University of Genoa
Department of Mathematics
Ph.D. in Mathematics and Applications
Curriculum Mathematics and Applications

Analysis and Generation of Quality Polytopal Meshes

with Applications to the Virtual Element Method

Supervisors:
Dr. Silvia Biasotti
Dr. Michela Spagnuolo
Dr. Gianmarco Manzini

Ph.D. Student: Tommaso Sorgente
Freshman Number 3779635

August 2022

Abstract

This thesis explores the concept of the quality of a mesh, the latter being intended as the discretization of a two- or three- dimensional domain. The topic is interdisciplinary in nature, as meshes are massively used in several fields from both the geometry processing and the numerical analysis communities. The goal is to produce a mesh with good geometrical properties and the lowest possible number of elements, able to produce results in a target range of accuracy. In other words, a good quality mesh that is also cheap to handle, overcoming the typical trade-off between quality and computational cost.

To reach this goal, we first need to answer the question: *"How, and how much, does the accuracy of a numerical simulation or a scientific computation (e.g., rendering, printing, modeling operations) depend on the particular mesh adopted to model the problem? And which geometrical features of the mesh most influence the result?"* We present a comparative study of the different mesh types, mesh generation techniques, and mesh quality measures currently available in the literature related to both engineering and computer graphics applications. This analysis leads to the precise definition of the notion of quality for a mesh, in the particular context of numerical simulations of partial differential equations with the virtual element method, and the consequent construction of criteria to determine and optimize the quality of a given mesh.

Our main contribution consists in a new mesh quality indicator for polytopal meshes, able to predict the performance of the virtual element method over a particular mesh before running the simulation. Strictly related to this, we also define a quality agglomeration algorithm that optimizes the quality of a mesh by wisely agglomerating groups of neighboring elements. The accuracy and the reliability of both tools are thoroughly verified in a series of tests in different scenarios.

Dedication and Acknowledgements

I would like to dedicate this thesis to Drs. Silvia Biasotti and Michela Spagnuolo, which have been so much more than tutors to me. Silvia has supported me unconditionally since the day I first walked into her office, giving me the courage to carry on in all the natural ups and downs of my Ph.D. Michela has been a constant source of inspiration and motivation, she always motivated me to push further or to try again.

A critical acknowledgment is due to Dr. Marco Manzini, who widened my horizons introducing me to the VEM world and a rigorous research discipline. I thank all the members of the 'E. Magenes' institute of CNR-IMATI for the numerous direct or indirect contributions, with a particular mention to Dr. Daniela Cabiddu for the technical support. Thanks to the reviewers of this manuscript, Prof. Natarajan Sukumar and Prof. Marcel Campen, for their corrections and precious suggestions, and thanks to the Ph.D. course coordinator Prof. Stefano Vigni. Thanks to every author and co-author that worked with me on my publications: these collaborations meant a lot to me and have been milestones of my research activity. I would like to thank the ERC Advanced Grant CHANGE, which partially funded my research during all these years.

Finally, thanks to all the people who shared with me these last years and unconsciously contributed to this work with unexpected hints, unsolicited aids, and undeserved patience. I avoid citing anyone here explicitly because very few of you will ever read this, but each one of you knows to be on the list.

Table of Contents

List of Tables

Table **Page**

List of Figures

Figure **Page**

CHAPTER 1

Introduction

The concept of a mesh as a discretization of space was originally developed in the field of numerical analysis, where it could be used in association with computational methods to obtain numerical solutions of partial differential equations [Richardson, 1922]. Generating a suitable mesh was long considered to be a rather tedious exercise and a minor part of the effort involved in solving a numerical problem. However, mesh generation has steadily evolved into a discipline in its own right, drawing on ideas from other fields, and gradually developing a distinct identity. In particular, it is important to recognize the growing interest and contribution of the computer science community in mesh-related problems [Boissonnat, 1984, De Floriani et al., 1985, Bern and Eppstein, 1992, Edelsbrunner et al., 2001]. Not only has this synergy brought new ideas and ways of viewing mesh-related questions, but it has also opened up whole new areas of application including medical imaging and segmentation, computer graphics and animation, and data interpolation and compression [Baker, 2005]. Nowadays, there exist international conferences devoted to mesh generation, adaptation, and/or analysis (e.g., *International Meshing Roundtable*), and almost all conferences on computational methods have sessions that feature these topics (e.g., *Symposium on Geometry Processing*, or *Symposium on Computational Geometry*).

Over the last fifty years, computer simulations of Partial Differential Equations (PDEs) have dramatically increased their impact on research, design, and production, and are now an indispensable tool for modeling and analyzing a number of phenomena arising in fields as diverse as physics, engineering, biology, and medicine. The Finite Element Method (FEM) is by large the most popular technique for the computer-based simulation of PDEs and relies on suitable descriptions of geometrical entities, such as the computational domain and its properties. In this context, increasing attention is being paid to adaptivity and multilevel modeling, which, in

real-life applications, are probably the only ways to efficiently obtain a solution with an accuracy that is "certified" to be in a certain desired range. The discretization of the computational domain is a key research topic in the ERC project CHANGE (*New CHallenges for (adaptive) PDE solvers: the interplay of ANalysis and GEometry*), from the European Union's Horizon 2020 research and innovation program, which has funded this work. CHANGE aims at

> *"Developing innovative mathematical tools for numerically solving PDEs and for geometric modeling and processing, the final goal being the definition of a common framework where geometrical entities and simulation are coherently integrated and where adaptive methods can be used to guarantee optimal use of computer resources, from the geometric description to the simulation."*

In this sense, the development of geometric tools to construct, manipulate and refine meshes is the first step toward the design of such an innovative adaptive framework. The full exploitation of the potential of adaptivity has been until now hindered, especially in 3D, by the difficulty in performing mesh refinement and coarsening in the framework of tetrahedral and hexahedral meshes. Indeed, there is a mismatch between what the analysis needs the geometry to do and what the geometry can actually do. We believe that, in the medium term, this gap will be filled by discretizations over polytopal partitions (or polytopal meshes), where polytopal means polygonal or polyhedral, according to the context. The extreme flexibility of polytopal meshes simplifies every mesh refinement or coarsening operation, giving a real boost to the concept of adaptivity. Polytopal Element Methods (PEMs) can be considered as extensions of the classical FEMs over polytopal meshes, and enhance enormously the current finite element analysis based on mesh generation.

1.1 Motivations

The wide range of contexts and fields in which meshes are currently employed justifies the vast (and often confusing, or even contradictory) vocabulary around this topic. The mesh itself is often called with different terms (e.g., tessellation, discretization, or grid), which can be simple synonyms but may even include additional information on the mesh typology, structure, or connectivity. Even higher confusion can be found in the literature on concepts related to the *quality* of a mesh, as visible in the several surveys on mesh quality metrics [Liu and Joe, 1994, Knupp, 2001, Stimpson et al., 2007]. What constitutes a good quality mesh, and exactly how fine the mesh should be are questions that have been around since the first mesh was generated. Only partial answers to these questions are currently available, and a comprehensive understanding of how the accuracy of a PDE approximation depends on the mesh is still lacking.

At the same time, the breathtaking explosion of the computational power available in modern computers seen in the last years opens up new possibilities, unimaginable so far. Since one of the few concepts universally accepted about mesh quality is that the finer the mesh is,

the more accurate results it is likely to produce, the idea of generating finer and finer meshes, regardless of the notion of quality, becomes terribly tempting. However, there are still situations in which simply decreasing the size of the elements is not enough to guarantee accurate results. Just to give a trivial example, let us imagine covering a triangular domain with some squared cells: no matter how small the squares are, they will never exactly cover the domain. Decreasing the size of the cells surely leads to a better approximation, but there is no way we could ever reach the same result as if we used triangular cells instead. In a more realistic scenario, in presence of particularly complex domains or problems, we may need to exponentially increase the number of elements in order to slightly improve the accuracy of the approximation. In such contexts, it may be worth considering spending some time analyzing the domain and the problem, to generate meshes with higher quality (whatever it means) and fewer elements.

We face the problem of discretizing a domain with well-shaped cells, keeping their number as low as possible, instead of massively generating a huge number of tiny identical cells. The goal is to produce a good quality mesh that is also cheap to handle, overcoming the typical trade-off between quality and computational cost. The first consideration is that, if we want the mesh to somehow adapt to the domain or the solution, we cannot afford to use only one type of element, as noted in [Shepherd and Johnson, 2008] for hexahedral meshes. Therefore, we need to consider polytopal meshes, as opposed to pure meshes, that are typically triangular, quadrangular, tetrahedral, or hexahedral. The study of polytopal meshes is overwhelmingly emerging as a research trend: we now have methods for solving PDEs over meshes composed of generic polygons/polyhedra [Beirão da Veiga et al., 2013, Cangiani et al., 2014, Di Pietro and Droniou, 2019], methods for generating *hybrid* meshes, containing mainly one type of element and few different ones in strategic positions [Gao et al., 2017, Schneider et al., 2019], and polytopal mesh generators, based on suitably regularized Voronoi tessellations [Lévy and Liu, 2010]. However, the extremely interesting results obtained through the usage of hybrid meshes and Voronoi tessellations are just a taste of the potentialities of fully polytopal meshes. In particular, hybrid mesh generators openly aim at producing the smallest possible number of non-standard elements, thus not exploiting the potential flexibility of polytopal meshes to its full.

We are scratching the surface of a brand new field of opportunities. What is currently still lacking is a robust method to automatically generate a fully polytopal mesh tailored for a particular domain. The main difficulty in this is that when we are asked to fill a domain or to connect some points in it, with elements of any imaginable shape, we have extreme freedom and infinite possible choices. A possible approach could be to build an initial pure mesh and then agglomerate its elements to make it polytopal. This would give us a starting point and limit the number of choices, at the additional cost of building the pure mesh. In any case, what we need is a guide, a criterion that can help us in deciding the shape of the elements from time to time. We believe that this guide should be nothing else but the concept of mesh quality.

