh $\bar{\mathcal{T}}^{\text{bg}}$ around a prismatic gear $\mathcal{B}_0$. 91
4.9 Background spatial mesh $\bar{\mathcal{T}}^{\text{bg}}$ around a wing $\mathcal{B}_0$. 92
4.10 Representation of the LIC filter of the viscous flow simulation a evolving geometry in Figure 4.8. 92
4.11 LIC representation of the viscous flow around Figure 4.9. 93

5.1 Unfitted FE representation in a distributed-memory computation. 98
5.2 Representation of the two-level propagation algorithm. 101
5.3 Strong scaling of the creation of distributed embedded discretizations. . 104
5.4 Weak scaling of the creation of distributed embedded discretizations. . . 105

List of Tables

2.1 Main features of the test geometries considered displayed in Figure 2.10. 36

List of Algorithms

1 $\texttt{pol}(\vec{\Gamma})$ . . . 17
2 $(P, \mathbf{H}) \cap \boldsymbol{h}$ . . . 21
3 $\texttt{walls}(S, \mathbf{H}_{S,S})$ . . . 22
4 $\texttt{convexify}(\mathcal{C}, (P, \mathbf{H}_{SW,P}), (S, \mathbf{H}_{SW,S}))$, . . . 23
5 $\texttt{colouring}(S, \mathbf{H}_{S,S})$ . . . 23
6 $(P, \mathbf{H}) \cap S \to (P, \mathbf{H})$ . . . 25
7 $\texttt{align\_surface}(\mathbf{H}_{SK,KS}) \to \mathbf{H}_{S,KS}$ . . . 27
8 $\texttt{align\_planes}(\mathbf{H}) \to \mathbf{H}$ . . . 28
9 $\texttt{merge}(\mathbf{H}) \to \mathbf{H}$ . . . 28
10 $\mathcal{T} \cap \mathcal{B} \to \mathcal{T}^{\text{cut}}, \mathcal{B}^{\text{cut}}$ . . . 30
11 ref_1: axis-aligned refinement of invariants . . . 56
12 ref_2: partition by connecting intersections . . . 58
13 $F \cap K$ . . . 59
14 $\texttt{connect}(\hat{F}, \hat{\gamma}_f)$ . . . 61
15 $\texttt{parametrize}(\mathcal{T}, D)$ . . . 65
16 $\partial K \cap \text{int}(\mathcal{B}_K^{\text{cut}}) \to (\partial K)^{\text{cut}}$ . . . 66
17 $\mathcal{T} \cap \text{int}(\mathcal{B}^{\text{CAD}})$ . . . 67
18 $\mathcal{T} \cap \mathcal{T}_- \cap \text{int}(\mathcal{B})$ . . . 88
19 $\texttt{distributed\_intersection}(\mathcal{T}_s^{\text{bg}}, K_s^{\text{coarse}}, \Omega_s)$ . . . 102

List of Abbreviations

AA	**A**xis-**A**ligned
AABB	**A**xis-**A**ligned **B**ounding **B**ox
AgFE	**Ag**gregated **F**inite **E**lement
AgFEM	**Ag**gregated **F**inite **E**lement **M**ethod
AMG	**A**lgebraic **M**ulti-**G**rid
AMR	**A**daptive **M**esh **R**efinement
ALE	**A**rbitrary **L**agrangian-**E**ulerian
BDDC	**B**alancing **D**omain **D**ecompomposition by **C**onstraints
BREP	**B**oundary **REP**resenentation
CAD	**C**omputer-**A**ided **D**esign
CAE	**C**omputer-**A**ided **E**ngineering
CG	**C**ontinuous **G**alerkin
CSG	**C**onstructive **S**olid **G**eometry
DG	**D**iscontinuous **G**alerkin
DOF	**D**egree **O**f **F**reedom
FE	**F**inite **E**lement
FEM	**F**inite **E**lement **M**ethod
FSI	**F**luid-**S**tructure **I**nteraction
HPC	**H**igh **P**erformance **C**omputing
IGA	**I**sogeometric **A**nalysis
JIT	**J**ust-**I**n-**T**ime
k-DOP	**D**iscrete **O**rientation **P**olytope

LIC	**L**ine **I**ntegral **C**onvolution
MPI	**M**essage **P**assing **I**nterface
NURBS	**N**on-**U**niform **R**ational **B**asis **S**pline
OBB	**O**riented **B**ounding **B**ox
OCCT	**O**pen **C**ASCADE **T**echnology
PDE	**P**artial **D**ifferential **E**quation
STEP	**S**tandard for the **T**he **E**xchange of **P**roduct model data
STL	**ST**ereo**L**ithography
XFEM	e**X**tended **F**inite **E**lement **M**ethod

Chapter 1

Introduction

1.1 Motivation

Nowadays, lightweight structures and shape optimization processes aim to significantly reduce costs via savings in construction materials on buildings and civil infrastructures. Complex structures can frequently be the product of esthetic demand. In all these cases, those structures can be particularly sensitive to wind effects and prone to related structural issues (e.g., reaching the limit stress of the material, large deflection of flexible structures, possible oscillations, etc.). Moreover, complex structural shapes can lead to possible side effects, such as complex wind flows in the surrounding area. Designing and validating these constructions through empirical experimentation is time-consuming, expensive, and often not feasible.

Computational simulation tools are very convenient since they can predict those behaviors early in the design process. A vast range of available techniques in the computer-aided engineering (CAE) address these problems. These tools involve mathematical models to describe the physical behavior utilizing partial differential equations (PDEs). The solution of these equations is not analytical in general. Instead, discretization techniques can approximate PDEs within a bounded domain. The geometry description generally derives from computer-aided design (CAD) tools. However, other techniques, e.g., topology optimization loops, 3D scanning, or analytical functions, can generate the domain.

Traditionally, simulation techniques require the tesselation of the domain around the target geometry. This step is not straightforward for complex geometries and often involves human intervention. Indeed, the mesh generation represents 80% of total simulation time in many industrial applications [63]. This task becomes especially challenging when the geometries evolve in time, e.g., deflexion, oscillations, and rotations caused by wind loads. The well-established arbitrary Lagrangian-Eulerian (ALE) formulations address this problem for small deformations. However, the method requires building new meshes during the simulation when the deformations are larger. Remeshing is an obvious limitation for the simulation of multiphysics problems. This limitation may have a backward effect on the design and quality of the final product. These limitations are inherently linked to generating body-fitted meshes, i.e., meshes

that conform to the geometry. Therefore, bypassing this step is essential for a fully automated simulation pipeline.

Furthermore, most of the available codes are based on serial algorithms that cannot efficiently exploit the computational resources of modern parallel hardware such as distributed-memory supercomputers. The efficient parallelization of the simulation tools is a challenging task that involves the entire pipeline. Consequently, computer-based simulations of complex phenomena are time-consuming, making analyzing real-world problems very challenging. Only the combination of (a) novel geometrical tools to bridge the gap between CAD and CAE, (b) robust formulations to simulate evolving geometries, and (c) highly scalable parallel algorithms will provide the means for an automated, rapid, and accurate design-to-simulation pipeline. Implementing these algorithms on modern (and imminent exascale) high-performance computing (HPC) resources will generate a qualitative step forward in engineering design and analysis.

1.2 Thesis objectives

The main objective of the thesis is to address the computational challenges preventing current simulation tools for the finite element (FE) simulation PDEs on complex geometries from CAD to generate rapid, accurate, and automatic results. The achievement of this objective breaks down into the following specific objectives:

O1: Address the gap between CAD and unfitted finite element methods to lead to accurate and robust results.

Generating body-fitted meshes around CAD models requires human intervention. Instead, unfitted (also known as embedded or immersed) FE methods embed the domain in a simple background mesh, e.g., uniform or adaptive Cartesian grids. The background mesh contains the functional discretization. The geometrical discretization only integrates the interior of the background cells with limited geometrical constraints (concerning body-fitted mesh generation). Generating integration meshes is straightforward with level-set defined geometries. However, the state-of-the-art unfitted methods do not successfully address geometries described by CAD models. Therefore, we consider the generation of embedded discretizations for linear CAD geometries, e.g., Stereolithography (STL) models.

O2: Formulate an automatic and precise framework for high-order CAD geometries with unfitted FE methods.

The CAD models are described by high-order geometrical representations, e.g., non-uniform rational basis spline (NURBS). Therefore, extending the previous objective (O1) to high-order discretizations is a natural step. This discretization is more involved since it requires non-linear operations and approximations of the

intersections, see [52]. Despite the increased complexity, we use the knowledge gained to develop the objective (O1). Furthermore, we consider tools to palliate the increment of computational cost, e.g., utilizing moment fitting techniques [107].

O3: Formulate and analyze a space-time method for moving explicit boundaries.

The flexibility of unfitted FE methods makes them very appealing for computations with moving geometries. However, they also pose significant challenges which need to be addressed. One of the most well-known difficulties is that degrees of freedom (DOFs) are activated and de-activated during the process, which makes it difficult to define the initial time step value in time integration schemes for the DOFs that are activated. The conventional way to circumvent this problem is to define artificial initial conditions in the newly created DOFs, but this might introduce spurious oscillations. For this reason, we explore space-time unfitted methods when the initial geometry is described by a CAD model.

O4: Formulate a parallel framework for simulating PDEs on explicit boundary representation on uniform and h-adaptivity meshes.

The solution of real-world problems can become excessively time-consuming. To this end, we explore the parallelization of the tools developed during the thesis (O1-3). Additionally, we exploit parallelism on background adaptive meshes, e.g., octree meshes like `p4est` [37]. The development of parallel algorithms to exploit available HPC resources, e.g., Marenostrum IV at Barcelona and Gadi at Canberra, aims to increase the impact of this thesis.

O5: Develop and contribute to open source libraries to distribute the developed methods.

As a transversal objective, we contribute to software tools available throughout the scientific community. We provide notable contributions to the in-house codes. Firstly, we develop code in FEMPAR [13], an object-oriented Fortran code that provides a wide range of FE methods and parallel implementations. Secondly, we contribute to Gridap [21] a Julia Language [27] package package for approximating PDEs. Finally, we develop Julia packages for the specific developments of this thesis. These open source packages enrich the Gridap ecosystem.

1.3 Document structure

The initial chapter of this thesis serves to introduce and provide a rationale for the subject of our research, outlining the objectives pursued within this study. The contents spanning from Chapter 2 to Chapter 4 encompass the principal contributions of this thesis. Specifically, Chapter 2 corresponds to a publication detailed in the next section. Chapters 3 and 4 are currently undergoing preparations for publication, while Chapter

5 has not been published yet but presented at a conference. Each of these chapters is self-contained, preserving the structure of their respective associated papers, thereby enabling independent reading. Therefore, each chapter may have specific notations and repeated definitions.

In Chapter 2, we describe a novel algorithm to numerically integrate over geometries bounded by explicit boundary representations, e.g., STL models. This work is defined in the framework of unfitted FE methods. We motivate and present a state-of-the-art review in Section 2.1. In Section 2.2, we present some unfitted FE methods and their geometrical requirements. In Section 2.3, we provide the geometrical algorithm that computes the intersection of background cells and oriented surface meshes. In Section 2.4, we report a thorough numerical experimentation of the proposed algorithms on almost 5,000 meshes in the Thingi10K database [123]. We show the remarkable robustness of the geometrical algorithm, providing very low geometrical error quantities in all cases. The algorithms are combined with unfitted FE methods to approximate PDEs on these complex geometries, showing the expected convergence orders of accuracy. We also analyse the computational performance of the proposed framework and provide details about the corresponding open source implementation [76]. Finally, some conclusions and future work lines are drawn in Section 2.5.

Chapter 3 is devoted to high-order unfitted FE methods for geometries descdibed by CAD models. After the introduction of Section 3.1, we introduce the unfitted FE methods and their requirements for handling high-order boundary representations (BREPs) in Section 3.2. Next, in Section 3.3, we provide the proposed geometric algorithms for computing the nonlinear intersections between background cells and oriented high-order BREPs, along with a surface parametrization method for integration purposes. Then, in Section 3.4, we present the numerical results obtained from applying the proposed method, including accuracy and robustness of the intersections, benchmark tests for validation of the unfitted FE pipeline, and simulations on CAD geometries. Finally, in Section 3.5, we draw the main conclusions and future work lines.

Chapter 4 we expose a space-time formulation for unfitted methods and explicit geometries. Firstly, we motivate the topic with the available literature in Section 4.1. Then, in Section 4.3, we present the variational formulation through a model problem. We also define the integration measures and the inter-slab transfer mechanism. In Section 4.4, we describe the intersection algorithm for time slab transfer. Then, in Section 4.5, we present numerical results for space and time convergence, condition number tests, and a numerical example of a moving domain. Finally, in Section 4.6, we present our conclusions and future work.

In Chapter 5, we present a parallel and fully distributed FE framework for the simulation of complex CAD geometries which may evolve over time. After the introduction in Section 5.1, we present the parallel unfitted FE methods used in this chapter in Section 5.2. In Section 5.3, we introduce the distributed intersection algorithm. In

Section 5.4, we present the numerical results on stability. Finally, in Section 5.5, we summarize the main conclusions of this chapter.

1.4 Research publications

The advancements outlined in this thesis have led to a publication within an international peer-reviewed journal, accompanied by two research papers prepared for imminent submission. Each of these publications directly aligns with a distinct chapter within the thesis framework:

Chapter 2

[17] S. BADIA, P. A. MARTORELL AND F. VERDUGO, *Geometrical discretisations for unfitted finite elements on explicit boundary representations*, Journal of Computational Physics, 460 (2022), p. 111162.

Chapter 3

[75] P. A. MARTORELL AND S. BADIA, *High order unfitted finite element discretizations for explicit boundary representations*, submitted.

Chapter 4

S. BADIA, P. A. MARTORELL AND F. VERDUGO, *Space-time unfitted finite elements on moving explicit geometry representations*, to be submitted.

1.5 Conference talks

Furthermore, the author has presented part of the contents of this thesis as a *presenting speaker*, in the following international conferences. The presentation of on each conference corresponds to a distinct chapter.

Chapter 2

2021 P.A. MARTORELL, S. BADIA AND F. VERDUGO, *From STLs to embedded integration meshes via robust polyhedra clipping*, IX International Conference on Computational Methods for Coupled Problems in Science and Engineering. Sardinia, Italy.

Chapter 5

2019 P.A. MARTORELL, S. BADIA, F. VERDUGO , *A Scalable Technique for Numerical Integration in Cut Cells Based on 3D CAD Models* , VII International Conference on Computational Methods for Coupled Problems in Science and Engineering. Sitges, Spain.

1.6 Research stays

Throughout the doctoral program, the author undertook a three-month research stay at Monash University, under the supervision of Prof. Santiago Badia. The tasks executed during this period significantly contributed to Chapter 3 and 4.

Chapter 2

Unfitted finite element discretizations for linear explicit boundary representations

The contents of this chapter correspond to the research publication

[17] S. BADIA, P. A. MARTORELL AND F. VERDUGO, *Geometrical discretisations for unfitted finite elements on explicit boundary representations*, Journal of Computational Physics, 460 (2022), p. 111162.

Unfitted (also known as embedded or immersed) finite element approximations of partial differential equations are very attractive because they have much lower geometrical requirements than standard body-fitted formulations. These schemes do not require body-fitted unstructured mesh generation. In turn, the numerical integration becomes more involved, because one has to compute integrals on portions of cells (only the interior part). In practice, these methods are restricted to level-set (implicit) geometrical representations, which drastically limit their application. Complex geometries in industrial and scientific problems are usually determined by (explicit) boundary representations. In this work, we propose an automatic computational framework for the discretisation of partial differential equations on domains defined by oriented boundary meshes. The geometrical kernel that connects functional and geometry representations generates a two-level integration mesh and a refinement of the boundary mesh that enables the straightforward numerical integration of all the terms in unfitted finite elements. The proposed framework has been applied with success on all analysis-suitable oriented boundary meshes (almost 5,000) in the Thingi10K database and combined with an unfitted finite element formulation to discretise partial differential equations on the corresponding domains.

2.1 Introduction

Many industrial and scientific applications are modelled by PDEs posed on a non-trivial bounded domain Ω. In these situations, Ω is described in terms of a BREP model. These CAD models are 2-variate, i.e., they are not a parameterisation of Ω but

its boundary $\partial\Omega$; $\partial\Omega$ must be an oriented manifold and Ω is defined as its interior. On the other hand, the numerical approximation of PDEs, e.g., using FE or finite volume schemes, relies on a partition (mesh) of Ω. The traditional simulation pipeline involves unstructured mesh generation algorithms [62, 101], which take as input the CAD representation of $\partial\Omega$ and return a mesh covering Ω (introducing some approximation error). The creation of analysis-suitable CAD models and body-fitted mesh generation is a non-automatic process that requires intensive human intervention and amounts for most of the simulation time [63]. This weak interaction between (geometry representation) and CAE functional discretisation is arguably the most serious problem in CAE, which has motivated the *isogeometric* analysis paradigm [63]. Isogeometric analysis has been one of the most active research topic in computational engineering for the last two decades. While this paradigm is sound for PDEs on manifolds (CAD representations are 2-variate), it does not solve the most ubiquitous situation in practice, i.e., 3D simulations of PDEs in the *bulk* of the domain Ω.

Besides, in order to exploit supercomputing resources for unstructured mesh simulations, mesh partitioning strategies must be used, which rely on graph partitioning techniques. Such algorithms are intrinsically sequential and have huge memory requirements [65]. The mesh partitioning step can easily become the bottleneck (if not a showstopper) of the simulation pipeline for parallel computations on distributed memory machines. Furthermore, the use of such framework in adaptive mesh refinement (AMR) codes with dynamic load-balancing is not an acceptable option in terms of performance, preventing the use of AMR in practical large-scale applications with non-trivial geometries. The geometrical discretisation is even more challenging in applications with geometries that evolve in time (like additive manufacturing) or free boundary problems [86], since they require 4D geometrical models (space and time). The generation of body-fitted meshes for complex 3D geometries is still an open problem, and to expect 4D body-fitted generators in a mid term is not reasonable.

In order to solve the current limitations, one could consider unfitted discretisations [81]. A background mesh is used for the discretisation (instead of a body-fitted one) and the geometrical discretisation only requires to generate meshes suitable for integration in the interior defined by $\partial\Omega$ (drastically reducing mesh constraints). An unfitted approach can use tree-based background meshes and exploit scalable and dimension-agnostic mesh generators and partitioners [10]. Octree-based meshes can be efficiently generated and load-balanced using space-filling curve techniques [6]; see, e.g., the highly scalable `p4est` framework [37] for handling forests of octrees on hundreds of thousands of processors. The extension to a space-time immersed boundaries is feasible since tree-based meshes and marching algorithms are dimension-agnostic.

Unfitted discretisations may lead to unstable and severe ill-conditioned discrete problems [43] unless a specific technique mitigates the problem. The intersection of a background cell with the physical domain can be arbitrarily small and with unbounded aspect ratio. Despite vast literature on the topic, unfitted finite element formulations

that solve these issues are quite recent. Stabilised formulations based on the so-called *ghost penalty* were originally proposed in [33] for Lagrangian continuous FEs, and has been widely used since [34]. The so-called *cell aggregation* or *cell agglomeration* techniques are an alternative way to ensure robustness with respect to cut location. This approach is very natural in discontinuous Galerkin (DG) methods, as they can be easily formulated on agglomerated meshes [84]. These techniques have been extended to $\mathcal{C}^0$ Lagrangian finite elements in [22] and to mixed methods in [14]; the method was coined aggregated finite element method (AgFEM). These unfitted formulations enjoy good numerical properties, such as stability, condition number bounds, optimal convergence, and continuity with respect to data. Distributed implementations for large scale problems have been designed [117] and error-driven h-adaptivity and parallel tree-based meshes have also been exploited [10].

Even though unfitted discretisation are motivated by their geometrical flexibility, the current state-of-the-art in unfitted finite elements falls short with respect to the complexity of the geometrics being treated in these publications. The core of the problem is the design of algorithms for the numerical integration in the interior of background mesh cells only. The vast majority of numerical frameworks rely on implicit level-set descriptions of geometries and marching cubes (or tetrahedra) algorithms, thus limiting their application. We refer the to [53] for a state-of-the-art review of geometrical discretisation techniques for level-set representations. Geometrical algorithms have also been developed for the intersection of 3D tetrahedral meshes for the unfitted discretisation of interface problems (see [64, 80] and references therein).

The main motivation of this work is to provide a new geometrical framework that covers all the needs of unfitted techniques and is amenable to arbitrarily complex 3D geometries represented by STL meshes, i.e., oriented faceted linear surface representations. This is one of the most common situations in CAE, in which STL meshes are used to define complex objects. In particular, the starting point of the algorithm, as in unstructured mesh generation, is a boundary mesh for $\partial\Omega$. We design an algorithm for computing the intersection of each cell in a background mesh and the interior of the boundary mesh. The number of faces in the boundary mesh intersecting a background cell can be in the order of hundreds or even thousands for very complex geometrical representations. As a result, the proposed algorithm must be resilient to rounding errors and provide answers accurate up to machine precision in all cases. With this algorithm, we complete an automatic simulation framework that takes a standard CAD representation, an STL mesh, and returns the PDE solution obtained from an unfitted discretisation. The procedure is fully automatic and allows us to exploit all the benefits of unfitted formulations described above on complex geometries defined by STL representations. On the other hand, with the proposed formulation, the geometrical error (determined by the boundary mesh) and the functional error (determined by the background mesh) are completely decoupled. This remarkable property is not shared by FEs on unstructured meshes (both geometry and functional discretisation rely on

the same geometrical discretisation) or standard level-set approaches with marching algorithms on background cells (the geometrical approximation is determined by the background mesh). We note that the geometrical framework proposed in this work can readily be applied to other numerical techniques that can be posed on general polytopal meshes, e.g., hybridised formulations on agglomerated meshes [8] or mollified FEs [48].

The first key ingredient of the proposed framework is a robust clipping algorithm for convex polytopes. A popular method for clipping is the one by Sutherland and Hodgman in [109] (see also [106]). Recent implementations and improvements of these algorithms can be found in [72] and an extension to non-convex polyhedra can be found in [73]. These methods require accuracy checks and the handling of all degenerate branches. Instead, the approach proposed by Sugihara and co-workers, called *combinatorial abstraction*, is an example of a numerically robust scheme for the intersection of convex polyhedra [108]. An implementation of this algorithm has been proposed in [93] for intersecting a tetrahedron and a background Cartesian mesh. Still, Sugihara's method relies on assumptions that are not true in general and current implementations are not designed to deal with a large number of clipping planes or non-convex geometries. In this work, we build on [93, 108] to design a robust algorithm and an efficient implementation for the clipping of a polyhedron and a plane that can naturally handle with degenerate possibly non-connected and non-convex outputs and is suitable for our specific target.

Since the boundary representation is not convex in general, the second key ingredient is a convex decomposition algorithm that transform a non-convex intersection into a set of convex ones. Different algorithms have been proposed for the convex decomposition of non-convex polyhedra [39, 58]. In this work, we use polyhedron decomposition ideas for the intersection problem at hand. The original intersection problem is decomposed into a set of convex clipping problems for which one can use our convex clipping strategy.

Finally, it is essential to design mechanisms that provide robustness of the algorithm with respect to rounding errors. The main problem is the potentially huge number of clipping planes to be processed. Our methods are based on a discrete level-set representation of planes (instead of a more traditional parametric representation) and a specifically oriented graph representations of polyhedra. The motivation for this choice is to maximise symbolic computations and define geometrical operations that are numerically robust under rounding errors. We also provide techniques that identify and merge quasi-aligned planes.

The outcomes of this chapter are the following:

- A robust and efficient intersection algorithm for computing the interior of cells given a boundary mesh representation;

- The combination of the intersection algorithm with unfitted finite element methods (FEMs) for a body-fitted mesh free computational framework that is applicable to the discretisation of PDEs on explicit representations of complex geometries;
- A detailed robustness analysis of the geometrical algorithms on the Thingi10K database with about 5,000 surface meshes [123];
- The numerical experimentation of an unfitted FE solver that relies on the proposed geometrical intersection engine;
- A performance analysis of the proposed framework and an open-source implementation [76].

The outline of this chapter is as follows. In Section 2.2, we present some unfitted FE methods and their geometrical requirements. In Section 2.3, we provide the geometrical algorithm that computes the intersection of background cells and oriented surface meshes. In Section 2.4, we report a thorough numerical experimentation of the proposed algorithms on almost 5,000 meshes in the Thingi10K database [123]. We show the remarkable robustness of the geometrical algorithm, providing very low geometrical error quantities in all cases. The algorithms are combined with unfitted FE methods to approximate PDEs on these complex geometries, showing the expected convergence orders of accuracy. We also analyse the computational performance of the proposed framework and provide details about the corresponding open source implementation [76]. Finally, some conclusions and future work lines are drawn in Section 2.5.

2.2 Unfitted finite element discretisations

Let us consider an open Lipschitz domain $\Omega \subset \mathbb{R}^3$ (the 2D case is an obvious restriction) in which we want to approximate a system of PDEs. In this work, we are interested in domains that are described as the interior of an oriented surface polygonal mesh $\mathcal{B}$ of $\partial\Omega$. PDEs usually involve Dirichlet boundary conditions on Γ_D and Neumann boundary conditions on Γ_N, where Γ_D and Γ_N are a partition of $\partial\Omega$. Such partition must be respected by the geometrical representation, e.g., the STL model. Thus, we consider that $\mathcal{B}_D$ and $\mathcal{B}_N$ are geometrical discretisations of Γ_D and Γ_N, resp., and $\mathcal{B} \doteq \mathcal{B}_D \cup \mathcal{B}_N$.

Our motivation in this work is to enable the use of grid-based unfitted numerical schemes that are automatically generated from $\mathcal{B}$, due to its industrial and scientific relevance. Embedded discretisation techniques alleviate geometrical constraints, because they do not rely on body-fitted meshes. Instead, these techniques make use of a background partition $\mathcal{T}^{\text{bg}}$ of an arbitrary artificial domain Ω^{art} such that $\Omega \subset \Omega^{\text{art}}$. The artificial domain can be trivial, e.g., it can be a bounding box of Ω. Thus, the

computation of $\mathcal{T}^{\text{bg}}$ is much simpler (and cheaper) than a body-fitted partition of Ω. In this work, we consider a Cartesian mesh $\mathcal{T}^{\text{bg}}$ for simplicity in the exposition, even though the proposed approach could readily be extended, e.g., to a tetrahedral structured background mesh obtained after simplex decomposition.

The abstract exposition of unfitted formulations considered in this work is general and accommodates different unfitted FE techniques that have been proposed in the literature, e.g. the extended finite element method (XFEM) [26] (for unfitted interface problems), the cutFEM method [34] based on ghost penalty stabilisation, the AgFEM [22], the finite cell method [95] and DG methods with cell aggregation [84], to mention a few.

In order to define a FE space on unfitted meshes, we do the following cell classification. The cells in the background partition with null intersection with Ω are *exterior* cells. The set of exterior cells in $\mathcal{T}^{\text{bg}}$ is denoted by $\mathcal{T}^{\text{out}}$ is not considered in the functional discretisation and can be discarded. $\mathcal{T} \doteq \mathcal{T}^{\text{bg}} \setminus \mathcal{T}^{\text{out}}$ is the *active* mesh (see Figure 2.1(a)). The above mentioned techniques make use of standard FE spaces on $\mathcal{T}$ to define the finite-dimensional space V in which to seek the solution and also test the weak form of the PDE. The unfitted problem reads as follows: find $u \in V$ such that

$$a(u,v) = \ell(v), \qquad \forall v \in \mathcal{V},$$

where

$$a(u,v) = \int_\Omega L_\Omega(u,v)\mathrm{d}\Omega + \int_{\Gamma^{\text{D}}} L_D(u,v)\mathrm{d}\Gamma + \int_{\mathcal{F}} L_{\text{sk}}(u,v)\mathrm{d}\Gamma,$$

and

$$\ell(v) = \int_\Omega F_\Omega(v)\mathrm{d}\Omega + \int_{\Gamma^{\text{N}}} F_N(v)\mathrm{d}\Gamma + \int_{\Gamma_D} F_D(v)\mathrm{d}\Gamma.$$

The bulk terms L_Ω and F_Ω include the differential operator (in weak sense), the source term and possibly some other numerical stabilisation terms. The operators L_D and F_D integrated on Γ_D represent the terms related to the weak imposition of Dirichlet boundary conditions, e.g., the so-called Nitsche's method, which is commonly used in unfitted formulations. The term F_N on Γ_N represents the Neumann boundary conditions of the problem at hand. We denote with $\mathcal{F}$ the skeleton of the active mesh, i.e., the set of interior faces of $\mathcal{T}$. The term L_{sk} collects additional penalty terms that include weak imposition of continuity in DG methods or ghost penalty stabilisation techniques.

Since FE methods are piecewise polynomials, the integration of all this terms rely on a cell-wise decomposition (of bulk and surface terms). However, in order to respect the geometry and solve the PDE on the right domain, one must perform these integrals on domain interiors. In particular, we have

$$\int_\Omega (\cdot)\mathrm{d}\Omega = \sum_{K\in\mathcal{T}} \int_{K\cap\Omega} (\cdot)\mathrm{d}\Omega.$$

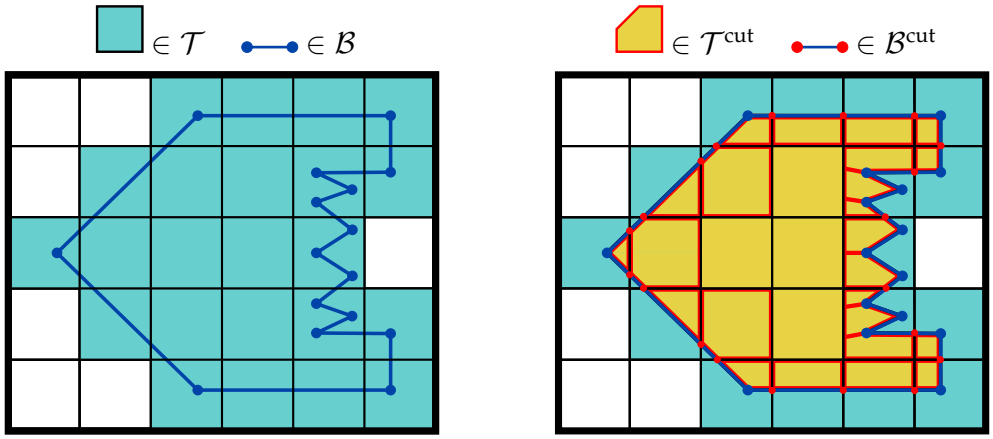


Figure 2.1: Example of an embedded non-convex domain in 2D. Left hand side figure (a) shows an active mesh $\mathcal{T}$ and an oriented skin mesh $\mathcal{B}$. The two-level meshes in (b), namely $\mathcal{T}^{\text{cut}}$ and $\mathcal{B}^{\text{cut}}$, are computed using the techniques proposed in this work to integrate unfitted formulations. Cut cells in $\mathcal{T}$ are split into a set of convex polytopes in $\mathcal{T}^{\text{cut}}$. We note that $\mathcal{T}^{\text{cut}}$ is not conforming across cells in 3D in general (only in 2D).

As commented above, the surface mesh $\mathcal{B}$ in which we aim to integrate the boundary terms and the background mesh that defines the cell-wise polynomial FE functions are not connected, i.e., one is not the boundary restriction of the other. Thus, the integration of boundary terms must be computed cell-wise as follows:

$$\int_{\Gamma_*} (\cdot) \mathrm{d}\Gamma = \sum_{K \in \mathcal{T}} \sum_{F \in \mathcal{B}_*} \int_{F \cap K} (\cdot) \mathrm{d}\Gamma, \qquad * \in \{D, N\}.$$

We note that, even though it does not represent any problem for the machinery we want to propose here, skeleton terms (common in ghost penalty and DG methods) can still be integrated on the whole skeleton faces, and there is no need in general to reduce these integrals to the domain interior.

As a result, the *only* geometrical complication of unfitted finite element schemes is the integral over $K \cap \Omega$ and $K \cap \mathcal{B}$ (or more specifically, $\mathcal{B}_N$ and $\mathcal{B}_D$). Such operations (and specially the first one) are hard for $\partial\Omega$ representing a complex shape explicitly determined by an STL model and will be the target of the next section. Let us stress the fact that the tools described below are *only used for integration purposes*. The functional spaces are defined in the background mesh, which provides lots of flexibility (no inter-cell consistency or shape regularity requirements) compared to unstructured mesh generation.

In particular, the geometrical queries that are required by unfitted formulations can be solved as follows. First, the intersection of the surface cells in $\mathcal{B}$ against all background cells in $\mathcal{T}$ produces a new surface mesh $\mathcal{B}^{\text{cut}}$ that is a refinement of $\mathcal{B}$ that describes the same geometry (up to machine precision). It can be indexed as a two-level mesh, in which first one computes its portion for a cell $K \in \mathcal{T}$, $\mathcal{B}^{\text{cut}}_K \doteq \{S \cap K : S \in \mathcal{B}\}$

and $\mathcal{B}^{\text{cut}} \doteq \bigcup_{K\in\mathcal{T}} \mathcal{B}_K^{\text{cut}}$. Thus, $\mathcal{B}^{\text{cut}}$ can readily be used for the integration of the boundary terms in (2.2); we can analogously use $\mathcal{B}_D$ (resp., $\mathcal{B}_N$) to produce $\mathcal{B}_D^{\text{cut}}$ (resp., $\mathcal{B}_N^{\text{cut}}$). In any case, $\mathcal{B}^{\text{cut}}$ preserves global conformity in 3D and can also be understood as a standard one-level polytopal mesh. Second, for each cell $K \in \mathcal{T}$, we want to compute a sub-mesh $\mathcal{T}_K$ (composed of convex polyhedra) of the interior of the cell, i.e., $K \cap \Omega$. We represent with $\mathcal{T}^{\text{cut}} \doteq \{\mathcal{T}_K : K \in \mathcal{T}\}$ the resulting two-level mesh such that $\bigcup_{K\in\mathcal{T}} \bigcup_{L\in\mathcal{T}_K} L = \Omega$ (up to machine precision). This two-level *integration* mesh can readily be used to compute the integrals in (2.2). $\mathcal{T}^{\text{cut}}$ is not conforming across background cells, since this mesh is is only needed for the numerical integration. Since the result of this algorithm is a set of meshes $\mathcal{T}_K$ composed of general convex polytopes, we can now use numerical quadratures for the integration on these polytopes. For these purposes, one can use quadrature rules on general polytopes (see, e.g., [40]) or a straightforward simplex decomposition and standard quadrature rules on triangles/tetrahedra. Figure 2.1 illustrates the construction of $\mathcal{B}^{\text{cut}}$ and $\mathcal{T}^{\text{cut}}$.

2.3 Intersection algorithms

In this section, we provide an algorithm that given the background mesh $\mathcal{T}$ and the oriented surface polygonal mesh $\mathcal{B}$ (resp., $\mathcal{B}_D$ and $\mathcal{B}_N$), it returns $\mathcal{T}^{\text{cut}}$ and $\mathcal{B}^{\text{cut}}$ (resp., $\mathcal{B}_D^{\text{cut}}$ and $\mathcal{B}_N^{\text{cut}}$). The problem when computing these meshes is the fact that the geometrical intersection and parametric distance computation algorithms are in general not robust for these purposes, due to inexact arithmetic. In this work, we aim at designing an algorithm that is robust and can be readily applied to any surface mesh with a well-defined interior. In order to attain such level of robustness, we work at different levels:

- First, we provide in Section 2.3.1 and Section 2.3.2 a computer representation of polyhedra (or surfaces) and planes, resp., that are suitable for intersection algorithms with inexact arithmetic.
- Second, in Section 2.3.3 we discuss a novel algorithm for the intersection of polyhedra and half-spaces. The algorithm is inspired by Sugihara's intersection algorithm [108] and Powell and Abell implementation in [93], but departs from these algorithms to make it suitable for our specific purposes. We provide the complete algorithm (up to minor implementation details) for the intersection of a convex polyhedron and a plane.
- Third, in Section 2.3.4 we consider the intersection of a convex polyhedron against a *non-convex* surface. In order to do that, we need to define a recursive convex decomposition algorithm that re-states the original intersection problem as a set of intersections between convex polyhedra and surfaces, for which we can use the algorithms in Section 2.3.3.

- Fourth, in order to have a robust algorithm, it is not enough with the proposed representation of polyhedra and planes and the proposed intersection algorithms. A common problem that appears in inexact arithmetic is the case of multiple planes that are quasi-aligned (conceptually, aligned up to machine precision). It has been proven that merging (enforcing the planes to be exactly aligned) dramatically improves the robustness of the overall algorithm. The approach we propose to merge planes is presented in Section 2.3.5.

- Finally, with all these ingredients, we can design the global intersection algorithm in Section 2.3.6, which returns $\mathcal{B}^{\text{cut}}$ and $\mathcal{T}^{\text{cut}}$ explained above.

2.3.1 Polyhedra and polygonal surface representations

In this work, we have to deal with hundreds (or even thousands in some limit cases) of planes clipping a cell. This situation makes robustness essential, which prevents us from using methods that require accuracy checks and the handling of all degenerate branches. For this reason, our starting point is Sugihara's method [108] and its implementation in [93]. However, the objective in [93] is to intersect a tetrahedron and a Cartesian mesh and thus restricted to a convex surface mesh and polyhedron. We propose below an algorithm that keeps robustness for a large number of clipping planes and non-convex situations.

Sugihara's method relies on the following assumption. A convex polyhedron intersected by a plane must produce two connected polyhedrons. As Sugihara pointed out in his seminal work, this is not the case in inexact arithmetic. In order to expose the problem, let us consider a polyhedron face with more than three vertices. In exact arithmetic, all these points belong to the same plane. However, this is not true in numerical computations, *co-planarity is only true up to machine precision*. (Below, we use the prefix *quasi-* to indicate a geometrical concept that is true for exact arithmetic but only approximate in finite precision.) Next, let us consider an oriented plane that is quasi-coplanar to the face up to machine precision. The classification of a vertex as interior, exterior, or on the plane is completely determined by rounding errors, thus unreliable. E.g., it can lead to a non-connected partition of the polyhedron that is impossible in exact arithmetic. In singular cases, Sugihara proposes an algorithm to re-classify the vertices on the two sides of the cutting plane based on logical arguments.

Despite Sugihara's method, we do not consider any re-classification of vertices to satisfy Sugihara's assumption in [108]. We consider an algorithm for the clipping of a polyhedron and a plane that can naturally handle possibly non-connected and non-convex outputs. In any case, the loss of convexity can only produce rounding errors and the resulting polyhedron is *quasi-convex*.

Let us start introducing some basic notation about graphs. Given a graph G, we denote with $\texttt{vert}(G)$ the set of vertices of the graph and with $\texttt{adj}(G)$ the adjacencies. The adjacency of a vertex $\alpha \in \texttt{vert}(G)$, i.e., the set of vertices connected to α by an edge

of the graph G, is denoted by $\texttt{adj}(G)(\alpha)$. We can extract the set of *connected components* (or sub-graphs) $\texttt{comp}(G)$ of a graph G.

A graph can readily be constructed from a set of vertices V and the vertices adjacencies E; we represent this construction with $\texttt{graph}(V, E)$. We can also make use of a constructor $\texttt{graph}(V, \texttt{C})$, where $\texttt{C}$ is a condition that determines whether two vertices $\alpha, \beta \in V$ are connected ($\texttt{C}(\alpha, \beta)$ is true) or not. This construction allows us to define both directed and undirected graphs, for symmetric or non-symmetric conditions, respectively.

In order to represent polyhedra, we need to make use of *rotation systems*, i.e., a subtype of graphs in which the adjacency of each vertex is a *cyclic order*. In a rotation system R, given a vertex $\alpha \in \texttt{vert}(R)$ and $\beta \in \texttt{adj}(R)(\alpha)$, there is a well-defined *previous* and *next* in $\texttt{adj}(R)(\alpha)$, defined by the cyclic ordering. Thus, we can define $\texttt{next}(\alpha; \beta)$ as the vertex after β in the cyclic ordering $\texttt{adj}(R)(\alpha)$.

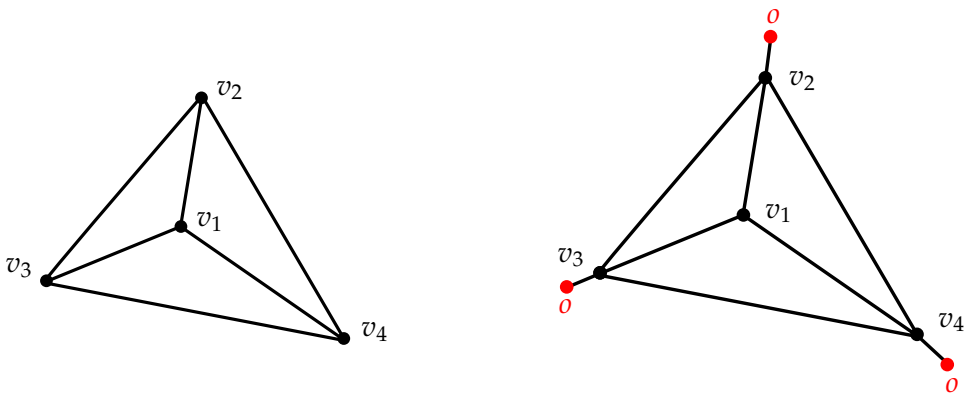
Definition 2.3.1 (Polyhedron representation)**.** *The boundary of a polyhedron P is an oriented closed surface made of polygons in which the edges around a vertex admit a cyclic ordering that encodes the surface orientation. The cyclic ordering of the adjacency (neighbours) of a vertex $\alpha \in \texttt{vert}(P)$ is determined by the clockwise ordering of edges as observed when positioned outside of P on α. Thus, a polyhedron P can be represented as a rotation system whose vertices are points in $\mathbb{R}^3$. This description of a polyhedron has been exploited in [93]. Figure 2.3 at step (i) shows a cube representation as a rotation system with clockwise ordering of neighbours.*

We can define a specific traversal of the polyhedron vertices using the definition of $\texttt{next}$ defined by the cyclic ordering above. Given an edge (α_0, α_1) of the polyhedron, subsequent vertices repeatedly applying $\alpha_{i+1} \leftarrow \texttt{next}(\alpha_i; \alpha_{i-1})$. The faces of the polyhedron are the *closed paths* determined by this graph traversal, i.e., a face is defined by α_0, α_1 and the iteration $\alpha_{i+1} \leftarrow \texttt{next}(\alpha_i; \alpha_{i-1})$ till the result is α_0; the face is a 2D polygon itself. We represent the set of faces in a polyhedron with $\texttt{faces}(P)$.

An open polygonal oriented surface $\vec{\Gamma}$ can also be represented as the polyhedron plus information about which vertices lie on the boundary, which are represented with $\texttt{bou}(\vec{\Gamma})$. It is convenient to *close* these open surfaces. We define the concept of *open* vertex o. Conceptually, o is a vertex at infinite distance of the surface and exterior to all the faces of the surface mesh $\vec{\Gamma}$. Algorithm 1 receives a surface mesh $\vec{\Gamma}$ and returns a polyhedron by modifying the surface graph by appending the artificial *open* node to the adjacency of boundary vertices (line 3). As we want closed paths to represent polyhedron faces, we need a mechanism to avoid vertices in $\texttt{bou}(\vec{\Gamma})$ to define a closed path; *open* vertices break this path. This construction is illustrated in Figure 2.2.

Algorithm 1 $\texttt{pol}(\vec{\Gamma})$

1: $V \leftarrow \texttt{vert}(\vec{\Gamma}), \quad E \leftarrow \texttt{adj}(\vec{\Gamma}), \quad \partial V \leftarrow \texttt{bou}(\vec{\Gamma}), \quad V \leftarrow V \cup \{o\}$
2: **for** $v \in \partial V$ **do**
3: $\quad E(v) \leftarrow (E(v), o)$
4: **end for**
5: **return** $\texttt{graph}(V, E)$



(a) Closed polyhedra (b) Open polyhedra

Figure 2.2: Example of Algorithm 1 that converts (a) a closed polyhedra into (b) an open surface polyhedra by adding *open* nodes to the boundary vertices $\partial V = \{v2, v3, v4\}$. The boundary vertices represent a graph cycle in (a), i.e., a polyhedron face. However, that cycle is broken in (b) because of the edges to *open* node, o. Hence, the polyhedron is open in (b) and represents a surface, as intended.

2.3.2 Half-space representations

Given an oriented plane $\vec{\pi}$, i.e., determined by a face of the polyhedron, one can define its corresponding *open* half-space as the set of points in the interior side of the plane. We use the following discrete representation of this space for a given set of vertices.

Definition 2.3.2 (Half-space representation)**.** *We represent the half-space corresponding to an oriented plane $\vec{\pi}$ using a* discrete level-set $\boldsymbol{h}$ *with respect to a set of vertices V, i.e., the set of signed distances of the vertices in V to the plane $\vec{\pi}$. $\boldsymbol{h}$ can be represented as an array of real numbers of length $|V|$. We use the convention that a negative sign means interior point (positive for exterior points). We assume that half-spaces are open and define their closure as $\overline{\boldsymbol{h}}$. We note that the only difference between the open and closed half-spaces is the definition of $\in$; vertices with zero distance belong to the closed half-space but not to the open one. Let us represent the plane with inverted orientation as $-\vec{\pi}$. Given the half-space $\boldsymbol{h}$ of $\vec{\pi}$, we define the one for $-\vec{\pi}$ as $-\boldsymbol{h}$. The complement of $\boldsymbol{h}$ is $\overline{-\boldsymbol{h}}$, i.e., $\boldsymbol{h} \oplus \overline{-\boldsymbol{h}} = \mathbb{R}^3$.*

The discrete level-set representation of a plane with respect to a set of vertices V can be determined by computing all the signed distances between vertices in V and

the plane. Let us consider a set S of planes. We represent the corresponding set of half-spaces as a *signed distance matrix* $\mathbf{H}$ in which the rows are half-spaces (the first index is the plane in S) and the columns are signed distances to all planes for a given vertex in V (the second index is the vertex). We also need to use block partitions of the matrix. E.g, if $S = X \cup Y$ and $V = W \cup Q$, we use the notation $\mathbf{H}_{XY,WQ}$ for the whole matrix while the blocks are represented using specific subscripts, e.g., the matrix block for planes X and vertices W is $\mathbf{H}_{X,W}$. We abuse of notation when dealing with polyhedra and distance matrices. E.g., given a polyhedron S, we use $\mathbf{H}_{S,*}$ instead of $\mathbf{H}_{\texttt{faces}(S),*}$ (or more accurately, the planes that contain $\texttt{faces}(S)$) and $\mathbf{H}_{*,S}$ instead of $\mathbf{H}_{*,\texttt{vert}(S)}$; the symbol $*$ means all indices in that dimension.

2.3.3 Clipping a polyhedron with a plane

We are in position to provide Algorithm 2, in which we compute the clipping of a convex polytope P against a half-space $\boldsymbol{h} \in \text{rows}(\mathbf{H})$. The result of this algorithm is (i) the new polytope obtained after clipping P with $\boldsymbol{h}$ and (ii) the new set of half-spaces $\mathbf{H}$ after eliminating $\boldsymbol{h}$, eliminating distances to vertices that are not in P anymore and adding distances to newly created vertices. The half-space $\boldsymbol{h}$ can be open or closed; it does not affect $\mathbf{H}$, since the same distances are required in both cases. The main steps in Algorithm 2 are illustrated in Figure 2.3, in which a cube is clipped by a plane.

Let us assume that $\boldsymbol{h}$ is open. The algorithm iterates over interior vertices (lines 2-3). At each vertex $\alpha \in \boldsymbol{h}$, we look for vertices in the adjacency of α that are exterior (4-5).[1] If we find an exterior vertex β in the adjacency of α. We have found an edge (α, β) that intersects the plane related to the half-space. In line 6, we compute the coordinates of the intersection vertex δ computing its coordinates as:

$$\boldsymbol{x}_\delta \doteq \xi_{\alpha\delta}\boldsymbol{x}_\alpha + \xi_{\beta\delta}\boldsymbol{x}_\beta \doteq \frac{-\boldsymbol{h}_\beta}{\boldsymbol{h}_\alpha - \boldsymbol{h}_\beta}\boldsymbol{x}_\alpha + \frac{\boldsymbol{h}_\alpha}{\boldsymbol{h}_\alpha - \boldsymbol{h}_\beta}\boldsymbol{x}_\beta.$$

This computation has also been used in [93] because it is much more robust than using an intersection algorithm between planes and edges in parametric form. The denominator is always positive, $\xi_{\alpha\delta}$ and $\xi_{\beta\delta}$ are non-negative by construction and the resulting vertex δ always lies between α and β. Furthermore, the computation of the signed distance between the new vertex δ and any half-space $\boldsymbol{h}' \in \mathbf{H}$ can readily be computed as:

$$\boldsymbol{h}'_\delta = \xi_{\alpha\delta}\boldsymbol{h}'_\alpha + \xi_{\beta\delta}\boldsymbol{h}'_\beta.$$

In order to illustrate the robustness of this approach, let us discuss what happens in the singular case in which a vertex is exactly on the intersecting plane. Using the definition above for an open half-space, this vertex is exterior. Any edge that connects it to an interior node will be intersected and a new vertex will be inserted. The new

[1] In line 4, we use the notation `enum` over an iterator to describe a new iterator that yields a tuple (i, a) in which i is a counter starting at 1 and a is the i-th value from the given iterator.

vertex distance to the planes (and coordinates) will have *exactly* the same coordinates as the vertex on the boundary, due to the expression in (2.3.3). In line 7 we create the adjacency (cyclic order) for δ with α in the first position and two additional positions not defined yet. On the other side, we replace the intersected edge (α, β) with (α, δ).

After this loop, we have identified all intersected edges, computed the new vertices after the intersection and modified the adjacencies. The adjacencies of the new vertices are not yet complete because we have not included the edges on the new face created after clipping; the edges of this face only include new vertices. We perform a new loop over new vertices in line 12. For each new vertex α, we start a graph traversal (lines 13-16) till we find another new vertex β. Then, we put β in the second position of $\texttt{adj}(P)(\alpha)$ and α in the third position of $\texttt{adj}(P)(\beta)$. When this loop finishes, we have the complete adjacencies of the new vertices.

It only remains to add to the polytope the new vertices (and their adjacencies) and to eliminate the exterior vertices (line 19-20). We also compute the signed distances of new vertices to half-spaces in $\mathbf{H}$ and eliminate the ones related to exterior vertices (inserting/removing rows to this matrix in line 21). The distance to any half-space are computed using (2.3.3). Since P has been clipped with $\boldsymbol{h}$, it is eliminated from $\mathbf{H}$.

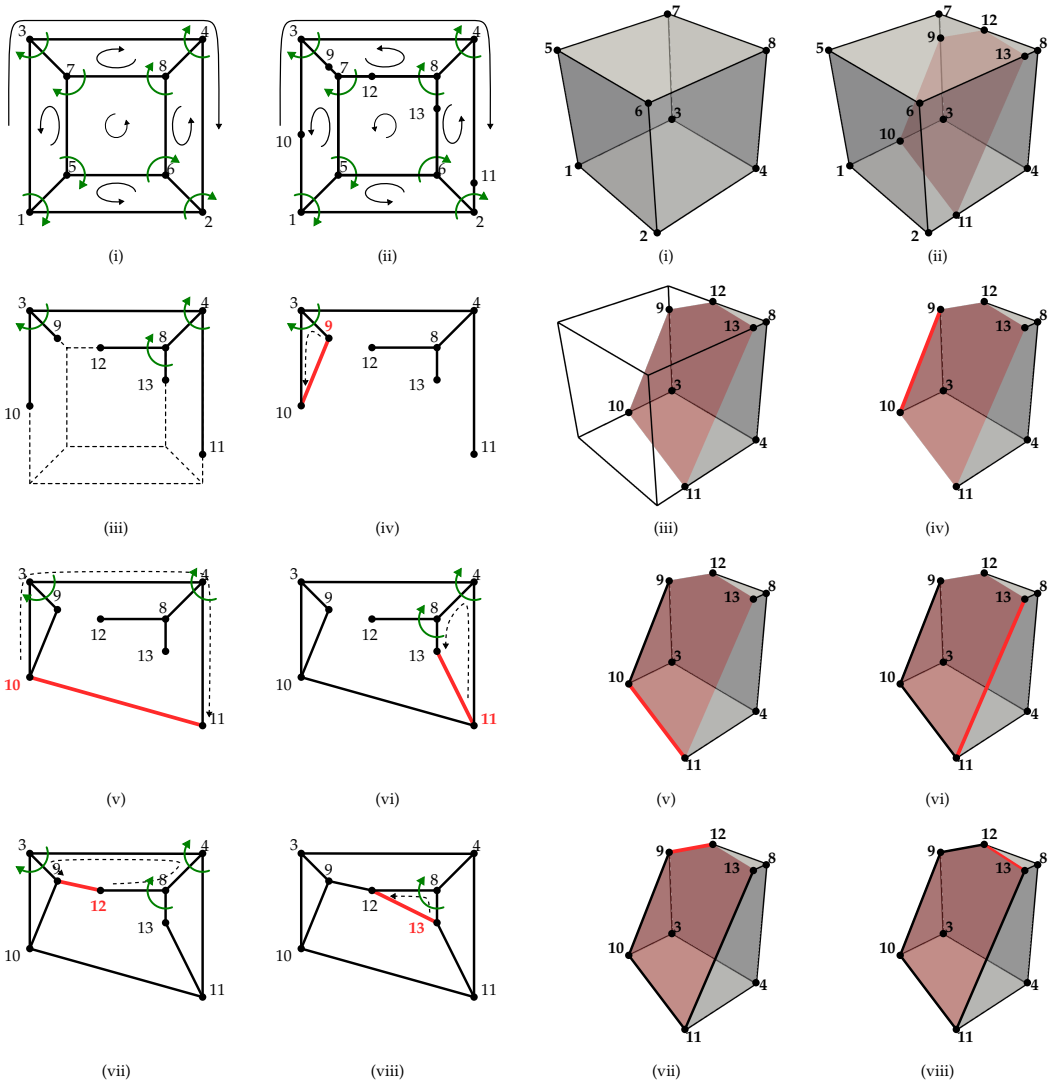
The most salient property of this algorithm is that most computations are symbolic, with the only exception of the new vertex coordinates in line 6 and new distances in line 21. However, the computation of these quantities has already been designed in such a way that they are well-posed in finite precision, using the expressions (2.3.3)-(2.3.3) discussed above.

We have considered the intersection with one half-space. But we can recursively use the algorithm to intersect with multiple planes, since we do not assume any specific topology of the initial quasi-convex polyhedron. With minor modifications, one can also extract not only the interior but also the exterior graph at the same time, reusing computations.

2.3.4 Intersecting a polyhedron with a surface

If the surface S we want to intersect with the polyhedron P is also convex, one can simply use Algorithm 2 for all the half-spaces corresponding to the faces of S. However, S is not convex for general geometries. In the final algorithm, we want to intersect an open oriented surface $\vec{\Gamma}$ (or its corresponding polyhedron representation $S \leftarrow \texttt{poly}(\vec{\Gamma})$) and a background mesh cell K. We note that, for non-convex geometries, it is not possible to avoid the appearance of cells intersected with non-convex surfaces by using standard refinement strategies.

In order to deal with general geometries, we perform a basic decomposition of the surface and the polyhedron into convex pieces [39]. First, we split the surface mesh



(a) Oriented graphs (b) Polyhedra representations

Figure 2.3: Illustration of Algorithm 2 for an example in which we intersect a polyhedron, step (i), by a half-space plane. On the left-hand side figure (a), we show the polyhedron as a rotation system and how the intersection algorithm modifies this graph during different steps of the algorithm. On the right-hand side figure (b), we show the geometrical representation of the same steps. First, the polyhedron of step (i) is defined as a rotation system as described in line 1 of the algorithm. Its edges are intersected by the half-space in step (ii) and the intersection points are computed (see line 6). Those new vertices are connected to the inside endpoints of the cut edges in step (iii) as indicated in line 7. In steps (v) to (viii), which correspond to the loop in line 12, the new vertices are connected to each other (see line 17) following an anti-clockwise path. The resulting polyhedron in step (viii) is represented with a new rotation system with all vertices inside the half-space.

Algorithm 2 $(P, \mathbf{H}) \cap \boldsymbol{h}$

1: $V_P \leftarrow \texttt{vert}(P), \quad E_P \leftarrow \texttt{adj}(P); \quad V_P^{\text{new}} \leftarrow \emptyset$
2: **for** $\alpha \in V_P$ **do**
3: **if** $\alpha \in \boldsymbol{h}$ **then**
4: **for** $(i, \beta) \in \texttt{enum}(E_P(\alpha))$ **do**
5: **if** $\beta \notin \boldsymbol{h}$ **then**
6: $\delta \leftarrow \vec{\alpha\beta} \cap \boldsymbol{h}$
7: $V_P^{\text{new}} \leftarrow V_P^{\text{new}} \cup \{\delta\}; \quad E_P \leftarrow E_P \cup \{(\delta, (\alpha, \varnothing, \varnothing))\}; \quad E_P(\alpha)[i] \leftarrow \delta$
8: **end if**
9: **end for**
10: **end if**
11: **end for**
12: **for** $\alpha \in V_P^{\text{new}}$ **do**
13: $(\beta, \delta) \leftarrow (\alpha, E_P(\alpha)[1])$
14: **while** $\delta \notin V_P^{\text{new}}$ **do**
15: $(\beta, \delta) \leftarrow (\delta, \texttt{next}(\delta; \beta))$
16: **end while**
17: $E_P(\alpha)[2] \leftarrow \delta, \quad E_P(\delta)[3] \leftarrow \alpha$
18: **end for**
19: $V^{\text{out}} \leftarrow \{\alpha \in V_P : \alpha \notin \boldsymbol{h}\}; \quad V_P \leftarrow V_P \cup V_P^{\text{new}} \setminus V_P^{\text{out}}; \quad E_P \leftarrow \{E_P(\alpha) : \alpha \in V_P\}$
20: $P \leftarrow \texttt{graph}(V_P, E_P)$
21: $\mathbf{H} \leftarrow \texttt{insert}(\mathbf{H}, V_P^{\text{new}}); \quad \mathbf{H} \leftarrow \texttt{remove}(\mathbf{H}, V^{\text{out}}); \quad \mathbf{H} \leftarrow \mathbf{H} \setminus \boldsymbol{h}$
22: **return** $(P, \mathbf{H})$

S into quasi-convex patches. In Algorithm 3 we compute a set of half-spaces that define such decomposition. If S is already a quasi-convex surface, the polytope is quasi-convex too, and we can proceed with the convex intersection as indicated above. Otherwise, we identify cells intersected by non-convex surfaces by identifying the *reflex edges* of the surfaces. A reflex edge is the one that connects faces that are not quasi-convex. They are determined by a dihedral angle larger than π. Using the half-space representation, the reflex edges can be simply determined by the signed distance matrix block $\mathbf{H}_{S,S}$, i.e., the distance of $\texttt{vert}(S)$ to the planes containing $\texttt{faces}(S)$. In line 2 we iterate over all edges and extract the faces that share each edge in line 3. An edge is reflex if the vertices of one face are exterior to the plane determined by the other face. We use this criterion to determine reflex edges in line 4. For quasi-coplanar faces, this definition can depend on the face being used to determine the plane. Besides, vertices of one face can lie on both sides of the plane, due to inexact arithmetic. In any case, these situations are not problematic when using the representations and algorithms discussed above. When two faces are quasi-aligned, considering the edge as reflex or not produces an error on the order of the machine precision. Besides, as discussed below, we consider a merging strategy to make quasi-aligned planes aligned in inexact arithmetic.

Standard methods to *convexify* (i.e., split a non-convex polyhedron into convex parts) rely on vertical *reflex walls*, i.e., vertical planes that contain the reflex edge. This algorithm is also denoted as *vertical decomposition*. See [58] for the application of this

method to polyhedra. In Algorithm 3, we do not consider a vertical wall. The definition of the plane without using information of the surface mesh is not a good choice in our case. We aim at reducing the intersections of the surface itself against these reflex walls and try to avoid quasi-aligned planes. Therefore, we consider the bisector of the two planes containing the faces sharing the reflex edge as the reflex wall (a *bisection wall*). Thus, we compute this plane for each reflex edge in line 5 of Algorithm 3. The result of this process for a surface S (in polyhedral form) is represented with $\texttt{walls}(S, \mathbf{H}_{S,S})$.

Algorithm 3 $\texttt{walls}(S, \mathbf{H}_{S,S})$

1: $R \leftarrow \emptyset$
2: **for** $e \in \texttt{edges}(S)$ **do**
3: $\quad (T, U) \leftarrow \texttt{faces}(e)$
4: $\quad$ **if** $\neg\,(\texttt{vert}(U) \subset \mathbf{H}_{T,S} \wedge \texttt{vert}(T) \subset \mathbf{H}_{U,S})$ **then**
5: $\quad\quad R \leftarrow R \cup \texttt{bisector}(T, U)$
6: $\quad$ **end if**
7: **end for**
8: **return** R

In Algorithm 4 we decompose the surface S and the polyhedron P at hand into convex parts via the recursive splitting of these by the bisection walls of S, i.e., $\texttt{walls}(S)$. In order to perform this algorithm, we need to compute first the signed matrix distance $\mathbf{H}_{SW,P}$. The first index S stands for $\texttt{faces}(S)$ planes while W for $\texttt{walls}(S)$ planes. The second index P stands for $\texttt{vert}(P)$ while S stands for $\texttt{vert}(S)$. We start the algorithm with $(\emptyset, (K, \mathbf{H}_{SW,K}), (S, \mathbf{H}_{SW,S}))$, where K is a cell in the background mesh and S the part of the whole surface mesh in touch with K. The recursivity in Algorithm 4 is illustrated in Figure 2.4, where the decomposition of both surface S and K by the corresponding walls lead to a tree of pairs of convex surface and polyhedra components.

Algorithm 4 recursively intersects a polyhedron P and surface S against the walls and returns pairs of convex polytopes and surfaces after these intersections. If in the call to this recursive function there are still walls to be processed, we recursively *convexify* P and S against each wall in lines 7-8. Note that we use closed half-space definitions for these intersections and that we convexify both sides after the intersection since both sides are of interest. We note that we can use either open or closed half-spaces in the intersections in lines 7-8.

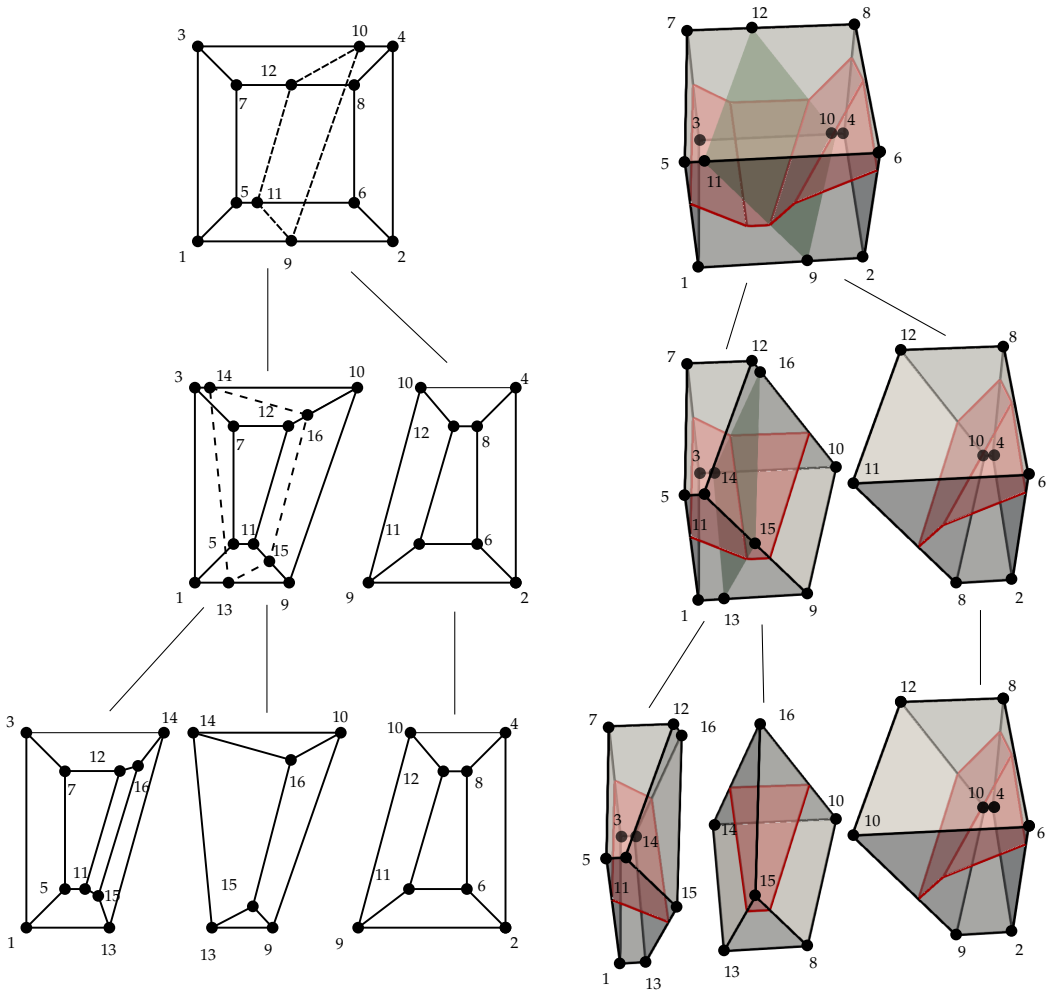
If the function is invoked with no walls, we stop the process, since we have reached the leafs of the tree. The surface component S that has been generated at this stage can still be disconnected, but what can be proved is that the connected components of S are convex. If S has disconnected components, we have to colour the surface into parts provide well-defined interiors of P. We do that in Algorithm 5. The reasoning behind this colouring is illustrated in Figure 2.5. We use these parts to colour S in line 3. We return tuples of P and each colour restriction of the surface S in line 4. By construction, the interior of P with respect to S in each tuple is convex.

Algorithm 4 $\texttt{convexify}(\mathcal{C}, (P, \mathbf{H}_{SW,P}), (S, \mathbf{H}_{SW,S}))$,

if $\mathbf{H}_{R,PS} = \emptyset$ **then**
  $S \leftarrow \texttt{colouring}(S, \mathbf{H}_{S,S})$
  return $\mathcal{C} \leftarrow \mathcal{C} \cup \{((P, \mathbf{H}_{T,P}), T) : T \in \texttt{colours}(S)\}$
else
  for $h \in \mathbf{H}_{R,PS}$ **do**
    $\mathcal{C}^+ \leftarrow \texttt{convexify}(\mathcal{C}, (P, \mathbf{H}_{SW,P}) \cap \overline{h_P}, (S, \mathbf{H}_{SW,S}) \cap \overline{h_S})$
    $\mathcal{C}^- \leftarrow \texttt{convexify}(\mathcal{C}, (P, \mathbf{H}_{SW,P}) \cap \overline{-h_P}, (S, \mathbf{H}_{SW,S}) \cap \overline{-h_S})$
    $\mathcal{C} \leftarrow \mathcal{C} \cup \mathcal{C}^+ \cup \mathcal{C}^-$
  end for
end if

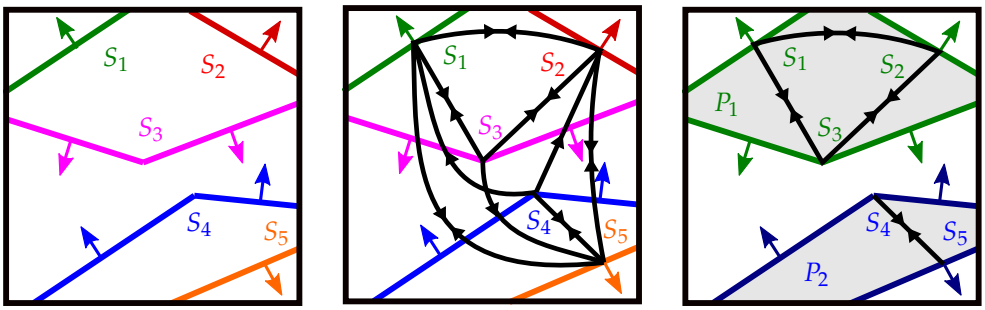
Algorithm 5 $\texttt{colouring}(S, \mathbf{H}_{S,S})$

$\vec{G} \leftarrow \texttt{graph}(\texttt{comp}(S), (T, U) \to \texttt{vert}(T) \subset \mathbf{H}_{U,S})$
$F \leftarrow \texttt{graph}(\texttt{comp}(S), (T, U) \to \texttt{vert}(U) \subset \mathbf{H}_{T,S} \wedge \texttt{vert}(T) \subset \mathbf{H}_{U,S})$
$V \leftarrow \texttt{vert}(\vec{G})$
$W \leftarrow \emptyset$
while $V \neq \emptyset$ **do**
  $v \leftarrow V[1], \quad NF \leftarrow v \cup \texttt{adj}(F)(v), \quad DG \leftarrow V \setminus (v \cup \texttt{adj}(\vec{G})(v))$
  $C \leftarrow NF \setminus \texttt{adj}(F)(DG), \quad D \leftarrow V \setminus C$
  $W \leftarrow W \cup \{C\}, \quad V \leftarrow D$
end while
$S \leftarrow \texttt{colour}(S, W)$
return S



(a) Oriented graphs (b) Polyhedra representations

Figure 2.4: Example of the application of Algorithm 4 to decomposes a cell polyhedron K and a non-convex surface S into convex parts. Both K and S are recursively split by the walls of S using Algorithm 2. The clipping of S is particularly simple since no extra vertex is introduced. Each row corresponds to a call of the algorithm. Recursion is introduced in line 7-8 of the algorithm. The result is the leaves of the tree-like decomposition, which are processed in line 3-4. In each leaf, a piece of K is associated with a convex piece of S.



(a) $\texttt{comp}(S)$ (b) $\vec{G}$ (F: bi-directed edges) (c) Colours

Figure 2.5: Illustration to explain Algorithm 5 for a 2D example, which is called in line 3 of Algorithm 4. Given a surface S with disconnected convex parts in (a), we build a directed graph G (see (b)). S_i is connected to S_j if S_i is inside S_j. Mutually connected components produce an undirected graph F. Let us discuss the first iteration of the while loop in line 5 for, e.g., v being component S_1 of S. We find mutually connected components NF to S_1 (including S_1) in line 6. $NF = \{S_1, S_2, S_3, S_5\}$ are the components that are in the interior of S_1 and S_1 is in their interior. In the same line, we find the components DG for which S_1 is outside. We get $DG = \{S_4\}$. We extract the mutually connected components to these ones, i.e., $\texttt{adj}(F)(DG)$ and extract them from NF in line 7. We get S_5 and extract if from NF to get the set $C = \{S_1, S_2, S_3\}$ that defines a convex polytope (have the same colour). We run the algorithm for the unprocessed components $D = \{S_4, S_5\}$ in the next iteration, which turn out to have the same colour. In this example, the algorithm returns two colours, namely $W = \{(S_1, S_2, S_3), (S_4, S_5)\}$, which define two convex domains P_1 and P_2. We colour the graph S with W in line 10.

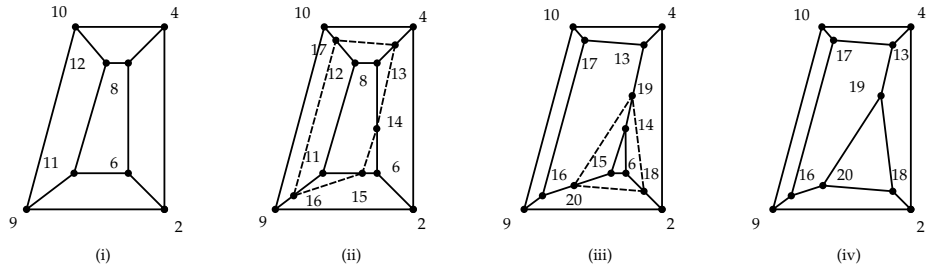
After Algorithm 4, we have a set of pairs of convex polyhedra and surfaces (P, S). We can now use Algorithm 2 for intersecting P against all the half-spaces related to the faces of S. We do this in Algorithm 6. $\mathbf{H}$ must include the signed distance between vertices in P and the half-spaces in S. The definition of open or close half-spaces depends on the definition of S (as closed set, open set, or a mixed situation). At each leaf of the tree in Figure 2.4, Algorithm 6 intersects P as in Figure 2.6.

Algorithm 6 $(P, \mathbf{H}) \cap S \rightarrow (P, \mathbf{H})$

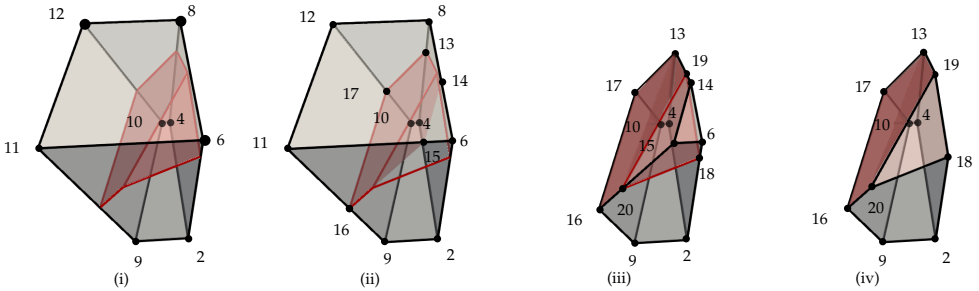
1: **for** $\boldsymbol{h} \in \mathbf{H}_S$ **do**
2: $\quad (P, \mathbf{H}) \leftarrow (P, \mathbf{H}) \cap \boldsymbol{h}$
3: **end for**
4: **return** $(P, \mathbf{H})$

2.3.5 Robust computation of signed distances

The main problem when running the previous algorithms in inexact arithmetic is the computation of the signed distance matrices when multiple planes are quasi-aligned. For complex geometries, the number of surface mesh faces intersecting a background cell can still be large. There is a chance that some of these faces and the respective walls will be quasi-aligned. Even though this is not an issue in exact arithmetic, it can be problematic in inexact arithmetic. In this section, we provide mechanisms to enforce



(a) Oriented graphs



(b) Polyhedra representations

Figure 2.6: Illustration of Algorithm 6, which intersects a convex volume P by a convex surface S. The input of step (i) represents one of the leafs of Figure 2.4. In steps (ii) and (iii) the volume polyhedra P is intersected by the half-spaces determined by $\texttt{faces}(S)$ as in the loop of line 1. The result in (iv) is the portion of P inside S. If this process is repeated for each leaf of Figure 2.4, the result is the portion of P in the interior defined by S, which is $P \cap \mathcal{B}$.

quasi-aligned half-spaces to be exactly aligned. Since we only make use of the signed distance matrix in our algorithms, the objective is to enforce the same entries for rows in $\mathbf{H}$ related to quasi-identical planes (with the same orientation) or times -1 for quasi-complimentary planes.

In a first step, we execute an algorithm $\texttt{dist}(\Pi, V)$ that returns the the signed distance matrix $\mathbf{H}$ after computing the plane to vertex signed distances using parametric representations. In this method, the distances are *snapped distances*. We define a *snap* tolerance ϵ_{sn} (e.g., 100 times the machine precision) and any distance within this tolerance is enforced to be zero. Geometrically, vertices extremely close to a plane are enforced to be on the plane in the half-space representation. It can happen that a node is snapped to multiple planes. We note that the snapping only affects the half-space representations; we do not perturb the vertices positions.

In order to make the algorithm more robust, we additionally provide a mechanism to identify half-spaces that are quasi-identical or quasi-complimentary and to make them exactly aligned in the discrete representation. Algorithm 7 merges (aligns) discrete level-set representations of half-spaces S (that represent surface faces) to the ones of a cell K of the background mesh if they are quasi-aligned. The S half-spaces, i.e., $\mathbf{H}_S$, can be perturbed in this process, but not the ones in K. Besides, the algorithm has been designed in such a way that the alignment of a half-space $\boldsymbol{h}^S$ against a half-space $\boldsymbol{h}^K$

of K is consistent among all cells containing $\boldsymbol{h}^K$. In line 4 we check whether a K half-space and an S half-space are quasi-aligned. A surface half-space $\boldsymbol{h}^S$ is aligned with a cell half-space $\boldsymbol{h}^K$ if the absolute value of their distance to all the surface vertices in S (in their discrete level-set representation) are below a given tolerance ϵ_{hs}. The absolute value is used to align not only two half-spaces that are quasi-coplanar but also the ones that are quasi-complementary. If the spaces are quasi-aligned, we run line 5. Given two quasi-aligned planes $\boldsymbol{h}^i$ and $\boldsymbol{h}^j$, $\texttt{sign}(\boldsymbol{h}^i, \boldsymbol{h}^j)$ returns +1 if they are quasi-coplanar and -1 if they are quasi-complementary. We note that this computation is numerically well-posed, e.g., comparing the sign of the distance to the furthest point in the discrete representation of the half-spaces. Finally, the S half-space is replaced by the K half-space times the sign, in order to keep consistency among cells.

Algorithm 7 $\texttt{align\_surface}(\mathbf{H}_{SK,KS}) \rightarrow \mathbf{H}_{S,KS}$

1: **for** $F_K \in \texttt{faces}(K)$ **do**
2: **for** $F_S \in \texttt{faces}(S)$ **do**
3: $\boldsymbol{h}^i \leftarrow \mathbf{H}_{F_K,S} : \boldsymbol{h}^j \leftarrow \mathbf{H}_{F_S,S}$
4: **if** $\texttt{min}(|\texttt{max.}(\boldsymbol{h}^i - \boldsymbol{h}^j)|, |\texttt{max.}(\boldsymbol{h}^i + \boldsymbol{h}^j)|) \leq \epsilon_{\text{hs}}$ **then**
5: $\mathbf{H}_{F_S,KS} \leftarrow \texttt{sign}(\boldsymbol{h}^i, \boldsymbol{h}^j) \cdot \mathbf{H}_{F_K,KS}$
6: **end if**
7: **end for**
8: **end for**
9: **return** $\mathbf{H}_{S,KS}$

Once we have aligned the surface half-spaces with the cell half-spaces, modifying $\mathbf{H}_{S,KS}$, we are in position to run the cell-wise intersection algorithms. But we still need to check whether wall and surface planes are quasi-aligned. In Algorithm 8 we provide an algorithm that aligns planes given a general signed distance matrix $\mathbf{H}$. First, the algorithm creates in line 1 a graph of half-spaces in which two half-spaces are connected if they are quasi-aligned, using the same condition as in Algorithm 7 but for all vertices in the discrete representation of the half-spaces. We extract the components of these graphs in line 2. Half-spaces in a component are considered to be all quasi-aligned and we enforce all their distances to be the same (times -1 for quasi-complementary half-spaces). The value of the distances that we have used is provided in Algorithm 9. In this algorithm, vertices that belong to one of the half-spaces in a component belong to the half-space after alignment, i.e., the distance is 0 (line 7). For other vertices, we just pick the signed distance from one of the half-spaces in line 8 (with the right sign, computed in line 4). In any case, as soon as the merge is performed, other reasonable choices could also be considered without affecting the robustness of the overall algorithm. After this process, all quasi-aligned components are exactly aligned.

Algorithm 8 $\texttt{align\_planes}(\mathbf{H}) \to \mathbf{H}$

1: $G \leftarrow \texttt{graph}(\mathbf{H}, (\boldsymbol{h}^i, \boldsymbol{h}^j) \to \texttt{min}(|\texttt{max.}(\boldsymbol{h}^i - \boldsymbol{h}^j)|, |\texttt{max.}(\boldsymbol{h}^i + \boldsymbol{h}^j)|) \leq \epsilon_{\text{hs}})$
2: $C \leftarrow \texttt{comp}(G)$
3: **for** $T \in C$ **do**
4: $\quad \texttt{merge}(\mathbf{H}_{T,*})$
5: **end for**
6: **return** $\mathbf{H}$

Algorithm 9 $\texttt{merge}(\mathbf{H}) \to \mathbf{H}$

1: $\boldsymbol{h}^0 \leftarrow \mathbf{H}[1,:]$
2: $\boldsymbol{s} \leftarrow \texttt{zeros}(\texttt{dims}(\mathbf{H})[1])$
3: **for** $(i, \boldsymbol{h}) \in \texttt{enum}(\texttt{rows}(\mathbf{H}))$ **do**
4: $\quad \boldsymbol{s}[i] \leftarrow \texttt{sign}(\boldsymbol{h}, \boldsymbol{h}^0)$
5: **end for**
6: **for** $\boldsymbol{c} \in \texttt{columns}(\mathbf{H})$ **do**
7: $\quad (\texttt{min}(\texttt{abs.}(\boldsymbol{c})) = 0)\ ?\ d \leftarrow 0 : d \leftarrow \boldsymbol{c}[1]$
8: $\quad \boldsymbol{c} \leftarrow d \cdot \boldsymbol{s}$
9: **end for**
10: **return** $\mathbf{H}$

2.3.6 Global intersection algorithm

We are in position to define Algorithm 10, the global algorithm we propose to intersects a background mesh $\mathcal{T}$ and a boundary mesh $\mathcal{B}$. The results is a partition of each cell in both meshes into sub-cells, denoted with $\mathcal{T}^{\text{cut}}$, $\mathcal{B}^{\text{cut}}$. Figure 2.9 illustrates all the steps being performed in this algorithm to intersect a cell $K \in \mathcal{T}$ with the boundary mesh $\mathcal{B}$.

First, we perform a background cell-wise intersection (see line 3). In general, the surface mesh can have a large number of cells but a background cell usually intersects a very small portion of these surface cells. For computational efficiency and robustness of the algorithm, it is essential to reduce the polyhedron clipping to the portion of the surface $\mathcal{B}$ that can be in touch with the cell. This step is denoted with `restrict` in line 3. It makes use of cheap geometrical predicates, since the result does not need to be precise; false positives do not pose any problem. In fact, in order to capture cells that are quasi-aligned to faces in the background cell K, we need to enlarge K at least a distance equal to ϵ_{hs}. Since these predicates are quite standard in computational geometry and can be found in computational geometry libraries like CGAL [112], they are not included here for the sake of conciseness.

After the restriction, we transform the portion of the surface mesh into a polyhedron in line 4, using Algorithm 1. In line 5, we compute the signed distance matrix between the vertices in $\texttt{vert}(K) \cup \texttt{vert}(S)$ and the planes in $\texttt{faces}(K) \cup \texttt{faces}(S) \cup \texttt{walls}(S)$ using standard algorithms.

The signed (snapped) distance matrix for all faces of K and S, and walls of S, and vertices in K and S is computed in line 5. Next, we align the surface half-spaces to the ones of the cell boundaries in a consistent way using Algorithm 7 in line 6.

Given the rectangular cell $K = [x^-, x^+] \times [y^-, y^+] \times [z^-, z^+]$, we define $K_{\circ\bullet} = (x^-, x^+] \times (y^-, y^+] \times (z^-, z^+]$. In order to perform the surface mesh cell-wise intersection, we precisely use $K_{\circ\bullet}$, not K, in line 7 using Algorithm 6. Otherwise, faces that lie on background cell boundaries would be processed twice. The result of this surface-cell intersection (line 8) for all cells returns a refinement of $\mathcal{B}$, denoted with $\mathcal{B}^{\text{cut}}$. Such intersection is illustrated in Figure 2.8.[2] We can readily use $\mathcal{B}_D$ and $\mathcal{B}_N$ instead, to compute $\mathcal{B}_D^{\text{cut}}$ and $\mathcal{B}_N^{\text{cut}}$.

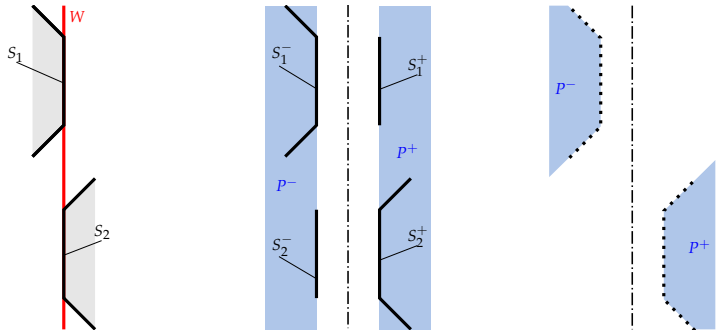


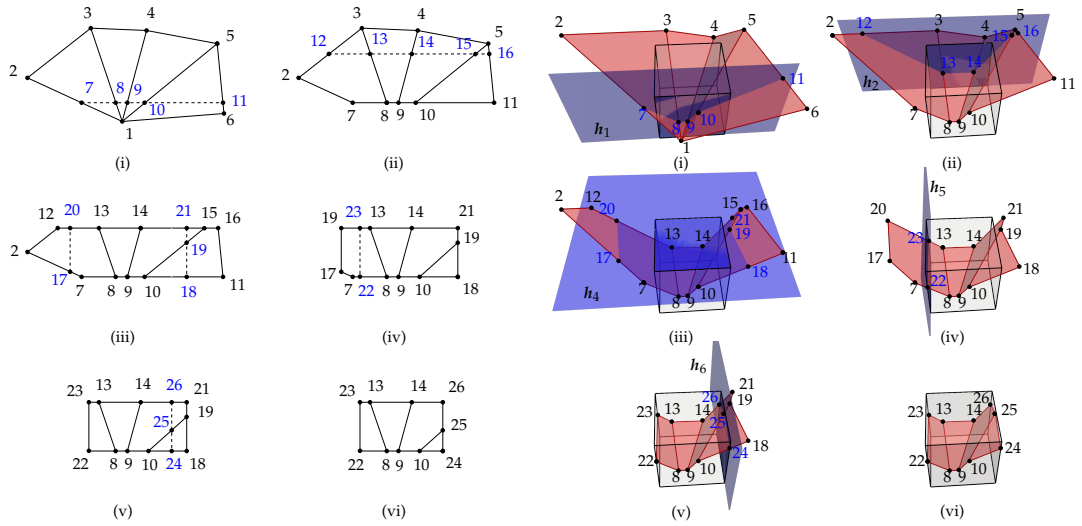
Figure 2.7: Simple 2D example to justify the choice of open and closed half-spaces in Algorithm 4 and 10. In (a) S_1 and S_2, surface faces are aligned with the wall W. When decomposing by W with closed half-spaces, i.e., $-\overline{\boldsymbol{h}_W}$ and $\overline{\boldsymbol{h}_W}$ in lines 7-8 of Algorithm 4, S_1 and S_2 are repeated in both sides as it is shown in (b). However, when intersecting $P \cap S$, since $\boldsymbol{h}_S$ is open in line 11 of Algorithm 10, S_1^+ (resp., S_2^-) does not belong to the open half-space $\boldsymbol{h}_{S_1}$ (resp., $\boldsymbol{h}_{S_2}$) and thus eliminated after the intersection. The resulting polyhedra after clipping are shown in (c). This specific choice is required in the case in which one wants to extract the boundary surface from the clipped polytopes, i.e., $\mathcal{B}^{\text{cut}} \leftarrow \partial\mathcal{T}^{\text{cut}}$.

In this intersection, since the half-spaces related to $\texttt{faces}(K)$ are processed, they are eliminated from $\mathbf{H}$ (see Algorithm 2). Finally, we merge quasi-aligned surface and wall half-spaces using Algorithm 8. With the resulting signed distance matrix, we run the convex decomposition in Algorithm 4 in line 10, starting with the polyhedron K and surface S. We note that this line is doing nothing if there are no walls, i.e., if the polytope is already quasi-convex. The resulting convex polyhedron-surface components are intersected using Algorithm 2 and added to the sub-mesh for K that represent its interior part in line 11.

[2] We note that one could also extract the surface mesh as the surface of the clipped polytopes obtained after intersecting again half-spaces in $\texttt{faces}(S)$ in line 12. This is the reason why we use $K_{\bullet\circ}$ in line 7 (to process surface faces aligned with background cells faces only once), closed spaces in lines 7-8 of Algorithm 4 (not to lose any surface face after splitting with wall half-spaces) and intersection against open half-spaces related to the surfaces in line 11 (to discard zero volume components after this decomposition and count surface faces on walls only once). A simple example in which we can encounter this situation is illustrated in Figure 2.7. In any case, these choices of open/closed half-spaces are not required when extracting the surface mesh as in line 8. The connection between the interior partition $\mathcal{T}^{\text{cut}}$ and $\mathcal{B}^{\text{cut}}$ is not important for the unfitted scheme being used later on because Dirichlet boundary conditions are weakly imposed. In any case, it could be useful for other embedded methods that would make use of a strong imposition of Dirichlet data.

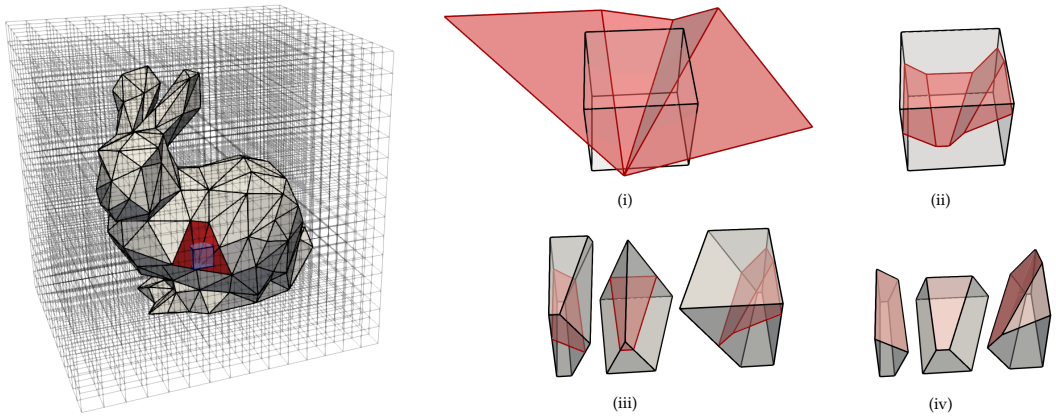
Algorithm 10 $\mathcal{T} \cap \mathcal{B} \to \mathcal{T}^{\text{cut}}, \mathcal{B}^{\text{cut}}$

1: $\mathcal{T}^{\text{cut}} \leftarrow \emptyset$; $\mathcal{B}^{\text{cut}} \leftarrow \emptyset$,
2: **for** $K \in \mathcal{T}$ **do**
3: $B \leftarrow \texttt{restrict}(\mathcal{B}, K)$
4: $S \leftarrow \texttt{pol}(B)$
5: $\mathbf{H}_{KSW,KS} \leftarrow \texttt{dist}([\texttt{faces}(K), \texttt{faces}(S), \texttt{walls}(S)], [\texttt{vert}(K), \texttt{vert}(S)])$
6: $\mathbf{H}_{S,KS} \leftarrow \texttt{align\_surface}(\mathbf{H}_{S,KS}, \mathbf{H}_{K,KS})$
7: $(S, \mathbf{H}_{S,KS}) \leftarrow (S, \mathbf{H}_{KS,KS}) \cap K_{\circ\bullet}$
8: $\mathcal{B}^{\text{cut}} \leftarrow \mathcal{B}^{\text{cut}} \cup S$
9: $\mathbf{H}_{SW,KS} \leftarrow \texttt{align\_planes}(\mathbf{H}_{SW,KS})$
10: $\mathcal{C} \leftarrow \texttt{convexify}(\emptyset, (K, \mathbf{H}_{SW,K}), (S, \mathbf{H}_{SW,S}))$
11: $\mathcal{T}_K \leftarrow \{P \cap S : ((P, \mathbf{H}_{S,P}), S) \in \mathcal{C}\}$; $\mathcal{T}^{\text{cut}} \leftarrow \mathcal{T}^{\text{cut}} \cup \mathcal{T}_K$
12: **end for**
13: **return** $\mathcal{T}^{\text{cut}}$, $\mathcal{B}^{\text{cut}}$



(a) Oriented graphs (b) Polyhedra representations

Figure 2.8: Illustration of Algorithm 6, i.e., $(S, \mathbf{H}) \cap K$, when calling line 7 of Algorithm 10. We consider S and K from step (i) in Figure 2.9. In this process, step (i) to (v), S is intersected by each half-space $\boldsymbol{h}_i \in \mathbf{H}_{S,K}$ related to $\texttt{faces}(K)$ using Algorithm 2, which is called in the loop of line 1 of Algorithm 6. Note that $\boldsymbol{h}_3$ (bottom plane) is excluded from the figure because the intersection is meaningless. The result, step (vi), is a new surface S inside K, which is introduced in step (ii) of Figure 2.9.