1.2 Contributions

In this work, we study in-depth the concept of *mesh quality*. After a general overview of the existing literature on this topic, we restrict our focus to numerical analysis and, in particular, to the Virtual Element Method (VEM) [Beirão da Veiga et al., 2013]. Each method has its pros and cons, but we believe that the method used for finding the solution to the problem should be taken into great account when defining the concept of quality. In other words, when deciding if a mesh is good or not, we should also consider who is going to work with that mesh, that is, the numerical method.

We define a *mesh quality indicator*, i.e., a mathematical tool to quantitatively measure the local and global quality of a mesh, specifically built for the VEM. In particular, this relationship lies in the fact that the indicator is directly deduced by the theoretical results on the convergence of the VEM. Besides the dependence on the numerical method, we want our indicator to be as general as possible: we define it for both two and three-dimensional domains and any kind of polygonal or polyhedral element (non-convex, non-star-shaped, with an undefined number of vertices, edges, and faces).

A second contribution is the development of a *quality agglomeration algorithm*. We define a polytopal mesh generation method, which takes an initial input mesh and agglomerates groups of neighboring elements in order to optimize the global mesh quality. The quality indicator drives the quality agglomeration algorithm. For each possible combination of adjacent elements, it indicates if its agglomeration will increase or decrease the global quality, and therefore if we should accept or reject it. Both the mesh quality indicator and the quality agglomeration algorithm undergo a thorough series of tests and experiments, to verify their accuracy and reliability. In particular, we exhibit their practical application in the context of Discrete Fracture Networks.

In addition, we describe in detail the routines that we have developed for the implementation of the above concepts, together with all the meshes built for the testing phase. In particular, significant work has been done on the implementation of an algorithm for computing the *kernel* of a polyhedron (i.e., the set of points from which the whole polyhedron is visible). The kernel is crucial information for the quality indicator, and a routine for the efficient computation of the kernel for high numbers of small polyhedra was not available in the literature.

1.3 Organization

The thesis is organized as follows. Chapter 2 introduces the reader to the wide world of two and three-dimensional meshes, discussing how to define and classify them, presenting the most developed methods for generating meshes of different types, and the most common tools for measuring their quality. In Chapter 3, we begin the presentation of the work published in [Sorgente et al., 2021b, Sorgente et al., 2022a, Sorgente et al., 2022b], defining the VEM

and analyzing its peculiarities and limitations. From such an analysis, we deduce the mesh quality indicators for two and three-dimensional meshes, specifically designed for this particular numerical method. The routines developed for implementing the indicators, published in [Sorgente et al., 2021a, Sorgente et al., 2022c], are reported in Appendix A. The quality indicators are then carefully tested throughout Chapter 4, where we appositely build a collection of two and three-dimensional meshes to investigate the correlation between the score provided by the indicator and the actual performance of the VEM. The algorithms developed for generating the two-dimensional datasets are reported in Appendix B. We then exploit the indicator to drive the quality agglomeration algorithm, which is able to navigate a mesh and agglomerate adjacent elements in order to optimize its global quality, and we exhibit several examples of the usage of such an algorithm. Last, Chapter 5 draws the conclusions of this work and addresses potential research directions for the future.

CHAPTER 2

Meshes

In this chapter, we introduce the concept of *mesh*, often also called *discretization*, *tessellation*, *computational domain*, or *grid*. A mesh is a discrete approximation of an object or a domain $\Omega \subset \mathbb{R}^d$, partitioned into a finite collection of disjoint cells. The mesh is called *planar* if $d = 2$, or *volumetric* if $d = 3$. The *cells* (or *elements*) of the mesh are d-dimensional subsets of $\mathbb{R}^d$ (polygons if $d = 2$, or polyhedra if $d = 3$) with no holes and no self-intersections. The boundary of a cell is composed of 2-dimensional *faces* (if $d = 3$), 1-dimensional *edges* and 0-dimensional *nodes* (or *vertices*), and two cells in a mesh can only share faces, edges, and nodes. Cells are defined in an abstract Euclidean space and then embedded into Ω (or *realized*) through proper mappings. There exists a half-way case, i.e., meshes in $\mathbb{R}^3$ made by polygonal cells, also known as *surface* meshes. We consider them together with planar meshes, as they are locally planar, and refer to both as "2D-meshes", as opposed to "3D-meshes", which refers to volumetric meshes. The foundations of the above concepts can be found in [Mäntylä, 1987].

Meshes are used as a tool for running simulations in a wide range of contexts, from numerical analysis to computer graphics, and come in a variety of types with different properties. The scope of this chapter is to present a general overview of the literature concerning meshes, mesh generation, and mesh quality. In Section 2.1, we propose a classification of the different mesh classes, which will be helpful throughout the work. We treat 2D-meshes and 3D-meshes in parallel throughout the chapter; distinctions will be made only at the bottom level. Section 2.2 is dedicated to mesh generation techniques currently available for each class. We include in our analysis meshes and methods from different application fields, from engineering to computer graphics. This may generate some confusion with notation, as mesh entities are often called with different names, according to the application context, but we will try our best to keep coherency and include all possible alternative terminologies from time to time. As these topics

are extremely wide, we align the treatment to the general focus of the thesis, which is the concept of mesh quality. In this sense, for instance, we do not include methods for generating meshes with curved cells, or cells with non-planar faces, as the quality analysis for such a type of meshes is still an ongoing research topic. Sections 2.1 and 2.2 lay the foundations for the introduction of the notions of mesh quality and mesh quality indicators. In Section 2.3 we analyze the concept of mesh quality, introducing local and global quality indicators for the different classes, and how they can be used in mesh quality optimization.

2.1 Mesh Classification

The mesh classification proposed in this section is based firstly on the typology of the cells, and secondly on the connectivity of the mesh nodes, namely, the mesh structure. We stress the fact that these two concepts are distinct and independent, i.e., a mesh with a certain structure can be generated with different types of cells, and vice-versa. We will see, however, that some topological constraints exist, and that some combinations are less common than others.

There is not a standard classification suited for every use one can have of a mesh, but rather, several alternative approaches to the problem according to the context. For instance, in the interesting paper [Ho-Le, 1988] the authors propose to classify meshes according to the chronological order in which the cells of the mesh are created: nodes first, cells first, or other hybrid approaches. Other works like [Baker, 2005] follow the historical order in which the several mesh generation techniques have been developed. Since the topic has become increasingly wide and complex in the last decades, some works only deal with particular types of meshes [Owen, 1998, Bommes et al., 2013b, Pietroni et al., 2022], focusing on specific application fields.

We anticipate the two following general definitions, which will be discussed more in detail in the following sections.

Definition 2.1 (Structured Mesh)**.** *A mesh is said to be structured if every internal node has the same (fixed) number of adjacent cells; otherwise, the mesh is called unstructured.*

Definition 2.2 (Conforming Mesh)**.** *A mesh is said to be conforming if two adjacent cells can only share a node, a whole edge, or a whole face. If the cells are allowed to overlap partially, the mesh is called non-conforming, and the intersections inside edges or faces are called T-junctions, or hanging nodes.*

2.1.1 Cell Typology

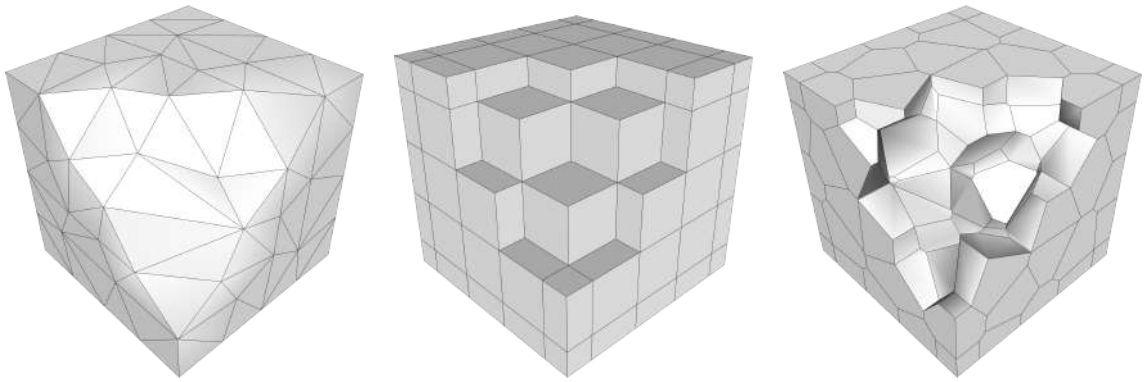
By cell typology we mean the number of edges (for 2D-meshes) or faces (for 3D-meshes) of a cell. This concept induces a straightforward classification of meshes into the following categories:

- *Simplicial Complexes* contain only simplices: triangular elements (in the following also noted *tris*) in 2D-meshes or tetrahedral elements (*tets*) in 3D-meshes. We call them, in short, tri-meshes (or triangulations) and tet-meshes;
- *Grid-Based Meshes* contain only quadrangular elements (*quads*) in 2D-meshes or hexahedral elements (*hexes*) in 3D-meshes. We call them, in short, quad-meshes and hex-meshes;
- *Polytopal Meshes* contain generic polytopes (*polys*): polygons in 2D-meshes or polyhedra in 3D-meshes. This class basically contains all the meshes which do not fall into the two above categories.

Visual examples of 3D-meshes belonging to these classes are given in Figure 2.1. We set out the following definition in order to further distinguish between the different types of polytopal meshes.

Definition 2.3 (Pure Mesh)**.** *A mesh is said to be pure if its cells are all of the same type.*

Simplicial complexes and grid-based meshes are always pure, while polytopal meshes may also be not.



(a) (b) (c)

Figure 2.1: Mesh classification according to the cell typology: (a) simplicial complex, (b) grid-based mesh, (c) polytopal mesh.

Simplicial Complexes The name *simplicial complex* is due to the fact that triangles and tetrahedra are formally *simplices*. From an algebraic topology point of view, we can reformulate the definition of mesh through the following notions.