(a) STL and cell sample (b) Main local steps

Figure 2.9: Illustration of Algorithm 10, which, given an STL $\mathcal{B}$ and a background mesh $\mathcal{T}$ (see (a)) intersects each background cell $K \in \mathcal{T}$ with $\mathcal{B}$. The steps described in (b) represent the loop in line 2. First, $\mathcal{B}$ is restricted to the faces touching K in (i) (line 3) and defined as a polyhedron S (line 4). Next, S is intersected by the half-spaces bounding K performed in step (ii) (see line 7). As the surface S may be non-convex, it is decomposed into convex parts in step (iii), as described in line 10, together with K. Finally, in line 11, each convex component P of K is intersected by the corresponding part of S. The result in step (iv) is $K \cap \Omega$ described as the union of convex polyhedra, represented as in Definition 2.3.1. Steps (ii), (iii) and (iv) are further detailed in Figure 2.8, Figure 2.4 and Figure 2.6 respectively.

Even though we have presented the algorithm the interior component only, i..e, $K \cup \Omega$, it is computationally efficient to compute the convex decomposition of both interior and exterior at the same time when the latter is needed, e.g., in interface problems. We also note that the definition of interior, exterior and boundary vertices is a straightforward side-result of the algorithm. The interior (resp., exterior) is determined in line 11 for cut cells and propagated globally to other interior (resp., exterior) cells.

Some algorithms, e.g., the numerical integration, could require a simplex decomposition of the polyhedra that define the interior or exterior. A convex decomposition of a convex polytope is straightforward and can be computed symbolically (see [93] for details). In any case, this step is optional. Even for numerical computations, one can use quadrature rules for general polytopes that do not require this step [40].

2.4 Numerical experiments

2.4.1 Objectives

In the numerical examples below, we analyse the algorithmic and computational performance of the intersection algorithm proposed in this chapter. In particular, we study the accuracy of the intersection method, its robustness, the scaling of CPU times with respect to the number of cells in the background mesh and the faces of the STL and its

usage in unfitted FE simulations. We consider three different numerical experiments. In the first one (Section 2.4.3), we run the intersection algorithm in a large set of geometries taken form the Thingi10K [123] collection of STL models. We apply the method to all models in this data-set that fulfil the input requirements of the intersection algorithm in order to evaluate its ability to deal with complex and arbitrary inputs. In the second experiment (Section 2.4.4), we analyse the robustness of the method with respect to perturbations in the background mesh, either with translations or rotations. And finally (Section 2.4.5), we apply the proposed intersection method to generate integration cells in an unfitted FE method to analyse the influence of the cutting algorithm in the quality of the FE solution. We have performed the simplex decomposition step in all experiments.

2.4.2 Experimental setup

The numerical experiments have been performed on TITANI, a medium size cluster at the Universitat Politècnica de Catalunya (Barcelona, Spain) and on Gadi, a high-end supercomputer at the NCI (Australia) with 3024 nodes, each one powered by a 2 x 24 core Intel Xeon Platinum 8274 (Cascade Lake) at 3.2 GHz and 192GB RAM. The timing experiments have been performed on Gadi exclusively, whereas TITANI has been considered for non-performance critical runs. In order to reduce the influence of external factors on the CPU timings, each time measure reported in the experiments is computed as the minimum of 5 runs in the same Julia session, i.e., one run for Julia just-in-time (JIT) compilation and four runs to measure run-time performance. The intersection algorithms have been implemented using the Julia programming language [27] and are freely available in the STLCutters.jl package [76]. The unfitted FE computations have been performed using the Julia FE library Gridap.jl [21] version 0.16.3 and the extension package for unfitted methods GridapEmbedded.jl [118] version 0.7. In order to parse the STL files, we have used the MeshIO.jl [41] Julia package version 0.4.

2.4.3 Batch processing the STL models of the Thingi10K data-set

We start the numerical experiments by processing a large number of real-world STL models to show the capacity of the proposed intersection algorithm to deal with complex and arbitrary data automatically. To this end, we consider the Thingi10K [123] data-base, which contains ten thousand 3D STL models, from simple to very complex, used mainly for real-world 3D printing purposes. Our goal is to show that our intersection algorithms are able to handle these geometries automatically and directly without any manual pre-process as a demonstration of the robustness and generality of the proposed algorithm.

Among all models within the Thingi10k set, we process the ones that fulfil the requirements of our method. In particular, we need closed surfaces that define a volume.

Not all geometries in the database fulfil this condition and, thus, we extract valid geometries by considering the ones tagged as *is closed* and *is manifold*. E.g., one can recover these geometries by typing "is closed, is manifold" in the search field of the Thingi10k web page. This results in a subset of 4963 models. Among them, we have found 211 cases that could not be processed either due to broken download links or corrupt STL files (i.e., the parser was not able to read the model into memory) and 20 cases that are not a manifold up to machine precision. By discarding these pathological cases, we recovered the 4732 geometries that have been processed in this test. As an example, Figure 2.10 shows some of the processed STL models, which illustrates the diversity of cases analysed in this experiment.

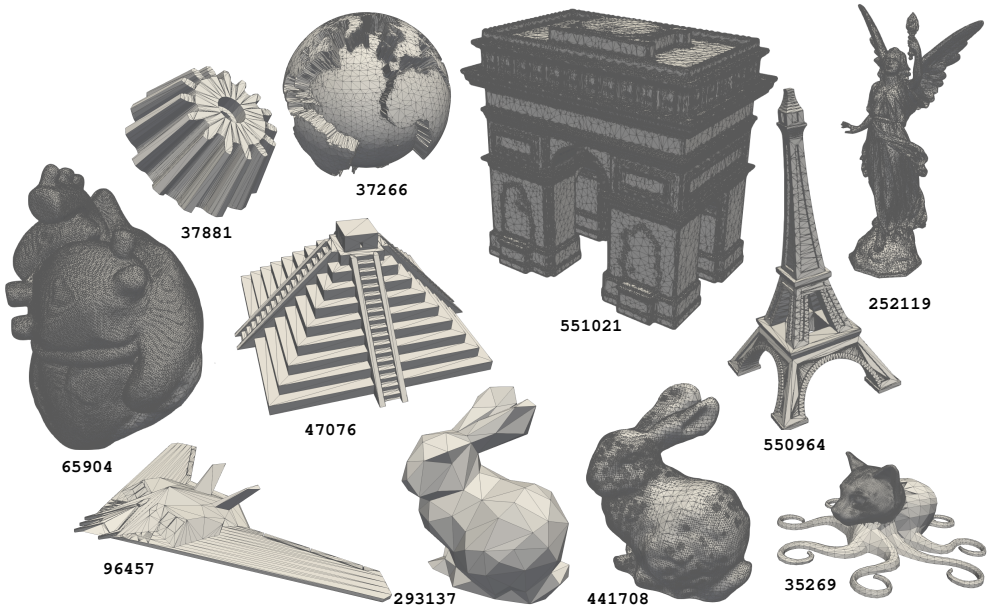


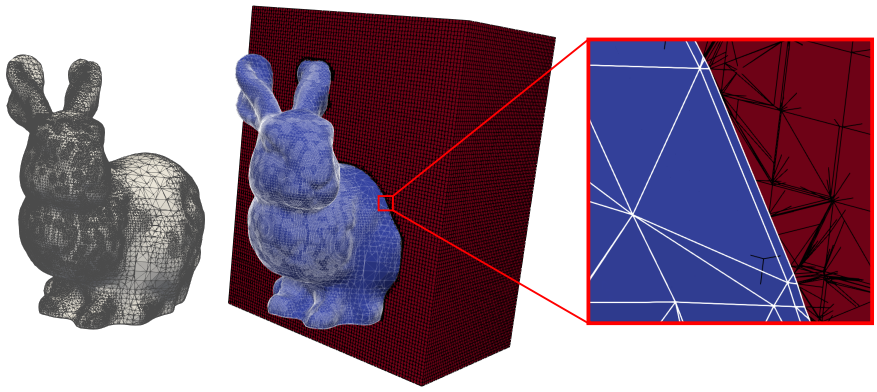
Figure 2.10: Selection of 11 STL models from the Thingi10K database processed in the numerical examples. They are displayed with their corresponding model id provided by the Thingi10K database.

Each of the considered models is processed automatically as follows. First, we parse the downloaded STL file and compute its bounding box extreme points $\mathbf{p}_{\text{stl}}^{\text{min}}$ and $\mathbf{p}_{\text{stl}}^{\text{max}}$. Then, we generate a 3D background Cartesian mesh, which covers a box approximately 40% larger in each direction than the bounding box of the STL. We generate the Cartesian mesh with at least $n^{\text{max}} = 100$ cells in the largest axis and $n^{\text{min}} = 10$ in the shortest. The element size h and the bounding box points of the background Cartesian mesh, $\mathbf{p}_{\text{msh}}^{\text{min}}$ and $\mathbf{p}_{\text{msh}}^{\text{max}}$, are respectively computed as

$$h = 1.4 \min \left\{ \max \left(\frac{\mathbf{p}_{\text{stl}}^{\text{max}} - \mathbf{p}_{\text{stl}}^{\text{min}}}{n^{\text{max}}} \right), \min \left(\frac{\mathbf{p}_{\text{stl}}^{\text{max}} - \mathbf{p}_{\text{stl}}^{\text{min}}}{n^{\text{min}}} \right) \right\},$$

and

$$\mathbf{p}_{\text{msh}}^{\text{min}} \doteq \mathbf{p}_{\text{stl}}^{\text{min}} - 0.2\left(\mathbf{p}_{\text{stl}}^{\text{max}} - \mathbf{p}_{\text{stl}}^{\text{min}}\right), \quad \mathbf{p}_{\text{msh}}^{\text{max}} \doteq \mathbf{p}_{\text{msh}}^{\text{min}} + \left\lceil \frac{1.4\left(\mathbf{p}_{\text{stl}}^{\text{max}} - \mathbf{p}_{\text{stl}}^{\text{min}}\right)}{h} \right\rceil h.$$

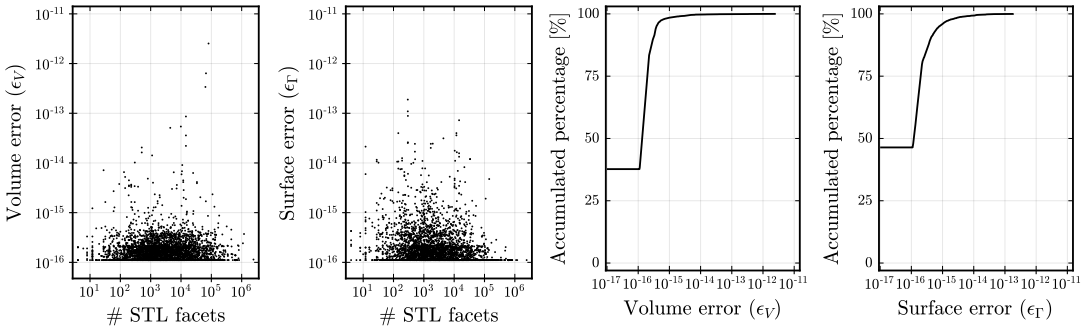


(a) (b)

Figure 2.11: Generated volume sub-triangulation for the STL model with id 441708: (a) shows the original STL geometry, while (b) shows a clipped portion of the volume sub-triangulation (red cells) with a detail of the cut cells near the STL faces defining the boundary (blue faces).

In a next step, the background mesh is intersected with the STL surface mesh using Algorithm 10. See, e.g., in Figure 2.11 a detail of the resulting volume sub-triangulation for one of the considered geometries. The intersection algorithm is applied with snap tolerance $\epsilon_{\text{sn}} = \ell_{\mathcal{B}}^{\text{max}} 10^2 \text{eps}$ and quasi co-planar tolerance $\epsilon_{\text{hs}} = \ell_{\mathcal{B}}^{\text{max}} 10^3 \text{eps}$, being $\ell_{\mathcal{B}}^{\text{max}}$ the length of the largest axis of the STL bounding box and eps the machine precision associated with 64-bit floating point numbers. The final step is to compute some indicators of the quality of the generated sub-triangulations. On the one hand, we measure Γ^{st}, the area of the boundary sub-triangulation and compare it with Γ^{STL}, the area of the original STL mesh. From these values, we compute the relative surface error $\epsilon_\Gamma = |\Gamma^{\text{STL}} - \Gamma^{\text{st}}|/\Gamma^{\text{STL}}$. As the input geometries are represented by surfaces, the original interior volume is unknown. Thus, we quantify the volume error by comparing the inside and outside volumes of the bulk sub-triangulation, V^{in} and V^{out} respectively, and compare it with the volume of the bounding box, V^{box}, leading to the relative volume error $\epsilon_V = \left|V^{\text{in}} + V^{\text{out}} - V^{\text{box}}\right| / V^{\text{box}}$.

Figure 2.12 reports the computed errors ϵ_Γ and ϵ_V for all processed geometries. Note that the intersection algorithm is able to successfully finish in all cases with relative volume and surface errors below 10^{-11} and 10^{-12} respectively, which confirms that the method is able to capture the given STL models accurately. Note also that the computed errors do not depend on the number of STL faces, even for geometries with millions of faces (see Figure 2.12(a) and 2.12(b)). In addition, the volume and surface errors ϵ_Γ and ϵ_V are below 10^{-15} for the virtual majority of cases (see Figure 2.12(c) and



(a) (b) (c) (d)

Figure 2.12: Volume and surface error distributions: (a) and (b) shows volume and surface errors vs the number of STL faces. Each single dot represents a geometry. The cumulative frequency of the volume and surface errors are represented in (c) and (d) respectively.

2.12(d)), which demonstrates that the algorithms are able to capture the given STL geometries exactly up to the tolerances as expected. Taking into account the large number and variety of STL models considered, the results of this experiment clearly show that the proposed intersection Algorithm 10 is able to deal with complex and arbitrary data automatically and provide volume and surface triangulation that capture the input STL exactly up to tolerances and close to machine precision.

2.4.4 Robustness test

In this second experiment, we study a sub-set of the models in the Thingi10K database in more detail to assess the robustness of the proposed method with respect to perturbations in the background mesh. We consider the STL geometries displayed in Figure 2.10 plus a toy STL model of a cube that will serve as a reference. These STL geometries are specifically chosen to cover a large range of shapes and model sizes, while keeping the number of considered cases relatively small in order to make feasible the computation of this example with the computational resources we have at hand. Table 2.1 contains a summary of the main features of the analysed geometries.

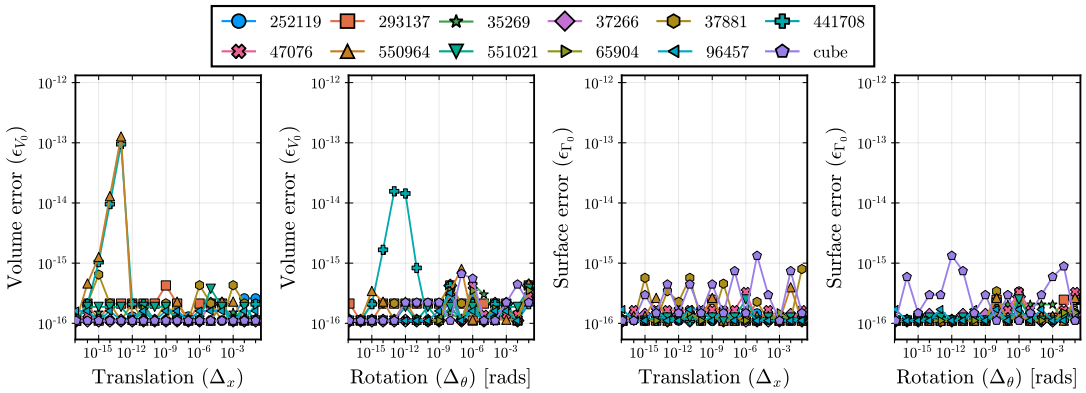
The setup of this experiment is as follows. For each STL model, we generate different background meshes by perturbing an initial grid, either using translations or rotations. The initial (unperturbed) mesh for a given STL is generated as in previous experiment, but now taking $n^{\max} = 112$. This value is chosen to stress the algorithm for the reference cube geometry since it leads to faces of the background mesh to be exactly aligned with the faces of the STL model. In this scenario, small perturbations of the background mesh lead to volume sub-triangulations with arbitrary small cells, which is a challenging degenerated case. In this regards, we want to analyse how the method behaves, when the perturbation magnitude approaches the machine precision. The first perturbation strategy is to apply a prescribed translation in all directions with magnitude $(\mathbf{p}_{\text{msh}}^{\max} - \mathbf{p}_{\text{msh}}^{\min})\Delta x$, where Δx is the perturbation coefficient computed

Model id	Num faces	Num vertices	Box Size	Surface
252119	49950	24979	(65.06, 37.371, 111.76)	12439.27
293137	292	148	(108.12, 86.625, 107.26)	29490.72
35269	40246	20125	(92.951, 93.426, 33.648)	8849.629
37266	29472	14738	(56.527, 52.541, 53.059)	13112.22
37881	3400	1700	(33.504, 33.688, 18.5)	3992.616
441708	112402	56203	(107.75, 87.802, 107.89)	29684.95
47076	1532	768	(93.095, 93.095, 42.0)	21864.75
550964	6156	3072	(20.0, 20.0, 45.0)	1715.988
551021	348128	174066	(38.365, 25.963, 37.438)	7282.98
65904	157726	78869	(532.5, 552.51, 490.47)	773637.9
96457	1634	813	(195.64, 120.13, 20.549)	21396.49
cube	12	8	(1.0, 1.0, 1.0)	6.0

Table 2.1: Main features of the test geometries considered displayed in Figure 2.10.

as $\Delta_x = 10^{-\alpha}$ with $\alpha = 1, \ldots, 17$. The second perturbation strategy is an imposed rotation composed by three individual rotations of angle Δ_θ, one over each Cartesian axis, taking the STL bounding box barycentre as the origin.

The perturbation angle is $\Delta_\theta = 10^{-\alpha}$ with $\alpha = 1, \ldots, 17$. As a result, we consider 34 different background meshes (17 translated + 17 rotated) for each of the STL models considered in this example. Finally, we run the intersection Algorithm 10 and compute the resulting volume and surface errors with respect to the non perturbed state ϵ_{Γ_0} and ϵ_{V_0}, which are defined as $\epsilon_{\Gamma_0} = |\Gamma - \Gamma_0|/\Gamma_0$ and $\epsilon_{V_0} = |V - V_0|/V_0$, where Γ and V are the respective surface at each point, and Γ_0 and V_0 are the respective surface and volume computed with the unperturbed mesh. In contrast to Figure 2.12, here we can take advantage a reference volume.



(a) (b) (c) (d)

Figure 2.13: Results of the robustness test: Volume and surface errors ϵ_{Γ_0} and ϵ_{V_0} in function of the perturbation coefficients Δ_x and Δ_θ for all the geometries of Figure 2.10.

As displayed in Figure 2.13, the volume and surface errors ϵ_{Γ_0} and ϵ_{V_0} are nearly independent on the perturbation coefficients and are below 10^{-15} in almost all cases.

Some outliers show some influence on the perturbation coefficients but the maximum volume and surface errors are below 10^{-13}, which is still close to the machine precision and can be attributed to propagation of round-off errors an the value of the tolerance ϵ_{hs}. Note that the errors for the cube geometry are always below 10^{-15} even though this test has been explicitly designed to render very pathological cases, when the perturbation coefficients tend to zero. At the view of these results, one can conclude that the quality of the computed sub-meshes is nearly independent to the location of the background mesh and, thus, the method is robust to perturbations.

2.4.5 Finite Element convergence test

In this last experiment, we explore the capacity of the proposed intersection algorithm to be coupled with unfitted FE methods in order to simulate complex geometries described by STL models without generating conforming unstructured grids. The main goal of this experiment is to check that the intersection algorithm does not introduce any spurious numerical artifacts that destroy the optimal convergence of the FE solver. We will also leverage this convergence test to evaluate the performance of the intersection method by studying the scaling of CPU times with respect to the number of cells in the background mesh.

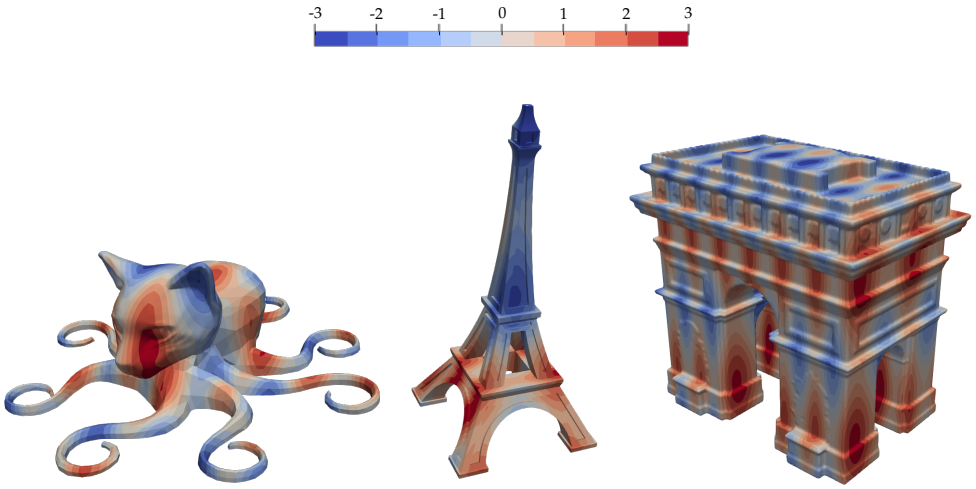
For the FE computation, we consider a Poisson equation with pure Dirichlet boundary conditions as the model problem. A numerical approximation $u_h \approx u$ is computed with the AgFEM method described in [22] for exactly the same model problem. In particular, the interpolation spaces are defined with continuous tri-linear Lagrangian shape functions. As an example, see in Figure 2.14, FE approximations computed on a sub-set of the studied STL models.

For the convergence test, the forcing term and Dirichlet boundary condition are defined such that the manufactured function $u(x,y,z) = x^2 + y^2 - z^2$ is the exact solution of the problem. Since this function is smooth and does not belong to the interpolation space, we expect that the H^1 and L^2 norms of the discretisation error $e_h \doteq u - u_h$, namely

$$\|e_h\|^2_{L^2(\Omega)} \doteq \int_\Omega e_h^2 \, \mathrm{d}\Omega \quad \text{and} \quad \|e_h\|^2_{H^1(\Omega)} \doteq \int_\Omega e_h^2 + \nabla e_h \cdot \nabla e_h \, \mathrm{d}\Omega,$$

converge with the optimal convergence rate. Our goal is to compute these error norms for different mesh sizes and confirm that they converge with the optimal slopes.

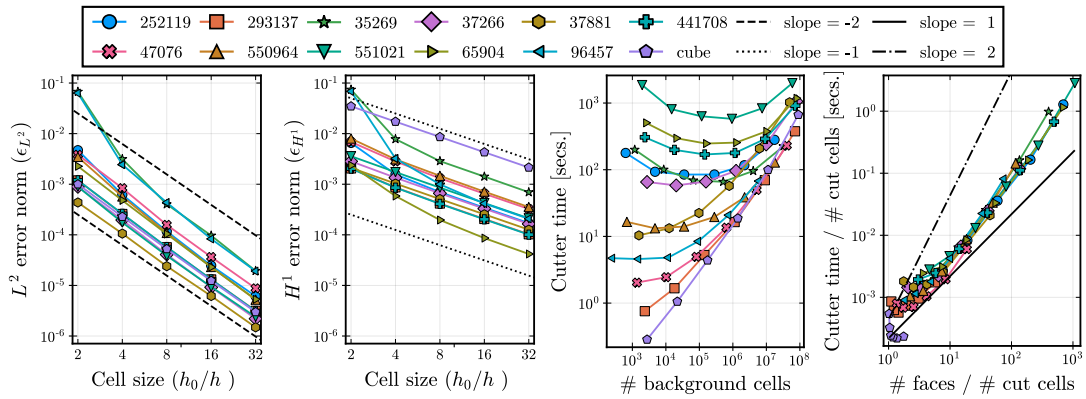
We build a family of background meshes for each STL model in Figure 2.10. Each mesh is generated by using a different value of $n^{\max}$, leading to several refinement levels. In particular, we use $n^{\max} = n_0^{\max} 2^\beta$ with $n_0^{\max} = 14$ and $\beta = 0, \ldots, 5$. The finest meshes generated in this way ($\beta = 5$) have 448 cells in the largest axis. In order to be able to solve the underlying system of linear algebraic equations for such problem sizes, we consider a conjugate gradient solver preconditioned with the Algebraic Multi-Grid (AMG) method in the Preconditioners.jl package version 0.3 [110]. We declare



(a) 35269 (b) 55094 (c) 551021

Figure 2.14: FE approximation computed with AgFEM on top of three of the STL models analysed in the experiments. Here, the underlying Poisson equation is defined using the manufactured solution $u(x,y,z) = sin(a\frac{2\pi}{T}x) + sin(b\frac{2\pi}{T}y) + sin(c\frac{2\pi}{T}z)$ with $T = 10h$, $(a,b,c) = (1,\frac{1}{2},\frac{1}{4})$, $h = \frac{1.4}{n^{\max}}\max(\mathbf{p}_{\text{stl}}^{\max} - \mathbf{p}_{\text{stl}}^{\min})$ and $n^{\max} = 100$.

convergence of the conjugate gradient solver, when the relative energy norm is below 10^{-10}.



(a) (b) (c) (d)

Figure 2.15: Results of the FE convergence test: (a), (b), and (c) show the L^2, and H^1 error norms, and CPU times in Algorithm 10 vs relative cell size for all STL models in Figure 2.10. (c) shows the scaling of CPU times vs number of STL faces per cut cell in the background mesh.

The results of the convergence test are displayed in Figure 2.15. The L^2 and H^1 error norms converge with the expected slopes for all the considered STL geometries, which confirms that the intersection algorithm is not affecting the quality of the FE solver. On the other hand, we measure the CPU time elapsed in the computation of the intersection Algorithm 10. Figure 2.15(c) shows the scaling of the CPU time with respect to the number of cells in the background mesh. The scaling tends to be linear as the

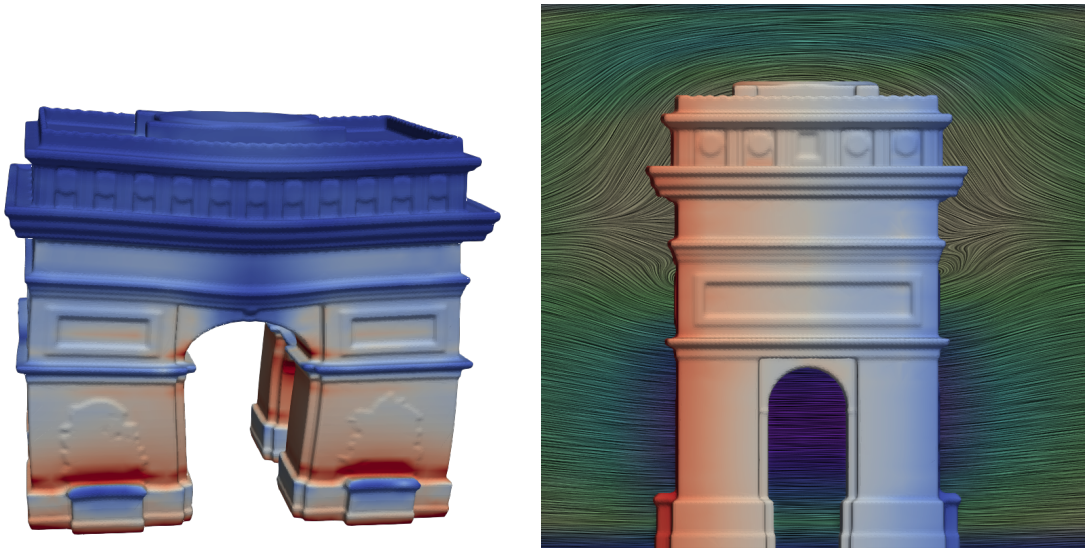
mesh is refined in all cases. The linear regime is reached at different speeds depending on the considered STL models. For the cube, which is the simplest geometry studied here, the linear regime is reached before the other ones, whereas the model with more STL faces (the *Arc de Triomphe* geometry with id 551021) is the latest one to achieve the linear regime. Note that the CPU times converge to similar values for all geometries, when the mesh is refined. This is because, in the limit, the cut algorithm only needs to intersect each cut background cell with a single plane independently of the number of faces in the STL. This suggest that the number of STL planes per cut cell in the background mesh is closely related with the performance of the method. To analyse the interplay between these two quantities, Figure 2.15(d) displays the scaling of CPU time with respect to the number of STL faces both averaged by the number of cut cell in the background mesh. Due to the nature of the algorithm, which involves searches between the STL and the background mesh, one can expect a superlinear scaling. This is indeed what is observed in Figure 2.15(d), but, in any case, the scaling is clearly not quadratic since the searches are efficiently computed using a tree partition. In particular, this allowed us to compute sub-triangulations of complex STL geometries with hundreds of thousands of STL faces in this example and even millions of STL faces in previous examples in Section 2.4.3.

The proposed geometrical treatment can readily be applied to other unfitted FEs methods and PDEs. In Figure 2.16, we solve a linear elasticity problem in the *Arc de Triomphe* geometry and an incompressible flow problem surrounding it. For linear elasticity, we used the formulation in [85]. The unfitted method for incompressible flows can be found in [14]. One can also observe in these two examples that we can readily use the meshes on both sides of the boundary representation.

2.5 Conclusions and future work

In this work, we have designed a fully automatic simulation pipeline for the numerical approximation of PDEs on general domains described by a boundary mesh. The algorithm makes use of a structured background mesh and an unfitted FE formulation on this mesh. The main complication of these methods is the numerical computation of integrals in the interior of the domain for background cells cutting the domain boundary. Boundary meshes for complex geometries can involve a huge number of faces intersecting background cells and the geometries are not convex in general.

We have designed a general clipping algorithm for cut cells that can deal with general surface meshes. They are based on convex decomposition algorithms, robust clipping of convex polyhedra, a graph-based representation of polyhedra, discrete level-set representation of planes and some merging techniques to reduce rounding error effects. The result of this algorithm is a refinement of the boundary mesh that can readily be used to integrate boundary terms and a two-level *integration* mesh. The two-level mesh



(a) (b)

Figure 2.16: We show the AgFEM approximations of two physical problems on both sides of the *Arc de Triomph* STL. In (a), we show the deformed configuration for linear elasticity and the colour map for the stress field. A vertical body force $(0, 0, -1)$ is applied to the volume, representing its own weight. The elastic modulus is set to 10^{-5}, the Poisson ratio is 0.3 and the deformation is magnified 500 times. We use a Cartesian mesh with $50 \times 50 \times 50$ cells. In (b), we show a line integral convolution of the velocity field for a viscous flow around the geometry and the pressure colour map on the surface. The inlet velocity is set as $v_{\text{in}}(x, y, z) = v_{\text{max}}\,(0, v_1(x)v_1(z), 0)$, where $v_{\text{max}} = 0.2$ and $v_1(x) = 4x - 4x^2$. The wall velocity is zero. We use a Cartesian mesh with $20 \times 60 \times 20$ cells.

combines the background mesh and a cell-wise partition of cut cell interiors into convex polyhedra and can straightforwardly be used to integrate the bulk terms in unfitted FE schemes.

The implementation of the algorithms are distributed as open source software and can be found in [76]. The algorithm implementation has been applied with success on all 3D analysis-suitable meshes in the Thingi10K database [123] (almost 5,000 meshes), showing the sound robustness of the approach. The reported integration errors are close to machine precision, which prove its accuracy. These integration meshes have been successfully combined with one unfitted formulation, the AgFEM [22], to discretise PDEs on these geometries, and convergence error plots are provided. Finally, the computational complexity and cost of the geometrical algorithm is reported and compared against the FE solver step.

Future work involves the extension of this approach to other background meshes, specially octree meshes, even though this extension is quite straightforward; the algorithms are cell-wise defined and can readily be applied to locally refined structured meshes. It is of practical relevance to extend the current (open source) implementation to distributed-memory computers. Since the algorithms mainly involve cell-wise computations, they are embarrassingly parallel, a great benefit compared to unstructured mesh generation algorithms that require global consistency and thus are very hard to parallelise. The extension to 4D (under homotopy assumptions or allowing topology changes in time) is of special relevance, since it would allow one to solve complex time-dependent problems (e.g., fluid-structure interaction or multi-fluid models) that involve moving interfaces, one of the main challenges in the field. Most of the ingredients in the current algorithms are dimension-agnostic and the polyhedron representation in terms of oriented graphs seems to be general enough (the formulation does not rely on planar graphs, which would prevent a 4D extension, since 4D polytopes cannot be represented as planar graphs in general). Another topic of interest is the extension of this approach to higher order boundary representations, e.g., connecting the algorithm with the BREP representation to attain higher levels of accuracy via nonlinear intersection algorithms.

Chapter 3

High-order unfitted finite element discretizations for explicit boundary representations

The contents of this chapter correspond to the research publication

[75] P. A. MARTORELL AND S. BADIA, *High order unfitted finite element discretizations for explicit boundary representations*, submitted.

When modeling scientific and industrial problems, geometries are typically modeled by explicit boundary representations obtained from computer-aided design software. Unfitted (also known as embedded or immersed) finite element methods offer a significant advantage in dealing with complex geometries, eliminating the need for generating unstructured body-fitted meshes. However, current unfitted finite elements on nonlinear geometries are restricted to implicit (possibly high-order) level set geometries. In this work, we introduce a novel automatic computational pipeline to approximate solutions of partial differential equations on domains defined by explicit nonlinear boundary representations. For the geometrical discretization, we propose a novel algorithm to generate quadratures for the bulk and surface integration on nonlinear polytopes required to compute all the terms in unfitted finite element methods. The algorithm relies on a nonlinear triangulation of the boundary, a kd-tree refinement of the surface cells that simplify the nonlinear intersections of surface and background cells to simple cases that are diffeomorphically equivalent to linear intersections, robust polynomial root-finding algorithms and surface parameterization techniques. We prove the correctness of the proposed algorithm. We have successfully applied this algorithm to simulate partial differential equations with unfitted finite elements on nonlinear domains described by computer-aided design models, demonstrating the robustness of the geometric algorithm and showing high-order accuracy of the overall method.

3.1 Introduction

A wide range of industrial and scientific applications requires solving PDEs on complex domains. These domains are commonly enclosed by BREP models and generated

in CAD software. CAD models are described by NURBS and boolean operations like constructive solid geometry (CSG). Due to the high-order nature of NURBS, it becomes essential to employ specialized tools capable of efficiently handling numerical simulations on these complex domains.

Despite the enduring popularity of body-fitted meshes in traditional simulation pipelines, they exhibit significant limitations. The generation of body-fitted unstructured mesh generation relies on manual intervention, resulting in significant bottlenecks in the process [63], especially when dealing with high-order representations. Additionally, simulating PDEs on body-fitted meshes with distributed memory machines necessitates mesh partitioning strategies based on graph partition techniques. These algorithms are inherently sequential and demand extensive memory resources [67]. Consequently, the mesh partitioning process represents a major bottleneck in the simulation pipeline and cannot be automated in general.

Unfitted FEMs, also known as embedded or immersed FEMs, offers a solution to the mesh generation bottlenecks by eliminating the need for body-fitted meshes. Unfitted methods rely on a simple background mesh, such as a uniform or adaptive Cartesian mesh. Traditionally, unfitted methods utilize implicit descriptions (level sets) to describe geometries. There are only a few works combining unfitted FEM with explicit boundary representations, such as the one described in [17], which is specifically designed for oriented linear triangulations as boundary representations (a.k.a. STL).

On the other side, several approaches have been introduced to combine unfitted methods with high-order implicit representations of geometries. The initial approach was presented in [51], and since then, there have been gradual improvements in subsequent works, including [50, 53, 69, 105]. These advancements have even extended to handling BREP models in [104]. However, it is worth noting that these methods still rely on level set representations.

In embedded FEMs the small cut cell problem is a significant limitation extensively discussed in the literature [43]. This problem arises when the intersection between physical and background domain cells becomes arbitrarily small, leading to ill-conditioning issues in the numerical solution. Although various techniques have been proposed to address this problem, only a few have demonstrated robustness and optimal convergence. One approach is the ghost penalty method [33], which is utilized in the CutFEM packages [34]. Alternatively, cell agglomeration techniques present a viable option to ensure robustness concerning the cut cell location, initially applied in DG methods [84]. Extensions to the $\mathcal{C}^0$ Lagrangian FE have been introduced in [14], while mixed methods have been explored in [22], where the AgFEM term was coined. AgFEM exhibits good numerical qualities, including stability, bounds on condition numbers, optimal convergence, and continuity concerning data. Distributed implementations have been exploited in [10, 117], also AgFEM has been extended to h-adaptive meshes [85] and higher-order FE with modal $\mathcal{C}^0$ basis in [19]. In [18], a novel

technique combining ghost penalty methods with AgFEM was proposed, offering reduced sensitivity to stabilization parameters. In [7] the AgFEM has been extended to solve transient problems in moving boundaries through space-time discretizations.

The development of isogeometric analysis (IGA) over the past two decades has been driven by the goal of improving the interaction between CAD and CAE [63]. While IGA techniques are suitable for PDEs on boundaries, they cannot readily handle PDEs in the volume of a CAD representation of the domain. Standard CAD representations are 2-variate (boundary representations) and they do not provide a parameterization of the volume. To overcome this limitation, some works [46, 47, 120, 121] propose constructing volume parametrizations based on Bernstein-Bézier basis using the Bézier projection techniques described in [113]. Nevertheless, these approaches still rely on high-order unstructured meshes, thereby inheriting the known limitations associated with them, such as tangling issues, lack of parallelization, and global graph partition bottlenecks.

Similarly to unfitted FE methods, immersed IGA [2, 3, 119] eliminates the need for unstructured meshes by utilizing the intersection of a background mesh. These methods utilize integration techniques for complex domains, including dimension reduction of integrands [40, 57], i.e., integrating over lines and surfaces. The precision of these methods is bounded by the approximation algorithms used on the *trimming curves*, which represent surface-surface intersections [91, 105] resulting from boolean CSG on the CAD models.

One of the primary challenges in the CAD to CAE paradigm is the approximation of trimming curves [71, 92, 100]. Trimming curves, in general, cannot be represented in the intersected surface patches. In the literature, there is a wide range of strategies to approximate trimming curves, see [25, 77] and references therein. These strategies can be categorized into analytical methods, lattice evaluation methods, subdivision methods, marching methods, or a combination thereof. The representation of the approximated trimming curves is also extensively studied. Once the trimming curves are approximated, various techniques can be employed. Untrimming techniques [1, 78, 79, 120, 121] are one approach, which involves a conformal reparametrization of the original surface. Another approach is the direct integration onto trimmed surfaces [57, 98].

In this work, we propose a computational framework that combines unfitted FEM and implicit CAD representations of the geometry. In order to do this, we propose a novel approach to numerically integrate on unfitted cut cells that are intersected by domains bounded by a high-order BREPs. Our method involves approximating the geometry using a set of Bézier patches, utilizing Bézier projection methods [29]. By leveraging the properties of Bézier curves, we can efficiently perform intersections. We can reduce the complexity of the collision interrogations and tangling prediction through the convex hull property. The variational diminishing property of Bézier curves enables root isolation for determining the intersection points [83]. We can employ efficient

multivariate root-finding techniques [82, 94] for polynomials on nonrational Bézier patches. To build an intersection method for nonlinear polytopes, we combine the intersection techniques with partition techniques typically used in level set methods [51] and linear polytopal intersection [17]. We propose a kd-tree refinement of the surface B'ezier triangulation that reduces nonlinear intersections against background cells to simple situations that are diffeomorphically equivalent to linear intersections. This allows us to handle the complexity of intersecting high-order geometries efficiently. Furthermore, we can approximate these intersections with a set of Bézier patches using least-squares techniques based on [30].

With the intersection method established, we are then able to integrate on the surface of these polytopes to solve PDEs on high-order unfitted FE meshes. We employ moment-fitting techniques based on Stokes theorem [19, 40] to ensure accurate and stable integration. By utilizing these techniques, we can effectively handle the integration process on high-order unfitted FE meshes, allowing for the solution of PDEs in domains bounded by complex high-order BREPs.

The outcomes of this work are as follows:

- An automatic computational framework that relies on a robust and accurate intersection algorithm for background cells and Bézier patches of arbitrary order (briefly described above), and the mathematical analysis of the correctness of the algorithm.
- Accuracy and robustness numerical experimentation of the intersection algorithm. The algorithms exhibit optimal convergence rates of the surface and volume integration. These errors are robust concerning the relative position of the background cells and the BREP.
- The numerical experimentation of a high-order unfitted FE method for high-order BREPs with analytical benchmarks. The results demonstrate optimal hp-convergence of the error norms.
- The demonstration of the application of the methods for problems defined in CAD geometries.

The outline of this chapter is as follows. Firstly, in Section 3.2, we introduce the unfitted FE methods and their requirements for handling high-order BREPs. Next, in Section 3.3, we provide the proposed geometric algorithms for computing the nonlinear intersections between background cells and oriented high-order BREPs, along with a surface parametrization method for integration purposes. Then, in Section 3.4, we present the numerical results obtained from applying the proposed method, including accuracy and robustness of the intersections, benchmark tests for validation of the unfitted FE pipeline, and simulations on CAD geometries. Finally, in Section 3.5, we draw the main conclusions and future work lines.

3.2 Unfitted finite element method

3.2.1 Unfitted finite element formulations

Let us consider an open Lipschitz domain $\Omega \in \mathbb{R}^3$ in which we want to approximate a system of PDEs. An oriented high-order surface mesh $\mathcal{B}$ defines the domain boundary $\partial\Omega$ and encloses the domain interior. The PDEs usually involve Dirichlet boundary conditions on Γ_D and Neumann boundary conditions on Γ_N, where $\partial\Omega \doteq \Gamma_D \cup \Gamma_N$ and $\Gamma_D \cap \Gamma_N = \emptyset$. These subsets, Γ_D and Γ_N, correspond to geometric discretizations of $\mathcal{B}_\mathcal{D}$ and $\mathcal{B}_\mathcal{N}$, resp., such that $\mathcal{B} \equiv \mathcal{B}_\mathcal{D} \cup \mathcal{B}_\mathcal{N}$.

The principal motivation of this work is to enable the utilization of grid-based unfitted numerical schemes that can be automatically generated from the oriented high-order surface mesh $\mathcal{B}$. This approach is valuable in industrial and scientific applications. We can alleviate the geometric constraints associated with body-fitted meshes by employing embedded discretization techniques. These techniques utilize a background partition $\mathcal{T}^{\text{bg}}$ defined over an arbitrary artificial domain $\Omega^{\text{art}} \supseteq \Omega$. The artificial domain can be a simple bounding box containing Ω, dramatically simplifying the computation of $\mathcal{T}^{\text{bg}}$ compared to a body-fitted partition of Ω. In this work, we adopt a Cartesian mesh $\mathcal{T}^{\text{bg}}$ for the sake of simplicity, although the proposed approach can readily be used on other types of background meshes, such as tetrahedral structured meshes obtained through simplex decomposition or adaptive mesh refinement (AMR) techniques.

The abstract exposition of unfitted formulations considered in this work is general and encompasses various unfitted FE techniques from the literature. These techniques include the XFEM [26], designed for handling unfitted interface problems. In order to have robustness with respect to small cut cells, the cutFEM method [34] and the finite cell method [96] add stabilization terms. The AgFEM [22] provides robustness via a discrete extension operator from interior to cut cells. Since DG methods can work on polytopal meshes, combined with cell aggregation [84], they are also robust unfitted FE techniques.

The definition of FE spaces on unfitted meshes requires a cell classification. The background cells $K \in \mathcal{T}^{\text{bg}}$ with a null intersection with Ω are classified as exterior cells and are denoted as $\mathcal{T}^{\text{out}}$. These exterior cells, which have no contribution to functional discretization and can be discarded. The active mesh, denoted as $\mathcal{T} = \mathcal{T}^{\text{bg}} \setminus \mathcal{T}^{\text{out}}$, represents the relevant mesh for the problem (Figure 3.1). The unfitted FE techniques stated above utilize FE spaces defined on $\mathcal{T}$ to construct the finite-dimensional space V. This space approximates the solution and tests the weak form of PDEs. An abstract unfitted FE problem reads as follows: find $u \in V$ such that

$$a(u, v) = l(v), \quad \forall v \in V,$$

where

$$a(u,v) = \int_\Omega L_\Omega(u,v)d\Omega + \int_{\Gamma_D} L_D(u,v)d\Gamma + \int_{\mathcal{F}} L_{\text{sk}}(u,v)d\Gamma,$$

and

$$l(v) = \int_\Omega F_\Omega(v)d\Omega + \int_{\Gamma_N} F_N(v)d\Gamma + \int_{\Gamma_D} F_D(v)d\Gamma.$$

The bulk terms L_Ω and F_Ω consist of the weak form of the differential operator, the source term, and possibly other numerical stabilization terms. On Γ_D, the operators L_D and F_D represent the enforcement of the Dirichlet boundary conditions, often implemented using Nitsche's method in unfitted formulations. The term F_N on Γ_N represents the Neumann boundary conditions of the given problem. The skeleton of the active mesh $\mathcal{F}$ corresponds to the interior faces of $\mathcal{T}$, while the term L_{sk} collects additional penalty terms, such as weak continuity enforcement in DG methods or ghost penalty stabilization techniques.

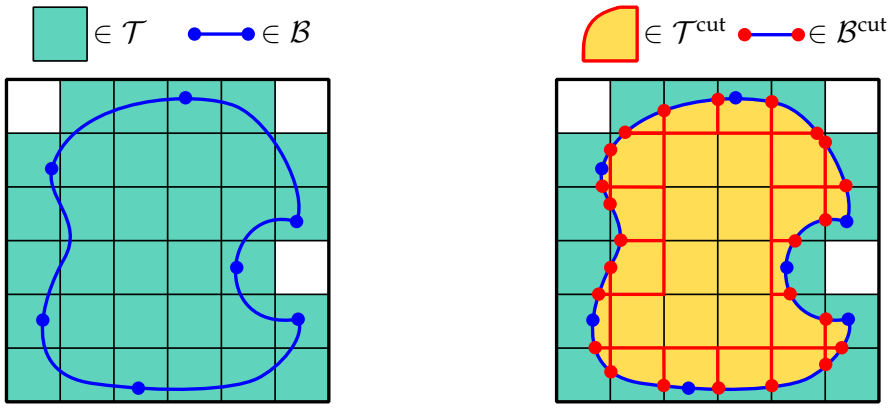


Figure 3.1: Example of the embedded nonlinear domain in 2D. Figure (a) presents a nonlinear oriented skin mesh $\mathcal{B}$ embedded in an active mesh $\mathcal{T}$. The intersections, computed with the techniques proposed in this work, result in the two-level partitions $\mathcal{T}^{\text{cut}}$ and $\mathcal{B}^{\text{cut}}$ shown in (b). These partitions are utilized for integrating unfitted formulations. It is important to note that the intersections in 2D are points that can be represented exactly. However, the intersections in 3D are trimming curves that must be approximated in general.

Since FE methods are piecewise polynomials, integrating these terms requires a cell-wise decomposition for both bulk and surface contributions. However, to accurately respect the geometry and solve the PDE on the proper domain, it is necessary to perform these integrals within the interiors of the domains. In particular, we have

$$\int_\Omega (\cdot)d\Omega = \sum_{K\in\mathcal{T}} \int_{K\cap\Omega} (\cdot)d\Omega. \tag{3.1}$$

Due to the inherent characteristics of unfitted FE methods, the surface mesh $\mathcal{B}$ is not the boundary restriction of the background mesh $\mathcal{T}$, which defines the cell-wise polynomial FE functions. Consequently, to integrate the boundary terms on $\mathcal{B}$, we must compute a cell-wise integral as follows:

$$\int_{\Gamma_*} (\cdot) d\Gamma = \sum_{K \in \mathcal{T}} \sum_{F \in \mathcal{B}_*} \int_{F \cap K} (\cdot) d\Gamma, \quad * \in \{D, N\}. \tag{3.2}$$

It is important to note that, in general, we do not need to restrict the integrals on the skeleton terms L_{sk} to the domain interior. In most applications, one can integrate these terms on the entire face skeleton, e.g., ghost penalty and DG methods. Therefore, in the unfitted FE methods, we must give particular attention to the geometrical operations involving $K \cap \Omega$ and $K \cap \mathcal{B}$ (or more specifically, $\mathcal{B}_N$ and $\mathcal{B}_D$). However, the methodology described below also handles $F \cap \Omega$ for $F \in \mathcal{F}$, in case it is required.

3.2.2 Geometrical ingredients for unfitted finite elements

This section aims to describe the geometrical entities that we need to compute the integrals (3.1)-(3.2) of unfitted FE formulations. These entities are easier to compute than unstructured meshes in body-fitted formulations. Our geometrical framework involves intersection algorithms that are cell-wise, and so, embarrassingly parallel. Since the FE spaces are defined on background meshes, the cut meshes do not require to be conforming or shape-regular.

The input of our geometrical framework is an oriented mesh of non-rational triangular Bézier elements $\mathcal{B}$ whose interior is the domain Ω. Each triangle $F \in \mathcal{B}$ is the image of a Bézier map $\boldsymbol{\phi}_F : \hat{F} \to F$ of order q acting on a reference triangle $\hat{F} \subset \mathbb{R}^2$.

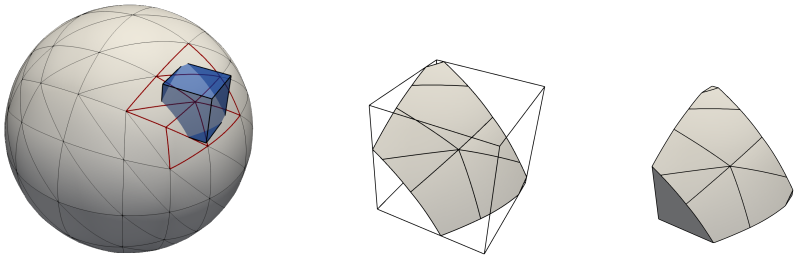
In the following exposition, we use $\hat{\boldsymbol{\xi}}$ to represent the coordinate system of the reference space of $\hat{F}$ and $\boldsymbol{x}$ to represent the coordinate system of the physical space. Given a set of points $\hat{f}$ in the reference space of $\hat{F}$ we can compute $f \doteq \boldsymbol{\phi}_F(\hat{f})$. Since $\hat{F}$ is diffeomorphic, we can also define f and compute $\hat{f} \doteq \boldsymbol{\phi}_F^{-1}(f)$. Given a function $\hat{\gamma}(\hat{\boldsymbol{\xi}})$ in the reference space of $\hat{F}$, we can compute $\gamma(\boldsymbol{x}) \doteq \hat{\gamma} \circ \boldsymbol{\phi}_F^{-1}(\boldsymbol{x})$. Reversely, we can define $\gamma(\boldsymbol{x})$ in the physical domain and compute its pull-back $\hat{\gamma}(\boldsymbol{\xi}) \doteq \gamma \circ \boldsymbol{\phi}_F(\boldsymbol{\xi})$. We will heavily use this notation to transform sets and functions between the reference and physical spaces.

In practical applications, geometries are described through a CAD model $\mathcal{B}^{\text{CAD}}$. Transforming $\mathcal{B}^{\text{CAD}}$ into $\mathcal{B}$ generally involves an approximation process, where we can utilize least-squares methods and third-party libraries, e.g., `gmsh` [54]. Next, we perform intersection algorithms between the surface representation and the background mesh, which consist of two steps as follows.

In the first step, we consider the intersection of the triangular Bézier elements that compose the surface $F \in \mathcal{B}$ with the background cells $K \in \mathcal{T}$. According to (3.2), the resulting mesh $\mathcal{B}^{\text{cut}}$ can be represented as a two-level mesh:

$$\mathcal{B}^{\text{cut}} \doteq \bigcup_{K \in \mathcal{T}} \mathcal{B}_K^{\text{cut}}, \qquad \text{where} \quad \mathcal{B}_K^{\text{cut}} \doteq \{F \cap K : F \in \mathcal{B}\}.$$

$\mathcal{B}^{\text{cut}}$ is a partition of $\mathcal{B}$ composed of general *nonlinear polytopes* (see Figure 3.2). We define a nonlinear polytope as the image of a polytope under a diffeomorphic map.



(a) $\mathcal{B}$, $\mathcal{B}_K$ and K (b) $\mathcal{B}_K^{\text{cut}}$ (c) K^{cut}

Figure 3.2: Representation of the surface $\mathcal{B} \doteq \partial\Omega$ (see (a)), its intersection $\mathcal{B}_K^{\text{cut}} \doteq \mathcal{B} \cap K$ for a background cell $K \in \mathcal{T}$ (see (b)), and the domain interior of the cell $K^{\text{cut}} \doteq K \cap \Omega$ (see (c)). In order to compute $\mathcal{B}_K^{\text{cut}}$, we identify first the subset of cells $\mathcal{B}_K \subseteq \mathcal{B}$ touching K (see cells with red edges in (a)). Next, for each triangle $F \in \mathcal{B}_K$, we compute the intersection of $F \cap K$ at the reference FE, i.e., we compute $\boldsymbol{\phi}_F^{-1}(F \cap K)$. Finally, we intersect K with the surface portion $\mathcal{B}_K^{\text{cut}}$ to obtain K^{cut}. It is worth to note that $\mathcal{B}_K^{\text{cut}} \subset \partial K^{\text{cut}}$.

In the second step, we compute a mesh of general nonlinear polyhedra that represent the domain interior of background cells (see Section 3.3.6):

$$\mathcal{T}^{\text{cut}} \doteq \bigcup_{K \in \mathcal{T}} K^{\text{cut}}, \qquad \text{where} \quad K^{\text{cut}} \doteq K \cap \Omega.$$

The surface cut cells in $\mathcal{B}_K^{\text{cut}}$ are nonlinear polygons that can be split into simplices. A simplex decomposition of the volumetric cells in $\mathcal{T}^{\text{cut}}$ is much more complex and expensive. Instead, we avoid a trivariate representation of $\mathcal{T}^{\text{cut}}$ and instead rely on a bivariate representation of $\partial\mathcal{T}^{\text{cut}}$. We consider a boundary representation (often abbreviated B-rep or BREP in solid modeling and computer-aided design) of K^{cut} as the interior of an oriented closed surface represented as the collection of connected oriented surface elements, i.e.:

$$\partial K^{\text{cut}} = (\partial K)^{\text{cut}} \cup \mathcal{B}_K^{\text{cut}}, \qquad (\partial K)^{\text{cut}} \doteq \bigcup_{F \in \Lambda^2(K)} F \cap \Omega.$$

3.2.3 Integration methods for cut cells

For the computation of the volume integrals in (3.1), since we define cut cells by its boundary representation in (3.2.2), we compute the integration of polynomials on volume cut cells by transforming them into surface integrals via Stokes theorem [19, 40]. Let us denote rth order polynomial differential k-forms in $\mathbb{R}^3$ with $\mathcal{P}_r\Lambda^k(\mathbb{R}^3)$ and d the exterior derivative (see, e.g., [4]). For any $\omega(\boldsymbol{x}) \in \mathcal{P}_r\Lambda^3(\mathbb{R}^3)$, one can readily find $\sigma \in \mathcal{P}_{r+1}\Lambda^2(\mathbb{R}^3)$ such that $\omega(\boldsymbol{x}) = d\sigma(\boldsymbol{x})$, due to the exactness of the polynomial de

Rham complex. Using Stokes theorem, one gets:

$$\begin{aligned}\int_{K^{\text{cut}}} \omega &= \int_{\partial K^{\text{cut}}} \sigma = \int_{\mathcal{B}_K^{\text{cut}}} \sigma + \int_{(\partial K)^{\text{cut}}} \sigma = \sum_{F\in\mathcal{B}_K} \int_{F\cap K} \sigma + \sum_{F\in\Lambda^2(K)} \int_{F\cap\Omega} \sigma \\ &= \sum_{F\in\mathcal{B}_K} \int_{\boldsymbol{\phi}_F^{-1}(F\cap K)} \boldsymbol{\phi}_F^*(\sigma) + \sum_{F\in\Lambda^2(K)} \int_{\boldsymbol{\phi}_F^{-1}(F\cap\Omega)} \boldsymbol{\phi}_F^*(\sigma).\end{aligned}$$

where $\boldsymbol{\phi}_F^*(\sigma)$ denotes the pull-back of σ by $\boldsymbol{\phi}_F$. In order to create a quadrature on K^{cut}, we combine this expression with a moment-fitting technique. First, we integrate all the elements in the monomial basis for $\mathcal{P}_r(\mathbb{R}^3)$; e.g., given a monomial 3-form $\omega(\boldsymbol{x}) = x^\alpha y^\beta z^\gamma dx\wedge dy\wedge dz$, we define $\sigma(\boldsymbol{x}) = \frac{1}{\alpha+1} x^{\alpha+1} y^\beta z^\gamma dy\wedge dz$ (which holds $\omega = d\sigma$) and use (3.2.3). After computing these integrals for the monomial basis at each face $F\in\mathcal{B}_K$, we can combine them to compute the integrals of Lagrangian polynomials, use the corresponding nodal interpolation, and end up with a quadrature on the cut cell K^{cut}.

To compute the surface integrals in (3.2) and the right-hand side of (3.2.3), we have several options. One approach is to generate a simplex nonlinear mesh of the boundary of the nonlinear polytopes in $\mathcal{T}^{\text{cut}}$, which is feasible on surfaces. We can use a moment-fitting method to compress the resulting quadrature. E.g., for the integral on $\mathcal{B}_K^{\text{cut}}$, we consider a nonlinear triangulation $\mathcal{S}_{\hat{F}}^{\text{cut}}$ of $\boldsymbol{\phi}_F^{-1}(F\cap K)$ and compute:

$$\int_{\mathcal{B}_K^{\text{cut}}} \sigma = \sum_{F\in\mathcal{B}_K} \sum_{S\in\mathcal{S}_{\hat{F}}^{\text{cut}}} \int_S \boldsymbol{\phi}_F^*(\sigma) = \sum_{F\in\mathcal{B}_K} \sum_{S\in\mathcal{S}_{\hat{F}}^{\text{cut}}} \int_{\hat{F}} \boldsymbol{\phi}_S^* \circ \boldsymbol{\phi}_F^*(\sigma)$$

where we rely on a map $\boldsymbol{\phi}_S : \hat{F}\to S$ to transform the integral to the reference triangle $\hat{F}$; S is a nonlinear triangle in $\mathbb{R}^2$ because it can have nonlinear edges. We proceed analogouly for the surface integral on $(\partial K)^{\text{cut}}$; this is a simpler case, since the surface is already contained in a plane in the physical space.

The edges of $S\in\mathcal{S}_{\hat{F}}^{\text{cut}}$ that result from intersections are implicitly defined. Thus, we must consider an approximation $\tilde{\phi}_S$ of the map ϕ_S in (3.2.3). This approximation is detailed in the next section. Since F (and as a result $S\in\mathcal{T}^{\text{cut}}$) is a smooth manifold, the polynomial approximation $\tilde{\phi}_S$ will introduce an error that is reduced by increasing the polynomial order of the approximation or by reducing the diameter of S. In turn, the diameter goes to zero with both the background mesh $\mathcal{T}$ and surface mesh $\mathcal{B}$ characteristic sizes.

Another approach is to integrate the pull-back $\boldsymbol{\phi}_F^*(\sigma)$ of the polynomial differential form σ in (3.2.3) on the surface to $\hat{F}$ (which are polynomials of higher order due to the factor $\det(\frac{\partial\boldsymbol{\phi}_F}{\partial\boldsymbol{\xi}})$ that comes from the pullback) and transform the integral to $\partial\hat{F}$ using Stokes theorem:

$$\begin{aligned}\int_{F\cap K} \sigma &= \int_{\boldsymbol{\phi}_F{}^{-1}(F\cap K)} \boldsymbol{\phi}_F^*(\sigma) = \int_{\boldsymbol{\phi}_F^{-1}(F\cap K)} d\varrho \\ &= \int_{\partial\boldsymbol{\phi}_F^{-1}(F\cap K)} \varrho = \int_{\boldsymbol{\phi}_F^{-1}(\partial F\cap K)} \varrho, \qquad \forall\varrho\in\mathcal{P}_{r+2q}\Lambda^1(\mathbb{R}^2)\ :\ d\varrho = \boldsymbol{\phi}_F^*(\sigma).\end{aligned}$$

We note that $\boldsymbol{\phi}_F^*(\sigma) \in \mathcal{P}_{r+2q-1}\Lambda^2(\mathbb{R}^2)$ and the existence of ϱ is assured by the exactness of the polynomial de Rham complex. In this case, we can extract the (nonlinear) edges E of $\partial F \cap K$ and define a map $\boldsymbol{\phi}_E : \hat{E} \to E$ from $\boldsymbol{\phi}_F$. Next, we can compute the integrals in a reference segment, i.e.,

$$\int_{F\cap K} \sigma = \sum_{E\in\partial F\cap K} \int_{\boldsymbol{\phi}_E^{-1}(E)} \varrho.$$

As above, since some edges $E \in \partial F \cap K$ are only implicitly defined, we must consider approximations $\tilde{\boldsymbol{\phi}}_E$ of $\boldsymbol{\phi}_E$ in (3.2.3). However, we do not need to compute approximated surface maps or triangulations of $\boldsymbol{\phi}_F^{-1}(F \cap K)$. Since we approximate each edge separately, and both F and the faces of K are smooth manifolds, a polynomial approximation provides error bounds that vanish by increasing the polynomial order or by reducing the edge sizes; edge sizes also go to zero with both the background mesh $\mathcal{T}$ and surface mesh $\mathcal{B}$ characteristics sizes.

3.3 Intersection algorithm

In this section, we describe an algorithm that takes a background mesh $\mathcal{T}$ and a high-order oriented surface $\mathcal{B}$ as inputs and returns $\mathcal{T}^{\text{cut}}$ and $\mathcal{B}^{\text{cut}}$. This algorithm is robust to the relative position of $\mathcal{T}$ and $\mathcal{B}$. Moreover, it provides an accurate description of the intersections. We list below the different steps considered in the definition of the algorithm:

1. In Section 3.3.1, we describe the nonlinear computations of the singular points utilized in the intersection algorithms. There, we utilize multivariate root-finding techniques [82].

2. In Section 3.3.2 and Section 3.3.3, we propose an accurate intersection algorithm for nonlinear polyhedra. It combines refinement strategies (to simplify the nonlinear intersection to simple cases) and linear clipping methods [17].

3. Section 3.3.4 describes a partition method for the intersected surface. This partition is composed of standard polytopes and is ready to be parametrized.

4. In Section 3.3.5, we introduce a parametrization method for intersected nonlinear polyhedra. This parametrization consists of a combination of least-squares methods [30] and sampling strategies [51]. It returns a set of nonrational Bézier patches.

5. In Section 3.3.6, we build the polyhedral representation of the intersected cells. In addition, we provide the tool to parametrize the resulting surface.

We describe a global algorithm that combines the previous algorithm in Section 3.3.7.

3.3.1 Intersection points

In this section, we perform some intersection algorithms that will be required in our geometrical framework. In order to compute surface intersections, we require the computation of three types of intersections, described below.

Consider an oriented plane π defined by a point $\boldsymbol{x}_\pi$ on the plane and its outward normal $\boldsymbol{n}_\pi$. The plane-point distance function can be computed as

$$\gamma_\pi(\boldsymbol{x}) \doteq \texttt{dist}(\pi, \boldsymbol{x}) \doteq (\boldsymbol{x} - \boldsymbol{x}_\pi) \cdot \boldsymbol{n}_\pi, \qquad \boldsymbol{x} \in \mathbb{R}^3.$$

In the following, we will also consider this distance function parametrized in the reference face $\hat{F}$ as $\hat{\gamma}_\pi(\hat{\boldsymbol{\xi}}) \doteq \gamma_\pi \circ \boldsymbol{\phi}_F(\hat{\boldsymbol{\xi}})$. Let us denote by $\text{int}(\gamma_\pi) = \{\boldsymbol{x} : \gamma_\pi(\boldsymbol{x}) < 0\}$ the interior of a level set. The zero level set of the distance function is represented by

$$\hat{\gamma}_\pi^0 \doteq \{\, \hat{\boldsymbol{\xi}} \in \hat{F} \;:\; \texttt{dist}(\pi, \boldsymbol{\phi}_F(\hat{\boldsymbol{\xi}})) = 0 \,\}.$$

Curve-plane intersection

First, we define the curve-plane intersections as the intersection of an edge $E = \boldsymbol{\phi}_E(\hat{E})$ (which can be described as a diffeomorphic polynomial map on a reference segment $\hat{E}$) with a plane π (see Figure 3.3(a)). We compute the intersection points in the reference segment $\hat{E}$ as the result of finding:

$$\hat{t} \in \hat{E} \;:\; \gamma_\pi \circ \boldsymbol{\phi}_E(\hat{t}) = 0.$$

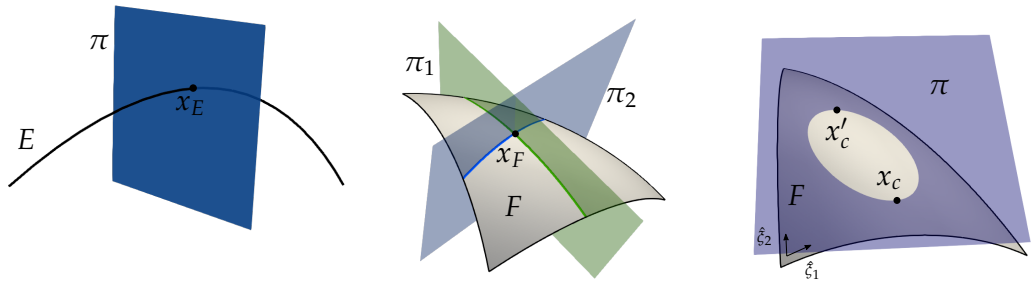
If $E \in \Lambda^1(F)$, the map $\boldsymbol{\phi}_E : \hat{E} \to E$ can be readily obtained from the corresponding face map $\boldsymbol{\phi}_F$. Since $\boldsymbol{\phi}_E$ is a polynomial, the computation of $\hat{t}$ involves finding the roots of a univariate polynomial system. One can utilize root isolation techniques combined with standard iterative solvers, e.g., the Newton-Rapson method. As the solution may be not unique, the root isolation techniques in [83] provide the means to find multiple roots by leveraging the variation diminishing property of Bézier curves.

Surface-line intersection

Next, we consider line-surface intersections. Since a line can be represented as the intersection of two half-spaces, we compute surface-line intersections as surface-plane-plane intersections (see Figure 3.3(b)). To compute these intersections, we find:

$$\hat{\boldsymbol{\xi}} \in \hat{F} \;:\; \gamma_{\pi_i} \circ \boldsymbol{\phi}_F(\hat{\boldsymbol{\xi}}) = 0, \quad \forall\, i \in \{1, 2\},$$

where π_1, π_2 are the planes whose intersection is the line we want to intersect. We note that $\hat{\boldsymbol{\xi}} \in \mathbb{R}^2$. It can be checked that this is a system of two bivariate polynomial equations of order q. The roots can be computed using bivariate root-finding algorithms for polynomials. The algorithms described in [82] bound and subdivide the reference



(a) $E \cap \pi$ (b) $F \cap \pi_1 \cap \pi_2$ (c) $\hat{\boldsymbol{\nabla}}\hat{\gamma}_\pi(\hat{\boldsymbol{\xi}}_c) \cdot \hat{\xi}_1 = 0$

Figure 3.3: Definition of intersection points. The intersection of curves and planes (see (a)) is computed through univariate root-finding methods. The surface-plane-plane intersection points in (b) also represent the intersection of surface-ine intersections. The AA critical points in (c) are defined in the reference space of F. These points split the surface-plane intersection curves into monotonic curves in the reference space (see an analogous reference space in Figure 3.4(a)). Both computations, surface-plane-plane intersection points and AA critical points, require solving a bivariate root system.

domain until the roots are isolated with a given precision. In this work, we adapt the implementation to simplices using the bounding techniques defined in [94].

Critical points of the zero isosurface of a distance function with respect to an edge.

Finally, we compute the points in which the zero let set curve $\hat{\gamma}_\pi^0$ has a zero directional derivative with respect to a given direction $\boldsymbol{t}$ (see Figure 3.3(c)). When the direction $\boldsymbol{t}$ is axis-aligned (AA), we call these points *AA critical points*. These points are obtained as the solution of the following system:

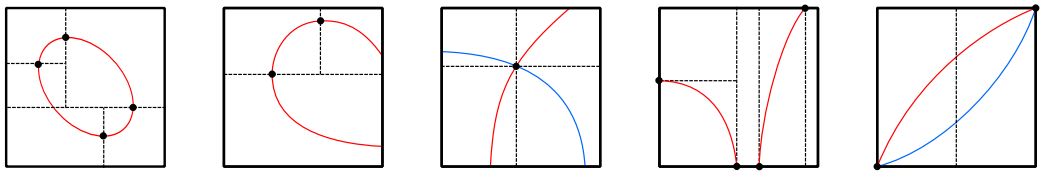
$$\hat{\boldsymbol{\xi}} \in \hat{F} \ : \ \hat{\boldsymbol{\nabla}}\hat{\gamma}_\pi(\hat{\boldsymbol{\xi}}) \cdot \boldsymbol{t} = 0, \quad \hat{\gamma}_\pi(\hat{\boldsymbol{\xi}}) = 0,$$

which is again a system of two bivariate polynomial equations of order q and can be computed as above. We note that the critical points are defined in the reference space. Thus, we take the gradient in the reference space.

3.3.2 Nonlinear trimming surface

Given a face $F \in \mathcal{B}_K$, we consider its intersection against a cell $K \in \mathcal{T}_h$. The objective is to create a partition of $\mathcal{B}_K$ by splitting its faces by intersection against the background cells $K \in \mathcal{T}_h$. We consider $\mathcal{T}_h$ to be a partition of Ω_{art} in a pointwise sense, i.e., every $\boldsymbol{x} \in \Omega_{\text{art}}$ belongs to only one $K \in \mathcal{T}_h$; K is neither open nor closed in general. We refer to [17] for more details. The pointwise partition is required to properly account for the case in which $F \in \mathcal{B}_K$ is (in machine precision) on K; otherwise, we could be counting more than once the same face F. We utilize the notation K° and $\overline{K}$ to refer to the interior and closure of K, resp.

Assumption 3.3.1. *We assume that* $F \cap K^\circ \neq \emptyset$.



(a) (b) (c) (d) (e)

Figure 3.4: Refinement steps of Algorithm 11. Step 1 is represented in (a) and (b). Steps 2-4 are represented in (c)-(e), resp. Red and blue dashed lines represent the $\hat{\gamma}_f^0$ and $\hat{\gamma}_{f'}^0$-curves, resp., $f, f' \in \Lambda^2(K)$. Interior black lines represent the proposed partition of $\hat{F}$ for a given invariant. Solid black lines are the edges of $\Lambda^1(\hat{F})$. There are multiple possible partitions, depending on the order the intersections are processed.

In [17], we propose an algorithm that determines whether the assumption holds, i.e., checks if $F \cap \pi_f = F$ for some $f \in \Lambda^2(K)$. Here, $f \in \Lambda^2(K)$ represents a face bounding K and π_f its corresponding half-space. If the assumption does not hold, $F \cap K = F \cap f$, and we perform the intersection in one dimension less, which is a simplification of the one below.

Assumption 3.3.2. *We assume that all intersection queries are well-posed, i.e., they return a finite number of points.*

For simplicity in the exposition, we postpone the ill-posed cases till the end of the section.

For each $f \in \Lambda^2(K)$ and its corresponding half-space π_f, we can define the distance function $\hat{\gamma}_{\pi_f}$ in the reference space using (3.3.1) , which we denote with $\hat{\gamma}_f$ for brevity. We can also consider the zero level set curves $\{\hat{\gamma}_f^0\}_{f \in \Lambda^2(K)}$, using (3.3.1), and their intersections:

$$\hat{\gamma}_f^0 \cap \hat{\gamma}_{f'}^0, \quad f, f' \in \Lambda^2(K).$$

We note that the intersection points of these curves are the intersections $\boldsymbol{\phi}_F^{-1}(\pi_f \cap \pi_{f'})$; $\pi_f \cap \pi_{f'}$ contains the edge $e \in \Lambda^1(K)$ such that $e \in f, f'$. We also note that $\hat{\gamma}_f^0 \cap \overline{\hat{F}}$, $f \in \Lambda^2(K)$, can have several disconnected parts. We represent with $C_f(\hat{F})$ the set of mutually disconnected components of $\hat{\gamma}_f^0 \cap \overline{\hat{F}}$.

In the following, we want to refine F in such a way that the nonlinear case is *diffeomorphically* equivalent to the linear one. Thus, we can use a linear clipping algorithm to compute the cyclic graph representation of the nonlinear polytope (see [17]). For this purpose, we consider the AA refinement ref_1 of F presented in Algorithm 11 and illustrated in Figure 3.4.

Let us consider the level set curve $\hat{\gamma}_{\varepsilon_d}^0$ associated with the diagonal edge $\varepsilon_d \in \Lambda^1(\hat{F})$. After ref_1, these edges cannot contain intersection points in their interior. Having intersection points only on AA edges simplifies the exposition of the next algorithms. We cannot do the same for AA edges since new AA edges are generated after each refinement step and new intersections can appear.

Algorithm 11 ref_1: axis-aligned refinement of invariants

Let us consider a *kd*-tree partition $\text{ref}_1^0(\hat{F})$ of $\hat{F}$ by refining against

1. $\hat{\xi}_1$-aligned lines that contain the $\hat{\xi}_2$-aligned critical points of $\hat{\gamma}_f^0$, $f \in \Lambda^2(K)$. Proceed analogously for $\hat{\xi}_2$-aligned lines and $\hat{\xi}_1$-aligned critical points.
2. $\hat{\xi}_1$ and $\hat{\xi}_2$-aligned lines containing each intersections $\hat{\gamma}_f^0 \cap \hat{\gamma}_{f'}^0$ and $\hat{\gamma}_f^0 \cap \hat{\gamma}_{\varepsilon_d}^0$, for all $f, f' \in \Lambda^2(K)$.

After this refinement, if $\hat{\gamma}_f^0 \cap \partial\hat{F}_R$, $f \in \Lambda^2(K)$, $\hat{F}_R \in \text{ref}_1^0(\hat{F})$, has more than two intersection points, consider the partition $\text{ref}_1^1(\hat{F}_R)$ by refining against

3. AA lines perpendicular to the AA edge $\varepsilon \in \Lambda^1(\hat{F}_R)$ containing each of these intersection points.

Next, if $\hat{\gamma}_f \cap \hat{\gamma}_{f'}$ or $\hat{\gamma}_f \cap \hat{\gamma}_{\varepsilon_d}^0$, $f, f' \in \Lambda^2(K)$, intersect more than once $\hat{F}_R \in \text{ref}_1^1 \circ \text{ref}_1^0(\hat{F})$,

4. Consider a partition of $\hat{F}_R$ by an AA line that passes between two vertices.

Return the final *kd*-tree partition $\text{ref}_1(\hat{F})$ after these four steps.

Proposition 3.3.3. *The number of faces in* $\text{ref}_1(\hat{F})$ *is bounded.*

Proof. First, we note that there is a limited number of intersections and critical points since $\boldsymbol{\phi}_F$ is a polynomial. Thus, the number of lines in the refinement strategy is bounded, so the resulting number of faces is also bounded. Furthermore, these points do not change with refinement (the sides of the children faces can only be parallel to the sides of the parent face). Only the intersection points in step (3) do change with refinement. However, we only need to perform step (3) once for the purpose of Proposition 3.3.4. □

Proposition 3.3.4. *After the refinement strategy above, for any* $f \in \Lambda^2(K)$, $C_f(\hat{F}_R)$ *is a set of smooth connected curves such that: given* $\hat{F}_R \in \text{ref}_1(\hat{F})$

(i) the curves in $C_f(\hat{F}_R)$ *cannot be tangent to any AA line in* $\hat{F}_R \setminus \Lambda^0(\hat{F}_R)$*;*

(ii) every curve in $C_f(\hat{F}_R)$ *is strictly monotonic with respect to the coordinates* $(\hat{\xi}_1, \hat{\xi}_2)$*, can only intersect once AA segments in* $\hat{F}_R \setminus \Lambda^0(\hat{F}_R)$ *and can only intersect diagonal edges on* $\Lambda^0(\hat{F}_R)$*;*

(iii) all the curves in $C_f(\hat{F}_R)$ *are strictly increasing with respect to the coordinates* $(\hat{\xi}_1, \hat{\xi}_2)$ *(or all strictly decreasing).*

Proof. First, we note that AA critical points do not change by refinement. The curves are smooth since they are the intersection of a smooth surface (image of a polynomial map) with a plane.

Let us prove (i) by contradiction. We assume that some $\hat{\gamma}_f^0$, $f \in \Lambda^2(K)$ is tangent to a $\hat{\xi}_1$-aligned AA line on $\hat{F}_R \setminus \Lambda^0(\hat{F})$. The point was a $\hat{\xi}_1$-critical point before the

refinement. As a result, after refinement, this point belongs to a $\hat{\xi}_2$-aligned axis of $\hat{F}_R$ (by (1) in ref_1). Then, the point can only be in $\Lambda^0(\hat{F}_R)$. We proceed analogously for $\hat{\xi}_2$-aligned lines.

Next, we prove (ii). By definition of the refinement rule, there cannot be AA critical points in $\hat{F}_R^\circ$. We also know that the curves cannot be tangent to AA sides. Thus, by the implicit function theorem, $\hat{\gamma}_f(\hat{\boldsymbol{\xi}}) = 0$ can be expressed as the graph of a function $d_1(\hat{\xi}_1) = \hat{\xi}_2$ and $d_2(\hat{\xi}_2) = \hat{\xi}_1$ in $\hat{F} \setminus \Lambda^0(\hat{F})$. Since these functions are smooth, by the mean value theorem, they must be strictly monotonic in $\hat{F}$ with respect to $\hat{\xi}_1$ and $\hat{\xi}_2$. Thus, they can intersect AA lines at most once. Besides, all intersections against diagonal sides are on $\Lambda^0(\hat{F}_R)$ after refinement (by (2) in ref_1); note that we are not inserting new non-AA edges by refinement.

Finally, we prove (iii). We note that the partition in step (3) or ref_1 prevents two curves $\hat{\alpha}_f, \hat{\alpha}'_f \in C_f(\hat{F}_R)$, strictly increasing and decreasing, resp., to be in the same $\hat{F}_R$. Let us assume that these two curves are in $\hat{F}_R$ before step (3). Then, by refining through the intersection points of $\hat{\alpha}_f$, we create the axis-aligned bounding box (AABB) of the curve. $\hat{\alpha}_f$ (which cannot touch the face boundary as proven above) splits the face into two parts, including the bottom-left and top-right vertices of this face, resp. A strictly decreasing curve in this face must connect the left-top and bottom-right edges intersecting $\hat{\alpha}_f$. However, this is not possible since $\hat{\alpha}_f$ and $\hat{\alpha}'_f$ are disconnected components of a non-self-intersecting curve $\hat{\gamma}_f^0$ (since $\boldsymbol{\phi}_F$ is a diffeomorphism).

□

Proposition 3.3.5. *Given $f, f' \in \Lambda^2(K)$, $f \neq f'$, and connected components $\hat{\alpha}_f \in C_f(\hat{F}_R)$ and $\hat{\alpha}_{f'} \in C_{f'}$, satisfy the following properties in $\hat{F}_R \in \text{ref}_1(\hat{F})$,*

(i) $\hat{\alpha}_f$ intersects at most once an edge $\hat{\varepsilon} \in \Lambda^1(\hat{F}_R)$ and is not tangent to the edge;

(ii) if $\hat{\alpha}_f$ intersects $\hat{F}_R^\circ$, $\hat{\alpha}_f$ intersects twice $\partial\hat{F}_R$;

(iii) $\hat{\alpha}_f$ and $\hat{\alpha}_{f'}$ do not intersect in $\overline{\hat{F}_R} \setminus \Lambda^0(\hat{F}_R)$;

(iv) $\hat{\alpha}_f$ and $\alpha_{f'}$ intersects at most once in $\overline{\hat{F}_R}$.

Proof. Statement (i) is a direct consequence of Proposition 3.3.4. Let us prove (ii) by contradiction. We assume that there exists a $\hat{\alpha}_f \in C_f(\hat{F}_R)$, $f \in \Lambda^2(K)$, that does not intersect any edge $\hat{\varepsilon} \in \Lambda^1(\hat{F}_R)$. $\hat{\alpha}_f$ is a closed curve in $\mathbb{R}^2$. This curve is not monotonic, which is in contradiction with Proposition 3.3.4. If we assume that $\hat{\alpha}_f \in C_f(\hat{F}_R)$, $f \in \Lambda^2(K)$ intersects only once $\partial\hat{F}_R$, then $\hat{\alpha}_f$ is tangent to an edge $\hat{\varepsilon} \in \Lambda^1(\hat{F}_R)$, which is in contradiction with (i). $\hat{\alpha}_f$ cannot intersect more than twice $\partial\hat{F}_R$ since $\hat{\alpha}_f$ is a connected and non-self-intersecting curve.

All intersections are on vertices of refined faces by step (2) in ref_1, which proves (iii). Step (3) in the refinement rule explicitly enforces (iv).

□

We note that we have not provided yet an algorithm that determines $C_f(\hat{F}_R)$, $f \in \Lambda^1(\hat{F}_R)$, i.e., the $\hat{\alpha}$-curves after ref_1. We can compute all the intersections (vertices) but it still remains open how to connect these vertices. We develop the details of the connectivity of the intersection points in the next section. In any case, assuming that the $\hat{\alpha}$-curves are connected, we can define the following refinement rule ref_2 to compute the intersection $\hat{F}_R \cap K$, where $\hat{F}_R \in \text{ref}_1(\hat{F})$. The ref_2 is not AA but it does not introduce new intersection points, as stated in Theorem 3.3.6 (see Figure 3.5).

Algorithm 12 ref_2: partition by connecting intersections

Split $\hat{F}_R \in \text{ref}_1(\hat{F})$ through each edge $\alpha \in C_f(\hat{F}_R)$, $\forall f \in \Lambda^2(K)$.

Theorem 3.3.6. *The previous refinement rule is such that $\hat{F}_R \in \text{ref}_2 \circ \text{ref}_1(\hat{F})$ has the following properties:*

(i) *$\Lambda^1(\hat{F}_R)$ is composed of AA edges and strictly monotonic nonlinear edges (including potential diagonal edges);*

(ii) *The vertices of $\hat{F}_R$ lay on the boundary of its AABB;*

(iii) *All the nonlinear edges are strictly increasing or all strictly decreasing;*

(iv) *$\hat{F}_R$ is diffeomorphically equivalent to a convex linear polygon P^{ln};*

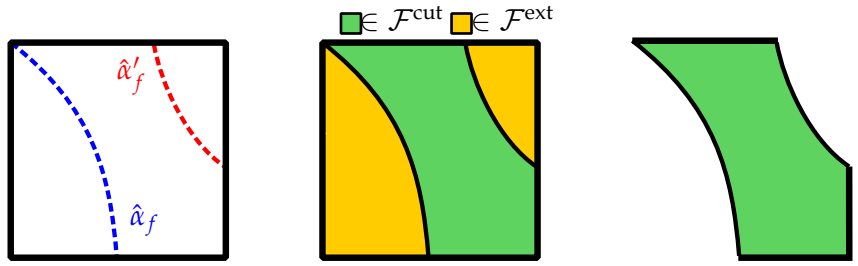
Proof. We note that the refinement ref_1 is a *kd*-tree partition of the square that contains the reference triangle. Let us consider the set of AABBs obtained by this refinement. By construction of the partition, $\hat{F}_R \in \text{ref}_2 \circ \text{ref}_1(\hat{F})$ is the clipping of an AABB by strictly monotonic curves that do not intersect among themselves (see Proposition 3.3.4 (ii) and Proposition 3.3.5). Thus, (i) and (ii) readily hold, while (iii) holds due to Proposition 3.3.4 (iii).

The nonlinear polygon can be represented as a cyclic graph, where the vertices are the intersection points of the edges. We can replace the nonlinear edges with linear ones connecting the vertices, creating a closed linear polygon. Since the nonlinear edges are smooth, the vertices are preserved, and the topology of the surface is preserved, the linear polygon is diffeomorphic to the nonlinear one. Thus, (iv) holds. □

After refining F, the $\hat{\gamma}^0$-curves do not intersect $\hat{F}_R^\circ$, i.e., they belong to $\partial\hat{F}_R$, for $\hat{F}_R \in \text{ref}_2 \circ \text{ref}_1(\hat{F})$. Thus, F_R° can only be inside or outside K. Now, we can classify the faces with respect to K as interior,

$$\mathcal{F}^{\text{cut}} \doteq \{\ \hat{F}_R \in \text{ref}_2 \circ \text{ref}_1(\hat{F})\ :\ \hat{F}_R \subseteq \bigcap_{f \in \Lambda^2(K)} \text{int}(\hat{\gamma}_f)\ \}, \tag{3.3}$$

and exterior, $\mathcal{F}^{\text{ext}} \doteq \{\ \hat{F}_R \in \text{ref}_2 \circ \text{ref}_1(\hat{F})\} \setminus \mathcal{F}^{\text{cut}}$. Remember that K is not open or closed in general, as stated at the beginning of this section. Thus, $\text{int}(\hat{\gamma}_f)$ (where $\text{int}(\cdot)$ denotes the interior), $f \in \Lambda^2(K)$ in (3.3) can be either open or closed (see more details



(a) $\hat{F}_R \in \text{ref}_1(\hat{F})$ (b) $\text{ref}_2(\hat{F}_R)$ (c) $\hat{F}_R \cap K$

Figure 3.5: Representation of clipping algorithm $F \cap K$. In (a), we start with a refined face $\hat{F}_R \in \text{ref}_1(\hat{F})$ such that the $\hat{\alpha}$-curves are strictly monotonic with respect to the axes in the reference space and do not intersect in $\hat{F}_R \setminus \Lambda^0(\hat{F}_R)$. Here, the $\hat{\alpha}$-curves are $\hat{\alpha}_f \in C_f(\hat{F}_R)$ (red) and $\hat{\alpha}_{f'} \in C_{f'}(\hat{F}_R)$ (blue) for $f, f' \in \Lambda^2(K)$. Then, in (b), we generate a partition $\text{ref}_2(\hat{F}_R)$ through the $\hat{\alpha}$-curves, in which we classify the faces with respect to K. Finally, in (c), we restrict $\text{ref}_2(\hat{F}_R)$ inside K (see Algorithm 12).

in [17]). Once the refined faces are classified, we define Algorithm 13 to obtain $F \cap K$ as a partition into nonlinear polygons (see Figure 3.5). We note that, due to Theorem 3.3.6, the linearization of $\mathcal{F}^{\text{ref}}$ can be represented with a cyclic graph (see [17]). Thus, $\mathcal{F}^{\text{cut}}$ can utilize such representation.

Algorithm 13 $F \cap K$

Compute the partition $\mathcal{F}^{\text{ref}} \doteq \text{ref}_2 \circ \text{ref}_1(F)$.
Return the faces $\mathcal{F}^{\text{cut}}$ in $\mathcal{F}^{\text{ref}}$ that are inside K (see (3.3)).

Remark 3.3.7. *For linear F and intersection curves, the AA critical point computation is ill-posed. However, there is no need to add any point in this case, since the intersection is linear. The curve-plane intersection is ill-posed if the curve is contained in the plane (linear edge); analogously for the surface-line intersection (linear face). These intersections can also be disregarded because they are not affecting the* meas_2 *of sets after trimming.*

Remark 3.3.8. *The partition generated by the refinement above may contain a level of refinement that is not required to fulfill all the requirements for Proposition 3.3.5 to hold. Thus, one can perform a coarsening step of $\mathcal{F}^{\text{ref}}$ that reduces the non-essential vertices, edges, and faces, while maintaining a diffeomorphically equivalent surface.*

Remark 3.3.9. *The presence of AA critical points can improve the accuracy of a parametrization step in Section 3.3.5, since the resulting curves are strictly monotonic with respect to the edges of the reference space. Additionally, one can consider another refinement of the γ_f^0-curves to bound the relative chord $\hat{\delta}_{\max}$ up to a given threshold.*

3.3.3 Connection algorithm

Let us consider a face $F \in \text{ref}_1(F^0)$ after the refinement in Algorithm 11. We stress that F hereafter denotes a refined face and F^0 the original face before refinement in the

previous section. We remark that the reference space is always the one of the original face, i.e., the map $\boldsymbol{\phi}_F$ is inherited from the unrefined face. Thus, the curves γ_f^0 are independent of the refinement.

As above, let us consider the set $C_f(\hat{F})$ of mutually disconnected components of $\hat{\gamma}_f^0 \cap \overline{\hat{F}}$ (now at the refined face). We omit the face sub-index, i.e., we use $C(\hat{F})$, to represent the union of all these sets for all faces $f \in \Lambda^2(K)$. In the reference space $\hat{F}$, the curves $\hat{\alpha}_f \in C_f(\hat{F})$ are strictly monotonic smooth curves in $\hat{F} \setminus \Lambda^0(\hat{F})$ (see Proposition 3.3.4); for brevity, we refer as monotonic the curves that are monotonic with respect to $(\hat{\xi}_1, \hat{\xi}_2)$. In Proposition 3.3.4, we prove that all the curves in $C_f(\hat{F})$ are strictly increasing or all curves are strictly decreasing. Moreover, two $\hat{\alpha}$-curves, $\hat{\alpha}_f \in C_f(\hat{F})$ and $\hat{\alpha}_{f'} \in C_{f'}(\hat{F})$, $f, f' \in \Lambda^2(K)$, cannot intersect in $\hat{F} \setminus \Lambda^0(\hat{F})$ (see Proposition 3.3.5). Let us define the set of intersection points:

$$I = \{\hat{\gamma}_f^0 \cap \partial\hat{F} \ : \ f \in \Lambda^2(K)\}.$$

There is an injective map between the intersection points in I and the α-curves. We assume that I is composed of isolated points, i.e., $\hat{\gamma}_f^0 \cap \partial\hat{F}$ is not an edge of $\hat{F}$. This singular case can readily handled, as discussed in Remark 3.3.7.

By construction (clipping of the reference triangle with AA lines), $\hat{F}$ has four edges and the bottom-left corner connects two AA edges. Thus, we can label the edges of $\Lambda^1(\hat{F})$ as $\hat{\varepsilon}_1, ..., \hat{\varepsilon}_4$ in the counterclockwise direction, starting with the leftmost vertical edge. We note that $\hat{\varepsilon}_3$ can be non-AA and $\hat{\varepsilon}_4$ empty, but it does not affect the discussion. The refinement rules ensure that the non-AA edges are not intersected by $\hat{\gamma}_f^0$. One can simply consider the AABB that contains the face, which cannot affect the intersection points, since the $\hat{\alpha}$-curves only connect AA edges that do not change by this step. Thus, we can assume that all edges are AA without loss of generality.

Let us consider $C_f(\hat{F})$ to be composed of strictly increasing curves. We define $\Gamma^+ \doteq \hat{\varepsilon}_1 \cup \hat{\varepsilon}_2$ and $\Gamma^- \doteq \partial\hat{F} \setminus \Gamma^+$ and $I^+ \doteq I \cap \Gamma^+$ and $I^- \doteq I \cap \Gamma^-$. We proceed analogously if $C_f(\hat{F})$ is composed of strictly decreasing curves, but defining $\Gamma^+ \doteq \hat{\varepsilon}_1 \cup \hat{\varepsilon}_4$.