Definition 2.4 (k-simplex)**.** *A $k-$simplex in $\mathbb{R}^n$, $0 \leq k \leq n$, is the $k-$dimensional polytope given by the convex hull of $k+1$ affinely independent points.*

In our case, mesh cells can be 3-simplices or 2-simplices, faces are 2-simplices, edges are 1-simplices, and nodes are 0-simplices, see Figure 2.2. A collection of 2-simplices (3-simplices)

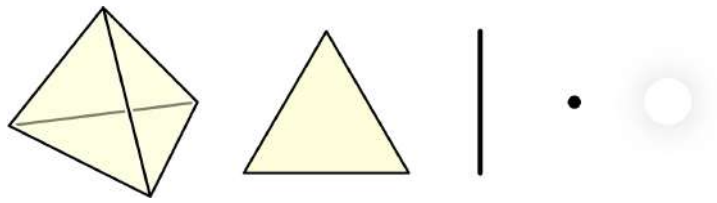


Figure 2.2: From left to right: a 3-simplex, a 2-simplex, a 1-simplex, a 0-simplex, and the (-1)-simplex.

defines a triangular (tetrahedral) mesh provided it satisfies some connectivity rules. In particular, we call such structures *simplicial complexes*.

Definition 2.5 (Simplicial Complex)**.** *A simplicial complex $\mathcal{K}$ is a finite set of simplices that satisfies:*

1. *every face of a simplex from $\mathcal{K}$ is also in $\mathcal{K}$;*
2. *the non-empty intersection of any two simplices $\sigma_1, \sigma_2 \in \mathcal{K}$ is a face of both σ_1 and σ_2.*

Conventionally, $\mathcal{K}$ also contains the empty set.

Each k-simplex has $k+1$ distinct vertices, and each set of $k+1$ vertices defines at most one k-simplex. To better adhere to the domain shape, Δ-complexes were defined. Intuitively, a Δ-complex is a topological space constructed by gluing simplices together along their boundaries. For more details on simplicial complexes refer to [Munkres, 2000, Biasotti et al., 2014, Hatcher, 2002].

Simplicial complexes have several good mathematical and geometrical properties, that make them preferable over other types of meshes in many contexts. First of all, simplicial complexes offer high flexibility in the domain representation, especially when arranged in unstructured combinations. They are therefore by far the most common choice for meshing particularly complicated geometric domains, where it could be difficult or impossible to compute a quad/hex-mesh without introducing low-quality elements or drastically increasing their number. Secondly, elements in a 2D simplicial complex and faces in a 3D simplicial complex are always planar, which is not always true for other element types. On one side, this may simplify operations on the mesh and can be a required property in some applications. In other contexts, however, non-planar faces could be exploited to better adapt to curved boundaries, e.g., in surface meshes. Last, every polygon can be easily triangulated and every polyhedron can be easily tetrahedrized (decomposed into a finite number of disjoint tetrahedra), and every point inside a simplex is expressible as a unique linear combination of its vertices, them being affinely independent. This local coordinate system, which is independent of the embedding, can be exploited in several geometric operations, for instance, intersection algorithms.

Algorithms for triangular and tetrahedral mesh generation are the most developed and robust among all the meshing techniques [Boissonnat, 1984, De Floriani et al., 1985, Edelsbrunner and Mücke, 1994, Boissonnat et al., 2000, Edelsbrunner et al., 2001, Meyer et al., 2003, Botsch et al., 2010]. A survey that illustrates the wide range of triangular and tetrahedral mesh generators was conducted by Owen [Owen, 1998] and described in the handbook of grid generation edited by Thompson et al. [Thompson et al., 1998]. More recent works can be found in the books of Frey [Frey and George, 2007], Lo [Lo, 2014] and Liseikin [Liseikin, 2017].

Grid-Based Meshes The term "grid" is due to the fact that the development of finite-difference techniques during the 1950s was confined to problems, typically two-dimensional, with simple boundary shapes. The first discretizations were therefore computed over planar rectangular regions, which can be naturally subdivided into smaller rectangles, and the resulting subdivision will look like a grid. The term was later extended to the three-dimensional equivalent of the quadrangular meshes, i.e. hexahedral discretizations. From the algebraic topology perspective, grid-based meshes can be defined as follows [Kovalevsky, 1989, Kong and Rosenfeld, 1989].

Definition 2.6 (k-cube)**.** *A $k-$cube in $\mathbb{R}^n$, $0 \leq k \leq n$, is a $k-$dimensional polytope that is combinatorially isomorphic to the standard cube $\square_k = [0,1]^k$.*

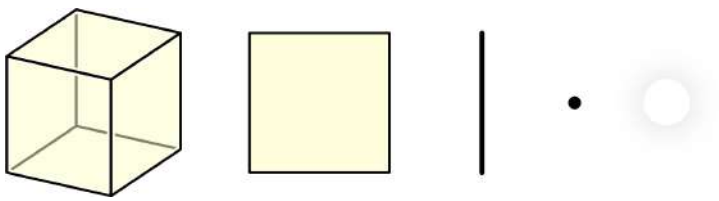


Figure 2.3: From left to right: a 3-cube, a 2-cube, a 1-cube, a 0-cube, and the (-1)-cube.

In our case, mesh cells can be 3-cubes or 2-cubes, faces are 2-cubes, edges are 1-cubes, and nodes are 0-cubes, see Figure 2.3.

Definition 2.7 (Cubical Complex)**.** *A cubical complex $\mathcal{K}$ is a finite set of k-cubes that satisfies:*

1. *every face of a k-cube from $\mathcal{K}$ is also in $\mathcal{K}$;*
2. *the non-empty intersection of any two k-cubes $\sigma_1, \sigma_2 \in \mathcal{K}$ is a face of both σ_1 and σ_2.*

Conventionally, $\mathcal{K}$ also contains the empty set.

The concept of cubical complex can be generalized to $\square$-complexes exactly like Δ-complexes generalize simplicial complexes, by simply replacing simplices with cubes. Despite this formal definition, in the rest of the work, we will address cubical complexes with the more familiar

term of grid-based meshes. We report, however, that it is not uncommon to find hexahedral meshes treated as polytopal meshes in the literature, due to the possible presence of non-planar faces.

Grid-based meshes are preferred in many applications for several reasons. Most geometry has two dominant local directions, typically associated with either principal curvature directions, or local sharp features. Quadrangular cells can be naturally aligned with such directions, while the use of triangular cells necessitates an arbitrary choice of a third edge direction. Analogous considerations can be done for hexahedral and tetrahedral cells. In computer graphics applications, the cell alignment to given directions (for instance, the dominant surface directions) is useful to capture shape features, as well as the semantics of the modeled objects, especially if they need to be segmented or animated [Bommes et al., 2013b]. Grid-based meshes are also well-suited to support tensor-product structures (e.g., tensor-product B-splines, or Catmull-Clark surfaces), even if extensions can be found also for tri- and tet-meshes. Moreover, quads and hexes allow for very large aspect ratios, while the same aspect ratio in tris and tets may lead to a very large *skewness* (see Section 2.3.1), which degrades the accuracy and convergence of the simulation. Grid-based meshes are also more economic than simplicial complexes with respect to the number of cells needed to tessellate a domain, with prescribed cell size. Since every quad can be split into two triangles, and every hex into six tets, the number of edges, faces, and cells (and therefore of degrees of freedom, in finite element analysis) in a grid-based mesh is significantly lower than in a simplicial complex. In the context of numerical simulations, linear hex-meshes are often superior to linear tet-meshes in terms of convergence time and accuracy of the simulation [Benzley et al., 1995, Tadepalli et al., 2011]. When it comes to quadratic elements, however, it has been shown that tet-meshes produce a given target error in less time with respect to hex-meshes [Schneider et al., 2022].

Grid-based meshes have been used for many years as the computational domain to solve partial differential equations (PDEs) that are relevant for the automobile, naval, aerospace, medical, and geological industries to name a few, and are at the core of prominent software tools used by such industries, such as [MeshGems, 2020, CoreForm, 2021, Cubit, 2021]. However, despite the huge effort that various scientific and industrial communities have spent so far (see the recent surveys [Bommes et al., 2013b, Pietroni et al., 2022]), the computation of high-quality quadrilateral and hexahedral meshes conforming to (or approximating) a target geometry remains a challenge with various open aspects for which no fully satisfactory solutions have been provided yet. Some of the known methods are extremely robust and scale well on complex geometries; some others produce high-quality meshes; some others are fully automatic. But no known method successfully combines all these properties into a single product.

Polytopal Meshes The definition of polytopal meshes is built around the concept of *polytope* (geometrical figure bounded by portions of lines, planes, or hyperplanes [Coxeter, 1973]).

Definition 2.8 (Polytopal Complex)**.** *A polytopal complex $\mathcal{K}$ is a finite set of polytopes in some Euclidean space $\mathbb{R}^n$ that satisfies:*

1. *every face of a polytope from $\mathcal{K}$ is also in $\mathcal{K}$;*
2. *the non-empty intersection of any two polytopes $\sigma_1, \sigma_2 \in \mathcal{K}$ is a face of both σ_1 and σ_2.*

Conventionally, $\mathcal{K}$ also contains the empty set.

The most general notion of polytopal complex is called *closure finite complexes with weak topology*, or more simply CW-complexes [Whitehead, 1949]. Technically, simplicial and cubical complexes are as well polytopal complexes, because tris, tets, quads, and hexes are all polytopes. Indeed, some authors refer to Δ-complexes as simplicial CW-complexes. We conventionally define the class of polytopal meshes as the collection of all the polytopal complexes that do not fall in the two previous categories. Examples of polytopal meshes are therefore hybrid tri/quad-meshes and tet/hex meshes, quad/hex-dominant meshes, pure hexagonal meshes, or meshes with mixed generic polytopal cells.

The main advantage of polytopal meshes is that they allow achieving a flexibility level analogous to that of simplicial complexes, but using a significantly smaller number of cells. This can become crucial in the treatment of complex geometries, as it allows to incorporate complex features at different scales without triggering mesh refinement. Moreover, not having to care about cell typology drastically simplifies refinement and coarsening operators, and automatically includes T-junctions [Myles et al., 2010]. A practical example of this property is given in Figure 2.2: according to the interpretation of the rightmost cell, the mesh can be conforming or not. If we need a pure quad-mesh, we are forced to insert a T-junction in correspondence of the red node, while, if we accept to have a polygonal mesh, we can simply have a pentagon with two adjacent quads.