Proposition 3.3.10. *Every node in I^+ is connected to one and only one node in I^-. The opposite is also true. Thus, we can connect all the nodes in I^+ and I^-.*

Proof. Let us consider the case in which $C_f(\hat{F})$ is composed of strictly increasing curves. A node in I^+ cannot be connected to a node in I^+ since one node in I belongs to only one curve, the curves are strictly increasing, and a strictly increasing function that starts in ε_1 or ε_2 cannot intersect again ε_1 or ε_2. Using an analogous argument, we can prove the result when $C_f(\hat{F})$ is composed of strictly decreasing curves. □

Proposition 3.3.11. *If I^+ and I^- are sorted clockwise and anti-clockwise, resp., then each node in I^+ can only be connected to the node in the same position in I^-.*

Proof. Let us assume that $i_1^+ \in I^+$ is connected to i_k^-, $k \neq 1$. The line that connects i_1^+ and i_k^- split $\hat{F}$ into two non-empty parts. We note that this is due to the fact that i_1^+ and

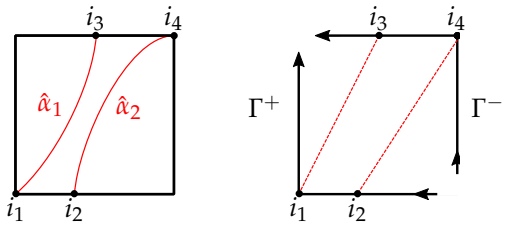


Figure 3.6: Representation of Algorithm 14 for strictly increasing curves. In (a), we present the intersections $i_1, ..., i_4 \in I$ of the curves $\hat{\alpha}_1, \hat{\alpha}_2 \in C_f(\hat{F})$ with $\partial\hat{F}$. These intersections are classified into $I^+ = \{i_3, i_4\}$ and $I^- = \{i_1, i_2\}$ according to their position in the boundary, Γ^+ or Γ^-, resp. These sets are sorted in clockwise and anticlockwise order, resp. This sorting leads to the connection shown in (b) (dashed red). If the curves in $C_f(\hat{F})$ were strictly decreasing, the figures would be symmetric to these ones.

i_k^+ cannot be on the same edge. One of the parts contains nodes $i_1^-, ..., i_{k-1}^- \in I^-$ and the other part contains $I^+ \setminus i_1^+$. By Proposition 3.3.10, nodes contained in the two different regions are connected by α-curves. But these α-curves then intersect the one connecting i_1^+ and i_k^-. This is not possible, since these are disconnected components of a non-self-intersecting curve. Thus, i_1^+ can only be connected to i_1^-. We proceed analogously for the other nodes. □

This way, we compute all the connections in the graph with Algorithm 14. First, in line 1, we compute the intersections between $\hat{\gamma}_f^0$ and $\partial\hat{F}$ using intersection algorithms (see Figure 3.6(a)). In line 3, we return the straightforward connection of two points. Otherwise, in line 5, we compute the derivative sign of the α-curves in $C_f(\hat{F})$ (e.g., evaluating the gradient $\boldsymbol{\nabla}(\hat{\gamma}_f^0)$ at any intersection point $i \in I$). Next, we connect[1] the nodes in the sorted sets I^+ and I^- according to Proposition 3.3.11 (line 6-8). The algorithm returns a set of edges $\mathcal{E}$ that connect the intersections I. These connections define entirely the $\hat{\alpha}$-curves, which are then used in Algorithm 12. We note that the full connection algorithm is barely used in practice if we start with a surface mesh with a small chordal error of its linearisation.

Algorithm 14 `connect`$(\hat{F}, \hat{\gamma}_f)$

1: $I \leftarrow \{\hat{\gamma}_f^0 \cap \partial\hat{F}\}$
2: **if** `length`$(I) = 2$ **then**
3: **return** $\mathcal{E} \leftarrow \{i_1, i_2 \in I\}$
4: **end if**
5: $s \leftarrow$ `derivative_sign`(γ_f^0)
6: $I^+, I^- \leftarrow$ `split_by_position`(I, s)
7: $I^+, I^- \leftarrow$ `cyclic_sort_by_sign`(I^-, I^+, s)
8: **return** $\mathcal{E} \leftarrow \{\ (i^+, i^-) \in$ `zip`$(\ I^+,\ I^-)\ \}$

[1] In line 8, we use the notation `zip` to iterate simultaneously over multiple iterators of the same length, e.g., $\boldsymbol{a}$ and $\boldsymbol{b}$. Each iteration returns a tuple of the i^{th} value of each interator, (a_i, b_i).

Remark 3.3.12. *In the computation of intersection points in Algorithm 14, we disregard the ones that do not logically represent an intersection. E.g., if the level set value γ_f does not change the sign around a vertex with zero level set value.*

One can observe that for this algorithm to work, I must have an even number of elements (after filtering I with Remark 3.3.12). It is true since the $\hat{\alpha}$-curves are monotonic. Thus, any $\hat{\alpha}_f \in C_f(\hat{F})$ that $\hat{\alpha}_f \cap \hat{F} \setminus \Lambda^0(\hat{F}) \neq \emptyset$ can only intersect twice $\partial\hat{F}$ (see Theorem 3.3.6).

3.3.4 Surface partition

Let us consider a nonlinear general polygon $\hat{F}$ in $\mathbb{R}^2$, e.g., $F \in \mathcal{F}^{\text{cut}}$. In order to parametrize F, we need a partition into regular polygons, e.g., simplices or quadrilaterals. This parameterization is needed to compute bulk and surface integrals when using (3.2.3). However, this is not needed when transforming these integrals into edge integrals using (3.2.3).

Let us recover the definition of a kernel point. A kernel point can be connected by a segment to any other point of a polytope without intersecting its boundary. The union of all possible kernel points is the kernel polytope, which is convex. The kernel polytope of a convex polytope is itself [124]. In a linear polytope P^{ln}, we can compute a kernel point by finding a point that belongs to the half-spaces defined by the faces of P^{ln}; see more details in [103].

Finding the kernel of a nonlinear polytope $\hat{F}$ is not trivial. One can find a lower bound of the kernel polytope with the convex hull of the nonlinear faces. If a kernel point exists for a given polytope $\hat{F}$, we can compute a simplex partition of $\hat{F}$ using linear edges. However, the kernel point does not exist in general.

Proposition 3.3.13. *If $\hat{F}$ has the properties of Theorem 3.3.6 then $\hat{F}$ can be partitioned into a set of nonlinear triangles and quadrilaterals by adding linear edges only.*

Proof. Let us assume that $\Lambda^1(\hat{F})$ is composed of two strictly increasing nonlinear curves and a set of AA edges. According to Theorem 3.3.6, these curves and edges can only intersect at the boundary of the AABB of $\hat{F}$. Thus, we connect the endpoints of the nonlinear curves with non-increasing linear edges. Since the new edges are non-increasing, they can only intersect once each curve, i.e., at the endpoints. This connection generates a nonlineal quadrilateral $\hat{Q} \subseteq \hat{F}$. The remaining parts $\hat{F} \setminus \hat{Q}$ are linear and convex (see Theorem 3.3.6), in which a simplex partition is straightforward. □

Therefore, we can compute a hybrid partition if a nonlinear polytope $\hat{F}$ has no kernel point We note that the nonlinear quadrilaterals of Proposition 3.3.13 are composed of two nonlinear edges and two linear edges. Thus, they can be represented with a tensor product map with anisotropic order, e.g., $q \times 1$ where q is the order of the nonlinear edges. If needed, we can decompose the nonlinear quadrilaterals into triangles within their reference space.

Remark 3.3.14. *We can consider a coarsening of $\mathcal{F}^{\text{cut}}$ that satisfies Proposition 3.3.5 (see Remark 3.3.8) if the resulting polygons have a kernel point. We note that the coarsening stated in Remark 3.3.8 can lead to non-convex polygons.*

3.3.5 Surface parametrization

The intersection $\mathcal{B}_K^{\text{cut}} \doteq \mathcal{B} \cap K$ is a nonlinear surface mesh in which the intersection curves are implicitly represented. Thus, we need a parametrization of the edges and surfaces for further operations. We utilize a least-squares method [30] combined with a sampling strategy [51]. This iterative process converges to an accurate parametrization of the intersection curves and trimmed surfaces.

In the least-squares method we can find the Bézier control points $X^b = \{x_j^b\}_{j=1}^m$ that minimize $X^\ell - \mathbf{B}X^b$ where $\{\mathbf{B}\}_{ij} = b_j(\xi_i)$ are the Bézier basis evaluated at the reference points $\{\xi_j\}_{j=1}^n$ and $X_\xi = \{x_j^\ell\}_{j=1}^n$ represents the set of sampling points (see [97] for more details). When $\mathbf{B}$ is not a square matrix, i.e., $n > m$, X_b is approximated through a linear least-squares operation. To isolate the approximation between the d-faces, one can recursively perform a least-squares operation on the interior points, increasing the dimension. Note that for square matrices, i.e., $n = m$, we are building the Bézier extraction operator [29].

The authors in [51] discussed several sampling strategies and demonstrated similar convergence in FE analysis. In each of the sampling strategies, they solve a nonlinear problem for every sampling point. In our case, we aim to parametrize the intersection curves $F \cap G,\ F \in \mathcal{B},\ G \in \Lambda^2(K)$. Thus, we define a sampling strategy to represent the intersection curves by solving a closest point problem. Specifically, we find

$$\hat{\boldsymbol{\xi}} \in \hat{F} \ : \ (\boldsymbol{x}_0 - \boldsymbol{\phi}_F(\hat{\boldsymbol{\xi}})) \cdot \boldsymbol{n}_\pi = 0, \qquad (\boldsymbol{x}_0 - \boldsymbol{\phi}_F(\hat{\boldsymbol{\xi}})) \cdot (\boldsymbol{n}_\pi \times (\boldsymbol{\partial}_{\hat{\xi}_1}(\boldsymbol{\phi}_F(\hat{\boldsymbol{\xi}})) \times \boldsymbol{\partial}_{\hat{\xi}_2}(\boldsymbol{\phi}_F(\hat{\boldsymbol{\xi}})))) = 0.$$

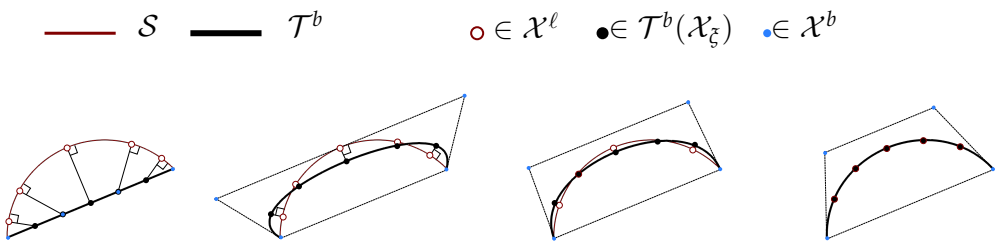
When we compute the closest point to a surface, e.g., when parametrizing the interior of a nonlinear face, the problem reads as follows: find

$$\hat{\boldsymbol{\xi}} \in \hat{F} \ : \qquad (\boldsymbol{x}_0 - \boldsymbol{\phi}_F(\hat{\boldsymbol{\xi}})) \cdot \boldsymbol{\nabla}(\boldsymbol{\phi}_F(\hat{\boldsymbol{\xi}})) = \mathbf{0}.$$

This algorithm requires a seed point in the physical space $\boldsymbol{x}_0$, which does not belong to the surface or curve, and an initial reference point $\hat{\boldsymbol{\xi}}_0 \in \hat{F}$. This sampling strategy, as well as others, assumes relatively small curvature in the curves and surfaces. The curvature has already been bounded in Section 3.3.2.

The least-squares method with a sampling strategy is insufficient for an accurate parametrization, see the example of Figure 3.7(b). The authors in [30] propose a method to optimize the reference points $\{\xi_j\}_{j=1}^n$ iteratively. We use a similar approach that fixes the reference points and recomputes the sampling points at each iteration. This approach allows us to compute the least-squares operator $\mathbf{B}^+$ only once. In addition, the sampling will be evenly spaced in the reference space. The example in Figure 3.7

shows how the approximation improves with the iterations. This method applies to the parametrization of curves and surfaces.



(a) it $= 0$ (b) it $= 1$ (c) it $= 5$ (d) it $= 15$

Figure 3.7: Example of iterative approximation of a curve. From the linear approximation in (a), we sample the closest points. These sampling points X^ℓ are approximated in a Bézier curve $\mathcal{X}^b$ in (b). In (b), we evaluate the approximated seed points $\mathcal{T}^b(\mathcal{X}_{\tilde{\xi}})$ to compute the new sampling points. Across the iteration, in (c) and (d), the approximation improves by reducing the distance between the sampling points and the curve. In (d), the sampling points are evenly spaced in the reference space (up to a stopping criterion). Note that this example corresponds to a particular case of the Algorithm 15 in which the approximation degree is fixed to $p = 3$.

The parameterization method is described in Algorithm 15. Even though this algorithm is designed for $\mathcal{B}_K^{\text{cut}}$, it is general to any nonlinear mesh, namely $\mathcal{S}$. We first generate a linear approximation of $\mathcal{S}$ in line 1 with a simplex partition (or a partition into standard polytopes, see Section 3.3.4). In line 2, we initialize the sampling points through the degree elevation, see [47]. Then, we parametrize the d-faces from lower to higher dimension line 3. Each d is parametrized from lower to higher order line 4, in simplices, we start with $p^0 = d$. The gradual increment of the approximation error improves the convergence. In each iteration, we compute a Bézier mesh with the least-squares method then we compute the sampling points from the evaluations of the Bézier mesh. When the variation of the error estimator is below a tolerance, we increase the degree. Finally, we return the Bézier mesh with the least-squares method of the highest degree.

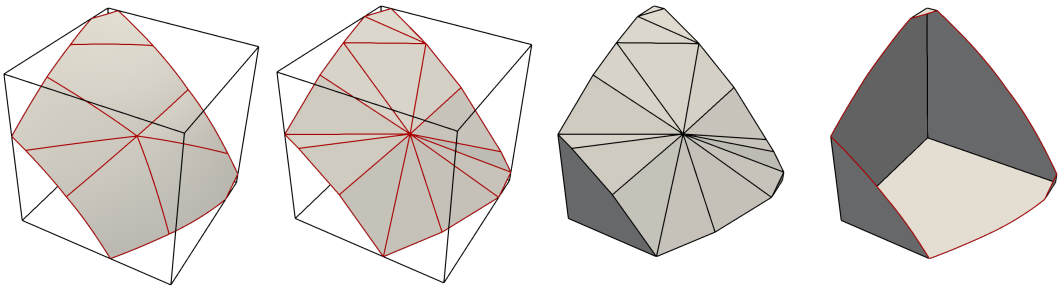
3.3.6 Cell intersection

The intersection of a cell with the domain $K^{\text{cut}} \doteq K \cap \Omega$ can be represented using its boundary $\partial K^{\text{cut}} \doteq (\partial K)^{\text{cut}} \cup \mathcal{B}_K^{\text{cut}}$, as stated in Section 3.2.2. In Algorithm 16 , we aim to compute $(\partial K)^{\text{cut}} \doteq \partial K \cap \Omega = \partial K \cap \text{int}(\mathcal{B}_K^{\text{cut}})$ using the linear algorithms from [17]. Here, $\text{int}(\mathcal{B}_K^{\text{cut}})$ represents the domain bounded by $\mathcal{B}_K^{\text{cut}}$. For this purpose, we first need a linearized surface $\mathcal{B}_K^{\text{lin}} \doteq \texttt{lin}(\mathcal{B}_K^{\text{cut}})$ partitioned into simplices (see line 1 and Figure 3.8(a) to Figure 3.8(b)). Since $\mathcal{B}_K^{\text{cut}}$ is composed of nonlinear polytopes that are diffeomorphically equivalent to linear and convex polytopes, both linearization and simplex decomposition are straightforward.

Algorithm 15 `parametrize`$(\mathcal{T}, D)$

1: $\mathcal{T}^{\text{lin}} \leftarrow$ `simplexify`$(\mathcal{S}, D)$
2: $\mathcal{T}^{\ell} \leftarrow$ `elevation`$(\mathcal{T}^{\text{lin}}, p^{\ell})$
3: **for** $d \in \{1, \dots, D\}$ **do**
4: **for** $p \in \{p^0, \dots, p^b\}$ **do**
5: $e^- \leftarrow \infty$; $\Delta e \leftarrow \infty$
6: **while** $\Delta e > \epsilon$ **do**
7: $\mathcal{T}^b \leftarrow \mathbf{B}_p^+ \cdot \mathcal{T}_d^{\ell}$
8: $\mathcal{T}_d^{\ell} \leftarrow$ `sample`$(\mathcal{T}^b, \mathcal{T}_d^{\ell}, \mathcal{S})$
9: $e \leftarrow \|\mathcal{T}^b - \mathcal{T}_d^{\ell}\|_{\ell^2}$; $\Delta e \leftarrow |e - e^-|/e$; $e^- \leftarrow e$
10: **end while**
11: **end for**
12: **end for**
13: **return** $\mathbf{B}_{p^b}^+ \cdot \mathcal{T}^{\ell}$

The main algorithm in [17] provides a linearized partition for K^{cut}, i.e., $\mathcal{T}_K^{\text{lin}} \doteq K \cap \mathcal{B}_K^{\text{lin}}$ in line 2 and Figure 3.8(c). From $\mathcal{T}_K^{\text{lin}}$, we can obtain $(\partial K)^{\text{lin}}$ by filtering the faces that belong to ∂K in line 3. Additionally, we merge cells $P \in (\partial K)^{\text{lin}}$ such that $\Lambda^1(P)$ is composed of cell edges $\varepsilon_K \subseteq \varepsilon \in \Lambda^1(K)$ and surface edges $\varepsilon_{\mathcal{B}} \in \Lambda^1(\mathcal{B}_K^{\text{lin}})$. It is important to note that the surface edges $\varepsilon_{\mathcal{B}} \in \Lambda^1(\mathcal{B}_K^{\text{lin}})$ have bijective map to $\varepsilon'_{\mathcal{B}} \in \Lambda^1(\mathcal{B}_K^{\text{cut}})$, namely $\Phi_{\mathcal{B}} : \varepsilon_{\mathcal{B}} \mapsto \varepsilon'_{\mathcal{B}}$. Therefore, we recover the nonlinear cell boundary intersection $(\partial K)^{\text{cut}}$ by replacing the $\varepsilon_{\mathcal{B}}$ edges with the ones parametrized in $\mathcal{B}_K^{\text{cut}}$ (see line 4 and Figure 3.8(d)).



(a) $\mathcal{B}_K^{\text{cut}}$ (b) $\mathcal{B}_K^{\text{lin}}$ (c) $\mathcal{T}_K^{\text{lin}}$ (d) $(\partial K)^{\text{cut}}$

Figure 3.8: Representation of the steps that generate $(\partial K)^{\text{cut}}$. First, the surface portion $\mathcal{B}_K^{\text{cut}}$ (see (a)) is linearized and partitioned into simplices ($\mathcal{B}_K^{\text{lin}}$ in (b)). Then, in (c), we intersect the background cell K with the half-spaces of the planes of $\mathcal{B}^{\text{lin}}$ using linear algorithms, resulting in $\mathcal{T}_K^{\text{lin}}$. This linear intersection is possible since the faces $F \in \mathcal{B}_K^{\text{lin}}$ are planar. Next, we extract the boundary of $\mathcal{T}_K^{\text{lin}}$ that belong to ∂K, leading to $(\partial K)^{\text{lin}}$. Finally, we replace the edges in $(\partial K)^{\text{lin}}$ by the ones in $\mathcal{B}_K^{\text{cut}}$ to obtain $(\partial K)^{\text{cut}}$ (see (d)). The boundary of $K \cap \Omega$ is represented by $(\partial K)^{\text{cut}} \cup \mathcal{B}_K^{\text{cut}}$.

The parametrization of the edges of $(\partial K)^{\text{cut}}$ is extracted from $\mathcal{B}_K^{\text{cut}}$. However, the surface parametrization requires a surface partition into standard polytopes. We utilize the methods described in Section 3.3.4 to build such a partition. In this case, the AA critical points and AA partitions are defined in the reference space of $P \in (\partial K)^{\text{cut}}$,

Algorithm 16 $\partial K \cap \text{int}(\mathcal{B}_K^{\text{cut}}) \to (\partial K)^{\text{cut}}$

1: $\mathcal{B}_K^{\text{lin}} \leftarrow \texttt{lin}(\mathcal{B}_K^{\text{cut}})$
2: $\mathcal{T}_K^{\text{lin}} \leftarrow K \cap \text{int}(\mathcal{B}_K^{\text{lin}})$
3: $(\partial K)^{\text{lin}} \leftarrow \{F \in \Lambda^2(P) : P \in T_K^{\text{lin}},\ F \subset \partial K\}$
4: **return** $(\partial K)^{\text{cut}} \leftarrow \texttt{replace\_edges}(\ (\partial K)^{\text{lin}},\ \Phi_{\mathcal{B}})$

i.e., in the reference space of $F \in \Lambda^2(K)$. It is worth noting that, in this partition, the intersections of the nonlinear edges $\varepsilon \in \Lambda^1(\mathcal{B}_K^{\text{cut}})$ with AA lines are computed in the reference space of ε. This fact ensures local conformity.

3.3.7 Global algorithm

The algorithms presented in the previous sections are defined cell-wise. In this section, we describe an algorithm that allows us to integrate FE functions in the whole domain and its boundary. Each cell of the background mesh is intersected by the domain bounded by a high-order Bézier surface mesh. Therefore, to proceed, we first need to generate this surface mesh from the given BREP, e.g., analytical functions or CAD representation. From CAD models we can generate high-order surface meshes with a third-party library, e.g., `gmsh` [54]. We can convert these meshes into Bézier patches with a Bézier projection operation or a least-squares approximation.

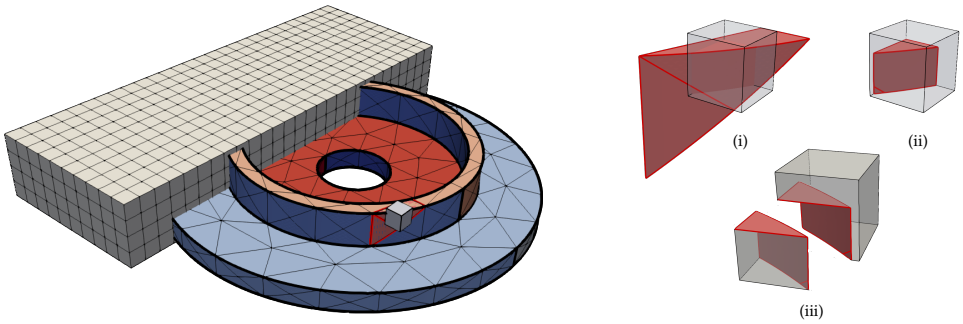
The intersection of a background cell $K \in \mathcal{T}$ with the whole surface mesh $\mathcal{B}$ would be inefficient. Therefore, we restrict the surface mesh to the faces colliding with the cell K. We perform this operation in a preprocessing step similar to [17]. The interrogations are approximated by linear operations on the convex hull of each Bézier patch. We can accelerate these queries with a hierarchy of simpler bounding domains, e.g., AABBs, oriented bounding boxes (OBBs) or discrete orientation polytopes (k-DOPs) [68, 122]. We note that these operations prioritize speed over accuracy as the subsequent operations can deal with false positives in the surface restriction.

The global algorithm is described in Algorithm 17 and demonstrated through an example in Figure 3.9. First, in line 2, the Bézier mesh is extracted from the given representation, e.g., CAD model in Figure 3.9(a). For each cell $K \in \mathcal{T}$, we restrict the faces $F \in \mathcal{B}$ touching K, see Figure 3.9(b) step (i) and line 4. The surface $\mathcal{B}_K$ is intersected by the walls of K, see step (ii) and line 5. Then, the cell boundary is intersected by the half-spaces of the intersected surface $\mathcal{B}_K^{\text{cut}}$ (line 6). Both boundary intersections $\mathcal{B}_K^{\text{cut}} \cup (\partial K)^{\text{cut}}$ represent the boundary of the cut cell K^{cut}, see step (iii). The parameterization of these is stored for integration purposes (line 7).

Once we have classified the non-intersected cells as described in [17], we can proceed with the integration over the entire domain. The integration strategy depends on the dimension of the parametrization used in line 7. In the methods described in this section, we utilize high-order 2-faces for parametrization, enabling us to employ Stokes theorem for integration [40] in combination with moment-fitting methods [19].

Algorithm 17 $\mathcal{T} \cap \text{int}(\mathcal{B}^{\text{CAD}})$

1: $\mathcal{T}^{\text{cut}} \leftarrow \emptyset$
2: $\mathcal{B} \leftarrow \texttt{extraction}(\mathcal{B}^{\text{CAD}})$
3: **for** $K \in \mathcal{T}$ **do**
4: $\quad \mathcal{B}_K \leftarrow \texttt{restrict}(\mathcal{B}, K)$
5: $\quad \mathcal{B}_K^{\text{cut}} \leftarrow \mathcal{B}_K \cap K$
6: $\quad (\partial K)^{\text{cut}} \leftarrow \partial K \cap \text{int}(\mathcal{B}_K^{\text{cut}})$
7: $\quad \mathcal{T}^{\text{cut}} \leftarrow \mathcal{T}^{\text{cut}} \cup \texttt{parametrize}(\mathcal{B}_K^{\text{cut}}) \cup \texttt{parametrize}((\partial K)^{\text{cut}})$
8: **end for**
9: **return** $\mathcal{T}^{\text{cut}}$



(a) CAD and cell sample (b) Local main steps

Figure 3.9: In (a) the CAD geometry (colored CAD entities) is first approximated into a high-order surface mesh. In each cell of the background mesh (b), in (i) we restrict the nonlinear faces touching the background cell. Then, in (ii) we clip the nonlinear faces by the background cell walls. Finally, in (iii) we build the polytopal representation of the intersected $\partial K \cap \Omega$. Afterward, we prepare the polytpes for the integration, e.g., with a surface parametrization.

However, when using edge parametrizations, it is necessary to employ Stokes theorem over trimmed faces [57].

3.4 Numerical experiments

In the numerical experiments, we aim to demonstrate the robustness of the method and optimal convergence of the geometrical and PDE solution approximations. First, we demonstrate *hp*-convergence of the intersection and parametrization methods in Section 3.4.2. Then, in Section 3.4.3, we show the robustness of the method concerning the relative position of the background mesh and the geometry. Next, in Section 3.4.4, we show the optimal *hp*-convergence of the FE analysis for a manufactured solution of the Poisson equation and an elasticity benchmark. Finally, we show the application of the method in real-world examples on CAD described in terms of standard for the exchange of product model data (STEP) files.

3.4.1 Experimental setup

The numerical experiments have been performed on Gadi, a high-end supercomputer at the NCI (Canberra, Australia) with 4962 nodes, 3074 of them powered by a 2 x 24 core Intel Xeon Platinum 8274 (Cascade Lake) at 3.2 GHz and 192 GB RAM. The algorithms presented in this work have been implemented in the Julia programming language [27]. The unfitted FE computations have been performed using the Julia FE library `Gridap.jl` [21, 116] version 0.17.17 and the extension package for unfitted methods `GridapEmbedded.jl` version 0.8.1 [118]. `STLCutters.jl` version 0.1.6 [76] has been used to compute intersection computations on STL geometries. The computations of the convex hulls have been performed with `DirectQhull.jl` version 0.2.0 [61], a Julia wrapper of the `qhull` library [23].

The CAD geometry preprocess is done in `gmsh` library [54] by using the `GridapGmsh.jl` Julia wrapper [115]. The `gmsh` library calls Open CASCADE Technology (OCCT) [90] as a parser for STEP files. The sample geometries are extracted from [90] and [56].

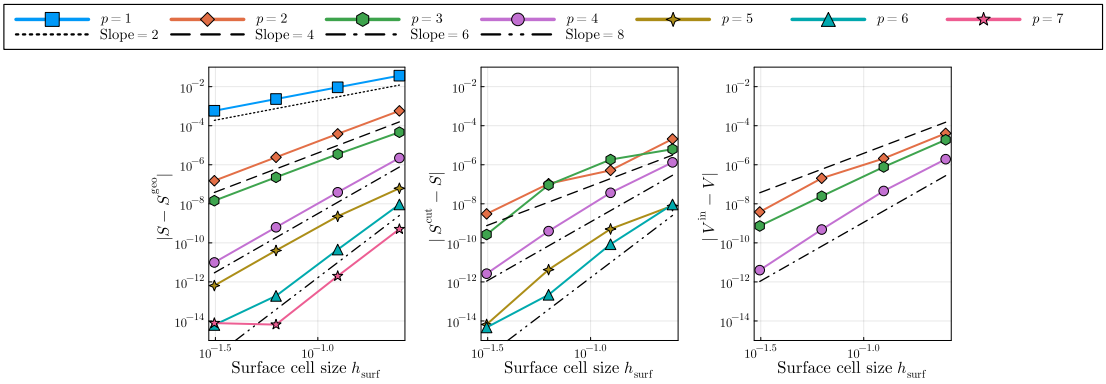
3.4.2 Approximation and parametrization analysis

In this section, we analyze the approximation of the geometry by a Bézier mesh $\mathcal{B}$. We use a sphere as test geometry. We define the sphere by the boundary of a reference cube bumped by $f(\hat{\boldsymbol{x}}) = \boldsymbol{x}_0 + R(\hat{\boldsymbol{x}} - \hat{\boldsymbol{x}}_0)/\|\hat{\boldsymbol{x}} - \hat{\boldsymbol{x}}_0\|$ where $\hat{\boldsymbol{x}}_0$ is the center of the reference cube, $\boldsymbol{x}_0$ is the center of the sphere and $R = 1$ is the radius of the sphere in our experiments. In the following experiments, we consider a triangular surface mesh for the reference cube boundary obtained from the convex decomposition of a Cartesian mesh. We define the relative cell size as $h_{\text{surf}} = 1/n_{\text{surf}}$ where n_{surf} is the number of elements in each Cartesian direction. The surface partition of the sphere is approximated by a Bézier mesh $\mathcal{B}$ of order p by using a least-squares operation. This sphere is embedded in a

cube of side $L = 3$. This domain is discretized with a background Cartesian mesh $\mathcal{T}$ of relative cell size $h = 1/n$ where n is the number of cells in each direction. During the intersection of the surface, the relative chord of the edges is bounded to $\hat{\delta}_{\max} < 0.1$. The surfaces are computed by integrating a unit function on the high-order surface mesh, while we use Stokes theorem in the volume computation.

In the surface approximation experiments of Figure 3.10(a), we test a matrix of surface cell sizes $h_{\text{surf}} = 2^{-\alpha}$, with $\alpha = 2,..,5$ and a range of Bézier orders $p = 1,..,7$. We compute the Bézier approximation error as the difference between the surface of $\mathcal{B}$ and the analytical surface. We observe that the convergence rate of the approximation errors is $p+1$ for odd orders and $p+2$ for even orders. In [1], one can observe similar convergence rates of the surface and volume errors.

In Figure 3.10(b), the surface mesh $\mathcal{B}$ is intersecting by each background cell $K \in \mathcal{T}$. This generates $\mathcal{B}^{\text{cut}}$. These intersections are performed for surfaces $\mathcal{B}$ of order p and parametrized with order p, where $p = 2,...,6$. In Figure 3.10(c), we present the results of intersecting each background cell $K \in \mathcal{T}$ with the domain bounded by $\mathcal{B}$, resulting in $\mathcal{T}^{\text{cut}}$. In this case, the approximation of $\mathcal{B}$ and surface parametrization are computed with order $p = 2,3,4$ for computational reasons. In both cases, the cell size of the background mesh is fixed to $h = 2^{-2}$. We observe in Figure 3.10(b) and Figure 3.10(c) that the convergence rates of the cut surface error, $|\texttt{surf}(\mathcal{B}^{\text{cut}}) - \texttt{surf}(\mathcal{B})|$, and the cut volume error, $|\texttt{vol}(\mathcal{T}^{\text{cut}}) - \texttt{vol}(\mathcal{B})|$, are similar to the convergence rates of Figure 3.10(a).



(a) (b) (c)

Figure 3.10: Surface and volume errors of the approximation and the reparametrization of a sphere. In (a), the surface error of approximating the geometry $\mathcal{B}^{\text{geo}}$ into a Bézier mesh $\mathcal{B}$, $|S - S^{\text{geo}}|$ where $S^{\text{geo}} = \texttt{surf}(\mathcal{B}^{\text{geo}})$ and $S = \texttt{surf}(\mathcal{B})$. In (b), the parametrization error of $\mathcal{B}^{\text{cut}}$, the intersection of $\mathcal{B}$ with each background cell $K \in \mathcal{T}$, $|S^{\text{cut}} - S|$ where $S^{\text{geo}} = \texttt{surf}(\mathcal{B}^{\text{geo}})$. In (c), the volume integration error of $\mathcal{T}^{\text{in}}$, the intersection of each background cell $K \in \mathcal{T}$ with the domain bounded by $\mathcal{B}$, $|V^{\text{in}} - V|$ where $V^{\text{in}} = \texttt{vol}(\mathcal{T}^{\text{in}})$ and $V = \texttt{vol}(\mathcal{B})$.

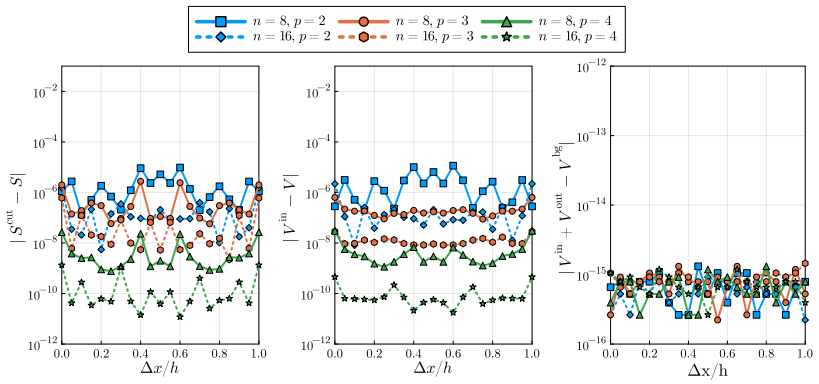
3.4.3 Robustness experiments

In this section, we demonstrate robustness concerning the relative position of $\mathcal{B}$ and $\mathcal{T}$. For these experiments, we consider a sphere with the same geometrical setup as

in Section 3.4.2. We perform the intersections with the same relative background cell size and surface cell size $h = h_{\text{surf}} = 2^{\alpha}$, $\alpha = 3, 4$. We test for the approximation and parametrization orders $p = 2, ..., 4$. In each combination, we shift the geometry $\Delta x = (i/N)h$, with $i = 1, ..., N$ where $N = 20$, from the origin.

In Figure 3.11(a) and Figure 3.11(b), we observe the surface and volume variation of the surface error, $|\texttt{surf}(\mathcal{B}^{\text{cut}}) - \texttt{surf}(\mathcal{B})|$, and the volume error, $|\texttt{vol}(\mathcal{T}^{\text{in}}) - \texttt{vol}(\mathcal{B})|$, resp. Here, $\mathcal{T}^{\text{in}} \doteq \mathcal{T}^{\text{cut}} \cup \{K \in \mathcal{T} : K \cap \partial\Omega = \emptyset\}$ is the physical volume mesh. The variations of the surface error are approximately two orders of magnitude and the variations of volume are one order of magnitude.

However, in Figure 3.11(c), we observe a machine precision error in the domain volume error $|\texttt{vol}(\mathcal{T}^{\text{in}}) + \texttt{vol}(\mathcal{T}^{\text{out}}) - \texttt{vol}(\mathcal{T}^{\text{bg}})|$, where $\mathcal{T}^{\text{out}} \doteq \mathcal{T}^{\text{bg}} \setminus \mathcal{T}^{\text{in}}$ is complementary of $\mathcal{T}^{\text{in}}$. The low error is due to the conformity between inside and outside polytopes of the cut cell described in Section 3.3.6.



(a) (b) (c)

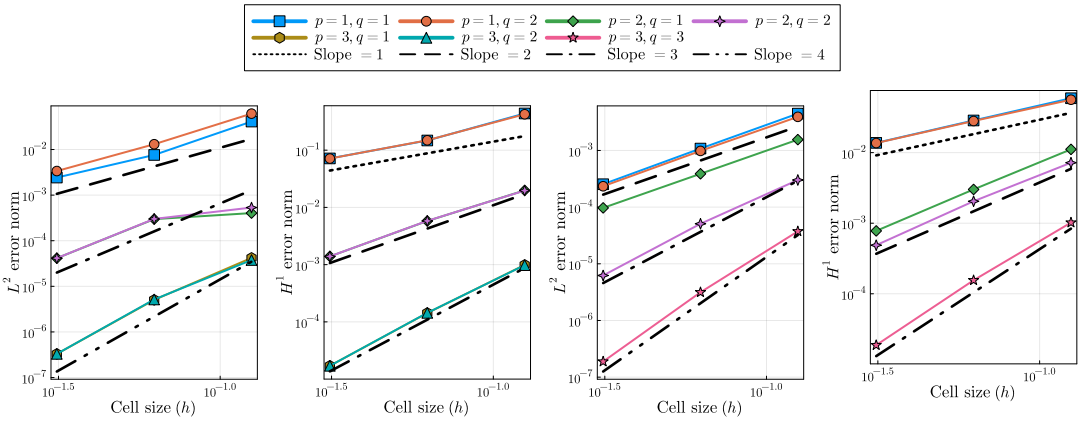
Figure 3.11: Demonstration of robustnes concerning the relative position of $\mathcal{B}$ and $\mathcal{T}$. Both plots have surface and volume errors when shifting a sphere in the embedded domain. Even though the surface and volume errors in (a) and (b) show variations, the errors are bounded. In (c), the domain volume errors are close to machine precision because the inside and outside polytopes of the cut cell are conforming, see Section 3.3.6

3.4.4 Unfitted FE experiments

In these experiments, we explore the behavior of the intersection algorithms by solving PDEs with unfitted FEs on two analytical benchmarks and two realistic examples. We analyze a Poisson equation with a manufactured solution on a sphere and a spherical cavity benchmark with an analytically derived solution [31]. Both experiments are performed with the geometrical setup described in Section 3.4.2. However, in the spherical cavity experiments, the domain is the outside of an octant of the sphere. We compute the FE solutions using AgFEM with modal $\mathcal{C}^0$ basis and moment-fitting quadratures [19], even though the proposed framework can be used with other unfitted methods like ghost penalty stabilization [33]. In AgFEM, we aggregate all cut cells.

In the Poisson experiments, we consider Dirichlet boundary conditions and a forcing term that satisfies the manufactured solution $u(x,y,z) = x^a + y^a$, with $a = 6$. We compute the convergence tests for $h = h_{\text{surf}} = 2^{-\alpha}$, with $\alpha = 3,4,5$, for the FE order $p = 1,2,3$ and the geometrical order $q = 1,2,3$ of the approximation and parametrizations. In Figure 3.12(a) and Figure 3.12(b), we can observe that the L^2 and H^1 errors converge with the optimal rate, $p+1$ and p, resp. This convergence is independent of the geometrical order q, as expected, the geometry description does not affect the manufactured solution problem.

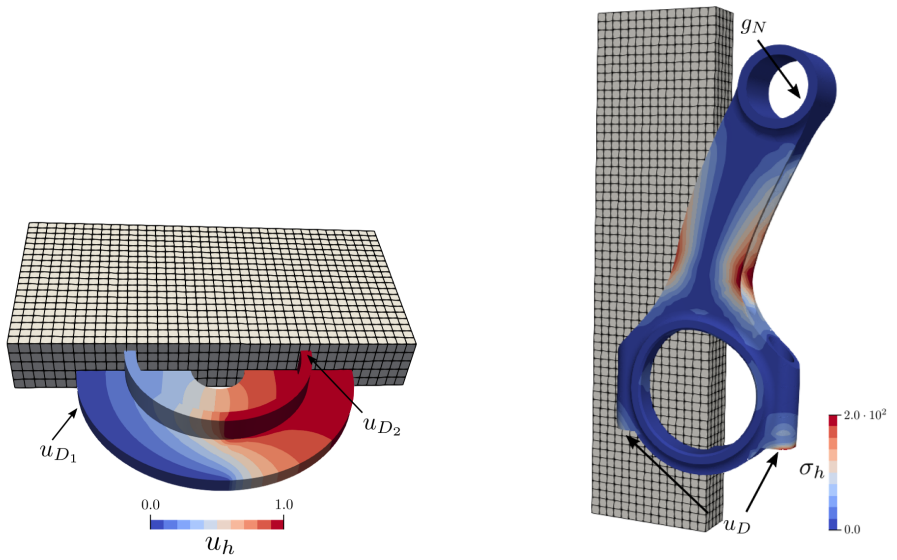
In the linear elasticity benchmark of the spherical cavity, we derive the potentials with automatic differentiation in Julia. We compute the tests for the same cell sizes $h = h_{\text{surf}} = 2^{-\alpha}$ and orders $p = 1,2,3$ and $q = 1,2,3$ than in the Poisson experiments. We set consider a Young modulus $E = 10^5$ and Poisson ratio $\nu = 0.3$. We observe in Figure 3.12(c) and Figure 3.12(d) the expected convergence rates for the L^2 and H^1 errors, $\min(p,q)+1$ and p resp. This demonstrates that we require high-order discretizations to accurately solve PDEs with high-order unfitted FE methods.



(a) Poisson L^2 (b) Poisson H^1 (c) Linear Elasticity L^2 (d) Linear Elasticity H^1

Figure 3.12: Convergence tests in AgFEM. Convergence of the manufactured solution with a Poisson equation in a sphere, in (a) and (b). Convergence of the linear elasticity benchmark in a spherical cavity, in (c) and (d). Some combinations of FE order p and geometry order q are not shown for the sake of conciseness.

Finally, we demonstrate the viability of the method in real-world geometries defined by CAD models in STEP files. These files are extracted from [90] and [56], resp. On each CAD geometry, we generated a high-order surface mesh using `gmsh`. Then, we converted this mesh into a Bézier mesh with a least-squares method. The indexing of topological entities of the CAD surface, $\mathcal{B}^{\text{CAD}}$, is preserved in the intersected mesh $\mathcal{B}^{\text{cut}}$. Thus, we can impose Dirichlet and Neumann boundary conditions over the entities in $\mathcal{B}^{\text{CAD}}$. We consider a heat problem and an elasticity problem in the two different geometries Figure 3.13. The experiments are described in the figure caption.



(a) (b)

Figure 3.13: Realistic examples on CAD geometries. In (a) we consider a heat equation with thermal conductivity $k = 1.0$ and Dirichlet boundary conditions on two opposite entities ($u_{D_1} = 0$ and $u_{D_2} = 1$). The background mesh is defined in a AABB 20% larger than the geometry with $40 \times 40 \times 5$ elements of order $p = 2$. The surface is approximated in 524 quadratic Bézier patches. In (b) we consider a linear elasticity problem with Young modulus $E = 10^5$ and Poisson ratio $\nu = 0.3$, with Dirichlet and Neumann boundary conditions ($u_D = (0, 0, 0)$ and $g_N = (1, 0, -1)$, resp.). The magnification factor of the deformation is 75. The background mesh is defined in an AABB 20% bigger than the geometry with $30 \times 10 \times 60$ elements of order $p = 2$. The surface is approximated in 1786 quadratic Bézier patches.

3.5 Conclusions and future work

In this work, we have designed an automated pipeline for numerically approximating PDEs in complex domains defined by high-order boundary meshes using unfitted FE formulations in a structured background mesh. The main challenge of the method lies in the numerical integration of the background cells intersected by the domain boundary. This requires handling the intersection between background cells and complex boundary meshes, including the computation of trimming curves and dealing with nonlinear and nonconvex domains.

We have presented a novel intersection algorithm for general high-order surfaces and polytopal cells, which is accurate and robust. The algorithm is based on mesh partition methods for nonlinear level sets, linear clipping algorithms for general polytopes, multivariate root finding, geometrical least-squares methods and the properties of Bézier patches. The result of this algorithm is a set of nonlinear general polyhedra that represent the cut cells of the background mesh. We parametrize the boundary of these polyhedra using sets of Bézier patches, taking advantage of concepts like convex hull and kernel point concepts. These Bézier patches can be used to integrate the bulk with moment-fitting quadratures.

The implementation, accuracy and robustness of the geometrical algorithm have been tested on high-order geometries defined by analytical methods and CAD models. In our tests, we observe optimal convergence of the numerical approximations, limited only by rounding errors. Additionally, we have observed the robustness of the method when varying the relative position of the background mesh. Furthermore, we have successfully solved PDEs on geometries defined by nonlinear boundary representations with high-order FE methods. We have demonstrated optimal convergence of the solutions in the designed benchmarks. We have also shown the viability of the method in real-world geometries defined by CAD models in STEP files. These results position the method as a pioneering computational framework for simulating PDEs on high-order geometries with unfitted FE methods. It provides an automatic geometrical and functional discretization that can be especially useful within shape and topology optimization loops, inverse problems with unknown boundaries and interfaces, and transient problems with moving domains.

Future work involves the extension of the method for other background discretizations, e.g., octree meshes for adaptive refinement. We also plan to extend the method to distributed memory [16], since the method is cell-wise parallel, allowing us to solve larger problems. We can optimize the method by parametrizing only the polytopal edges and using moment-fitting integration on the surfaces [57]. Further extension of the method involves solving boundary layer problems with a separate discretization, see [119]. Finally, we plan to extend the method to more complex scenarios and practical applications, such as fluid-structure interaction (FSI) and transient problems.

Chapter 4

Space-time unfitted finite element method for moving explicit boundary representations

This work proposes a novel variational approximation of partial differential equations on moving geometries determined by explicit boundary representations. The benefits of the proposed formulation are the ability to handle large displacements of explicitly represented domain boundaries without generating body-fitted meshes and remeshing techniques. For the space discretisation, we use a background mesh and an unfitted method that relies on integration on cut cells only. We perform this intersection by using clipping algorithms. To deal with the mesh movement, we pullback the equations to a reference configuration (the spatial mesh at the initial time slab times the time interval) that is constant in time. This way, the geometrical intersection algorithm is only required in 3D, another key property of the proposed scheme. At the end of the time slab, we compute the deformed mesh, intersect the deformed boundary with the background mesh, and consider an exact transfer operator between meshes to compute jump terms in the time discontinuous Galerkin integration. The transfer is also computed using geometrical intersection algorithms. We demonstrate the applicability of the method to fluid problems around rotating (2D and 3D) geometries described by oriented boundary meshes. We also provide a set of numerical experiments that show the optimal convergence of the method.

4.1 Introduction

Space-time formulations for FEMs are valuable techniques for solving transient numerical problems. Unlike standard time stepping schemes, which employ different discretizations for space and time, space-time formulations discretize the problem simultaneously in both space and time. However, generating 4D space-time body-fitted meshes becomes extremely challenging on moving complex geometries. There are no general-purpose tools to generate 4D meshes. Besides, the re-meshing process due to moving domains introduces projection errors when transferring between meshes.

The bottleneck of mesh generation can be addressed by employing unfitted FEMs. Unfitted FEMs, also known as embedded or immersed FEMs, offers a solution that eliminates the need for body-fitted mesh generation, relying instead on a simple background grid, such as a uniform or adaptive Cartesian grid. For large-scale simulations on distributed-memory platforms, one can replace expensive (both in terms of computational time and memory) unstructured mesh partitioning [66] by efficient tree-mesh partitioners with load balancing [36, 37]. Unfitted FEMs have gained increasing popularity in various applications, including FSI [35, 49, 99], fracture mechanics [44, 55], additive manufacturing [38, 87], and stochastic geometry problems [9]. Traditionally, unfitted problems utilize level sets to describe geometries. However, recent work [17] has extended unfitted discretizations to simulate geometries described by CAD models, i.e., explicit geometry representations.

The small cut cell problem, commonly discussed in the literature [43], is a significant limitation of unfitted FEMs. The intersection between the physical domain and the background cells can become arbitrarily small, thereby giving rise to ill-conditioning issues. While several authors have attempted to address this problem, only a few techniques have demonstrated robustness and optimal convergence. The ghost penalty methods [33], employed within the so-called CutFEM framework [34], represent one such approach. Alternatively, cell agglomeration techniques have emerged as viable options to ensure robustness in the presence of cut cells, naturally applied on DG methods [84]. Extensions to the C^0 Lagrangian FE have been introduced in [22], while mixed methods have been explored in [14], where the AgFEM term was coined. AgFEM exhibits good numerical qualities, including stability, bounds on condition numbers, optimal convergence, and continuity concerning data. Distributed implementations have been exploited in [11, 117], while AgFEM has also been extended to h-adaptive meshes [85] and higher-order FE [19]. In [18], a novel technique combining ghost penalty methods with AgFEM was proposed, offering reduced sensitivity to stabilization parameters compared to standard ghost methods.

In the body-fitted case, one can consider variational space-time formulations [24, 111, 114] to avoid the generation of 4D meshes. In these formulations, a body-fitted mesh is extruded using a geometric mapping technique to represent the temporal evolution of the boundary displacements. However, when dealing with large displacements, the geometric mapping process becomes ill-posed, often necessitating re-meshing. Similar challenges arise in body-fitted ALE [45, 89] schemes, where frequent re-meshing becomes necessary due to large changes in topology. Consequently, none of these formulations are optimal when confronted with large displacements. Furthermore, the re-meshing process introduces bottlenecks within the simulation procedure and incurs in projection errors when transferring between meshes.

Despite the notable advantages of unfitted FEMs, its application in the space-time domain [7, 60] is currently limited to implicit level set geometrical representations (in which 4D geometrical treatment is attainable), 2D explicit representations (in which 3D

geometrical tools, e.g., in [17], can readily be used for space-time) or piecewise constant approximations of the geometry in time [70]. This limitation stems from the complexity involved in developing 4D geometrical tools for unfitted methods.

The present work introduces a novel variational space-time formulation for unfitted FEMs that eliminates the need for complex 4D geometrical algorithms. Instead, our formulation relies on time-extruded FE spaces. Furthermore, we incorporate an *exact* inter-slab transfer mechanism through an intersection algorithm, which removes projection errors between time slabs. The main contributions of our work are as follows:

- We develop space-time formulations for unfitted FEMs on moving 3D geometries described explicitly.
- We propose to pull back the problem to a reference configuration that is constant in each time slab to avoid 4D geometrical algorithms.
- We propose and implement an exact time-slab mechanism transfer that relies on 3D intersection algorithms.
- We compute the solution of transient problems on complex unfitted domains with large displacements.

The outline of this chapter is as follows. In Section 4.2, we introduce the geometry description and the space-time FE spaces employed in this work. In Section 4.3, we present the variational formulation through a model problem. We also define the integration measures and the inter-slab transfer mechanism. In Section 4.4, we describe the intersection algorithm for time slab transfer. Then, in Section 4.5, we present numerical results for space and time convergence, condition number tests, and a numerical example of a moving domain. Finally, in Section 4.6, we present our conclusions and future work.

4.2 Space time unfitted finite element method

In this section, we introduce the geometry description by utilizing a geometrical map defined in a space-time FE space. We also define the unfitted space-time FE spaces that are necessary to support our proposed formulation.

4.2.1 Geometry description for moving domains

Let us consider an initial Lipschitz domain Ω_0 in a description suitable for unfitted FEM. Following the methodology described in [17], we assume that Ω_0 is represented by a parameterized surface, e.g., a STL mesh or a CAD model. For simplicity, we consider an oriented surface mesh $\mathcal{B}_h^0$ where the interior corresponds to the domain Ω_0. Let $[0, T]$ be the time interval of interest.

We consider the coordinates of the mesh nodes of $\mathcal{B}_h$ to be time-depedendent. Consequently, $\mathcal{B}_h(t)$ and its corresponding interior $\Omega(t)$. This variation of coordinates can be represented by a map $\boldsymbol{D} : \mathcal{B}_h(0) \times [0,T] \to \mathcal{B}_h(t)$ such that $\boldsymbol{D}(\cdot, 0)$ is the identity.

Next, we define the space-time domain $Q = \{x \in \Omega(t) : t \in [0,T]\}$, in accordance with the nomenclature in [7], see Figure 4.1. Additionally, we partition the boundary of the domain $\partial\Omega(t)$ into Dirichlet and Neumann boundaries, $\partial\Omega_D(t)$ and $\partial\Omega_N(t)$, resp. This partition is such that $\partial\Omega(t) = \partial\Omega_D(t) \cup \partial\Omega_N(t)$ and $\partial\Omega_D(t) \cap \partial\Omega_N(t) = \emptyset$. The space-time boundaries are defined as $\partial Q_* \doteq \bigcup_{t\in[0,T]} \partial\Omega_*(t) \times \{t\}$ for $* \in \{N, D\}$. Consequently, the boundary of Q is given by $\partial Q \doteq \Omega(0) \cup \Omega(T) \cup \partial Q_D \cup \partial Q_N$.

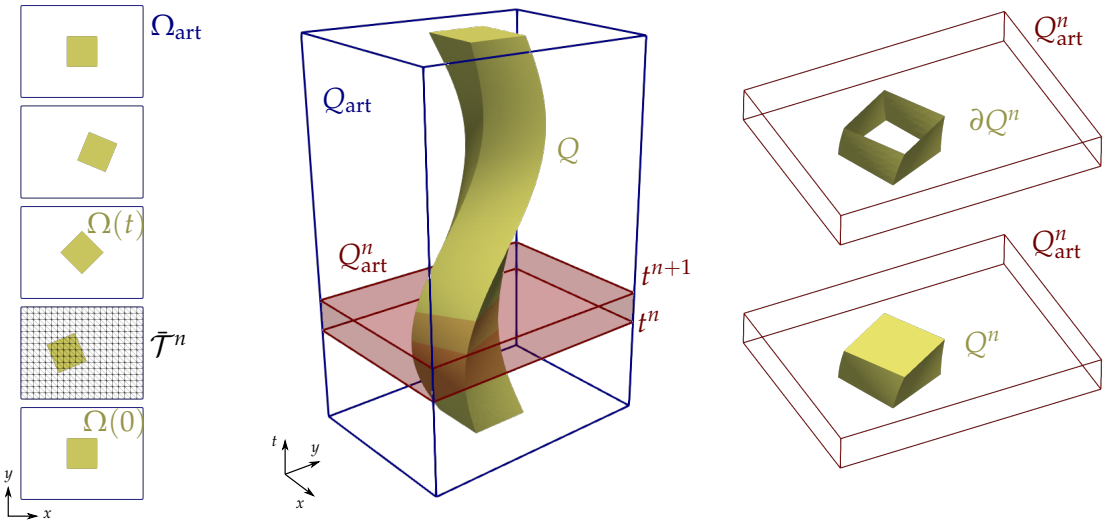


Figure 4.1: Representation of the space-time domain Q embedded within an artificial space-time domain Q_{art}. The spacial domains $\Omega(t)$ and Ω_{art} in 2D are extruded in the time dimension. The solution is computed in each time-slab $J^n = (t^n, t^{n+1})$ on the space-time domain Q^n, which is embedded in Q_{art}.

In addition, we introduce an artificial spatial domain Ω_{art}, such that $\Omega(t) \subset \Omega_{\text{art}}$ for all $t \in [0,T]$. This artificial domain can be a simple geometric shape, such as a bounding box, that can conveniently be meshed with a regular grid, such as a Cartesian mesh. Subsequently, we define the artificial space-time domain $Q_{\text{art}} \doteq \Omega_{\text{art}} \times [0,T]$, such that $Q \subset Q_{\text{art}}$.

We define a partition in time, $\{J^n\}_{n=1}^N$ for $[0,T]$, in which the time domain is divided into time slabs. The time slabs are defined as $J^n = (t^n, t^{n+1}), 1 < n < N$ where $t^n < t^{n+1}, \forall n \in 1, ..., N$. Within each time slab, the time-step size is defined as $\tau^n = t^{n+1} - t^n$ and the domain time-step size is defined as $\tau = \max_{n=1,...,N} \tau^n$. In each time-slab, we define an artificial space-time domain $Q^n_{\text{art}} \doteq \Omega_{\text{art}} \times J^n$. In a time slab, the space-time domain is determined by $Q^n \doteq Q^n_{\text{art}} \cap Q$. Similarly, the space-time boundary is denoted as $\partial Q^n_{\{D,N\}}$. We also introduce the notation $\Omega^n \doteq \Omega(t^n),\ n = 1, ..., N+1$.

Finally, we define an undeformed space-time domain at each time slab as $\hat{Q}^n \doteq \Omega^n \times J^n$, see Figure 4.2. Here, we note that Ω^n is the initial spatial domain of the time

slab $J^n = (t^n, t^{n+1})$. In order to define the reference configuration, we require a map $\boldsymbol{\varphi}_h^n : \hat{Q}^n \to Q^n$ that must satisfy

$$\boldsymbol{\varphi}_h^n(\partial\Omega(t)) = \boldsymbol{D}(t)(\mathcal{B}_h^0),$$

i.e., respect the boundary position determined by $\boldsymbol{D}$.

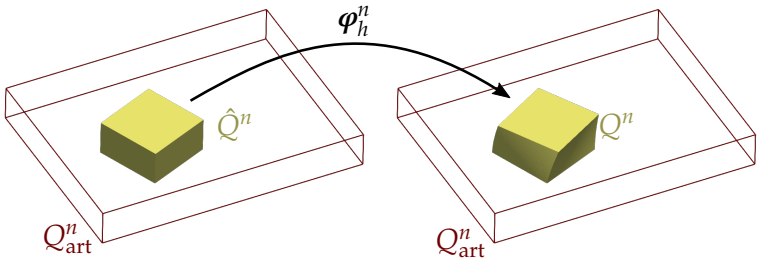


Figure 4.2: Representation of the deformation map $\boldsymbol{\varphi}_h^n$ resulting from extending the variation of the surface map $\boldsymbol{D}$ in the time slab domain Q^n (or in the time slab artifical domain Q^n_{art}). The undeformed configuration $\hat{Q}^n$ will be used for the FE analysis.

Let $\bar{\mathcal{T}}^n_{\text{art}}$ represent a simple partition of Ω_{art} at time t^n. Even though Ω_{art} does not evolve in time, the partition $\bar{\mathcal{T}}^n_{\text{art}}$ can differ across time slabs, e.g., when using AMR techniques. We define the partition of Q^n_{art} as a Cartesian product of $\bar{\mathcal{T}}^n_{\text{art}}$ and J^n. Specifically, $\mathcal{T}^n_{\text{art}} \doteq \{\bar{K} \times J^n : \bar{K} \in \bar{\mathcal{T}}^n_{\text{art}}\}$. We can classify the spatial artificial cells $K \in \bar{\mathcal{T}}^n_{\text{art}}$ as interior, cut, and exterior depending on the relative position concerning the domain boundary $\partial\Omega^n$. Since there is a one-to-one map between cells in $\bar{\mathcal{T}}^n_{\text{art}}$ and $\mathcal{T}^n_{\text{art}}$, it already provides a classification of the cells in the space-time artificial mesh $\mathcal{T}^n_{\text{art}}$, which corresponds to the in-out classification for the extruded space-time domain boundary $\partial\hat{Q}^n$.

In the context of unfitted FEM, we are interested in solving PDEs using the active portion of $\mathcal{T}^n_{\text{art}}$, i.e., the cells touching Ω^n at t^n. Thus, we remove the exterior cells $\bar{\mathcal{T}}^n_{\text{out}} \doteq \{\bar{K} \in \bar{\mathcal{T}}^n_{\text{art}} : \bar{K} \cap \Omega^n = \emptyset\}$ from the artificial spatial partition $\bar{\mathcal{T}}^n_{\text{art}}$, leading to an active spatial partition $\bar{\mathcal{T}}^n \doteq \bar{\mathcal{T}}^n_{\text{art}} \setminus \bar{\mathcal{T}}^n_{\text{out}}$. From the spatial active partition, we compute the active space-time partition $\mathcal{T}^n \doteq \{\bar{K} \times J^n : \bar{K} \in \bar{\mathcal{T}}^n\}$ by extrusion.

4.2.2 Space-time finite element spaces

In this section, we define space-time unfitted FE spaces for the discretization of the model problem on moving domains. However, a main difference with respect to [7], the original problem on Q^n will be re-state in the undeformed domain $\hat{Q}^n$. As a result, in this section, we only need a construction that works on time-extruded domains. Thus, the active mesh $\mathcal{T}^n$ is the one that intersects Ω^n at t^n, as defined above. We note that $\mathcal{T}^n$ may change across time slabs in evolving domains.

Firstly, we define the FE space $\mathcal{X}_h^n$ in the spatial domain. Then, for each DOF in $\mathcal{X}_h$, we introduce a 1D FE basis $\mathcal{Y}_\tau^n$. Thus, we construct the space-time FE space composed as the tensor product of these two spaces, $\mathcal{V}_h^n \doteq \mathcal{X}_h^n \otimes \mathcal{Y}_\tau^n$.

In each space-time cell $T^n = \bar{T}^n \times J^n \in \mathcal{T}^n$, we can define the FE local interpolation, which consists of tensor product DOFs $\Sigma \doteq \Sigma_X \otimes \Sigma_Y$ and shape functions $\Phi \doteq \Phi_X \otimes \Phi_Y$. Specifically, any shape function $\phi \in \Phi$ can be expressed as $\phi^{(\alpha_X,\alpha_Y)}(\boldsymbol{x},t) = \phi_X^{\alpha_X}(\boldsymbol{x}) \otimes \phi_Y^{\alpha_Y}(t)$.

To address the ill-conditioning issues in unfitted FEM, we will utilize as a model example the AgFEM, even though the proposed numerical framework can readily be used for other techniques, e.g., ghost penalty stabilization (see [7] for space-time unfitted formulations). The AgFEM eliminates problematic DOFs by constraining them to well-posed DOFs using a discrete extension operator $\mathcal{E}$. We define the spatial extension operator $\bar{\mathcal{E}} : \bar{\mathcal{V}}_{h,\text{in}} \mapsto \bar{\mathcal{V}}_h$ between the the spatial interior $\bar{\mathcal{V}}_{h,\text{in}}$ and active $\bar{\mathcal{V}}_h$ FE spaces . The spatial aggregated finite element (AgFE) space is defined as $\bar{\mathcal{V}}_{h,\text{ag}} \doteq \bar{\mathcal{E}}(\bar{\mathcal{V}}_{h,\text{in}})$

We define a slab-wise space-time discrete operator between $\mathcal{V}_{h,\text{in}}^n$ and $\mathcal{V}_h^n$. In each time slab, we aggregate the cut cells in the spatial active mesh $\bar{\mathcal{T}}^n$ using the aggregation techniques from the AgFEM. For a given time slab J^n, the space-time extension operator $\mathcal{E}^n : \mathcal{V}_{h,\text{in}}^n \mapsto \mathcal{V}_h^n$ is defined as

$$\mathcal{E}^n(\mathcal{V}_{h,\text{in}}^n) = \mathcal{E}^n(\bar{\mathcal{V}}_{h,\text{in}}^n \otimes \mathcal{Y}_\tau^n) = \bar{\mathcal{E}}^n(\bar{\mathcal{V}}_{h,\text{in}}^n) \otimes \mathcal{Y}_\tau^n,$$

where $\bar{\mathcal{E}}^n : \bar{\mathcal{V}}_{h,\text{in}}^n \mapsto \bar{\mathcal{V}}_h^n$ represents the spatial extension operator, and $\mathcal{Y}_\tau^n$ the FE space in time at the time slab J^n. We define the space-time AgFE space on the time-slab. Now, we can define the global AgFE space as $\mathcal{V}_{h,\text{ag}} \doteq \mathcal{V}_{h,\text{ag}}^1 \times \cdots \times \mathcal{V}_{h,\text{ag}}^N$. Note that no continuity is imposed across time slabs, i.e., the discrete time space is discontinuous.

In ghost penalty formulations, one can readily use the non-aggregated spaces on the active mesh and add stabilization terms (see, e.g., [42]) to make the problem robust with respect to cut locations. Thus, the space-time FE spaces to be used in this case is simply $\mathcal{V}_h^n$.

4.2.3 Extension of the deformation map

In the case in which the domain is represented in time by its explicit boundary representation and its boundary displacement $\boldsymbol{D}(t)$, we must extend $\boldsymbol{D}(t)$ defined on $\mathcal{B}(t)$ to the space-time domain $\hat{Q}^n$ of each time slab J^n. For this purpose, we solve the following linear elasticity problem in $\hat{Q}^n$, even though other extension operators could be considered. Find $\hat{\boldsymbol{u}} \in \mathbb{R}^d$ such that,

$$\begin{cases} \hat{\boldsymbol{\nabla}}_x \cdot \boldsymbol{\sigma}(\hat{\boldsymbol{u}}) = 0 & \text{in } \hat{Q}^n, \\ \hat{\boldsymbol{u}} = \hat{\boldsymbol{u}}_D^n & \text{on } \partial\hat{Q}_D^n, \\ \hat{\boldsymbol{u}} = 0 & \text{on } \partial\hat{Q}_{D_0}^n, \\ \mathbf{n}_x \cdot \boldsymbol{\sigma}(\hat{\boldsymbol{u}}) = 0 & \text{on } \partial\hat{Q}_N^n. \end{cases}$$

where $\boldsymbol{\sigma}$ is the stress tensor, $\hat{\boldsymbol{\nabla}}_x$ is the spatial gradient, $\mathbf{n}_x$ is the spatial component of outward normal vector to $\partial\hat{Q}^n$, and

$$\hat{\boldsymbol{u}}_D^n(t) = (\boldsymbol{D}(t) - \boldsymbol{D}(t^n)) \circ (\boldsymbol{D}(t^n))^{-1}$$

is the Dirichlet boundary condition on $\partial\hat{Q}_D^n = \mathcal{B}_h(t^n) \times J^n$, which imposes the deformation near the unfitted boundary (see Figure 4.3). On the boundary of the artificial domain and at the initial spatial domain, $\partial\hat{Q}_{D_0}^n = \partial\Omega_{\text{art}} \times J^n \cup \Omega^n \times \{t^n\}$, the deformation is fixed to zero. The Neumann boundary is defined on $\partial\hat{Q}_N^n = \Omega^n \times \{t^{n+1}\}$.

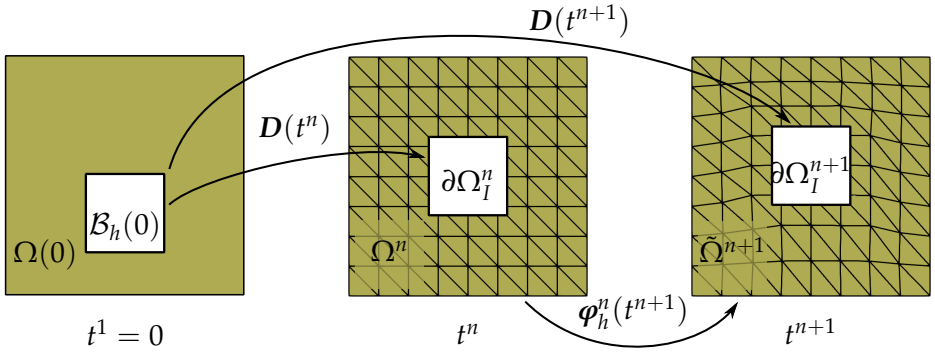


Figure 4.3: Representation of the deformation map $\boldsymbol{D}$ and the extended map $\boldsymbol{\varphi}_h^n$ in the time slab $J^n = (t^{n-1}, t^n)$. This example represents the external domain. The spatial deformation map $\boldsymbol{D}(t) : \mathcal{B}_h(0) \mapsto \mathcal{B}_h(t)$, $t \in [0, T]$, is defined on the surface mesh, and the extended map $\boldsymbol{\varphi}_h^n(t) : \Omega^n \mapsto \tilde{\Omega}(t)$, $t \in J^n$, is defined on the spatial domain. At t^{n+1}, the description of the inner boundary $\partial\Omega_I^{n+1}$ may differ across maps: $\boldsymbol{\varphi}_h^n(t^{n+1})(\partial\Omega_I^n) \approx \boldsymbol{D}(t^{n+1})(\mathcal{B}_h(0))$, where $\partial\Omega_I^n = \boldsymbol{D}(t^n)(\mathcal{B}_h(0))$. The approximation error decreases with the spatial discretization size h.

This problem is approximated with a continuous Galerkin (CG) method and weak imposition of the Dirichlet boundary conditions on the unfitted boundary $\partial\hat{Q}_D$ using Nitsche's method [88]. In the weak formulation, we find $u \in \mathcal{V}_{h,\text{ag}}^n$ such that,

$$a(\hat{\boldsymbol{u}}, \hat{\boldsymbol{v}}) = l(\hat{\boldsymbol{v}}), \quad \forall \hat{\boldsymbol{v}} \in \mathcal{V}_{h,\text{ag}}^n$$

where the bilinear form and the linear form are defined as,

$$a(\hat{\boldsymbol{u}}, \hat{\boldsymbol{v}}) = \int_{\hat{Q}^n} \boldsymbol{\varepsilon}(\hat{\boldsymbol{v}}) : \boldsymbol{\sigma}(\boldsymbol{\varepsilon}(\hat{\boldsymbol{u}}))d\hat{\boldsymbol{x}}dt + \int_{\partial\hat{Q}_D} \left(\tau_D \hat{\boldsymbol{v}} \cdot \hat{\boldsymbol{u}} - v \cdot (\mathbf{n}_x \cdot \boldsymbol{\sigma}(\boldsymbol{\varepsilon}(\hat{\boldsymbol{u}})))\right) d\hat{\boldsymbol{x}}dt,$$

and

$$l(\hat{\boldsymbol{v}}) = \int_{\partial\hat{Q}_D} \left(\hat{\boldsymbol{v}} \cdot \hat{\boldsymbol{u}}_D - \mathbf{n}_x \cdot \boldsymbol{\sigma}(\boldsymbol{\varepsilon}(\hat{\boldsymbol{v}})) \cdot \hat{\boldsymbol{u}}_D\right) d\hat{\boldsymbol{x}}dt,$$

resp., where $\boldsymbol{\varepsilon}$ is the symmetric (spatial) gradient operator.

From the approximated solution $\hat{\boldsymbol{u}}_h^n \in \mathcal{V}_{h,\text{ag}}^n$ we extract the map $\boldsymbol{\varphi}_h^n(\hat{\boldsymbol{x}}, t) = \hat{\boldsymbol{x}} + \hat{\boldsymbol{u}}_h^n(\hat{\boldsymbol{x}}, t)$. One can alternatively use a ghost penalty stabilization to solve the elasticity problem. Since the deformation is imposed weakly, the deformation $\hat{\boldsymbol{u}}_h^n$ is not equal to $\hat{\boldsymbol{u}}_D^n$ on $\mathcal{B}_h(t)$. Thus, the deformed space-time domain Q^n is approximated by $\tilde{Q}^n =$

$\boldsymbol{\varphi}_h^n(\hat{Q}^n)$. In any case, the geometrical error is expected to decrease with the mesh size h. For given time $t \in J^n$, we can extract the spatial map $\boldsymbol{\varphi}_h^n(t) : \Omega^n \mapsto \tilde{\Omega}(t)$ (see Figure 4.3). Here, the spatial domain is also approximated, $\Omega(t) \approx \tilde{\Omega}(t)$.

We assume that the computed map is one-to-one, i.e., $\det(\boldsymbol{\nabla}\boldsymbol{\varphi}_h^n) > 0$ at all times. Assuming that $\boldsymbol{D}$ is such that it admits an extension that is diffeomorphic, this requirement can be attained for small enough time steps in the time-discrete problem. One can also move back to the Cartesian mesh after several time steps, as soon as the map remains bijective.

4.2.4 Extended active mesh

As discussed above, in the case in which the deformation map has to be computed, $\boldsymbol{\varphi}_h^n$ is not equal to $\boldsymbol{D}$ on $\mathcal{B}_h(t)$, since unfitted methods usually make use of weak imposition of boundary conditions.

At the end of the time slab J^n, we can define the solution in the approximated domain $\tilde{Q}^n$ as

$$u_h^{n+1,-}(\boldsymbol{x}) = \hat{u}_h^n(\boldsymbol{\varphi}_h^n(t^{n+1})^{-1}(\boldsymbol{x}), t^{n+1}), \quad \forall \boldsymbol{x} \in \tilde{\Omega}^{n+1},$$

which is a function defined on the undeformed domain Ω^n; since we integrate the forms on $\hat{Q}^n$, the inverse map is never computed in practice. On the other side, with the method proposed below, we have to compute an inter-slab integral on Ω^{n+1} that involves $u_h^{n+1,-}$.

In order for $u_h^{n+1,-}$ to be defined on Ω^{n+1}, we proceed as follows. First, we extend the active mesh by $\bar{\mathcal{T}}_{\text{ext}}^n = \bar{\mathcal{T}}^n \cup \{K \in \bar{\mathcal{T}}_{\text{art}}^n : \boldsymbol{\varphi}^n(t^{n+1})(K) \cap \Omega^{n+1} \neq \emptyset\}$. Assuming that the geometrical map is one-to-one, the inverse $\boldsymbol{\varphi}_h^n(t^{n+1})^{-1}$ is well-defined on Ω^{n+1}.

To accommodate a FE space $\bar{\mathcal{V}}_{h,\text{ext}}^n$ in $\bar{\mathcal{T}}_{\text{ext}}^n$, one can consider the modification of the extension operator. In the AgFEM framework, we can simply redefine the extension operator $\bar{\mathcal{E}}_{\text{ext}}^n : \bar{\mathcal{V}}_{h,\text{in}}^n \mapsto \bar{\mathcal{V}}_{h,\text{ext}}^n$. In non-aggregated methods like CutFEM, we can utilize the same extension on the solution $\bar{\mathcal{E}}_{\text{ext}}^n(\bar{\mathcal{V}}_h^n)$.

We note that the implementation of the extended triangulation and FE space can be computed a posteriori, on demand, whenever it is needed. One can mark the additional cells that are needed for the extension, and then extend the aggregates to these cells.

4.3 Variational formulation on a model problem

In this section, we establish the space-time variational formulation by employing a model problem, specifically the heat equation. Although we use the heat equation for demonstration purposes, it is essential to note that a similar approach can be applied to other PDEs.

4.3.1 Weak formulation

In order to define the space-time variational formulation, we first define the convection-diffusion equation in a space-time domain Q, as find u such that,

$$\begin{cases} \partial_t u + (\boldsymbol{w} \cdot \boldsymbol{\nabla}_x) u - \mu \Delta_x u = f & \text{in } Q, \\ u = u_D & \text{on } \partial Q_D, \\ \mu \mathbf{n}_x \cdot \boldsymbol{\nabla}_x u = g_N & \text{on } \partial Q_N, \\ u = u_0 & \text{on } Q(0). \end{cases} \tag{4.1}$$

where μ is the diffusion coefficient, $\boldsymbol{w}$ the advection velocity, f the source term, u_D the Dirichlet boundary condition, and g_N boundary flux on ∂Q_N. $\boldsymbol{\nabla}_x$ and Δ_x denote the spatial gradient and spatial Laplacian, resp. Well-posedness requires that $\boldsymbol{w} \cdot \mathbf{n}_x + n_t \geq 0$ on the Neumann boundary ∂Q_N. Here, $\mathbf{n} = (\mathbf{n}_x, n_t)$ is the outward normal to ∂Q.

We discretize this problem with the AgFEM in space (or a ghost penalty stabilization) and a DG method in time. We weakly impose the Dirichlet boundary conditions using Nistche's method [88]. Since the coupling between time slabs respects causality, we analyze the problem on a single time slab, assuming we know the solution of the previous one (see also [7]). To analyze each time slab $J^n = (t^n, t^{n-1})$, we define the problem in the reference domain $\hat{Q}^n = (\boldsymbol{\varphi}_h^n)^{-1}(\tilde{Q}^n)$ as: find $\hat{u} \in \mathcal{V}_{h,\text{ag}}^n$ such that

$$B_h^n(\hat{u}, \hat{v}) = L_h^n(\hat{v}), \quad \forall \hat{v} \in \mathcal{V}_{h,\text{ag}}^n,$$

with $u = \hat{u} \circ (\boldsymbol{\varphi}_h^n)^{-1}$. The bilinear form reads as:

$$\begin{aligned} B_h^n(\hat{u}, \hat{v}) &= \int_{\hat{Q}^n} \hat{v} \partial_t^n \hat{u} |J_{\tilde{Q}^n}| d\hat{\boldsymbol{x}} dt + \int_{\Omega^n} \hat{v}(t^n) \hat{u}(t^n) |J_{\Omega^n}| d\hat{\boldsymbol{x}} + a_h(\hat{u}, \hat{v}), \\ a_h(\hat{u}, \hat{v}) &= \int_{\hat{Q}^n} (\mu \boldsymbol{\nabla}_x^n \hat{u} \cdot \boldsymbol{\nabla}_x^n \hat{v} \; + \; \hat{v}(\boldsymbol{w} \circ \boldsymbol{\varphi}_h^n \cdot \boldsymbol{\nabla}_x^n) \hat{u}) \, |J_{\tilde{Q}^n}| d\hat{\boldsymbol{x}} dt \\ &\quad + \int_{\partial \hat{Q}_D^n} (\beta_h \hat{v} \hat{u} - \hat{v} \, (\mathbf{n}_x^n \cdot \mu \boldsymbol{\nabla}_x^n \hat{u}) - (\mathbf{n}_x^n \cdot \mu \boldsymbol{\nabla}_x^n \hat{v}) \, \hat{u}) \, |J_{\partial \tilde{Q}_D^n}| d\hat{\boldsymbol{x}} dt, \end{aligned} \tag{4.2}$$

and the linear form reads as,

$$\begin{aligned} L_h^n(\hat{v}) &= \int_{\Omega^n} \hat{v}(t^n) \hat{u}^{n-1}(t^n) |J_{\Omega^n}| d\hat{\boldsymbol{x}} + l_h(\hat{v}), \\ l_h(\hat{v}) &= \int_{\hat{Q}^n} \hat{v} \, (f \circ \boldsymbol{\varphi}_h^n) \, |J_{\tilde{Q}^n}| d\hat{\boldsymbol{x}} dt + \int_{\partial \hat{Q}_N^n} \hat{v} \, (g_N \circ \boldsymbol{\varphi}_h^n) \, |J_{\partial \tilde{Q}_N^n}| d\hat{\boldsymbol{x}} dt \\ &\quad + \int_{\partial \hat{Q}_D^n} (\beta_h \hat{v} \, (u_D \circ \boldsymbol{\varphi}_h^n) - (\mathbf{n}_x^n \cdot \mu \boldsymbol{\nabla}_x^n \hat{v}) \, (u_D \circ \boldsymbol{\varphi}_h^n)) \, |J_{\partial \tilde{Q}_D^n}| d\hat{\boldsymbol{x}} dt, \end{aligned} \tag{4.3}$$

Let us define the norms used in [7] to prove stability and convergence results, which we will computed in the numerical experiments. In [7] the space-time accumulated DG

norm of $\mathcal{V}_{h,\text{ag}}$ is defined as follows,

$$|||v|||^2_{n,*} \doteq \|v^n(t^{n+1})\|^2_{L^2(\Omega^n)} + \sum_{i=1}^{n} \|v^i(t^i) - v^{i-1}(t^i)\|^2_{L^2(\Omega^i)} + c_\mu \int_0^{t^{n+1}} \|v\|^2_{\bar{\mathcal{V}}^n(h)} dt,$$

where c_μ is the coercivity constant and $\bar{\mathcal{V}}^n(h) \doteq \bar{\mathcal{V}}^n_{h,\text{ag}} + H^2(\Omega(t))$ is the norm of the FE space at a time step t given by

$$|||v|||^2_{\bar{\mathcal{V}}^n(h)} \doteq \mu \|\boldsymbol{\nabla} v\|^2_{L^2(\Omega(t))} + \sum_{\bar{T} \in \bar{\mathcal{T}}_{h,\text{act}}} \beta_{\bar{T}} \|v\|^2_{L^2(\bar{T} \cap \partial\Omega_D(t))} + \sum_{\bar{T} \in \bar{\mathcal{T}}^n_{h,\text{act}}} \mu h^2_{\bar{T}} \|v\|^2_{H^2(\bar{T} \cap \Omega(t))}.$$

The proof of this stability result follows the ideas in [7] in the case in which the deformation map $\boldsymbol{\varphi}^n_h(t)$ is equal to $\boldsymbol{D}(t)$ on $\Gamma(t)$. Otherwise, the analysis is more technical and would require to use ideas similar to the ones in [59, 70]. The analysis therein assumes a constant deformation map and a level-set description of the domain. In our case, the deformation map can be of higher-order (time-dependent) and the domain is represented by its boundary. We will leave this analysis for future work.

In the variational form, the Dirichlet boundary conditions are imposed weakly with the Nistche method, with a penalty term β_h that depends on the spatial cell size h. The initial value is also imposed weakly with DG in time, where the jump is given by $[\hat{u}(t^n) - \hat{u}^{n-1}(t^n)]$. The evaluation of the solution of the previous time slab $\hat{u}^{n-1}(t^n)$ in Ω^n requires special attention since it is computed using a different discretization. Further discussion is provided in Section 4.3.2 and Section 4.4.

The derivatives are projections of the space-time gradient into space and time, $\partial^n_t = (\boldsymbol{\nabla}^n)_t$, and $\boldsymbol{\nabla}^n_x = (\boldsymbol{\nabla}^n)_x$, resp. They are obtained by transporting the space-time gradient to the deformed domain $\boldsymbol{\nabla}^n = \mathbf{F}^{-T}\hat{\boldsymbol{\nabla}}$, where $\mathbf{F} = \boldsymbol{\nabla}\boldsymbol{\varphi}^n_h$. The operators $(\,)_x$ and $(\,)_t$ represent the projection to the space and time directions, resp. The boundary normal is also transported to the deformed domain. It is computed as follows,

$$\mathbf{n}^n_x = \left(\frac{\mathbf{F}^{-T}\mathbf{n}_{\partial\hat{Q}^n}}{\|\mathbf{F}^{-T}\mathbf{n}_{\partial\hat{Q}^n}\|} \right)_x,$$

where $\mathbf{n}_{\partial\hat{Q}^n}$ is the normal vector in the undeformed space-time domain.

The integration measures to define the domain change are defined by the jacobians of the deformed domain. The integration measure change of the space-time domain $\tilde{Q}^n$ is given by $J_{Q^n} = J$, where $J = det(\mathbf{F})$. The initial boundary Jacobian is defined as $J_{\Omega^n} = det(\mathbf{F}_x(t^n))$. In the given case $J_{\Omega^n} = 1$. Note that for small deformations, where $\min_\Omega(J) \approx 1$, a simpler approach can be considered by assuming a deformed initial state, e.g., $\Omega^n = \boldsymbol{\varphi}^{n-1}_h(t^n)(\Omega^{n-1})$. In this situation, the evaluation $\hat{u}^{n-1}_h(t^n)$ does not require the change of reference space described in Section 4.3.2. This approach is equivalent to a time slab with more than one cell in time and continuous FE spaces in time could be considered within this macro-cell.

The pullback of the area differential form to the reference domain is expressed as

$$da_{\partial\tilde{Q}^n} = J\sqrt{\mathbf{n}_{\partial\hat{Q}^n}^T \cdot \mathbf{C}^{-1} \cdot \mathbf{n}_{\partial\hat{Q}^n}}\, da_{\partial\hat{Q}^n}$$

where $\mathbf{C} = \mathbf{F}^T\mathbf{F}$ (see, e.g., [28]).

The space-time gradients are transported to the deformed domain, $\boldsymbol{\nabla}^n = \mathbf{F}^{-T}\hat{\boldsymbol{\nabla}}$. The map gradient, $\mathbf{F} = \boldsymbol{\nabla}\boldsymbol{\varphi}_h^n$, contains the terms as follows:

$$\mathbf{F} = \begin{bmatrix} \mathbf{F}_x & 0 \\ \partial_t \boldsymbol{\varphi}_{h,x}^n & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{F}_x & 0 \\ \mathbf{w}^T & 1 \end{bmatrix},$$

where $\mathbf{F}_x$ is the space gradient and $\mathbf{w}^T = \partial_t \boldsymbol{\varphi}_{h,x}^n$ the deformation velocity. Then, the inverse gradient is given by

$$\mathbf{F}^{-T} = \begin{bmatrix} \mathbf{F}_x^{-1} & 0 \\ -\mathbf{w}^T\mathbf{F}_x^{-1} & 1 \end{bmatrix}.$$

Then, by decomposing the space-time gradient into space and time derivatives, we obtain

$$\begin{bmatrix} \boldsymbol{\nabla}_x^n \\ \partial_t^n \end{bmatrix} = \mathbf{F}^{-T} \begin{bmatrix} \hat{\boldsymbol{\nabla}}_x \\ \hat{\partial}_t \end{bmatrix} = \begin{bmatrix} \mathbf{F}_x^{-1}\hat{\boldsymbol{\nabla}}_x \\ \hat{\partial}_t - \mathbf{w}^T\mathbf{F}_x^{-1}\hat{\boldsymbol{\nabla}}_x \end{bmatrix},$$

which already recovers the derivative terms used in ALE formulations.

4.3.2 Inter-slab integration

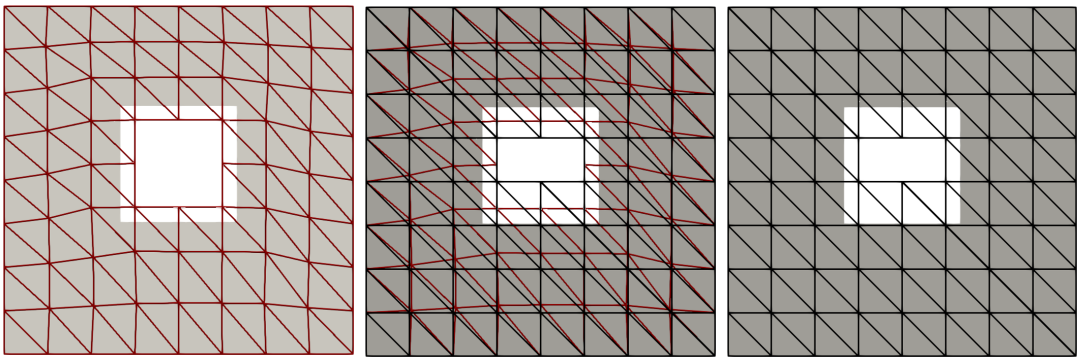
In the formulation (4.2)-(4.3), a DG method is used in time, where the initial value at the time slab is imposed weakly through an inter-time slab jump $[\hat{u}^n(t^n) - \hat{u}^{n-1}(t^n)]$. However, integrating this jump is not straightforward, since $\hat{u}^n(t^n)$ and $\hat{u}^{n-1}(t^n)$ in (4.3) are expressed in different discrete spaces (and meshes).

To address this evaluation, an intermediate unfitted discretization $\bar{\mathcal{T}}_{\text{int}}^n$ is introduced. $\bar{\mathcal{T}}_{\text{int}}^n$ is a partition of Ω^n that results of intersecting the embedded discretization $\mathcal{T}_{\text{cut}}^n$ of Ω^n and $\bar{\mathcal{T}}_-^n = \boldsymbol{\varphi}_h^{n-1}(t^n)(\bar{\mathcal{T}}^{n-1})$, i.e., $\bar{\mathcal{T}}_{\text{int}}^n$ results from intersecting $\bar{\mathcal{T}}^n$, $\bar{\mathcal{T}}_-^n$ and Ω^{n-1}. Each cell $K_{\text{int}} \in \bar{\mathcal{T}}_{\text{int}}^n$ has an injective map to $K^{n-1} \in \bar{\mathcal{T}}^{n-1}$ and $K^n \in \bar{\mathcal{T}}^n$. A representation of $\bar{\mathcal{T}}_{\text{int}}^n$ is depicted in Figure 4.4, and its construction is detailed in Section 4.4.

The integration of the jump in the time slab interface is performed in $\bar{\mathcal{T}}_{\text{int}}^n$. Thus, we need to define the cell maps from $\bar{\mathcal{T}}_{\text{int}}^n$ to $\bar{\mathcal{T}}^n$ and $\bar{\mathcal{T}}^{n+1}$. For $\hat{u}^{n-1}(t^n)$ we need a cell map from $\hat{K}_{\text{int}} \in \bar{\mathcal{T}}_{\text{int}}^n$ to $\hat{K}_1 \in \bar{\mathcal{T}}^n$ such that

$$\psi^- = (\phi_{K_1})^{-1} \circ \left(\boldsymbol{\varphi}^{n-1}\right)^{-1} \circ (\phi_{K_{\text{int}}}),$$

where ϕ_{K_1} maps from the reference space to the undeformed physical space of $K_1 \in \bar{\mathcal{T}}^n$ and $\phi_{K_{\text{int}}}$ maps from the reference space to the physical space of K_{int}, see Figure 4.5. In



(a) $\boldsymbol{\varphi}_h^{n-1}(t^n)(\bar{\mathcal{T}}^{n-1})$ (b) $\bar{\mathcal{T}}_{\text{int}}^n$ (c) $\bar{\mathcal{T}}^n$

Figure 4.4: Mesh sequence for solution transfer between time slabs. The solution obtained in $\bar{\mathcal{T}}^{n-1}$ in (a) is then evaluated at $\bar{\mathcal{T}}^n$ (c). Acting as a bridge, the intersected mesh $\bar{\mathcal{T}}_{\text{int}}^n = \boldsymbol{\varphi}_h^{n-1}(t^n)(\bar{\mathcal{T}}^{n-1}) \cap \bar{\mathcal{T}}^n \cap \Omega^n$ in (b) facilitates the evaluation by providing injective cell maps to both active meshes.

the case of $\hat{u}^n(t^n)$ we need a cell map from $\hat{K}_{\text{int}} \in \bar{\mathcal{T}}_{\text{int}}^n$ to $\hat{K}^n \in \bar{\mathcal{T}}^n$ such that

$$\psi^+ = (\phi_{K_2})^{-1} \circ (\phi_{K_{\text{int}}}) .$$

where ϕ_{K_2} maps from the reference space to the physical space of $K_2 \in \bar{\mathcal{T}}^n$. Now, we can numerically integrate the jump by evaluating the following integral,

$$\sum_{K_{\text{int}} \in \bar{\mathcal{T}}_{\text{int}}} \sum_{\hat{q}} \left(\hat{u}^n(t^n) \circ \psi^+ - \hat{u}^{n-1}(t^n) \circ \psi^- \right) (v^n(t^n) \circ \psi^+ (\hat{q})) |J_{K_{\text{int}}}| w_{K_{\text{int}}},$$

where $J_{K_{\text{int}}}$, $\hat{q}$ and $w_{K_{\text{int}}}$ are the Jacobian, quadrature points and quadrature weights of K_{int}, resp.

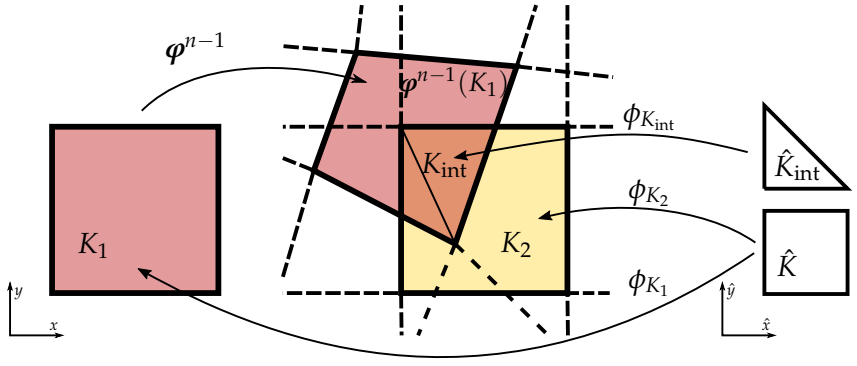


Figure 4.5: Representation of the cell maps used in the time slab interface integration. The maps ϕ_{K_1} and ϕ_{K_2} send the reference element $\hat{K}$ to the physical space of $K_1 \in \bar{\mathcal{T}}^{n-1}$ and $K_2 \in \bar{\mathcal{T}}^n$, resp. The map $\phi_{K_{\text{int}}}$ sends the reference element $\hat{K}_{\text{int}}$ to the physical space of $K_{\text{int}} \in \bar{\mathcal{T}}_{\text{int}}^{n-1}$.