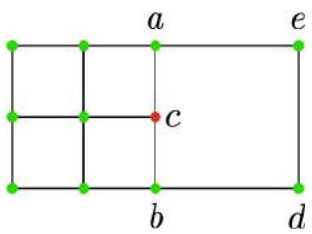


Figure 2.4: If the rightmost cell is defined as *acbde*, then the mesh is conforming. If it is defined as *abde* instead, the mesh is non-conforming, and the node c is a T-junction.

The origins of the concept of polytopal meshes can be traced back to two independent works by Nakahashi [Nakahashi, 1987] and Weatherill [Weatherill, 1988b] about *hybrid* meshes. They observed that the real advantages of grid-based meshes coincide with the disadvantages of simplicial complexes. Promoters of grid-based meshes highlight the efficiency and accuracy that is attained through the employment of regularly arranged quads and hexes. Supporters of

simplicial complexes emphasize the geometric flexibility and suitability for adaptation inherent to the use of irregularly connected tris and tets. Hybrid meshes contain both triangular and quadrangular (tetrahedral and hexahedral) cells, in a combination of the two approaches that try to capitalize on the merits of both of them. Hybrid mesh generators produce a structured quad/hex-mesh that covers the majority of the domain, and in correspondence of boundaries or singularities, few tris or tets are inserted. Some mesh generators also accept a limited number of other notable cell types, such as pyramids and prisms [Geuzaine and Remacle, 2009]. For this reason, such tessellations are often called *quad/hex-dominant*. In the last years, people started to think outside the strict "tet-hex" binomial and began to produce hex-dominant meshes, that, besides a high percentage of hexes, contain some generic polyhedra [Gao et al., 2017]. Meanwhile, thanks to the development of extremely general numerical methods, fully polygonal and polyhedral meshes have become increasingly studied. While finite elements methods were only designed to work on standard elements like tris/tets or quads/hexes [Ciarlet, 2002], recent literature widely explored Polytopal Element Methods (PEMs), i.e., methods for the numerical solution of PDEs based on polytopal grids. The PEM family already counts quite a few methods, such as Mimetic Finite Differences [Brezzi et al., 2005, Beirão da Veiga et al., 2014], Virtual Elements Method (VEM) [Beirão da Veiga et al., 2013], Discontinuous Galerkin-Finite Element Method (DG-FEM) [Cangiani et al., 2014, Antonietti et al., 2016], Hybridizable and Hybrid High-Order Methods [Cockburn et al., 2008, Di Pietro and Droniou, 2019], Weak Galerkin Method [Wang and Ye, 2013], BEM-based FEM [Rjasanow and Weiser, 2012], Poly-Spline FEM [Schneider et al., 2019], and Polygonal FEM [Sukumar and Tabarraei, 2004] to name a few.

Last, a significant example of pure polytopal meshes is represented by hexagonal meshes, also called honeycombs, that have been studied, among others, in the context of communication algorithms [Stojmenovic, 1997].

2.1.2 Mesh Structure

The structure of the mesh reflects the way in which its cells are organized. Recalling Definition 2.1, in a structured mesh, every internal node has the same number of adjacent cells. The number of adjacent cells (equivalently, incident edges in 2D, or incident faces in 3D) is called the *valence* of the node. Structured meshes are typically pure (see Definition 2.3), and the fixed valence is commonly assumed to be 4 for quad-meshes and tet-meshes, and 6 for tri-meshes and hex-meshes. In pure structured meshes, the local organization of the nodes, and the type of the cells, do not depend on their position, but are constant across the domain, see Figure 2.5.

As soon as we relax the connectivity constraint, *singular* nodes begin to appear, whose valence is different from that of the majority of the other nodes (called *regular*). When the connection of the nodes varies from point to point, the mesh is generically called unstructured, even if we can further distinguish between different types of unstructured meshes, see for instance [Alliez et al., 2008]. A very general consideration is that, historically, structured meshes

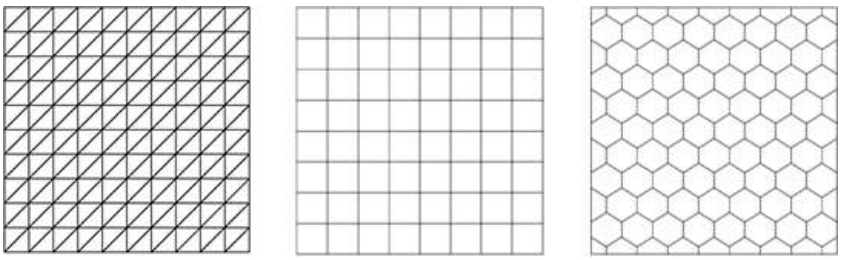


Figure 2.5: Examples of pure structured 2D-meshes, with different cell types.

have been associated with *finite difference* methods, since a regular lattice structure provides easy identification of neighboring points to be used in the representation of derivatives. On the other side, the *finite element* approach has always been, by the nature of its construction on discrete cells of general shape, considered well-suited for unstructured meshes. Indeed, a network of irregular cells can be made to fill any arbitrarily-shaped region, and the representation of the finite element functions is built independently on each cell, not across them.

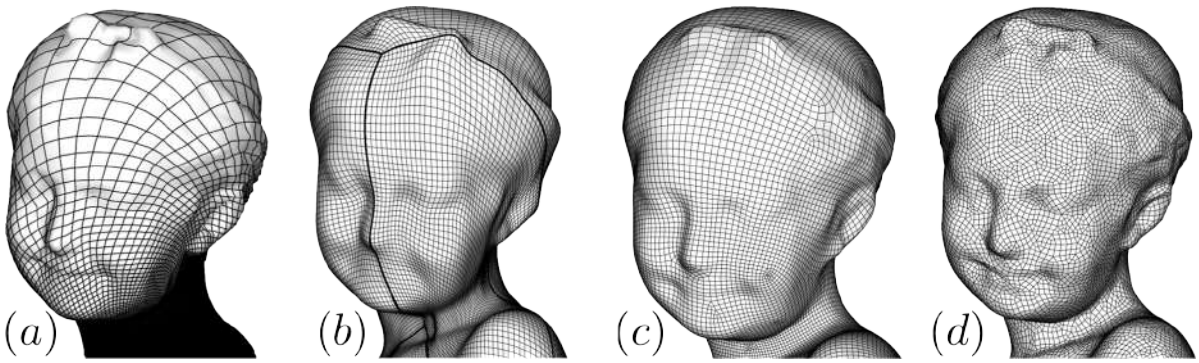


Figure 2.6: Quad-mesh structure classification according to [Bommes et al., 2013b]: (a) structured, (b) block-structured, (c) semi-structured, (d) unstructured.

The classification presented in [Bommes et al., 2013b] for quadrangular meshes can be extended to the general case, see Figure 2.6:

- A mesh is *structured* if its nodes are all regular. Structured meshes are suitable, for instance, to model 2-manifolds of genus 0 and 1 only (a mesh with toroidal topology can be obtained by identifying the opposite sides of a structured mesh, without introducing singular nodes).

- A mesh is *block-structured* if it is obtained by gluing, in a conforming way (see Definition 2.2), several structured arrays of adjacent elements, called blocks. In 2D-meshes, all nodes that are internal to blocks or lie along their boundary edges are regular, while only nodes that lie at corners of blocks may possibly be singular. For 3D-meshes, regularity is

required also for nodes internal to block boundary facets, and nodes at block edges may be singular.

- A mesh is *semi-structured* if most of its nodes are regular. All block-structured meshes are semi-structured, but not every semi-structured mesh can be partitioned into a small number of blocks.

- A mesh is completely *unstructured* if a large fraction of its nodes is singular. Note that block-structured and semi-structured meshes are technically unstructured as well, but have a limited number of singular nodes. We reserve the term "unstructured" for meshes whose nodes are mainly singular.

As explained in [Pietroni et al., 2022], where a similar classification is proposed for hexahedral meshes, *"the whole taxonomy can be understood in terms of the ratio between the number of singular nodes and the total amount of mesh nodes r_V, and the ratio between the number of blocks and the total number of mesh elements r_B"*. In fact, when both r_V and r_B are high, the mesh is unstructured; if r_V is low and r_B is high, the mesh is semi-structured; if both r_V and r_B are low the mesh is block-structured; if the number of blocks is exactly 1, the mesh is structured. Providing actual thresholds to precisely define what high and low mean, however, is an application-dependent matter. In the literature, the term structured is sometimes replaced by the term *regular*, and semi-structured meshes are also called *valence semi-structured* (or *valence semi-regular*). From an applicative perspective, there is a substantial difference between these three classes and there exists a variety of structure enhancement algorithms that are specifically designed to improve mesh regularity.

Given a mesh M, the total numbers of cells E, faces F, edges E, and nodes V in M are regulated through the Euler formula [Richeson, 2012]:

$$\chi(M) = \begin{cases} V - E + C & \text{if } M \text{ is a 2D-mesh;} \\ V - E + F - C & \text{if } M \text{ is a 3D-mesh.} \end{cases} \tag{2.1}$$

The quantity $\chi(M)$, called *Euler characteristic*, is defined by the alternating sum of the Betti numbers $\chi(M) := \sum_{i=0}^{\infty}(-1)^i B_i$ [Munkres, 2000]. In the case of 2D-meshes, $\chi(M)$ is a topological invariant (i.e., it only depends on the topology of M and not on the particular discretization), and we have $\chi(M) = 2 - 2g - b$, where g is the genus of M and b the number of boundary components [Biasotti et al., 2014]. Therefore, $b = 0$ for any closed surface mesh, and $b = 1$ in the case of planar, or open surface meshes. When applying the above formulas to meshes with small genus and significant numbers of cells, faces, edges, and nodes, the approximation $\chi(M) \approx 0$ is typically assumed for both 2D-meshes and 3D-meshes.