4.4 Intersection algorithm for time slab transfer

In this section, we will present one of the key novelties of this work. We will develop the geometrical algorithms required in Section 4.3.2 to evaluate the inter-time-slab solution. These algorithms are based on the methods presented in [17]. Specifically, we compute a linear intersection $\bar{\mathcal{T}}^n_{\text{int}}$ of $\mathcal{T}^n$, $\bar{\mathcal{T}}^n_- = \boldsymbol{\varphi}^{n-1}_h(t^n)(\bar{\mathcal{T}}^{n-1})$ and $\Omega^n = \text{int}(\mathcal{B}_h(t^n))$. It is important to note that, to ensure the linearity of the intersections, we need to avoid bilinear terms in $\boldsymbol{\varphi}^{n-1}_h$ using a P_k space, e.g., decomposing $K \in \bar{\mathcal{T}}^n$ into simplices.

To simplify the exposition of the intersection algorithm Algorithm 18, we redefine $\mathcal{T} = \bar{\mathcal{T}}^n$, $\mathcal{T}_- = \bar{\mathcal{T}}^n_-$ and $\mathcal{B} = \mathcal{B}_h(t^n)$. Within the cell loop of Algorithm 18, we first consider the cells close to $K \in \mathcal{T}$ by restricting $\mathcal{B}$ and $\mathcal{T}_-$ accordingly (line 3 and line 4). These restriction queries are computed during a preprocessing stage before entering the loop. Next, in line 5, we compute the intersection of K with the interior of $\mathcal{B}$ using the algorithms in the loop-body of Algorithm 10 in [17].

These algorithms assume that $\mathcal{B}$ is a linear polytope, which is non-convex in general. Thus, its domain interior $\text{int}(\mathcal{B})$ is bounded by the set of planar faces of $\mathcal{B}$. The intersection $K \cap \text{int}(\mathcal{B})$ requires a convex decomposition of $\mathcal{B}$ and K before intersecting the half-spaces defined by the planar faces (see [17]).

Finally, we intersect each polytope in $\mathcal{T}^K_{\text{cut}}$ by the subset of $\mathcal{T}_-$ around K (line 8). These intersections are performed employing convex linear clipping algorithms [108] described in Algorithm 2 in [17]. Alternatively, in line 10, we intersect the cells $K \in \mathcal{T}$ within the domain $\text{int}(\mathcal{B})$ bounded by $\mathcal{B}$ that are not intersected by the domain boundary $\mathcal{B}$. The information about the cells inside $\text{int}(\mathcal{B})$ is obtained from the propagation through the untouched cells (see [17]). The returned triangulation $\mathcal{T}_{\text{int}}$ not only covers the cells cut cells intersected by the boundary $\mathcal{B}$ but the entire domain enclosed by $\mathcal{B}$. This process guarantees that each cell $K_{\text{int}} \in \mathcal{T}_{\text{int}}$ has an injective map to $K \in \mathcal{T}$ and $K_- \in \mathcal{T}_-$. The cells in $\mathcal{T}_{\text{int}}$ are general polytopes that cannot use standard quadrature rules. In order to numerically integrate in these cells, one can perform a decomposition of the cells into simplices. Alternatively, one can reduce the dimension of the integrals with Stokes theorem and moment-fitting methods; see more details in [75] and references therein.

We emphasize that the intersection algorithm is as robust as the core cut algorithm in line 5. Furthermore, it is important to note that, for evaluation purposes, accurate tolerance management is not required in the intersection process of line 8. While the algorithm is described for the core cut algorithm with an exact embedded discretization of explicit domain representation [7], Algorithm 18 is general enough to be used with other unfitted discretizations.

Algorithm 18 $\mathcal{T} \cap \mathcal{T}_- \cap \text{int}(\mathcal{B})$

$\mathcal{T}_{\text{cut}} \leftarrow \emptyset, \quad \mathcal{T}_{\text{in}} \leftarrow \emptyset$
for $K \in \mathcal{T}$ **do**
  $\mathcal{B}^K \leftarrow \texttt{restrict}(\mathcal{B}, K)$
  $\mathcal{T}_-^K \leftarrow \texttt{restrict}(\mathcal{T}_-, K)$
  $\mathcal{T}_{\text{cut}}^K \leftarrow K \cap \text{int}(\mathcal{B}^K)$
  for $K_- \in \mathcal{T}_-^K$ **do**
    if $K \cap \mathcal{B} \neq \emptyset$ **then**
      $\mathcal{T}_{\text{cut}}^{K,K_-} \leftarrow \{K_{\text{cut}} \cap K_- : K_{\text{cut}} \in \mathcal{T}_{\text{cut}}^K\}; \quad \mathcal{T}_{\text{cut}} \leftarrow \mathcal{T}_{\text{cut}} \cup \mathcal{T}_{\text{cut}}^{K,K_-}$
    else if $K \subset \text{int}(\mathcal{B})$ **then**
      $\mathcal{T}_{\text{in}} \leftarrow \mathcal{T}_{\text{in}} \cup (K \cap K_-)$
    end if
  end for
end for
return $\mathcal{T}_{\text{int}} \leftarrow \mathcal{T}_{\text{cut}} \cap \mathcal{T}_{\text{in}}$

4.5 Numerical experiments

4.5.1 Objectives

In the experiments of this section, we aim to demonstrate the effectiveness of the presented methods. In particular, we examine the following aspects within our formulation:

- We evaluate the *hp*-convergence on both 2D and 3D spatial domains, compared to the results and analysis in [7].
- We explore numerical stability concerning the cut location and the approximation degrees.
- We assess the applicability of our formulation to complex 2D and 3D moving domains derived from STL models.

It is important to note that these experiments focus on comparing our formulation with the one presented in [7]. Recall that [7] uses space-time embedded discretizations on implicit geometries determined by level sets, and the results are presented exclusively in 2D+1D domains (as the geometrical algorithms for 3D+1D domains are significantly more complex). In contrast, we design a space-time formulation that works on explicit boundary representations and only require geometrical intersection algorithms in space only, e.g., 2D and 3D vs. 3D and 4D in [7]. This is possible by pulling back the problem into an extruded space-time domain using the formulation in Sect. 4.3.1.

4.5.2 Environment setup

The numerical experiments have been performed on Gadi, a high-end supercomputer at the NCI (Australia) with 4962 nodes, 3074 of them powered by a 2 x 24 core Intel Xeon

Platinum 8274 (Cascade Lake) at 3.2 GHz and 192 GB RAM. The algorithms presented in this work have been implemented in the Julia programming language [27]. The unfitted FE computations have been performed using the Julia FE library `Gridap.jl` [116] version 0.17.17 and the extension package for unfitted methods `GridapEmbedded.jl` version 0.8.1 [118]. `STLCutters.jl` version 0.1.6 [76] has been used to compute intersection computations on STL geometries. To mitigate excessive computational costs, the condition numbers are computed in the 1-norm using `cond()` Julia function.

4.5.3 Space-time convergence tests

To demonstrate optimal convergence rates we solve a simple PDE with a manufactured solution out of the FE space. Inspired by [7], we solve the Poisson equation (4.1) with the following manufactured solution

$$u(x,t) = \sin\left(\frac{\pi\alpha t}{T}\right)\prod_{i=1}^{D}\sin\left(\frac{\pi x_i}{L_i}\right),$$

where $\{L_1,...,L_D,T\}$ are the cartesian dimensions of the space-time domain and the time parameter is set to $\alpha = 0.5$ for the experiments shown in Figure 4.6 and Figure 4.7. Furthermore, in the equation (4.1) we set the diffusion term as $\mu = 1$. The space domain is a n-cube with a n-cubic hole in the center. The hole is described by the STL of a cube. The lengths at the domain sides are $L = 3$ while the lengths of the hole sides are $l = 1$. The time domain has size $T = 1$. The hole is linearly translated in the x direction. The displacement map is described as $\boldsymbol{D}(x,t) = (0.2t, 0, 0)$. In all the cases, the domain discretization is a regular Cartesian grid, with the same number of elements n in each direction, both space and time.

For implementation reasons, the two-dimensional examples are computed with a three-dimensional STL. Thus, we build a pseudo-two-dimensional domain that has only one cell in the z direction. The space-time domain dimensions are $\{L, L, \frac{L}{n}, T\}$ while the number of elements in each direction is $\{n, n, 1, n\}$. The number of elements per direction in convergence tests is $n = 2^i, i = 3,...,6$ in two-dimensional runs, while $n = 2^i, i = 3,...,5$ for three-dimensional runs.

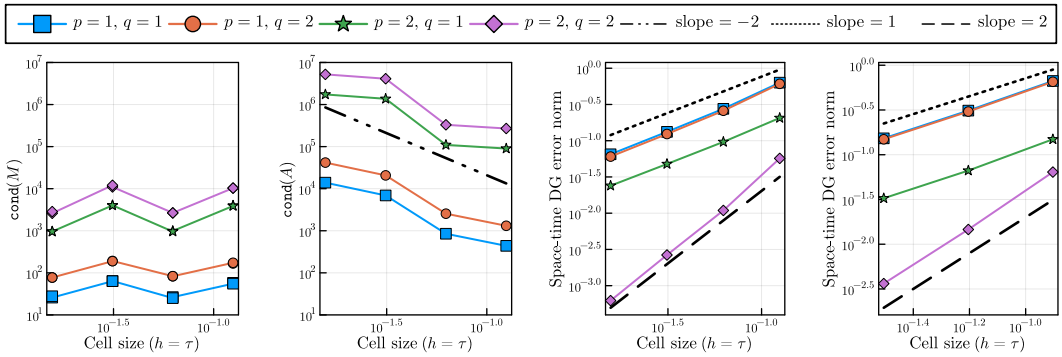
We analyze the condition number of the system to be inverted in each time-slab. Since the system matrix is nonsymmetric, as suggested in [7], a preconditioner for DG in time [102] can be considered. The effectiveness of this preconditioner depends on the condition numbers the mass and stiffness matrices, which are defined as follows,

$$\mathbf{M}_{ab} = \int_{\hat{Q}^n} \mathcal{E}^n(\mathbf{\Phi}^a)\mathcal{E}^n(\mathbf{\Phi}^b)|J_{\Omega^n}|d\hat{\mathbf{x}}dt, \quad \mathbf{A}_{ab} = a_h\left(\mathcal{E}^n(\mathbf{\Phi}^a), \mathcal{E}^n(\mathbf{\Phi}^b)\right).$$

The condition numbers presented in Figure 4.6(a) and Figure 4.6(b) are computed in the initial time slab of the two-dimensional convergence experiments. We observe that the condition number of the mass matrix remains nearly constant, while the condition

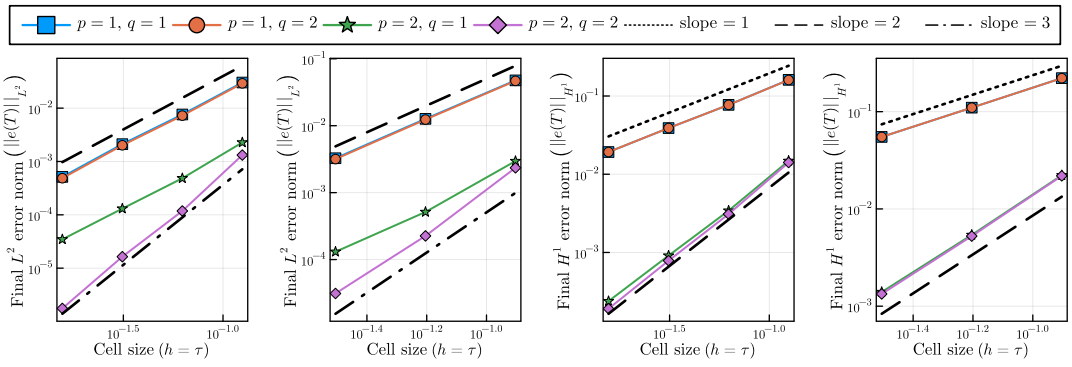
number of the stiffness matrix scales with $\mathcal{O}(h^{-2})$. These observations align with the behavior expected for AgFEM in space-time domains analyzed in [7].

Now, let us analyze the convergence of the error norms. In Figure 4.6, we can observe the accumulated DG error norm convergence with coercivity constant $c_\mu = 1$. We observe that using the AgFEM and constant h/τ the error converges with $\mathcal{O}(h^r)$, where the convergence rate $r = \min(p, q)$. Figure 4.7 shows the $L^2(\Omega^n)$ and $H^1(\Omega^n)$ norms at the final time $t = T$. The convergence rate of $L^2(\Omega^n)$ norm is $r = \min(p, q) + 1$ while the convergence rate of $H^1(\Omega^n)$ norm is $r = p$. These results are in agreement with the theoretical and analytical space-time AgFEM results in [19].



(a) Mass matrix (b) Stiffness matrix (c) 2D (d) 3D

Figure 4.6: Scaling of the condition numbers of the mass and stiffness matrices in the initial time slab, (a) and (b). Convergence of the space-time DG norm error ($e = u - u_h$) in two and three-dimensional space domains (c) and (d). Here, p and q represent the space and time approximation order, resp.



(a) 2D (b) 3D (c) 2D (d) 3D

Figure 4.7: Convergence of the $L^2(\Omega^n)$ and $H^1(\Omega^n)$ norms of the error ($e = u - u_h$) at the final time $t = T$ in two and three-dimensional space domains. Here, p and q represent the space and time approximation order, resp.

4.5.4 Moving domains examples

To demonstrate the applicability of the presented algorithms for the simulation of fluids around moving boundaries, we solve the flow around two rotating geometries in 2D and 3D. We solve the Stokes equations in both examples with a kinematic viscosity $\nu = 10^{-2}$. Regardless of the dimension, we run similar setups. Both share the same discretization order for pressure $p_p = 1$, velocity $p_u = 2$ and time q. We utilize the AgFEM aggregating all cut cells.

We improve the accuracy of both meshes by clustering the cells around the geometry. See Figure 4.8 and Figure 4.9 for 2D and 3D, resp. We map each direction $i \in 1, ..., d$ of the Cartesian meshes as follows,

$$\phi_M^i(\hat{x}) = \begin{cases} \hat{\boldsymbol{x}}_0^i \left(\frac{\hat{x}^i}{\hat{x}_0^i}\right)^\alpha & \text{if } \hat{x}^i < \hat{x}_0^i, \\ 1 - (1 - \hat{x}_0^i)\left(\frac{1-\hat{x}^i}{1-\hat{x}_0^i}\right)^\alpha & \text{otherwise.} \end{cases} \tag{4.4}$$

Here, $\hat{x}^i = x^i - x_0^i / L^i$ is the reference axis of the direction x^i where x_o^i is the lower value in the i direction, and L^i is the i-length. In (4.4), we use $\hat{x}_0^i = 0.5$ to determine the region of element condensation and the exponential factor $\alpha < 1$ for the smoothness of the map.

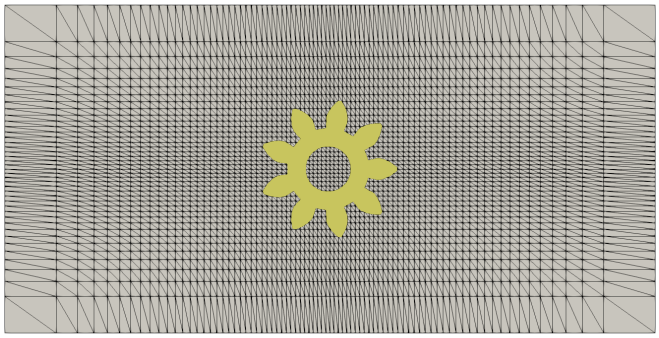


Figure 4.8: Background spatial mesh $\bar{\mathcal{T}}^{\text{bg}}$ around a prismatic gear $\mathcal{B}_0$ (geometry id 71711 from *Thingi10k* [123]). This mesh is the simplex partition of a mapped Cartesian mesh. The uniform elements are mapped in each direction with ϕ_M (4.4) and an exponential factor $\alpha = 0.5$. The Cartesian mesh has 80×40 elements before the simplex decomposition. The artificial domain Ω^{art} is a box of size $4.8L_x \times 2.4L_y$. Here, L_x and L_y represent the bounding box size of $\mathcal{B}_0$.

In the 2D example of Figure 4.8 and Figure 4.10, we set up the boundary conditions of a viscous flow benchmark. We define a parabolic inlet flow in the x-direction,

$$u(\boldsymbol{x}) = U_{\max}(4x_2 - 4x_2^2),$$

where $U_{\max} = (1, 0, 0)$. We set zero velocity on the y-faces and zero z-velocity in the z-faces, i.e., slipping conditions. We weakly impose the displacement velocity $\boldsymbol{\nabla\varphi}^n$ with Nistche's method on the geometry $\mathcal{B}(t)$. See the description of the displacement $\boldsymbol{D}$ in Figure 4.10.

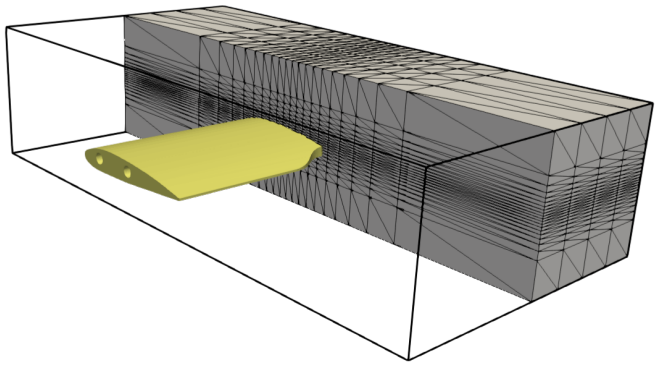
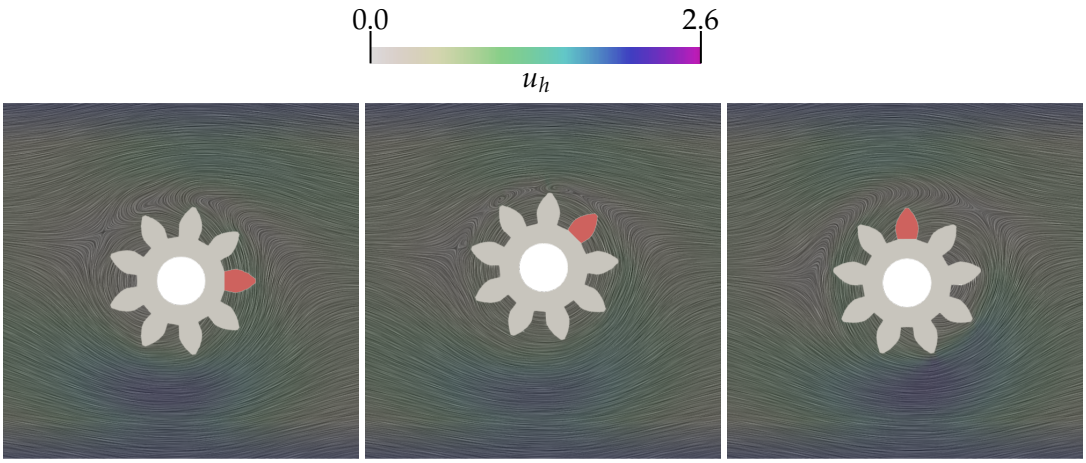


Figure 4.9: Background spatial mesh $\bar{\mathcal{T}}^{\text{bg}}$ around a wing $\mathcal{B}_0$ (id 65604 in *Thingi10k*). This simplex mesh comes from a mapped Cartesian mesh. The coordinates of the mesh are mapped in the x and y directions with ϕ_M (4.4) and an exponential factor $\alpha = 0.3$. The Cartesian mesh has $20 \times 20 \times 8$ elements. The artificial domain Ω^{art} is a box of size $4L_x \times 4L_y \times 0.8L_z$. Here, L_x, L_z and L_y represent the bounding box size of $\mathcal{B}_0$.



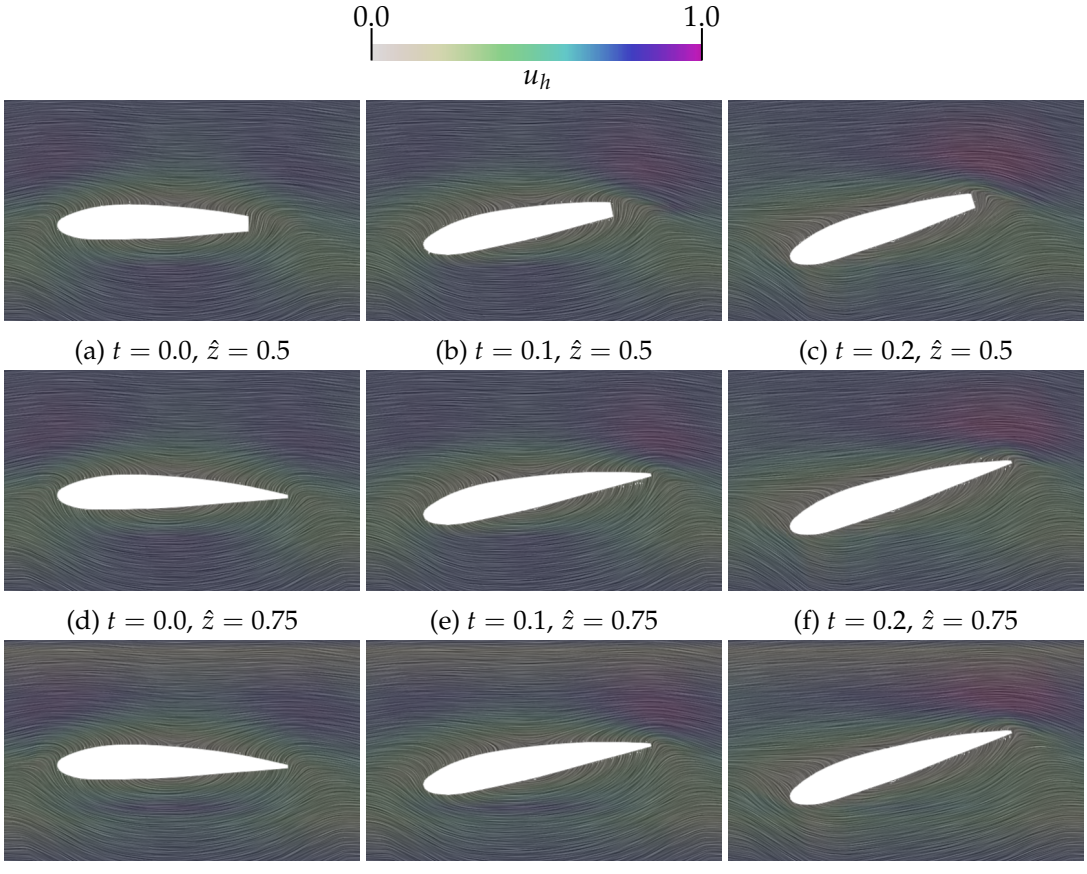
(a) $t = 0.0$ (b) $t = 0.5$ (c) $t = 1.0$

Figure 4.10: Representation of the line integral convolution (LIC) filter of the viscous flow simulation a evolving geometry in Figure 4.8. The time step size is $\tau = 1/60$. The geometry combines rotation and displacements. The initial geometry $\mathcal{B}_0$ is mapped by $\boldsymbol{D}(\boldsymbol{x}, t) = \boldsymbol{x}_0 + \mathbf{R}(t) \cdot (\boldsymbol{x} - \boldsymbol{x}_0) + \mathbf{A}_x \sin(\omega_x t)$, where $\mathbf{R}(t)$ is the rotation matrix with angular velocity $\omega = \pi$ rad/s, $\boldsymbol{x}_0$ is the center of the mesh, $\mathbf{A}_x = (0, 0.1)$ and $\omega_x = \pi$. The red tooth serves as a reference for visualizing the rotation.

In the 3D experiment of Figure 4.9 and Figure 4.11, we utilize the same boundary conditions as in the 2D experiment. However, we define a different inlet profile,

$$u(\boldsymbol{x}) = U_{\max}(4x_2 - 4x_2^2)(4x_3 - 4x_3^2),$$

that utilizes the 3D domain. The differences in the z-direction are clearly shown in Figure 4.11. Equally, the z-faces have a slipping condition to hold the rotation of the wing in the z-direction. The rotation is defined by the displacement $\boldsymbol{D}$ described in Figure 4.11. We slightly optimize the number of time slab transfer operations in these experiments. Above a deformation threshold, e.g., $\max(\nabla \boldsymbol{\varphi}^n) < 0.8$, we maintain the spatial active mesh $\bar{\mathcal{T}}^n$ between time slabs. In these cases, the initial value evaluation does not require mesh intersections, reducing computational cost.



(a) $t = 0.0$, $\hat{z} = 0.5$ (b) $t = 0.1$, $\hat{z} = 0.5$ (c) $t = 0.2$, $\hat{z} = 0.5$

(d) $t = 0.0$, $\hat{z} = 0.75$ (e) $t = 0.1$, $\hat{z} = 0.75$ (f) $t = 0.2$, $\hat{z} = 0.75$

(g) $t = 0.0$, $\hat{z} = 1.0$ (h) $t = 0.1$, $\hat{z} = 1.0$ (i) $t = 0.2$, $\hat{z} = 1.0$

Figure 4.11: LIC representation of the viscous flow around Figure 4.9. The matrix representation shows several time slabs on different slices. Here, $\hat{z} = (z - z_0)/L_z$ represents the relative position in z where z_0 is the lower the z-coordinate and L_z the z-length. The step size of the simulation is $\tau = 1/120$. The wing geometry is rotating with the map $\boldsymbol{D}(\boldsymbol{x}, t) = \boldsymbol{x}_0 + \mathbf{R}_\theta(t)(\boldsymbol{x} - \boldsymbol{x}_0)$, where $\boldsymbol{x}_0$ is the center of the mesh and $\mathbf{R}_\theta$ is the rotation matrix of the angle $\theta(t) = \theta_{\max} \sin(\omega_\theta t)$ over the z-axis. Here, $\theta_{\max} = \pi/10$ and $\omega_\theta = \pi$. Note that the wing is not an extrusion of an airfoil. The central section ($\hat{z} = 0.5$) has a flat trailing edge (see Figure 4.9).

4.6 Conclusions and future work

In this work, we have introduced a novel space-time formulation for unfitted FEM that can handle large displacements of moving domains. This formulation relies on a space-only embedded discretization. Therefore, we eliminate the requirement for four-dimensional geometric algorithms. We have achieved this through an intersection mesh that allows us to integrate the time slab interface exactly.

We have validated this method through an hp-convergence analysis of a manufactured solution using AgFEM. We have observed optimal convergence rates for two-dimensional and three-dimensional space domains in these experiments. The convergence has been validated by comparing it with another space-time analysis for unfitted FEM [7]. Additionally, we have observed the expected scaling of the condition number of the mass and stiffness matrices. Furthermore, we have demonstrated the practical applications of this methodology to the simulation of incompressible flows around rotating geometries in two examples, one with a two-dimensional geometry and another with a three-dimensional geometry.

Future work involves the numerical analysis of the method in the case in which the deformation maps do not much the boundary displacement by designing a high-order extension of the work in [70] and the extension of this method to distributed memory machines [16]. The geometrical component of this extension will be highly scalable, given that the presented algorithms are defined cell-wise and thus embarrasingly parallel, as they are in [17]. Distributed computations combined with background mesh refinement, e.g., using octree meshes, will allow us to solve larger real-world simulations. Additional developments include applying this method to FSI simulations. This extension will develop the full potential of the method in dynamic interface-coupling multiphysics simulations.

Chapter 5

Distributed unfitted finite element discretizations for explicit boundary representations

Approximating partial differential equations for extensive industrial and scientific applications requires leveraging the power of modern high-performance computing. In large-scale parallel computations, unfitted finite element methods offer an advantageous solution for mesh partitioning compared to standard body-fitted formulations. These methods do not necessitate unstructured body-fitted meshes. Nevertheless, their application is constrained to implicit (level-set) geometrical representations. This chapter presents an efficient parallel implementation for unfitted finite element methods for explicit boundary representations. Such geometries can be generated using standard computer-aided design tools. The proposed algorithms utilize a multilevel approach to overlapping computations, effectively eliminating bottlenecks in large-scale computations. The numerical results demonstrate perfect weak scalability over 12,000 processors and one billion cells.

5.1 Introduction

Nowadays, computing power does not follow Moore's Law, and a performance increment of scientific simulations is only achievable through parallel algorithms on distributed-memory machines. Large-scale parallel computations involve efficient communications to exploit the power of current HPC resources. Many FE methods efficiently approximate the solution of PDEs in parallel. The balancing domain decomposition by constraints (BDDC) [12] and AMG [32] methods are popular solvers that have demonstrated high parallel scalability.

Distributed-memory FE computations require body-fitted meshes and partitioning

them. In practical applications, generating unstructured body-fitted meshes over complex geometries is challenging, especially in parallel. Additionally, the mesh partitioning algorithms rely on graph partitioning techniques [65] that are inherently serial. Thus, the mesh partitioning step can become the bottleneck, even a showstopper, of the simulation pipeline for parallel computations on distributed-memory machines. Furthermore, this framework is unsuitable for practical large-scale applications on non-trivial domains in AMR since dynamic load-balancing has an unacceptable performance overhead.

Unfitted (also known as embedded or immersed) FE methods [81] can overcome the current limitations of body-fitted meshes. Instead, unfitted methods use a background mesh for the functional discretization and a geometrical discretization for integrating the interior of the domain Ω. This approach drastically reduces the mesh constraints. An unfitted approach can use tree-based background meshes and exploit scalable and dimension-agnostic mesh generators and partitioners [10, 20]. The generation and load-balancing of octree-meshes is efficient thanks to space-filling curve techniques [6]; see, e.g., the highly scalable `p4est` framework [37] for handling forests of octrees on hundreds of thousands of processors.

Unfitted discretizations may lead to unstable and severe ill-conditioned discrete problems [43] unless a specific technique mitigates the problem. The size and aspect ratio of the intersection of a background cell and the physical domain are not bounded. Despite the vast literature on the topic, unfitted FE formulations that solve these issues are quite recent. Stabilized formulations based on the so-called *ghost penalty* were originally proposed in [33] for Lagrangian continuous FEs and has been widely used since [34]. The so-called *cell aggregation* or *cell agglomeration* techniques are an alternative way to ensure robustness concerning cut location. This approach is very natural in DG methods, as their formulation on agglomerated meshes is straighforward [84]. These techniques present extensions with $\mathcal{C}^0$ Lagrangian FE in [22] and mixed methods in [14]; the method was coined AgFEM. These unfitted formulations enjoy good numerical properties, such as condition number bounds, stability, optimal convergence, and continuity concerning data. Distributed-memory implementations for large-scale problems have been designed [117], and error-driven h-adaptivity and parallel tree-based meshes have also been exploited [10].

Most of the unfitted FE methods utilize implicit geometry representations, e.g., level sets, which dramatically reduces their applications. In Chapter 2, we presented an extension for explicit BREP. In Chapter 3, we extended this method to high order BREPs, e.g., CAD models. Unlike level set methods, explicit geometry representations require global operations to define the inside and outside domains, e.g., ray-tracing and propagation techniques. These operations are not trivial to parallelize; they can become a bottleneck in large-scale simulations.

This chapter presents a scalable algorithm to parallelize the generation of embedded discretizations for explicit geometry representations. Apart from the definition of

inside and outside, the intersection algorithms in Chapter 2 and Chapter 3 are embarrassingly parallel. We define a multilevel propagation algorithm that overlaps coarse and fine computations to achieve perfect scalability. The presented algorithm, inspired by the multilevel BDDC [32], has demonstrated weak scalability over 10k processors.

The outline of this chapter is as follows. In Section 5.2, we introduce the parallel unfitted FE methods used in this chapter. In Section 5.3, we present the distributed intersection algorithm. In Section 5.4, we expose the numerical results on stability. Finally, in Section 5.5, we will summarize the main conclusions of this chapter.

5.2 Distributed unfitted finite element method

Let us consider a Lipschitz domain $\Omega \subset \mathbb{R}^d$, with $d \in \{2,3\}$ the number of spatial dimensions. We describe the boundary of the domain $\partial\Omega$ with a parametric oriented surface mesh $\mathcal{B}$. We aim to solve a system of PDEs that can involve Dirichlet boundary conditions on Γ_D and Neumann boundary conditions on Γ_N. Γ_N and Γ_D are a partition of $\partial\Omega$. The geometrical representation, e.g., CAD model, must respect this partition. Therefore, we consider a partition of $\mathcal{B}_D$ and $\mathcal{B}_N$ a partition of $\mathcal{B}$, such that represent Γ_N and Γ_D repectively.

This work aims to define an efficient method for the parallel implementation of unfitted FE discretizations generated from $\mathcal{B}$. Unfitted discretizations utilize a background mesh $\mathcal{T}^{\text{bg}}$ mesh instead of body-fitted meshes. This background mesh is an arbitrary partition of an artificial domain Ω^{art} such that $\Omega^{\text{art}} \supset \Omega$ (see Figure 5.1). Ω^{art} can be as simple as a bounding box. The $\mathcal{T}^{\text{bg}}$ is a simpler partition than a body-fitted mesh, e.g., a Cartesian grid or a refinement of a hexahedral mesh.

In distributed-memory computation, we subdivide the domain Ω^{art} into S subdomains Ω_s^{art}, $s = 1, \dots, S$ (see Figure 5.1). Subdomain partition of uniform Cartesian meshes is straightforward. However, we can efficiently aggregate the subdomain cells in adaptive meshes like octrees using space-filling curves [6]. In any case, serial graph partition algorithms [65] are not necessary to partition background meshes.

The presented unfitted FE formulation accommodates different methods from the literature, e.g., the extended FE method (XFEM) [26], the cutFEM cite[34], the AgFEM [22], or the finite cell method [95]. Furthermore, the presented algorithm is agnostic to the solver utilized for parallel unfitted FE methods, e.g., BDDC [20] or AMG [32].

To solve PDEs on the unfitted discretization, we need to classify the background cells into interior, exterior, and cut, $\mathcal{T}^*$, $* \in \{\text{in}, \text{out}, \text{cut}\}$, respectively (see Figure 5.1). The functional discretization does not consider the exterior cells $\mathcal{T}^{\text{out}}$. Thus, we consider the active mesh $\mathcal{T} \doteq \mathcal{T}^{\text{bg}} \setminus \mathcal{T}^{\text{out}}$ for the FE discretization. The unfitted FE techniques utilize standard FE spaces on $\mathcal{T}$, V, to solve and test the weak form of the PDEs.

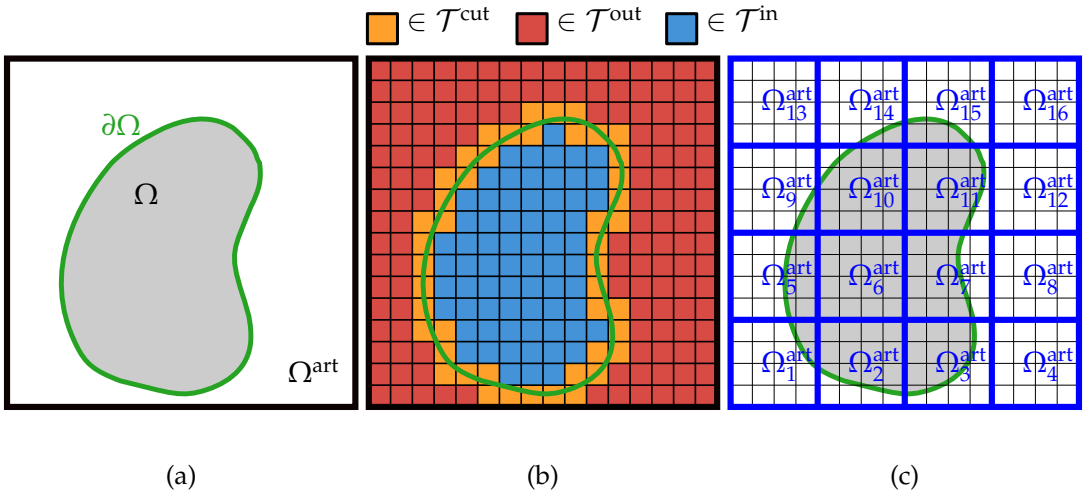


Figure 5.1: Unfitted FE representation in a distributed-memory computation. In (a), a surface mesh $\mathcal{B}$ represents the boundary of the domain $\partial\Omega$. The domain Ω is embedded in Ω^{art}. The artificial domain Ω^{art} is discretized in a Cartesian background mesh $\mathcal{T}^{\text{bg}}$. In (b), We classify the cells in $\mathcal{T}^{\text{bg}}$ interior, exterior, and cut cells. For parallel computations, in (c), we divide Ω^{art} in S subdomains Ω_s^{art}, $s = 1, \dots, S$. The background mesh is partitioned accordingly. The partition into subdomains coincides with the coarse mesh $\mathcal{T}^{\text{coarse}}$.

The unfitted problem reads as follows: find $u \in V$ such that

$$a(u, v) = (f, v)_{\Omega^{\text{art}}}, \quad \forall v \in V,$$

where

$$a(u, v) = \int_{\Omega} L_{\Omega}(u, v) d\Omega + \int_{\Gamma_D} L_D(u, v) d\Omega + \int_{\mathcal{F}} L_{sk}(u, v) d\Gamma,$$

and

$$l(v) = \int_{\Omega} F_{\Omega}(v) d\Omega + \int_{\Gamma_N} F_N(v) d\Gamma + \int_{\Gamma_D} F_D(v) d\Gamma.$$

Here, we include the differential operator, the source term, and additional stabilization terms within the bulk terms F_Ω and L_Ω. The integration of the operations F_D and L_D on Γ_D are related to the weak imposition of the Dirichlet boundary conditions, commonly utilizing Nische's method. The terms F_N and F_N integrated on Γ_N impose the Neumann boundary conditions. The skeleton $\mathcal{F}$ represents the interior faces of the active mesh $\mathcal{T}$. Integrating L_{sk} on $\mathcal{F}$, we introduce additional penalty terms, e.g., ghost penalty stabilization techniques or weak imposition of continuity in DG methods.

In FE methods, the integration of piecewise-wise polynomials relies on a cell-wise decomposition of bulk and surface terms. In unfitted FE methods, the integration is defined only in the domain's interior. Bulk integration meshes $\mathcal{T}^{\text{int}} \doteq \{K \cap \Omega : K \in \mathcal{T}\} = \mathcal{T}^{\text{in}} \cup \mathcal{T}^{\text{clip}}$, where the clipped mesh reads as $T^{\text{clip}} \doteq \{K \cap \Omega : K \in \mathcal{T}^{\text{cut}}\}$. A cell-wise geometrical algorithm, e.g., algorithms in Chapter 2 and Chapter 3, performs the intersection of the cut cells $K \in \mathcal{T}^{\text{cut}}$. However, the definition of the interior cells $K \in \mathcal{T}^{\text{in}}$ is not trivial in distributed memory computations. The propagation of inside cells

can become a bottleneck in large-scale computations. The integration of the surface terms requires the intersection of the boundary with the background mesh $\mathcal{B}^{\text{int}} \doteq \{\mathcal{B} \cap K : K \in \mathcal{T}^{\text{cut}}\}$. Again, the algorithms described in previous chapters provide the algorithms for these computations.

The parallel unfitted FE methods each background cell $K \in \mathcal{T}^{\text{bg}}$ owns to a single task $s \in 1, ..., S$ (see Figure 5.1(c)). Therefore, one can define the above meshes in each of the S tasks, e.g., $\mathcal{T}_s^*$, $* \in \{\text{in}, \text{out}, \text{cut}, \text{bg}, \text{int}\}$, $\mathcal{B}_s$ and $\mathcal{B}_s^{\text{int}}$. In the proposed algorithms, each coarse cell $K_s^{\text{coarse}} \in \mathcal{T}^{\text{coarse}}$ represents an artificial subdomain Ω_s^{art}. The distributed FE solvers also need to utilize *ghost* cells and DOFs. However, we do not deal with ghost elements in this work, we focus on the geometrical side.

5.3 Distributed intersection algorithm

In this section, we define an algorithm that intersects the background distributed cells $K \in \mathcal{T}^{\text{bg}}$ and a physical domain Ω described by an oriented surface mesh $\mathcal{B}$. First, in Section 5.3.1, we expose the algorithms for the bulk intersections $K \cap \Omega$ and the boundary intersections $K \cap \mathcal{B}$. Then, in Section 5.3.2, we state a propagation method for classifying non-intersected cells $K \cap \partial\Omega = \emptyset$. Finally, in Section 5.3.3, we describe a global algorithm that overlaps classifications and intersections in a two-level distributed mesh.

5.3.1 Local intersection

The main complication of the unfitted FE methods resides in the integration of the cut cells. For this purpose, we need to compute the intersections of the background cells $K \in \mathcal{T}^{\text{bg}}$ with the domain Ω. Each integration method relies on a different underlying data structure to store the resulting intersection $\mathcal{T}^{\text{clip}}$. Generating a two-level simplex mesh is an option, and building a mesh of general polytopes is another. Depending on the parametriation of $\mathcal{B}$, $\mathcal{T}^{\text{clip}}$ can hold linear or nonlinear elements (see Chapter 2 and Chapter 3, respectively).

Chapter 2 and Chapter 3 present algorithms to generate $\mathcal{T}^{\text{clip}}$, and also $\mathcal{B}^{\text{int}}$. Both linear and nonlinear algorithms rely on 3D polytopal clipping. However, this work considers a different 2D intersection algorithm for demonstration purposes. This 2D algorithm considers a mesh enrichment of each cell $K \in \mathcal{T}$ with the elements of the surface mesh $F \in \mathcal{B}$. After this enrichment, we have a two-level mesh that is cell-wise conformal. The subcells can be classified as interior and interior through a depth-first traversal algorithm based on edges (assuming the conformity within each cell). It is important to note that extending this enrichment algorithm to 3D is not trivial. Inserting 3D edges may introduce inaccuracies requiring a more complex development (see [101]). We can consider other 2D algorithms, e.g., the 2D restriction of the algorithms in Chapter 2.

5.3.2 Local classification

The definition of the non-intersected background cells $K \in \mathcal{T}^{\text{bg}} \setminus \mathcal{T}^{\text{cut}}$ is a simple task in serial computations. Taking advantage of the information given by the intersected cells $K \in \mathcal{T}^{\text{cut}}$, a depth-first traversal propagation is an efficient choice. In contrast to ray tracing algorithms [5], this method does not require floating point operations.

The algorithm utilizes nodal propagation of the relative positions, i.e., interior and exterior. Nodal propagation is trivial in conformal meshes, e.g., Cartesian meshes. However, we may need to propagate across the hanging nodes in non-conformal meshes, e.g., octree meshes from `p4est`. It is worth noting that we do not need to consider the hanging nodes in 2:1 k-balanced octrees, i.e., a maximum of one hanging node per edge. The 2:1 k-balance constraint improves the parallelization and scalability of distributed tree-based meshes (see [10]). In the 2:1 k-balance, a minimum of one true node connects every two neighboring cells.

5.3.3 Global distributed algorithm

Each task in a distributed-memory environment computes local computations in a subdomain $\Omega_s^{\text{art}} \subset \Omega^{\text{art}}$. This subdomain is discretized with a local background mesh $\mathcal{T}_s^{\text{bg}}$ in each task. A single task can not contain the global mesh $\mathcal{T}^{\text{bg}}$ in large-scale computations. One additional task contains a coarse background mesh $\mathcal{T}^{\text{coarse}}$. Each cell of this coarse mesh $K_s^{\text{coarse}} \in \mathcal{T}^{\text{coarse}}$ represents a subdomain Ω_s^{art}, thus K_s^{coarse} and Ω_s^{art} belong to a single task s. This coarse mesh can be reused in the solver stage, e.g., in the BDDC solver [32]. Similarly to BDDC, one can consider a multi-level coarse mesh to handle a larger number of tasks. However, in this work, we consider a single coarse mesh (see Figure 5.2).

The main advantage of using a two-level distributed algorithm is the reduction of communications. The centralized communications go through the root processor that contains the coarse mesh. The processors communicate with the root processor using `gather` and `scatter` MPI commands. The processors can also communicate with the nearest neighbors using `send`, `recv`, or `sendrecv` MPI commands. The nearest-neighbor communications enforce the consistency in the elements near the interface between subdomains. We perform the least number of nearest-neighbor communications to maintain the scalability of the algorithm.

The algorithm Algorithm 19 performs the intersection of the background cells and the inside propagation in a distributed-memory computation. We developed an efficient and scalable algorithm by overlapping computations. Each MPI task runs the algorithm. The fine tasks execute the code blocks within $s \neq s^{\text{coarse}}$ (see line 1, 12 and 20) and the coarse task runs the blocks in $s = s^{\text{coarse}}$ (see line 15). All the tasks execute the code outside these conditions.

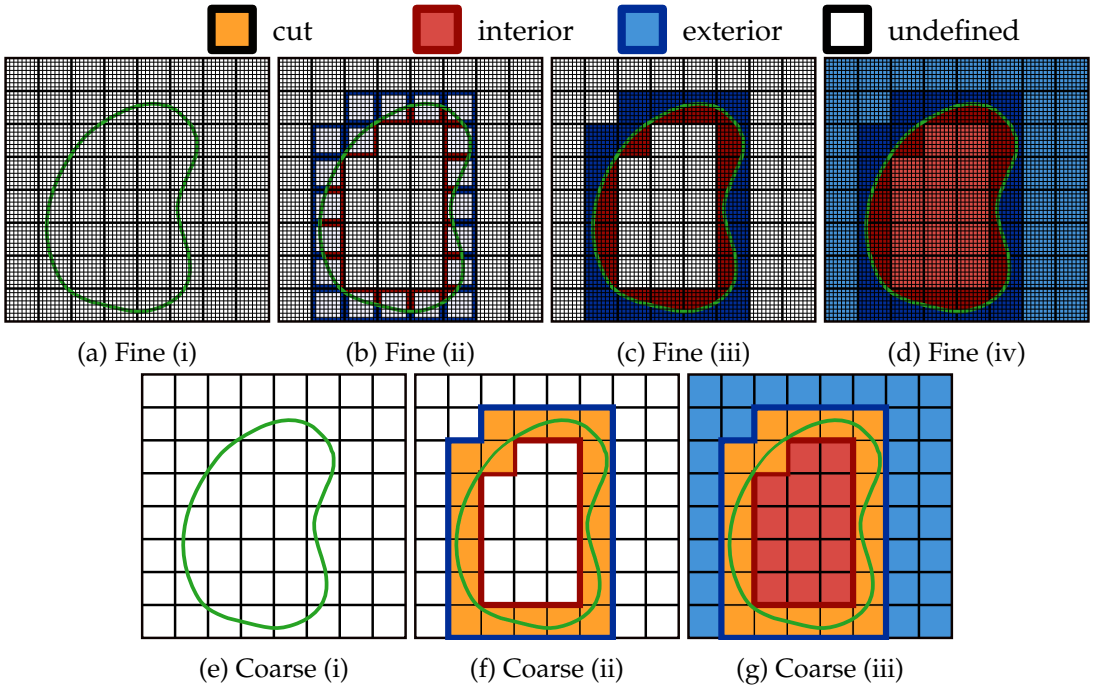


Figure 5.2: Representation of the two-level propagation algorithm. The domain boundary $\partial\Omega$ intersects the fine (a) and coarse mesh (e). First, we intersect the domain boundaries Ω_s^{art} (b) to define the coarse cells $K_s^{\text{coarse}} \in \mathcal{T}^{\text{coarse}}$ (f). The `gather` MPI command performs the fine to coarse communications. Then, we propagate the coarse cells (g) while intersecting the rest of the fine subdomains (c). Finally, after a coarse to fine communication (`scatter`), we define the cells in the untouched subdomains (d).

Algorithm 19 `distributed_intersection`$(\mathcal{T}_s^{\text{bg}}, K_s^{\text{coarse}}, \Omega_s)$

if $s \neq s^{\text{coarse}}$ **then**
  $\mathcal{T}_s^{\text{bnd}} \leftarrow \{K \in \mathcal{T}_s : K \cap \partial K_s^{\text{coarse}} \neq \emptyset\}$
  $\mathcal{T}_s^{\text{clip,bnd}} \leftarrow \{K \cap \Omega_s : K \in \mathcal{T}_s^{\text{bnd}}, K \cap \partial\Omega_s \neq \emptyset\}$
  $\mathcal{L}_s \leftarrow$ `location_map`$(\mathcal{T}_s^{\text{clip,bnd}})$
  $\mathcal{L}_s \leftarrow$ `propagate_location`$(\mathcal{T}_s^{\text{bnd}}, \mathcal{L}_s)$
  $\mathcal{L}_s \leftarrow$ `sendrecv`$(\mathcal{L}_s)$
  $\mathcal{L}_s^{\text{coarse}} \leftarrow \{(F, \mathcal{L}_s(\mathcal{F}(F))) : F \in K_s^{\text{coarse}}\}$
end if
$\mathcal{L}^{\text{coarse}} \leftarrow$ `gather`$(\mathcal{L}_s^{\text{coarse}})$
if $s \neq s^{\text{coarse}}$ **then**
  $\mathcal{T}_s^{\text{bulk}} \leftarrow \mathcal{T}_s \setminus \mathcal{T}_s^{\text{bnd}}$
  $\mathcal{T}_s^{\text{clip}} \leftarrow \mathcal{T}_s^{\text{clip,bnd}} \cup \{K \cap \Omega_s : K \in \mathcal{T}_s^{\text{bulk}}, K \cap \partial\Omega_s \neq \emptyset\}$
  $\mathcal{L}_s \leftarrow$ `location_map`$(\mathcal{T}_s^{\text{cut}})$
  $\mathcal{L}_s \leftarrow$ `propagate_location`$(\mathcal{T}_s^{\text{bg}}, \mathcal{L}_s)$
else
  $\mathcal{T}^{\text{coarse}} \leftarrow \mathcal{T}_s$
  $\mathcal{L}^{\text{coarse}} \leftarrow$ `propagate_location`$(\mathcal{T}^{\text{coarse}}, \mathcal{L}^{\text{coarse}})$
end if
$\mathcal{L}_s^{\text{coarse}} \leftarrow$ `scatter`$(\mathcal{L}^{\text{coarse}})$
if $s \neq s^{\text{coarse}}$ **then**
  if $\mathcal{T}_s^{\text{clip}} = \emptyset$ **then**
    $\mathcal{L}_s \leftarrow \{(F, \mathcal{L}_s^{\text{coarse}}(K_s^{\text{coarse}})) : F \in \mathcal{T}_s\}$
  end if
  $\mathcal{T}_s^{\text{in}} \leftarrow \{K \in \mathcal{T}_s^{\text{bg}} : \mathcal{L}_s(K) = \text{in}\}$
end if
return $\mathcal{T}_s^{\text{in}} \cup \mathcal{T}_s^{\text{clip}}$

The first part of the algorithm, line 2-3, performs the intersection and classification of the cells near the boundary of each subdomain $\partial\Omega_s^{\text{art}}$ (or $\partial K_s^{\text{coarse}}$); see Figure 5.2(b). Here, we define the location map of the faces as $\mathcal{L} : F \mapsto \{\text{in}, \text{out}, \text{cut}\}$ for $F \in \texttt{faces}(\mathcal{T})$. In line 4, we extract the location map from the clipped boundary mesh $\mathcal{T}_s^{\text{clip,bnd}}$ in a straightforward operation. The map $\mathcal{L}_s$ is indexed with the local faces of $\mathcal{T}_s^{\text{bg}}$. Then, in line 5, we propagate the location $\mathcal{L}_s$ through the domain boundary cells $\mathcal{T}_s^{\text{bnd}}$. The line 6 enforces consistency of the interface faces with nearest-neighbor communications, e.g., `sendrecv` MPI command. At this point, the location of the subdomains is defined with minimum operations and communications. Thus, we build a map for the location of the local coarse entities $\mathcal{L}_s^{\text{coarse}}$ in line 7. For non-interested subdomains, i.e., $\mathcal{T}_s^{\text{clip}} = \emptyset$, the location of the entire subdomain is set as *undefined*. The coarse task collects the local locations of the coarse entities in line 9 with a `gather` communication (see Figure 5.2(f)).

In the second part, we overlap the intersection of the rest of the fine cells (line 12) and the classification of the coarse mesh (line 15). This overlapping is crucial to avoid the idling of the main bunch of processors. The intersection of the bulk cells $\mathcal{T}_s^{\text{bulk}}$ in line 12 is analogous to the intersections of the boundary cells $\mathcal{T}_s^{\text{bnd}}$ in line 2. Here, we intersect and classify the entire subdomains. Classifying the coarse cells $\mathcal{T}^{\text{coarse}}$ in line 17 utilizes serial propagation algorithms. After this part, the coarse sends the coarse location to each processor through the `scatter` command (line 19). Finally, we define the local cells in the untouched domains (line 22). This algorithm returns a mesh ready for integration $\mathcal{T}^{\text{in}} \cup \mathcal{T}^{\text{clip}}$.

5.4 Numerical experiments

5.4.1 Experimental setup

The numerical experiments have been performed in the Marenostrum IV supercomputer at the Barcelona Supercomputing Center. The supercomputer has 3.456 nodes with 2 Intel Xenon Platinum chips of 24 cores at 2.1 GHz. The experiment times are calculated from the lowest of 20 runs to minimize the impact of external factors on CPU timings. The implementation is done in FEMPAR [13], a parallel FE object-oriented Fortran library that scales up to hundreds of thousands of processors. In FEMPAR, we generate octree meshes with `p4est` library [37]. The parallelization is done with the Intel MPI library available in Marenostrum IV.

5.4.2 Parallel scalability

In this section, we focus the results on the parallel performance, specifically on the scalability of the algorithm. The geometry is a polygonal circle of 100 evenly spaced points. This circle of radius $R = 0.4$ is embedded in a unit square artificial domain Ω^{art}. The background mesh $\mathcal{T}^{\text{bg}}$ is built with `p4est`. Thus, the number of cells is a power of

4^l where l is the number of uniform refinement levels. In the presented data points, we execute a manufactured solution unfitted FE of the Poisson equation.

We perform strong scalability tests to estimate the local size in the weak scaling tests. These tests are performed with a fixed number of cells, 1.048.576. As expected, the strong scalability in Figure 5.3 is not optimal. We do not aim to design a strongly scalable algorithm. The coarse task becomes more significant with the number of tasks, and eventually, it will limit the task overlapping. In the Algorithm 19, $\mathcal{T}_s^{\text{bulk}} \cap \Omega_s$ and the propagation in the coarse mesh $\mathcal{T}^{\text{coarse}}$ are overlapped. Larger loads in the coarse propagation will force idling to the rest of the processors.

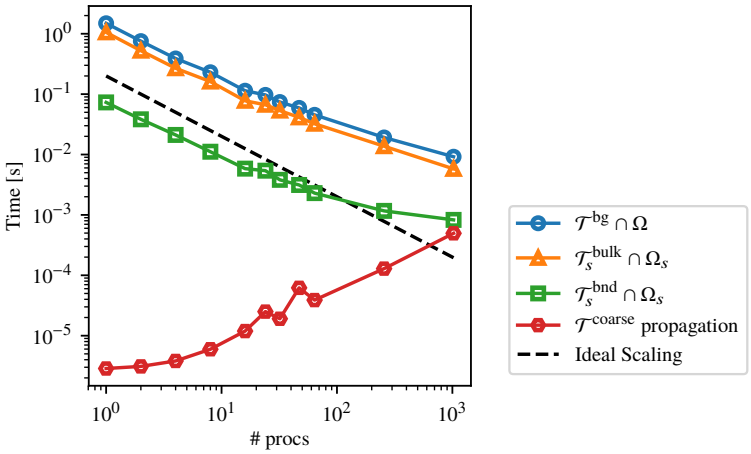


Figure 5.3: Strong scaling of the creation of distributed embedded discretizations with fixed background mesh $\mathcal{T}^{\text{bg}}$ (1.048.576 cells). The figure shows the strong scaling of the stages of the algorithm. The $\mathcal{T}^{\text{bg}} \cap \Omega$ represent entire Algorithm 19, $\mathcal{T}_s^{\text{bnd}} \cap \Omega_s$ corresponds to line 1, $\mathcal{T}_s^{\text{bulk}} \cap \Omega_s$ executes line 12 and $\mathcal{T}^{\text{coarse}}$ runs line 17.

The algorithm is designed to be weakly scalable. Weak scaling is essential in large-scale simulations. In the weak scaling tests, we increase the number of processors P with the number of background cells N while keeping the local number of cells N/P constant. Based on the strong scaling tests, we test the following local sizes $N/P = 4^l/48$ with $l = 9, 10, 11$, namely $5k$, $22k$, and $87k$, respectively. In Figure 5.4, we show perfect weak scalability for local problems of $22k$ and $87k$ cells. However, for smaller local problems, e.g., $5k$ cells, the scalability is slightly reduced. The tests are performed for $P = 48 \cdot 4^m$, $m = 0, ..., 4$ processors. Therefore, we have tested perfect weak scalability up to 1,073,741,824 cells in 12,288 processors.

It is important to note that, in these tests, the intersection algorithm represents around 5% of the simulation workflow. Even though the intersection stage may become more significant in the more complex 3D geometries, the weight of the intersection step only depends on the size of the local problem.

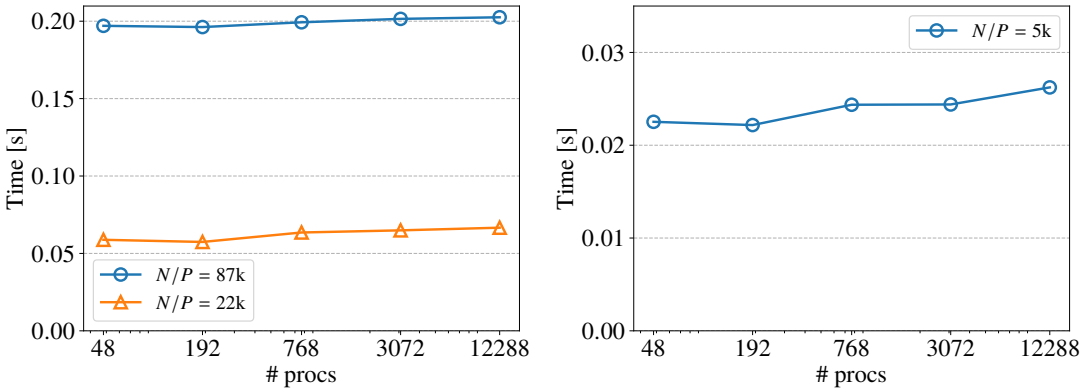


Figure 5.4: Weak scaling of the creation of distributed embedded discretizations. In (a), the algorithm presents perfect weak scalability for larger local loads, e.g., 87k cells per processor. In (b), for smaller local loads, the scalability is reduced.

5.5 Conclusions and future work

In this chapter, we have presented a distributed extension of the algorithms of the document in this thesis, exploiting the embarrassingly parallel implementations. This extension presents perfect weak scalability over ten thousand processors and 10^9 background cells. Such extension accelerates the pipeline from CAD models to FE simulations of PDEs.

The presented results are a proof of concept of the distributed extension. The algorithm is implemented in FEMPAR, an object-oriented Fortran code that handles large-scale FE computations. Even though the tests are performed with a simple 2D intersection algorithm in `p4est` background meshes, the propagation algorithms are dimension-independent. Therefore, complex 3D intersections can readily utilize this algorithm.

The future work involves implementing these algorithms in the Julia implementations, like `STLCutters.jl` with Gridap meshes. Such implementation will make use of `GridapDistributed.jl` [15] utilities. The Gridap ecosystem also provides tools for AMR, e.g., `GridapP4est.jl` [74]. The Julia implementations will exploit the potential of the other chapters of this thesis, e.g., the space-time methods and the nonlinear geometry representations. In general, the distributed extension will allow us to solve PDE in more extensive problems with more complex geometries in an automated unfitted FE pipeline.

Chapter 6

Conclusions and future work

6.1 Conclusions

This thesis has introduced innovative techniques to unlock the potential of unfitted FE methods for the numerical approximation of PDEs in complex geometries represented by CAD models. These techniques can be applied across a broad spectrum of physics in science and engineering problems, e.g., FSI in civil engineering projects, additive manufacturing simulations, and topology optimization. The state-of-the-art unfitted tools fail to address the challenges of the geometries represented by CAD models. The presented algorithms tackle these challenges with a scalable, accurate, and robust approach. Real-world applications often involve transient dynamics. In response, we have formulated and analyzed a space-time framework to simulate evolving domains. Parallel computing is of paramount importance to simulate practical applications. The presented distributed algorithms facilitate efficiently exploiting the available HPC resources.

In this section, we summarize the main conclusions. The conclusions of each chapter are independent from each other since each they are self-contained.

In Chapter 2, we have designed a fully automated simulation pipeline for the numerical approximation of PDEs on general domains described by a linear boundary mesh, e.g., STL models. The algorithm utilizes a structured background mesh and an unfitted FE formulation on this mesh. The presented algorithms successfully address the intricate task of integrating the background cells intersected by the domain boundary. These intersections become particularly challenging when dealing with complex and non-convex geometries with many faces. A numerical analysis of this algorithm has demonstrated accuracy and robustness on unfitted FE simulations, in particular with the AgFEM [22]. Notably, the algorithm implementation has achieved successful applications across all 3D analysis-suitable meshes in the Thingi10K database [123] (almost 5,000 meshes). These computations exhibit scalability and are performed cell-wise, thus, suitable for distributed memory machines. The algorithm implementation is available as open source software [76].

In Chapter 3, we have extended the automated pipeline of Chapter 2 to high-order geometries defined by CAD models. This high-order extension dramatically complicates the geometry description and its intersections. The challenge of numerically

integrating over these domains involves handling trimming curves, nonlinear intersections, and non-convex domains. The corresponding analysis has underscored robustness and optimal *hp*-convergence achieved in the tailored benchmarks. Moreover, the methods have been successfully employed in simulating real-world geometries defined by CAD models in STEP files. This compressive analysis positions these methods as cutting-edge tools for simulating PDEs on high-order geometries with unfitted FE methods.

In Chapter 4, we have introduced a novel space-time formulation for unfitted FEM that can handle large displacements of moving domains. This formulation relies on space-only embedded discretizations. Therefore, we circumvent the necessity for 4D geometric algorithms. This achievement is realized by integrating the time slab interface via an intersection mesh, allowing exact integration. The efficacy of this formulation has been substantiated through a comparative assessment with an alternative space-time analysis employing unfitted FEM [7]. This validation underscores the practical feasibility of implementing the method within real-world simulations. The approach's potential has been unveiled by successfully simulating two flow dynamics examples around a rotating geometry, one featuring a 2D geometry and the other involving a 3D geometry.

In Chapter 5, we have introduced a distributed extension of the algorithms expounded within this thesis. This extension leverages the cell-wise implementations detailed in Chapter 2-4, which are embarrassingly parallel. The parallelization significantly accelerates the design pipeline proposed in this thesis. CAD to CAE simulations substantial allocation of computational resources, particularly when addressing complex geometries characterized by a high level of detail. The proposed parallel algorithms effectively distribute the workloads and diminish potential bottlenecks, such as the classification of the background cells. We have demonstrated optimal weak scalability over ten thousand processors and 10^9 background cells. These tests were performed upon `p4est` quadtree meshes in FEMPAR, an object-oriented Fortran code engineered for managing large-scale parallel computations. These tests were not proven on the concrete implementations of Chapter 2-4. However, the underlying algorithms are agnostic to the mesh's dimension and topology.

6.2 Future work

The novelties introduced within this thesis constitute a substantial step forward in the field of unfitted FE methods. Despite their significance, these advancements are insufficient to replace the prevailing design workflows in numerous scientific and engineering applications. Our numerical framework still has several limitations that need to be addressed to achieve a high level of competitiveness compared to manual pipelines. In this section, we discuss the following lines of research to develop the latent potential of the algorithms expounded herein.

- **Formulate boundary layer problems on high-order unfitted methods**

  In flow dynamics, the simulation of the region near the boundary assumes paramount importance. Unfitted methods fail to capture the boundary layer dynamics in large Reynold numbers. The AMR techniques do not solve this problem efficiently. In the context of body-fitted meshes, a prevalent strategy involves the utilization of tailored boundary layer meshes. Future work explores the combination of boundary layer meshes with unfitted FE methods. This approach involves coupling separate discretizations as proposed in [119]. The boundary layer discretizations should be extruded from high-order surfaces to provide an accurate solution of the flow around complex geometries.

- **Exploit parallel framework for large-scale simulations**

  The algorithms presented in Chapter 5 have demonstrated high scalability on distributed memory machines. However, the integration and testing of these algorithms with those detailed in Chapter 2-4 remain pending. Such integration is key to utilizing the inherent potential for solving larger and more accurate problems. It is imperative to optimize the workflow to benefit from the available HPC resources. The usage AMR, e.g., `p4est` grids [37], in space and time with proper error estimator will dramatically reduce the number of DOFs and the computational cost of the simulations. The efficiency of distributed computations with AMR requires proper load balancing. Furthermore, the optimization horizon extends to the enhancement of integration steps. The integration process can further benefit from moment fitting techniques [40, 57].

- **Test multiphysics capabilities of the framework**

  One of the main motivations of Chapter 4 and the entire thesis is providing tools for efficiently simulating FSI problems. Even though direct validation through multiphysics scenarios remains pending, the presented methods are general enough for this purpose. The exigencies of practical FSI problems in complex domains demand extensive computational resources. Therefore, the parallel extension (Chapter 5) is a mandatory prerequisite before undertaking extensive FSI simulations. The proposed FSI framework will be able to handle large displacements of the interface using unfitted FE methods. The utilization of space-time methods in Chapter 4 increases robustness compared to standard body-fitted ALE approaches. The simulation of FSI problems utilizing the methods presented in this thesis will demonstrate their inherent potential.

Bibliography

[1] P. ANTOLIN, A. BUFFA, AND M. MARTINELLI, *Isogeometric analysis on v-reps: First results*, Computer Methods in Applied Mechanics and Engineering, 355 (2019), pp. 976–1002.

[2] P. ANTOLIN AND T. HIRSCHLER, *Quadrature-free immersed isogeometric analysis*, Engineering with Computers, 38 (2022), pp. 4475–4499.

[3] P. ANTOLIN, X. WEI, AND A. BUFFA, *Robust numerical integration on curved polyhedra based on folded decompositions*, Computer Methods in Applied Mechanics and Engineering, 395 (2022), p. 114948.

[4] D. N. ARNOLD, *Finite Element Exterior Calculus*, Society for Industrial and Applied Mathematics, Dec. 2018.

[5] J. ARVO AND D. KIRK, *A survey of ray tracing in*, in An Introduction to Ray Tracing, Academic Press, Ltd., 1989, pp. 201–262.

[6] M. BADER, *Space-Filling Curves: An Introduction With Applications in Scientific Computing*, Springer Science & Business Media, 2012. Google-Books-ID: eIe_OdFP0WkC.

[7] S. BADIA, H. DILIP, AND F. VERDUGO, *Space-time unfitted finite element methods for time-dependent problems on moving domains*, Computers & Mathematics with Applications, 135 (2023), pp. 60–76.

[8] S. BADIA, J. DRONIOU, AND L. YEMM, *Conditioning of a hybrid high-order scheme on meshes with small faces*, (2021).

[9] S. BADIA, J. HAMPTON, AND J. PRINCIPE, *EMBEDDED MULTILEVEL MONTE CARLO FOR UNCERTAINTY QUANTIFICATION IN RANDOM DOMAINS*, International Journal for Uncertainty Quantification, 11 (2021), pp. 119–142.

[10] S. BADIA, A. F. MARTÍN, E. NEIVA, AND F. VERDUGO, *The aggregated unfitted finite element method on parallel tree-based adaptive meshes*, SIAM Journal on Scientific Computing, 43 (2021), pp. C203–C234.

[11] S. BADIA, A. F. MARTÍN, E. NEIVA, AND F. VERDUGO, *The aggregated unfitted finite element method on parallel tree-based adaptive meshes*, SIAM Journal on Scientific Computing, 43 (2021), pp. C203–C234.

[12] S. BADIA, A. F. MARTÍN, AND J. PRINCIPE, *A highly scalable parallel implementation of balancing domain decomposition by constraints*, SIAM Journal on Scientific Computing, 36 (2014), pp. C190–C218.

[13] S. BADIA, A. F. MARTÍN, AND J. PRINCIPE, *FEMPAR: An Object-Oriented Parallel Finite Element Framework*, Archives of Computational Methods in Engineering, 25 (2018), pp. 195–271.

[14] S. BADIA, A. F. MARTÍN, AND F. VERDUGO, *Mixed aggregated finite element methods for the unfitted discretisation of the Stokes problem*, SIAM Journal on Scientific Computing, 40 (2018), pp. B1541–B1576.

[15] S. BADIA, A. F. MARTÍN, AND F. VERDUGO, *GridapEmbedded. Version 0.2.*, Nov. 2021. Available at `https://github.com/gridap/GridapDistributed.jl`.

[16] S. BADIA, A. F. MARTÍN, AND F. VERDUGO, *GridapDistributed: a massively parallel finite element toolbox in julia*, Journal of Open Source Software, 7 (2022), p. 4157.

[17] S. BADIA, P. A. MARTORELL, AND F. VERDUGO, *Geometrical discretisations for unfitted finite elements on explicit boundary representations*, Journal of Computational Physics, 460 (2022), p. 111162.

[18] S. BADIA, E. NEIVA, AND F. VERDUGO, *Linking ghost penalty and aggregated unfitted methods*, Computer Methods in Applied Mechanics and Engineering, 388 (2022), p. 114232.

[19] S. BADIA, E. NEIVA, AND F. VERDUGO, *Robust high-order unfitted finite elements by interpolation-based discrete extension*, Computers & Mathematics with Applications, 127 (2022), pp. 105–126.

[20] S. BADIA AND F. VERDUGO, *Robust and scalable domain decomposition solvers for unfitted finite element methods*, Journal of Computational and Applied Mathematics, 344 (2018), pp. 740–759.

[21] S. BADIA AND F. VERDUGO, *Gridap: An extensible Finite Element toolbox in Julia*, Journal of Open Source Software, 5 (2020), p. 2520.

[22] S. BADIA, F. VERDUGO, AND A. F. MARTÍN, *The aggregated unfitted finite element method for elliptic problems*, Computer Methods in Applied Mechanics and Engineering, 336 (2018), pp. 533–553.

[23] C. B. BARBER, D. P. DOBKIN, AND H. HUHDANPAA, *The quickhull algorithm for convex hulls*, ACM Transactions on Mathematical Software, 22 (1996), pp. 469–483.

[24] G. L. BEAU, S. RAY, S. ALIABADI, AND T. TEZDUYAR, *SUPG finite element computation of compressible flows with the entropy and conservation variables formulations*,

Computer Methods in Applied Mechanics and Engineering, 104 (1993), pp. 397–422.

[25] G. BEER, B. MARUSSIG, AND C. DUENSER, *Simulation with trimmed models*, in The Isogeometric Boundary Element Method, Springer International Publishing, sep 2019, pp. 185–216.

[26] T. BELYTSCHKO, N. MOËS, S. USUI, AND C. PARIMI, *Arbitrary discontinuities in finite elements*, International Journal for Numerical Methods in Engineering, 50 (2001), pp. 993–1013.

[27] J. BEZANSON, A. EDELMAN, S. KARPINSKI, AND V. B. SHAH, *Julia: A fresh approach to numerical computing*, SIAM Review, 59 (2017), pp. 65–98.

[28] J. BONET AND R. D. WOOD, *Nonlinear continuum mechanics for finite element analysis*, Cambridge university press, 1997.

[29] M. J. BORDEN, M. A. SCOTT, J. A. EVANS, AND T. J. R. HUGHES, *Isogeometric finite element data structures based on bézier extraction of NURBS*, International Journal for Numerical Methods in Engineering, 87 (2010), pp. 15–47.

[30] C. F. BORGES AND T. PASTVA, *Total least squares fitting of bézier and b-spline curves to ordered data*, Computer Aided Geometric Design, 19 (2002), pp. 275–289.

[31] A. BOWER, *Continuum mechanics, elasticity.* Brown Universit, School of Engineering, 2012. Available at `https://www.brown.edu/Departments/Engineering/Courses/En221/Notes/Elasticity/Elasticity.htm`, Accessed: May, 2023.

[32] M. BREZINA AND P. S. VASSILEVSKI, *Smoothed Aggregation Spectral Element Agglomeration AMG: SA-ρAMGe*, in Large-Scale Scientific Computing, I. Lirkov, S. Margenov, and J. Waśniewski, eds., no. 7116 in Lecture Notes in Computer Science, Springer Berlin Heidelberg, jun 2011, pp. 3–15.

[33] E. BURMAN, *Ghost penalty*, Comptes Rendus Mathematique, 348 (2010), pp. 1217–1220.

[34] E. BURMAN, S. CLAUS, P. HANSBO, M. G. LARSON, AND A. MASSING, *CutFEM: Discretizing Geometry and Partial Differential Equations*, International Journal for Numerical Methods in Engineering, 104 (2015), pp. 472–501.

[35] E. BURMAN AND M. A. FERNÁNDEZ, *An unfitted nitsche method for incompressible fluid–structure interaction using overlapping meshes*, Computer Methods in Applied Mechanics and Engineering, 279 (2014), pp. 497–514.

[36] C. BURSTEDDE AND J. HOLKE, *A tetrahedral space-filling curve for nonconforming adaptive meshes*, SIAM Journal on Scientific Computing, 38 (2016), pp. C471–C503.

[37] C. BURSTEDDE, L. C. WILCOX, AND O. GHATTAS, `p4est`*: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees*, SIAM Journal on Scientific Computing, 33 (2011), pp. 1103–1133.

[38] M. CARRATURO, J. JOMO, S. KOLLMANNSBERGER, A. REALI, F. AURICCHIO, AND E. RANK, *Modeling and experimental validation of an immersed thermo-mechanical part-scale analysis for laser powder bed fusion processes*, Additive Manufacturing, 36 (2020), p. 101498.

[39] B. CHAZELLE, *Convex partitions of polyhedra: A lower bound and worst-case optimal algorithm*, SIAM Journal on Computing, 13 (1984), pp. 488–507.

[40] E. B. CHIN AND N. SUKUMAR, *An efficient method to integrate polynomials over polytopes and curved solids*, Computer Aided Geometric Design, 82 (2020), p. 101914.

[41] S. DANISCH, *MeshIO. Version 0.4.*, Apr. 2020. Available at `https://github.com/JuliaIO/MeshIO.jl`.

[42] F. DE PRENTER, C. V. VERHOOSEL, E. H. VAN BRUMMELEN, M. G. LARSON, AND S. BADIA, *Stability and conditioning of immersed finite element methods: Analysis and remedies*, Archives of Computational Methods in Engineering, 30 (2023), p. 3617–3656.

[43] F. DE PRENTER, C. V. VERHOOSEL, G. J. VAN ZWIETEN, AND E. H. VAN BRUMMELEN, *Condition number analysis and preconditioning of the finite cell method*, Computer Methods in Applied Mechanics and Engineering, 316 (2017), pp. 297–327.

[44] R. DEKKER, F. MEER, J. MALJAARS, AND L. SLUYS, *A cohesive XFEM model for simulating fatigue crack growth under mixed-mode loading and overloading*, International Journal for Numerical Methods in Engineering, 118 (2019), pp. 561–577.

[45] J. DONEA, S. GIULIANI, AND J. HALLEUX, *An arbitrary lagrangian-eulerian finite element method for transient dynamic fluid-structure interactions*, Computer Methods in Applied Mechanics and Engineering, 33 (1982), pp. 689–723.

[46] L. ENGVALL AND J. A. EVANS, *Isogeometric triangular bernstein–bézier discretizations: Automatic mesh generation and geometrically exact finite element analysis*, Computer Methods in Applied Mechanics and Engineering, 304 (2016), pp. 378–407.

[47] L. ENGVALL AND J. A. EVANS, *Isogeometric unstructured tetrahedral and mixed-element bernstein–bézier discretizations*, Computer Methods in Applied Mechanics and Engineering, 319 (2017), pp. 83–123.

[48] E. FEBRIANTO, M. ORTIZ, AND F. CIRAK, *Mollified finite element approximants of arbitrary order and smoothness*, Computer Methods in Applied Mechanics and Engineering, 373 (2021), p. 113513.

[49] L. FORMAGGIA, F. GATTI, AND S. ZONCA, *An XFEM/DG approach for fluid-structure interaction problems with contact*, Applications of Mathematics, 66 (2021), pp. 183–211.

[50] T. FRIES, *Higher-order conformal decomposition FEM (CDFEM)*, Computer Methods in Applied Mechanics and Engineering, 328 (2018), pp. 75–98.

[51] T.-P. FRIES AND S. OMEROVIĆ, *Higher-order accurate integration of implicit geometries*, International Journal for Numerical Methods in Engineering, 106 (2015), pp. 323–371.

[52] T. P. FRIES, S. OMEROVIĆ, D. SCHÖLLHAMMER, AND J. STEIDL, *Higher-order meshing of implicit geometries—Part I: Integration and interpolation in cut elements*, Computer Methods in Applied Mechanics and Engineering, (2017).

[53] T. P. FRIES, S. OMEROVIĆ, D. SCHÖLLHAMMER, AND J. STEIDL, *Higher-order meshing of implicit geometries—Part I: Integration and interpolation in cut elements*, Computer Methods in Applied Mechanics and Engineering, 313 (2017), pp. 759–784.

[54] C. GEUZAINE AND J.-F. REMACLE, *Gmsh: A 3-d finite element mesh generator with built-in pre- and post-processing facilities*, International Journal for Numerical Methods in Engineering, 79 (2009), pp. 1309–1331.

[55] B. GIOVANARDI, L. FORMAGGIA, A. SCOTTI, AND P. ZUNINO, *Unfitted FEM for modelling the interaction of multiple fractures in a poroelastic medium*, in Lecture Notes in Computational Science and Engineering, Springer International Publishing, 2017, pp. 331–352.

[56] *Grabcad*, 2023. Available at `https://grabcad.com/library/connecting-rod-416`, Accessed: May, 2023.

[57] D. GUNDERMAN, K. WEISS, AND J. A. EVANS, *High-accuracy mesh-free quadrature for trimmed parametric surfaces and volumes*, Computer-Aided Design, 141 (2021), p. 103093.

[58] P. HACHENBERGER, *Exact minkowksi sums of polyhedra and exact and efficient decomposition of polyhedra into convex pieces*, Algorithmica, 55 (2008), pp. 329–345.

[59] F. HEIMANN AND C. LEHRENFELD, *Geometrically higher order unfitted space-time methods for pdes on moving domains: Geometry error analysis*, arXiv, (2023).

[60] F. HEIMANN, C. LEHRENFELD, AND J. PREUSS, *Geometrically higher order unfitted space-time methods for pdes on moving domains*, SIAM Journal on Scientific Computing, 45 (2023), p. B139–B165.

[61] J. HEISKALA, *DirectQhull. Version 0.2.0.*, Dec. 2022. Available at `https://github.com/JuhaHeiskala/DirectQhull.jl`.

[62] Y. HU, Q. ZHOU, X. GAO, A. JACOBSON, D. ZORIN, AND D. PANOZZO, *Tetrahedral meshing in the wild*, ACM Transactions on Graphics, 37 (2018).

[63] T. J. R. HUGHES, J. A. COTTRELL, AND Y. BAZILEVS, *Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement*, Computer Methods in Applied Mechanics and Engineering, 194 (2005), pp. 4135–4195.

[64] A. JOHANSSON, B. KEHLET, M. G. LARSON, AND A. LOGG, *Multimesh finite element methods: Solving PDEs on multiple intersecting meshes*, Computer Methods in Applied Mechanics and Engineering, 343 (2019), pp. 672–689.

[65] G. KARYPIS, *A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices. Version 5.1.0*, tech. rep., University of Minnesota, Department of Computer Science and Engineering, Minneapolis, MN, 2013. Available at `http://glaros.dtc.umn.edu/gkhome/fetch/sw/metis/manual.pdf`.

[66] G. KARYPIS, K. SCHLOEGEL, AND V. KUMAR, *ParMETIS: Parallel graph partitioning and sparse matrix ordering library*, tech. rep., Department of Computer Science and Engineering, University of Minnesota, 1997.

[67] V. KARYPIS, GEORGE; KUMAR, *Metis: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices*, tech. rep., University of Minnesota, Department of Computer Science and Engineering, 1997. Available at `https://hdl.handle.net/11299/215346`.

[68] J. KLOSOWSKI, M. HELD, J. MITCHELL, H. SOWIZRAL, AND K. ZIKAN, *Efficient collision detection using bounding volume hierarchies of k-DOPs*, IEEE Transactions on Visualization and Computer Graphics, 4 (1998), pp. 21–36.

[69] G. LEGRAIN AND N. MOËS, *Adaptive anisotropic integration scheme for high-order fictitious domain methods: Application to thin structures*, International Journal for Numerical Methods in Engineering, 114 (2018), pp. 882–904.

[70] C. LEHRENFELD AND M. OLSHANSKII, *An eulerian finite element method for pdes in time-dependent domains*, ESAIM: Mathematical Modelling and Numerical Analysis, 53 (2019), p. 585–614.

[71] X. LI AND F. CHEN, *Exact and approximate representations of trimmed surfaces with NURBS and bézier surfaces*, in 2009 11th IEEE International Conference on Computer-Aided Design and Computer Graphics, IEEE, aug 2009.

[72] J. LÓPEZ, J. HERNÁNDEZ, P. GÓMEZ, AND F. FAURA, *VOFTools - a software package of calculation tools for volume of fluid methods using general convex grids*, Computer Physics Communications, 223 (2018), pp. 45–54.

[73] J. LÓPEZ, J. HERNÁNDEZ, P. GÓMEZ, AND F. FAURA, *Non-convex analytical and geometrical tools for volume truncation, initialization and conservation enforcement in VOF methods*, Journal of Computational Physics, 392 (2019), pp. 666–693.

[74] A. F. MARTÍN, *GridapP4est. Version 0.2.*, Feb. 2023. Available at `https://github.com/gridap/GridapP4est.jl`.

[75] P. A. MARTORELL AND S. BADIA, *High order unfitted finite element discretizations for explicit boundary representations*, arXiv preprint arXiv:2311.14363, (2023).

[76] P. A. MARTORELL, S. BADIA, AND F. VERDUGO, *STLCutters*, Zenodo, (2021).

[77] B. MARUSSIG AND T. J. R. HUGHES, *A review of trimming in isogeometric analysis: Challenges, data exchange and simulation aspects*, Archives of Computational Methods in Engineering, 25 (2017), pp. 1059–1127.

[78] F. MASSARWI, P. ANTOLIN, AND G. ELBER, *Volumetric untrimming: Precise decomposition of trimmed trivariates into tensor products*, Computer Aided Geometric Design, 71 (2019), pp. 1–15.

[79] F. MASSARWI, B. VAN SOSIN, AND G. ELBER, *Untrimming: Precise conversion of trimmed-surfaces to tensor-product surfaces*, Computers & Graphics, 70 (2018), pp. 80–91.

[80] A. MASSING, M. G. LARSON, AND A. LOGG, *Efficient implementation of finite element methods on nonmatching and overlapping meshes in three dimensions*, SIAM Journal on Scientific Computing, 35 (2013), pp. C23–C47.

[81] R. MITTAL AND G. IACCARINO, *Immersed Boundary Methods*, Annual Review of Fluid Mechanics, 37 (2005), pp. 239–261.

[82] B. MOURRAIN AND J. PAVONE, *Subdivision methods for solving polynomial equations*, Journal of Symbolic Computation, 44 (2009), pp. 292–306.

[83] B. MOURRAIN, F. ROUILLIER, AND M.-F. ROY, *Bernstein's basis and real root isolation*, Research Report RR-5149, INRIA, 2004.

[84] B. MÜLLER, S. KRÄMER-EIS, F. KUMMER, AND M. OBERLACK, *A high-order discontinuous Galerkin method for compressible flows with immersed boundaries*, International Journal for Numerical Methods in Engineering, 110 (2017), pp. 3–30.

[85] E. NEIVA AND S. BADIA, *Robust and scalable h-adaptive aggregated unfitted finite elements for interface elliptic problems*, Computer Methods in Applied Mechanics and Engineering, 380 (2021), p. 113769.

[86] E. NEIVA, S. BADIA, A. F. MARTÍN, AND M. CHIUMENTI, *A scalable parallel finite element framework for growing geometries. application to metal additive manufacturing*, International Journal for Numerical Methods in Engineering, 119 (2019), pp. 1098–1125.

[87] E. NEIVA, M. CHIUMENTI, M. CERVERA, E. SALSI, G. PISCOPO, S. BADIA, A. F. MARTÍN, Z. CHEN, C. LEE, AND C. DAVIES, *Numerical modelling of heat transfer and experimental validation in powder-bed fusion with the virtual domain approximation*, Finite Elements in Analysis and Design, 168 (2020), p. 103343.

[88] J. NITSCHE, *Über ein variationsprinzip zur lösung von dirichlet-problemen bei verwendung von teilräumen, die keinen randbedingungen unterworfen sind*, Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg, 36 (1971), p. 9–15.

[89] F. NOBILE AND L. FORMAGGIA, *A stability analysis for the arbitrary lagrangian eulerian formulation with finite elements*, East-West Journal of Numerical Mathematics, 7 (1999), pp. 105–132.

[90] *Open cascade technology*, 2022. Available at `https://dev.opencascade.org/`, Accessed: Jan, 2023.

[91] Y. PARK, S.-H. SON, M.-S. KIM, AND G. ELBER, *Surface–surface-intersection computation using a bounding volume hierarchy with osculating toroidal patches in the leaf nodes*, Computer-Aided Design, 127 (2020), p. 102866.

[92] N. M. PATRIKALAKIS AND T. MAEKAWA, *Shape interrogation for computer aided design and manufacturing*, Springer Berlin Heidelberg, 2010.

[93] D. POWELL AND T. ABEL, *An exact general remeshing scheme applied to physically conservative voxelization*, Journal of Computational Physics, 297 (2015), pp. 340–356.

[94] M. REUTER, T. S. MIKKELSEN, E. C. SHERBROOKE, T. MAEKAWA, AND N. M. PATRIKALAKIS, *Solving nonlinear polynomial systems in the barycentric bernstein basis*, The Visual Computer, 24 (2007), pp. 187–200.

[95] D. SCHILLINGER AND M. RUESS, *The Finite Cell Method: A review in the context of higher-order structural analysis of CAD and image-based geometric models*, Archives of Computational Methods in Engineering, 22 (2015), pp. 391–455.

[96] D. SCHILLINGER, P. K. RUTHALA, AND L. H. NGUYEN, *Lagrange extraction and projection for NURBS basis functions: A direct link between isogeometric and standard nodal finite element formulations*, International Journal for Numerical Methods in Engineering, 108 (2016), pp. 515–534.

[97] R. SCHMIDT, R. WÜCHNER, AND K.-U. BLETZINGER, *Isogeometric analysis of trimmed NURBS geometries*, Computer Methods in Applied Mechanics and Engineering, 241-244 (2012), pp. 93–111.

[98] F. SCHOLZ AND B. JÜTTLER, *Numerical integration on trimmed three-dimensional domains with implicitly defined trimming surfaces*, Computer Methods in Applied Mechanics and Engineering, 357 (2019), p. 112577.

[99] B. SCHOTT, C. AGER, AND W. A. WALL, *Monolithic cut finite element–based approaches for fluid-structure interaction*, International Journal for Numerical Methods in Engineering, 119 (2019), pp. 757–796.

[100] J. SHEN, L. BUSÉ, P. ALLIEZ, AND N. DODGSON, *A line/trimmed NURBS surface intersection algorithm using matrix representations*, Computer Aided Geometric Design, 48 (2016), pp. 1–16.

[101] H. SI, *TetGen, a delaunay-based quality tetrahedral mesh generator*, ACM Transactions on Mathematical Software, 41 (2015), pp. 1–36.

[102] I. SMEARS, *Robust and efficient preconditioners for the discontinuous galerkin time-stepping method*, IMA Journal of Numerical Analysis, (2016), pp. 1961–1985.

[103] T. SORGENTE, S. BIASOTTI, AND M. SPAGNUOLO, *A geometric approach for computing the kernel of a polyhedron*, Smart Tools and Apps for Graphics - Eurographics Italian Chapter Conference, (2021).

[104] J. STANFORD AND T. FRIES, *A higher-order conformal decomposition finite element method for plane b-rep geometries*, Computers & Structures, 214 (2019), pp. 15–27.

[105] J. W. STANFORD AND T. P. FRIES, *Higher-order accurate meshing of nonsmooth implicitly defined surfaces and intersection curves*, Computational Mathematics and Mathematical Physics, 59 (2019), pp. 2093–2107.

[106] M. B. STEPHENSON AND H. N. CHRISTIANSEN, *A polyhedron clipping and capping algorithm and a display system for three dimensional finite element models*, ACM SIGGRAPH Computer Graphics, 9 (1975), pp. 1–16.

[107] Y. SUDHAKAR, J. P. MOITINHO DE ALMEIDA, W. A. WALL, J. P. DE ALMEIDA, AND W. A. WALL, *An accurate, robust, and easy-to-implement method for integration over arbitrary polyhedra: Application to embedded interface methods*, Journal of Computational Physics, 273 (2014), pp. 393–415.

[108] K. SUGIHARA, *A Robust and Consistent Algorithm for Intersecting Convex Polyhedra*, Computer Graphics Forum, 13 (1994), pp. 45–54.

[109] I. E. SUTHERLAND AND G. W. HODGMAN, *Reentrant polygon clipping*, Communications of the ACM, 17 (1974), pp. 32–42.

[110] M. TAREK, *Preconditioners. Version 0.3.*, Oct. 2019. Available at `https://github.com/mohamed82008/Preconditioners.jl`.

[111] T. E. TEZDUYAR, S. SATHE, R. KEEDY, AND K. STEIN, *Space–time finite element techniques for computation of fluid–structure interactions*, Computer Methods in Applied Mechanics and Engineering, 195 (2006), pp. 2002–2027.

[112] THE CGAL PROJECT, *CGAL User and Reference Manual*, CGAL Editorial Board, 5.3 ed., 2021.

[113] D. THOMAS, M. SCOTT, J. EVANS, K. TEW, AND E. EVANS, *Bézier projection: A unified approach for local projection and quadrature-free refinement and coarsening of NURBS and t-splines with particular application to isogeometric design and analysis*, Computer Methods in Applied Mechanics and Engineering, 284 (2015), pp. 55–105.

[114] L. L. THOMPSON AND P. M. PINSKY, *A space-time finite element method for structural acoustics in infinite domains part 1: Formulation, stability and convergence*, Computer Methods in Applied Mechanics and Engineering, 132 (1996), pp. 195–227.

[115] F. VERDUGO, *GridapGmsh. Version 0.6.1.*, July 2022. Available at `https://github.com/gridap/GridapGmsh.jl`.

[116] F. VERDUGO AND S. BADIA, *The software design of gridap: A finite element package based on the julia JIT compiler*, Computer Physics Communications, 276 (2022), p. 108341.

[117] F. VERDUGO, A. F. MARTÍN, AND S. BADIA, *Distributed-memory parallelization of the aggregated unfitted finite element method*, Computer Methods in Applied Mechanics and Engineering, 357 (2019), p. 112583.

[118] F. VERDUGO, E. NEIVA, AND S. BADIA, *GridapEmbedded. Version 0.7.*, Oct. 2021. Available at `https://github.com/gridap/GridapEmbedded.jl`.

[119] X. WEI, B. MARUSSIG, P. ANTOLIN, AND A. BUFFA, *Immersed boundary-conformal isogeometric method for linear elliptic problems*, Computational Mechanics, 68 (2021), pp. 1385–1405.

[120] S. XIA AND X. QIAN, *Isogeometric analysis with bézier tetrahedra*, Computer Methods in Applied Mechanics and Engineering, 316 (2017), pp. 782–816.

[121] S. XIA AND X. QIAN, *Generating high-quality high-order parameterization for isogeometric analysis on triangulations*, Computer Methods in Applied Mechanics and Engineering, 338 (2018), pp. 1–26.

[122] X. XIAO, M. SABIN, AND F. CIRAK, *Interrogation of spline surfaces with application to isogeometric design and analysis of lattice-skin structures*, Computer Methods in Applied Mechanics and Engineering, 351 (2019), pp. 928–950.

[123] Q. ZHOU AND A. JACOBSON, *Thingi10K: A Dataset of 10,000 3D-Printing Models*, arXiv preprint arXiv:1605.04797, (2016).

[124] G. M. ZIEGLER, *Lectures on Polytopes*, Springer New York, 1995.