When the number of singular nodes in a mesh is limited, we can make assumptions about the average node valence. In a tri-mesh with fixed valence 6, we have 3 nodes in each cell and

approximately 6 cells incident to each node, hence it holds $3C \approx 6V$, and $C \approx 2V$. Moreover, we have exactly 2 cells around each edge and 3 edges in each cell, therefore $2E = 3C \approx 6V$, and $E \approx 3V$. Formula (2.1) is satisfied because $V - E + C \approx V - 3V + 2V = 0$, and we discover that the ratio between the numbers of edges, cells, and nodes is approximately $E : C : V = 3 : 2 : 1$ (if the genus of the domain is reasonably low). This result holds for structured, block-structured and semi-structured tri-meshes, even though the estimate becomes less precise as the number of singular nodes increases. The same reasoning for quad-meshes leads to $C \approx V$ and $E = 2C \approx 2V$ (in accordance to (2.1)). The ratio in a quad-mesh with low genus is $E : C : V = 2 : 1 : 1$, and this gives a hint on the smaller number of edges and cells required by a structured quad-mesh with respect to a structured tri-mesh with the same number of nodes. In 3D-meshes, we can only obtain relationships between cells and nodes, or cells and faces, because we have no information on the valence of the edges, nor on the number of faces incident to a node. For a tet-mesh with fixed valence 4 we obtain $C \approx V$ and $F = 2C \approx 2V$. However, we can substitute these relations into equation (2.1) and isolate the E term, to get $E \approx 2V$. The ratio $C : F : E : V$ for tet-meshes is therefore $1 : 2 : 2 : 1$, and a similar count for hex-meshes leads to $0.5 : 1.5 : 2 : 1$. As for the 2D case, structured hex-meshes require fewer elements, faces, and edges than structured tet-meshes.

2.2 Mesh Generation

This section presents an overview of the different existing techniques for the generation of two and three-dimensional tessellations. In Table 2.1 we summarize the combinations between cell typology and mesh structure obtainable with the most common generation methods. Simplicial complexes can be arranged in any type of structure, even though they are more commonly found in unstructured meshes. Grid-based meshes are preferably generated in structured (or almost structured) configurations, due to their intrinsic geometrical properties. There is one major exception, given by quad/octree grid generation methods [Yerry and Shephard, 1984, Shephard and Georges, 1991]. These methods produce grids with a high number of singular nodes but are still used because, at least, they guarantee the construction of topologically-valid meshes. On the other hand, polytopal meshes containing generic-shaped elements cannot present fixed patterns in their structure and are intrinsically unstructured. The only known example of structured polytopal mesh is represented by hexagonal meshes, while some methods for the generation of semi-structured meshes produce hybrid tessellations.

2.2.1 Structured Meshes

An easy way to visualize the correspondence of a structured curvilinear grid in the physical field with a logically rectangular grid in the computational field is through the “sponge” analogy [Thompson, 1982]. Consider a rectangular sponge within which an equally spaced Cartesian grid

Table 2.1: Most common combinations between cell typology and mesh structure.

	Simplicial	Grid-Based	Polytopal
Structured	X	X	
Block-Structured	X	X	
Semi-Structured	X	X	X
Unstructured	X		X

has been drawn, Figure 2.7(a). Now wrap the sponge around a circular cylinder and connect the two ends of the sponge together, as shown in Figure 2.7(b). The original Cartesian grid inside the sponge has now become a curvilinear grid fitted to the cylinder, but the rectangular logical form of the grid lattice is still preserved. Such a sponge could just as well be enclosing a cylinder of non-circular cross-section, regardless of the cross-sectional shape, or it could be expanded and compressed to fill any region (e.g. a sphere), again producing a curvilinear grid filling the region and having the same correspondence to a logically rectangular grid.

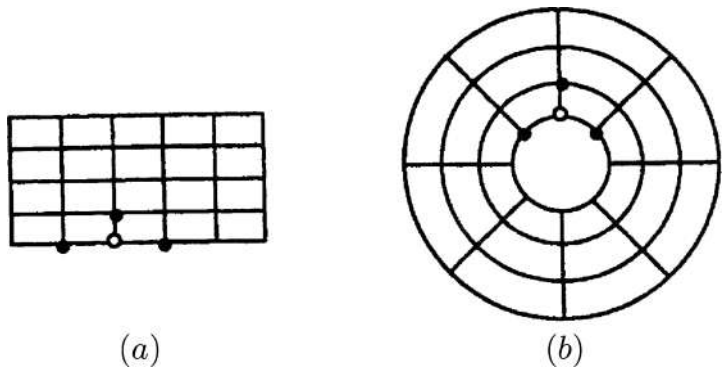


Figure 2.7: Rectangular sponge (a) wrapped around a circular cylinder (b).

Structured meshes are typically generated through a mapping approach [Mäntylä, 1987]. The basic idea, common to all structured mesh generation methods, consists of meshing a reference $n-$dimensional domain Ξ^n of a simple shape and mapping this mesh to a physical $n-$dimensional domain or geometry X^n through some transformation $\mathbf{x}(\boldsymbol{\xi}) : \Xi^n \to X^n$, $\boldsymbol{\xi} = (\xi_1, \ldots, \xi_n)$. Even though there are in principle no limitations on the shape of Ξ^n, it is very common to use a rectangular or hexahedral reference domain, and this choice leads to the generation of structured quad and hex-meshes. It is a standard requirement for the resulting mesh to be conformal to the boundary of the domain. A boundary-conforming coordinate grid in the region X^n is commonly generated first on the boundary of X^n with a map $\partial\mathbf{x}(\boldsymbol{\xi}) : \partial\Xi^n \to \partial X^n$, and then successively extended from the boundary to the interior of X^n. Since this process is analogous to the interpolation of a function from a boundary or to the solution of a differential boundary value problem, we can divide the mapping methods into three basic groups [Liseikin, 2017]:

- *algebraic methods*, where points of the reference domain are directly mapped onto the

physical domain through various forms of transfinite interpolation [Gordon and Hall, 1973, Gordon and Thiel, 1982] or special sweeping [Shepherd et al., 2000] or dragging [Park and Washam, 1979] functions;

- *differential methods*, based on the solution of elliptic [Thompson, 1982], parabolic [Nakamura, 1982], or hyperbolic [Cordova and Barth, 1988] equations to generate contours where points can be located;

- *variational methods*, based on the optimization of grid quality properties like smoothness, departure from orthogonality or conformality, cell skewness, and cell volume [Warsi and Thompson, 1990].

A visual overview of the differences between meshes resulting from the different methods is given in Figure 2.8. Basically, the algebraic generation systems are faster, but meshes generated from partial differential equations are generally smoother. The hyperbolic generation systems are faster than the elliptic systems but are more limited in the geometries that can be treated. The elliptic systems are the most generally applicable with complicated boundary configurations, but transfinite interpolation is also effective in a multi-block framework (Section 2.2.2). Variational methods are particularly powerful in generating high-quality meshes, with the disadvantage that some effort is required to solve the equations related to the mesh [Thompson et al., 1985, Thompson et al., 1998].

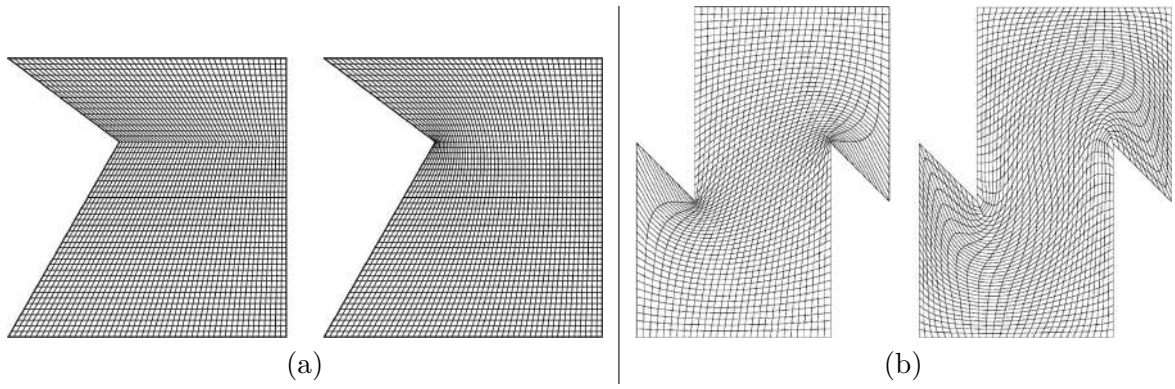


Figure 2.8: Meshing of a two-dimensional domains [Liseikin, 2017]: (a) on the left a quad-mesh obtained by means of transfinite interpolation; on the right a quad-mesh obtained by means of elliptic equations. (b) Variational meshing: on the left an harmonic quad-mesh; on the right a quasi-isometric quad-mesh.

Even though less used, there exists a whole class of triangular and tetrahedral structured meshes. The mapping methods presented above can be easily extended to generate triangular or tetrahedral grids by mapping a standard triangular or tetrahedral domain, respectively (Figure 2.9).

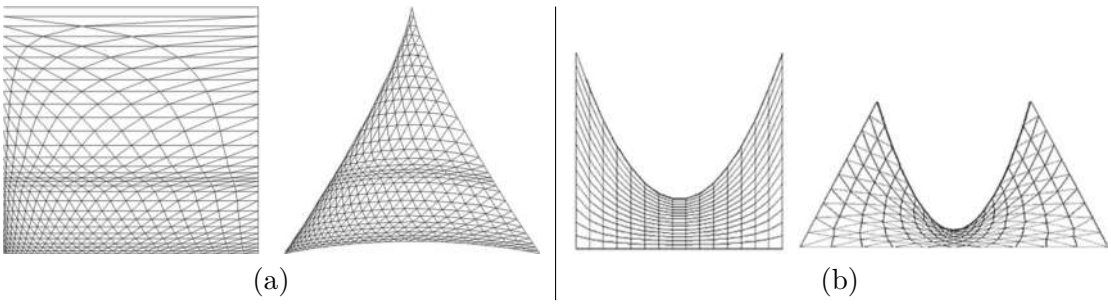


Figure 2.9: (a) Example of a structured triangular mesh (right) and the corresponding mesh on the parametric domain (left) generated by the algebraic method; (b) Quadrilateral and triangular structured meshes generated with the hyperbolic method [Liseikin, 2017].

Structured meshing is particularly effective in generating a large number of elements over regular domains, where the quality and the speed of mesh generation are guaranteed. A structured mesh requires significantly less memory -say a factor of three less- than an unstructured mesh with the same number of elements because array storage can define neighbor connectivity implicitly [Bern and Plassmann, 2000]. In fact, it does not need the storage of any connectivity table, as the mesh is defined according to a specified pattern. The storage and computational costs for different mesh representations are analyzed in [Garimella, 2002], where the authors compute the relative storage cost (i.e., divided by the number of nodes) of a mesh, in terms of the number of entities M_e (cells, faces, edges, and nodes) and the number of connections between them M_c. It turns out that, with the most efficient mesh representations, an unstructured tet-mesh occupies $25M_e + 119M_c$, while a structured hex-mesh only takes $8M_e + 42M_c$, thus obtaining the factor 3 stated above. A more detailed discussion on mesh representation can be found in [Beall and Shephard, 1997, Cignoni et al., 2003]. A structured mesh can also save time: to access neighboring cells when computing a finite-difference stencil, the software simply increments or decrements array indices, as data for elements that are close geometrically is also close in memory by design. Compilers produce quite efficient code for these operations; in particular, they can optimize the code for vector machines. However, this approach is rather restrictive for mesh generation in several ways: (i) the number of divisions has to be equal on opposite edges or faces; (ii) the elements may be distorted due to the mapping; (iii) a progressive change of element size is less flexible as element connections are fixed; and (iv) element distortion at the corner and in the interior of the physical domain cannot be predicted before the application of the mapping.

2.2.2 Block-Structured Meshes

Recalling the sponge analogy, it is not hard to see that, for some shapes with particularly complex boundary, the sponge may have to be greatly deformed. In such cases, the curvilinear grid of Figure 2.7(b) could become so highly skewed and twisted to be unusable. One solution

to this problem is to use not one, but rather a group of sponges to fill the physical field: each sponge has its own logically rectangular grid that deforms into a curvilinear grid when the sponge is put in place in the field.

The block-structured (or multi-block) strategy [Lee et al., 1980, Armstrong et al., 2015] is based on this multiple-sponge analogy, with the physical field being filled with a group of grid blocks with correspondence of grid lines, and in fact complete continuity, across the interfaces between blocks. Grids of this kind can thus be considered as locally structured at the level of an individual block, but globally unstructured when viewed as a collection of blocks (Figure 2.10). All nodes that are internal to blocks or lie along their boundary edges are regular, while only nodes that lie at corners of blocks may possibly be singular. This procedure is equivalent to the extraction of a parametrization of the domain [Botsch et al., 2010]. In several works on block-structured mesh generation, however, the parametrization is not explicitly defined.

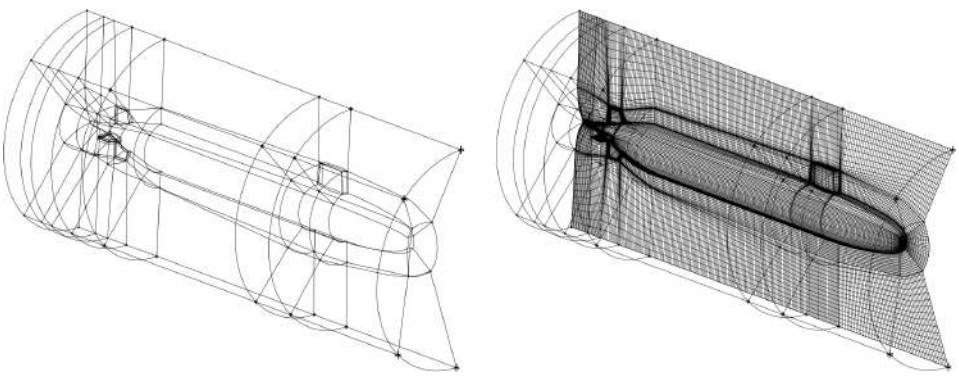


Figure 2.10: A block-structured hexahedral mesh of a submarine, showing the block structure and a vertical slice through the mesh [Bern and Plassmann, 2000]

The generation problem is split into two independent steps: first, the domain is divided without holes or overlaps into a few contiguous sub-domains. Then, a separate structured mesh is generated in each block with any of the methods from Section 2.2.1, allowing the most appropriate mesh configuration to be used in each region. The most common techniques for automatically splitting the domain into blocks typically rely on the geometry or the topology of the shape. The key idea is to divide the domain in correspondence to shape features or topological changes. For two-dimensional domains, the decomposition can consist of a clustering of the nodes based on their normal orientation [Boier-Martin et al., 2004]. A successful class of methods for quad-mesh generation are those based on an *integer-grid map* [Tong et al., 2006, Kälberer et al., 2007, Bommes et al., 2013a]. Another strategy is to induce the decomposition through a Morse-Smale complex [Dong et al., 2006], a Reeb graph [Sorgente et al., 2018], or discrete Ricci flow [Chen et al., 2019]. The polycube strategy (Section 2.2.3) can be applied to a surface model in order to generate a quadrangular block-decomposition. In *multi-charts decompositions*, cells are clustered into "charts" (just another word for "blocks") according to their distance from a

set of seeds on the surface [Carr et al., 2006, Sander et al., 2003]. For three dimensional domains instead, more advanced topological tools are needed, such as medial descriptors: geodesic functions [Dey and Sun, 2006], skeletons [Livesu et al., 2016], foliations [Campen et al., 2016], or the recent decomposition called *motorcycle complex* [Brückler et al., 2022].

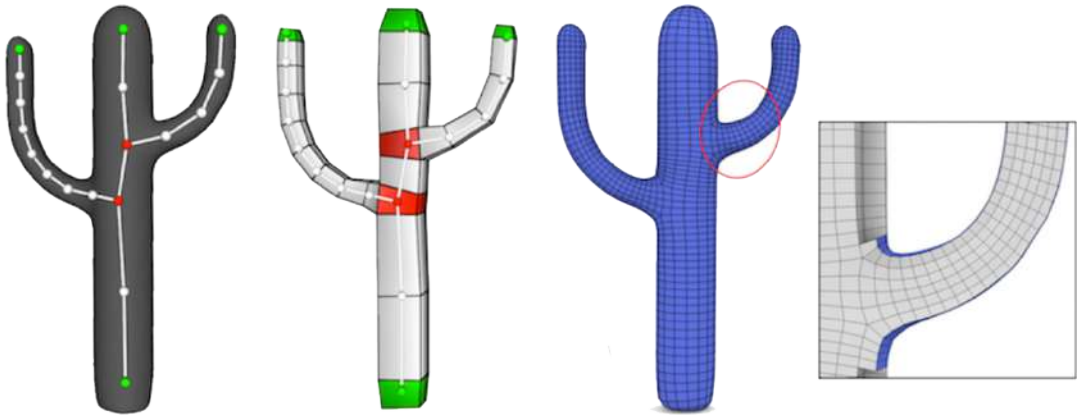


Figure 2.11: Skeleton driven hex-meshing [Livesu et al., 2016] starts from an input surface mesh and line skeleton (left), around which a tubular structure composed of hexahedral blocks is initialized (middle left). Refining this structure and projecting it on the target surface yields a hexahedral mesh (middle right) where the distribution of the mesh elements aligns with the skeleton guiding curves (right closeup).

Block-structured meshes are considerably more flexible than the structured ones in handling complex geometries. Since these meshes retain the simple regular connectivity pattern of a structured mesh on a local level, they maintain nearly the same compatibility with efficient finite-difference or finite-volume algorithms as structured meshes. The few singular nodes define a coarse block layout that can be exploited by dedicated data structures for cheaper storage and fast querying [Tautges, 2004], and it is also useful in a variety of applications that exploit the tensor product structure of its elements (e.g., IGA [Hughes et al., 2005]). This type of mesh is particularly suited if the physical problem is heterogeneous relative to some of the physical quantities, so that different mathematical models are required in different zones of the domain to adequately describe the physical phenomena. They are also particularly indicated when the solution to the problem behaves non-uniformly, as zones of smooth and rapid variation of different scales may exist. Unfortunately, the mentioned methods for the generation of block-structured meshes do not always succeed in generating topologically valid meshes. A fair amount of user interaction may be required for particularly complex models.

2.2.3 Semi-Structured Meshes

A mesh is semi-structured (or valence semi-structured) if most of its nodes are regular. In these cases, nodes are not connected in a way that induces a coarse block decomposition into a few

regular blocks. All block-structured meshes are semi-structured, but not every semi-structured mesh can be partitioned into a small number of blocks. The distinction is important as the difference between these two classes can be really dramatic and has a significant impact on possible applications. Differentiating these two classes of meshes allows us to differentiate more precisely the algorithms that aim at producing meshes with a block structure, from those algorithms that minimize the number of singular nodes only.

Advancing Front Methods A common approach to the generation of semi-structured meshes is through advancing front methods, also called moving or marching front. They allow the generation of triangular and tetrahedral meshes [Lo, 1985, Lo, 2013] as well as quadrangular (also referred to with the term *paving* [Blacker and Stephenson, 1991]) and hexahedral ones (also referred to with the term *plastering* [Staten et al., 2010]).

Such techniques define a mesh on the domain in the form of advancing layers, starting from the boundary and proceeding until the whole region has been covered with grid cells. The region separating the part of the domain already meshed from those that are still unmeshed is referred to as a front (Figure 2.12(a)). The name of this class of methods refers to a strategy that consists of creating the mesh sequentially, element by element, creating new points, and connecting them with previously created elements, thus advancing into as-yet-unmeshed space and sweeping a front across the domain. Advancing-front techniques need some discretization of the boundaries of the geometry, which constitutes the initial front which remains intact throughout the mesh generation process. In two dimensions, the boundary discretization consists of the initial placement of nodes offsetting from a boundary edge, while in three dimensions it can be a triangular or quadrangular mesh. New elements are built by connecting the nodes of a front face to either other points on the front or some inserted new points. A new mesh point is placed at a position that is determined to result in an element with prescribed optimal quality features. For instance, an advancing front approach can be combined with a Delaunay criterion (Section 2.2.4) which drives the insertion of new points [Marcum and Weatherill, 1995]. The process stops when the front is empty, i.e., when the domain is entirely meshed.

The advancing-front approaches offer the advantages of high-quality point placement and integrity of the boundary. The efficiency of the marching process largely depends on the arrangement of mesh points in the front, especially at sharp corners. A particular difficulty of this method occurs in the closing stages when the front is collapsing on itself and the last vestiges of empty space are replaced by new elements. In such cases, poor elements have to be used to fill up the interior just for building up a mesh of correct topological structure, and sometimes existing elements already placed have to be removed or modified from time to time to cater to the formation of new elements (Figure 2.12(c)). In practice, there is rarely any difficulty in completing the process for a planar domain. In three dimensions, however, the remaining region of space can have an extremely complicated shape, which may not yield an

acceptable covering by tets or hexes, thus preventing the tessellation from filling the entire region to be meshed.

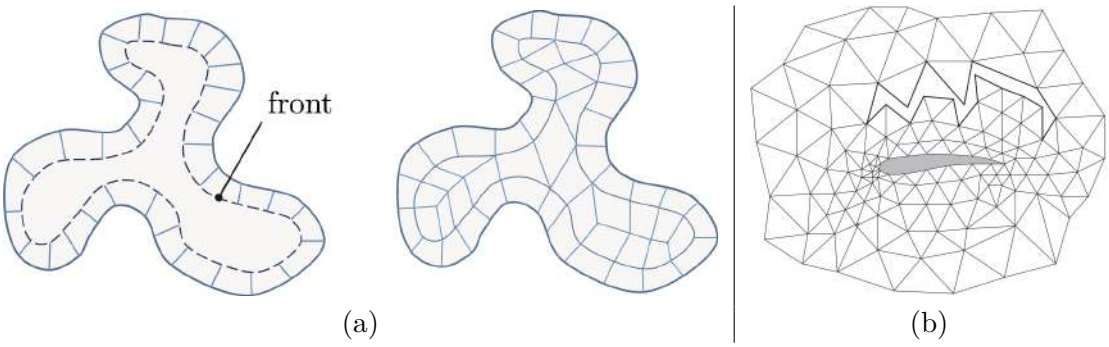


Figure 2.12: Planar meshes generated by the advancing front method: (a) initial and final stage for a quadrangular mesh: deformed quads are found at the interior of the domain; (b) closing stage for a triangular mesh [Baker, 2005].

Field-Guided Methods A promising research direction for semi-structured mesh generation is offered by field-guided methods, presented in Figure 2.13. They are characterized by explicit control over the local properties of elements in the mesh by means of some guiding fields. Due to their formulation, they are specific to grid-based mesh generation.

For two-dimensional grids, the most interesting local properties are the orientation and the size of elements, which can be specified by a cross (or frame) field which smoothly varies over the entire surface [Alliez et al., 2003, Bommes et al., 2009, Ray et al., 2009, Zhang et al., 2010]. A cross-field exhibits the same types of singularities that can be observed in a quad-meshes and consequently the generation of a highly regular quad-mesh is strongly related to the generation of a cross-field with few singular points. In the three-dimensional case, a frame consists of three linearly independent vectors that represent a parallelepiped, i.e., the orientation and shape of a linearly deformed cube. In a typical hex-generation algorithm, the initial step is the computation and optimization of a boundary-aligned frame field [Palmer et al., 2020]. Then we can generate an integer-grid map that resembles the frame field [Nieser et al., 2011], and extract the integer level-sets that form the explicit hexahedral mesh [Lyon et al., 2016]. A particular class of field-guided methods are the ones based on a *distance field* [Martin et al., 2009, Chen et al., 2019]. In this case, the field is defined by the geodesic distance from particular shape features or from the model boundary towards its interior.

The peculiarity of field-guided methods is that they can achieve superior mesh quality, specifically if complex feature alignment is required. In particular, they are able to express alignment not only to the boundary of a domain but also to arbitrary internal structures, which is, for example, important in multi-material applications or in the simulation of fluid-structure interaction. On the other hand, volumetric frame fields may exhibit additional types

of singularities that cannot occur in hexahedral meshes [Viertel et al., 2016]. Such singularity configurations are said to be "non-meshable", and their fixing requires complex techniques or user interaction. As a consequence, field-guided methods are successfully adopted when the mesh to be generated is allowed to be hex-dominant [Gao et al., 2017, Sokolov et al., 2016] (i.e., to contain few non-hexahedral elements).

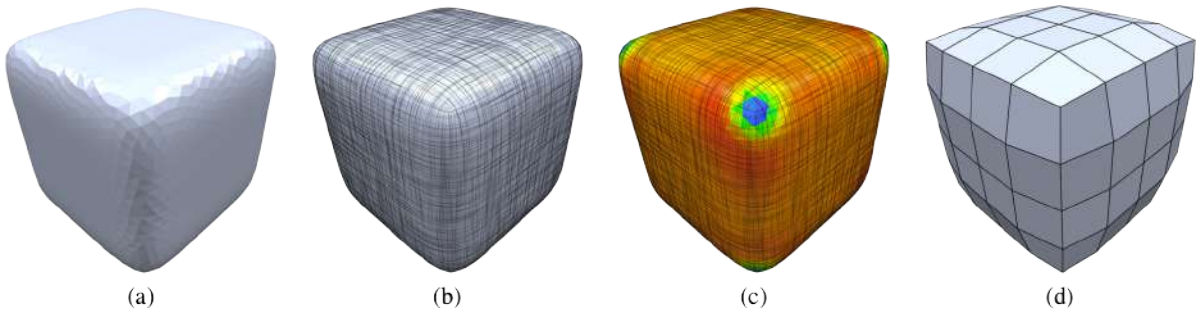


Figure 2.13: Prototype of a field-guided method from [Bommes et al., 2013b]: Given an input triangle mesh (a) in the first step an orientation field (b) is computed which represents the local rotation of quad elements. In the second step, a sizing field (c) is determined that specifies the sample density, which in this example is isotropic and close to uniform. In the third step, a consistent quad-mesh (d) is generated that closely reproduces both guiding fields.

Polycube Methods Another notable example of semi-structured grid-based mesh generation is provided by polycube methods. This class of methods was originally designed for volumetric tessellations, but they have also been applied to surface multi-block meshing as a tool to decompose the domain into well-shaped patches with few irregular nodes and uniform tessellation density (Section 2.2.2).

Polycube algorithms work by volumetrically mapping a shape to an orthogonal polyhedron (or polycube [Tarini et al., 2004]) embedded in $\mathbb{Z}^3$. Sampling the polycube at a dense integer lattice gives regular hexahedral connectivity, whose nodes can be positioned inside the initial object following the inverse map, see Figure 2.14. Polycube methods are based on two fundamental building blocks: the definition of the polycube structure, and the generation of the volumetric map [Gregson et al., 2011, Livesu et al., 2013]. These two objectives can be pursued separately (i.e., defining a valid polycube structure first, and then computing the map) or together, using mesh deformation to explore the space of shapes and find the orthogonal polyhedron closest to the input object [Fang et al., 2016].

Polycube methods have received increasing attention from the meshing community and have now reached a discrete maturity level [Pietroni et al., 2022]. The most recent algorithms allow to process big datasets (containing more than a hundred shapes), producing hex-meshes of good quality [Fu et al., 2016]. In terms of structure, meshes resulting from these methods are typically semi-structured, or occasionally block-structured if singularities (i.e., polycube corners)

align [Cherchi et al., 2016]. A particular feature of polycube-based hex-meshes is that their singular structure is confined to the surface and consists of all polycube edges and corners. This inability to position singularities in the interior inherently limits the map, and may occasionally be the source of unnecessary distortion.

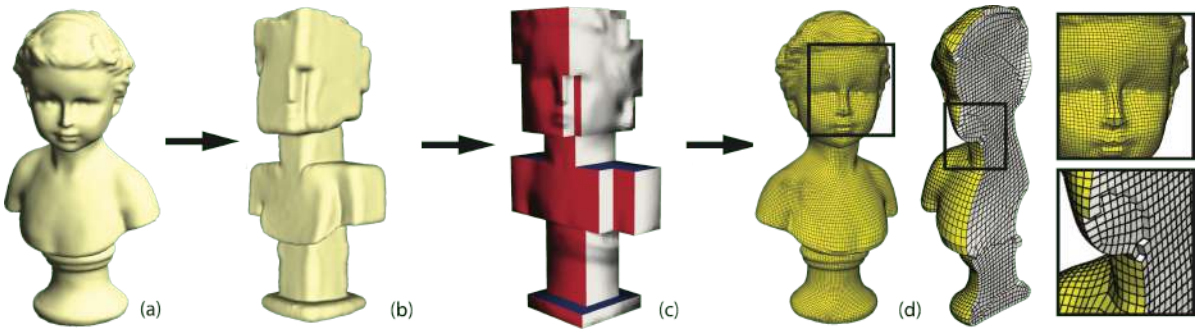


Figure 2.14: Polycube pipeline from [Gregson et al., 2011]: (a) initial shape; (b,c) definition of the polycube structure; (d) generation of the volumetric map.

2.2.4 Unstructured Meshes

Unstructured meshes are preferable when the shape of the domain does not exhibit much regularity, making it useless to exploit a coarse block structure which would cause a very high number of singular nodes. Among them, the most used and studied are undoubtedly the simplicial meshes. Triangular and tetrahedral unstructured mesh generation is an extensively studied topic, with spectacular theoretical and practical achievements obtained in the last years. Generic polytopal discretizations also fall in the class of unstructured meshes. They constitute an interesting field, that is being explored in the very last years and it is yielding interesting results.

As we have already seen, in contrast to structured or partially-structured meshes, the major feature of unstructured meshes consists of much larger flexibility in terms of cell types, mesh organization or mesh structure. The concept of unstructured mesh allows one to place the nodes locally, irrespective of any prescribed coordinate directions, so that curved boundaries can be handled with ease. Moreover, local regions in which the solution is turbulent or its variations are large can be resolved with a selective and local insertion of new points without unduly affecting the resolution in other parts of the physical domain. This matches perfectly with the versatility of simplicial and polytopal elements, which, differently from quadrangles and hexahedra, do not need to be aligned to specific directions. The advantages of these meshes lie in their ability to deal with complex geometries while allowing to provide natural mesh adaptation through the insertion of new nodes.

Simplicial meshes have been studied extensively in the literature related to computational geometry [Preparata and Shamos, 2012] together with other geometric constructs, Voronoi

tessellations, that are particularly well suited to partition domains into polytopal elements with good geometric properties. In the following, we will review briefly the definitions and relations among the most important geometric structures in this context. A survey on data structures for unstructured meshes is given in [Löhner, 1988].

Voronoi Tessellations Let $\mathcal{P} = \{\mathbf{p}_1, \ldots, \mathbf{p}_l\}$ be a set of points (so-called *sites*) in $\mathbb{R}^n$. We associate to each site $\mathbf{p}_i$ its *Voronoi cell*, or region, $V(\mathbf{p}_i)$ such that

$$V(\mathbf{p}_i) := \{\mathbf{x} \in \mathbb{R}^n : \|\mathbf{x} - \mathbf{p}_i\| \leq \|\mathbf{x} - \mathbf{p}_j\|, \forall j \neq i\}.$$

The collection of the nonempty Voronoi regions and their faces, together with their incidence relations, constitute a polytopal complex called the *Voronoi tessellation*, or diagram, of $\mathcal{P}$ [Aurenhammer, 1991, Fortune, 1995, Boissonnat and Yvinec, 1998]. The Voronoi tessellation of $\mathcal{P}$ is a partition of $\mathbb{R}^n$ because any point of $\mathbb{R}^n$ belongs to at least one Voronoi region. A Voronoi cell of a site $\mathbf{p}_i$ is also defined as the intersection of closed half-spaces bounded by bisectors (the locus of points that are equidistant to two sites $\mathbf{p}_i$ and $\mathbf{p}_j$). This implies that all Voronoi cells are convex since the intersection of convex sets remains convex. Note that, when a site $\mathbf{p}_i$ is on the boundary of the convex hull of $\mathcal{P}$, the relative Voronoi cell $V(\mathbf{p}_i)$ is unbounded (Figure 2.15(b)). A *clipped* Voronoi tessellation [Boots et al., 2009, Yan et al., 2013] is the intersection between the Voronoi tessellation and a domain $\Omega \subset \mathbb{R}^n$ (Figure 2.15(c)). Voronoi tessellations have been used extensively for the geometric properties of their cells, whose definition encapsulates important information about relative distance among points. This notion has also been used to produce very well-balanced partitions of the space into polytopal elements by the introduction of the *Centroidal Voronoi Tessellation* (CVT) [Du et al., 1999]. A CVT is a special type of clipped Voronoi tessellation, where the site of each Voronoi cell is also its center of mass. CVTs are used to partition the space into a given number n of polytopes: intuitively, n points are cast into the space, the Voronoi tessellation is constructed (n polytopes are created) and the n points are moved to the centroids of their relative cell. The process is repeated until the tessellation converges to an almost balanced set of polytopes covering the space. The computation of a CVT consists of the following three steps, summarized in Figure 2.15 [Wang et al., 2016]:

- Initialization: sample a user-defined number of sites $\mathcal{P}$ inside Ω. Sites can be generated randomly or with apposite techniques [Moriguchi and Sugihara, 2006, Quinn et al., 2012].
- Clipping: compute the intersection between the Voronoi tessellation of $\mathcal{P}$ and Ω. This is not an easy problem, in particular for volumetric domains [Lévy, 2014, Abdelkader et al., 2020].
- Optimization: update the position of the sites for obtaining a tessellation with uniform and regular cells. This is obtained by minimizing a CVT energy function which expresses,

to some extent, the compactness of the cells. Finding an optimal CVT appears difficult since the energy function is usually non-linear and non-convex [Liu et al., 2009].

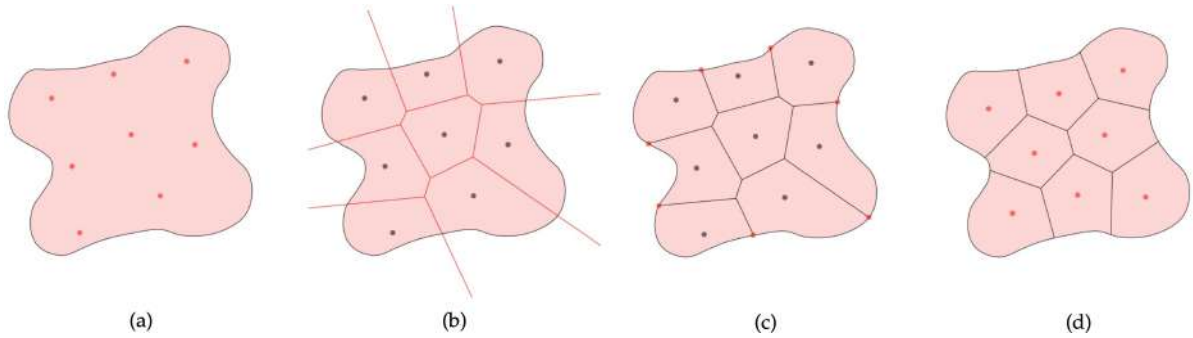


Figure 2.15: Overview of the computation of a Voronoi-based tessellation in two dimensions [Wang, 2017]. (a): Initialization: sample the sites inside the input shape. (b): Voronoi tessellation of the sites. (c): Clipping: compute the clipped Voronoi tessellation. (d): Optimization: update the position of the sites by minimizing the CVT energy function.

CVTs are widely used to generate uniform discretizations of shapes and structures in various scientific domains that include quantization, sensor networks, crystallography, and shape modeling, among others [Du et al., 1999]. Since the dual of a Voronoi tessellation is a Delaunay triangulation (defined in the next paragraph), algorithms for computing the Voronoi tessellations are also widely used for tri and tet-mesh generation, see Figure 2.16. Moreover, extensions of the Voronoi tessellation can be defined by replacing the Euclidean distance with other distance metrics, such as the weighted Voronoi tessellation [Boots et al., 2009], the power diagram [Aurenhammer, 1987], and the L_p Voronoi tessellation [Lévy and Liu, 2010]. In particular, the L_p Voronoi tessellation produces a set of (mostly) square Voronoi cells from which a quad-dominant semi-structured mesh can be extracted by triangle merging [Boier-Martin et al., 2004]. A downside to Voronoi-based mesh generation methods is the fact that they do not allow for full control over the node placement in the mesh. In fact, the user can only specify the location of the centroids, the optimization criterion, and, possibly, some boundary constraints, but it is not possible to ensure that the generated mesh will contain a node in a particular position in the interior of the domain.

Delaunay Triangulations By far the most popular among the triangle and tetrahedral meshing techniques are those derived by a *Delaunay triangulation*, which is the dual structure of the Voronoi tessellation [Weatherill, 1988a, Botsch et al., 2010]. More specifically, the Delaunay triangulation of a set of sites $\mathcal{P}$ is a simplicial complex such that $k+1$ points in $\mathcal{P}$ form a Delaunay simplex if their Voronoi cells have non-empty intersection. Note that, like the Voronoi tessellation, the Delaunay triangulation is defined for any space dimension; therefore the term "triangulation" here may refer to both tri- and tet-meshes. In two dimensions, each Delaunay

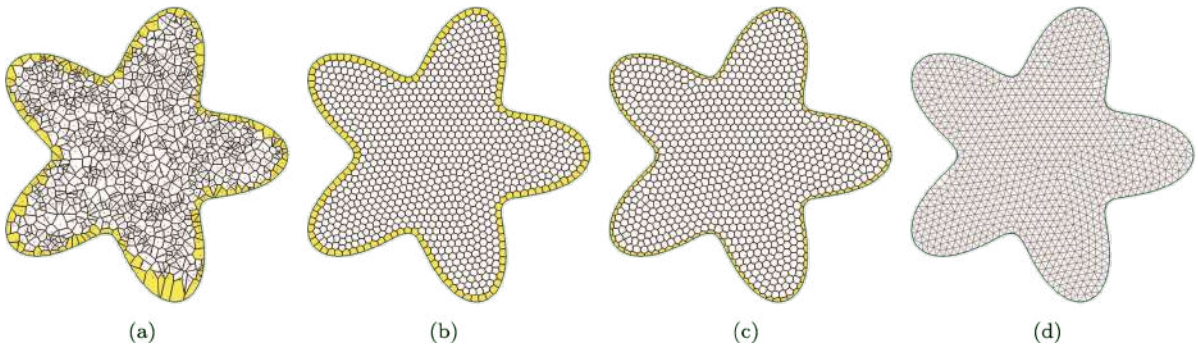


Figure 2.16: 2D CVT-based meshing from [Yan et al., 2013]. (a) The clipped Voronoi tessellation of initial sites; (b) the result of CVT; (c) the result of constrained optimization, with boundary seeds constrained on the border; (d) the dual triangle mesh.

triangle $(\mathbf{p}, \mathbf{q}, \mathbf{r})$ is dual to a Voronoi vertex where $V(\mathbf{p})$, $V(\mathbf{q})$, and $V(\mathbf{r})$ meet; each Delaunay edge $(\mathbf{p}, \mathbf{q})$ is dual to a Voronoi edge where $V(\mathbf{p})$ and $V(\mathbf{q})$ meet; and each Delaunay vertex $\mathbf{p}$ is dual to its Voronoi face $V(\mathbf{p})$, see Figure 2.17. Another key notion used in Delaunay-based meshing algorithms is the *restricted Delaunay triangulation* [Edelsbrunner and Shah, 1994, Dey, 2006]. Let Ω denote a subset of $\mathbb{R}^n$; $\mathcal{P}$ a point set of $\mathbb{R}^n$; and $D(\mathcal{P})$ the Delaunay triangulation of $\mathcal{P}$. We call the Delaunay triangulation restricted to Ω the sub-complex of $D(\mathcal{P})$, denoted $D_\Omega(\mathcal{P})$, whose dual Voronoi faces intersect Ω. The 3D Delaunay triangulation restricted to a surface $\mathcal{S}$ is the set of Delaunay facets (triangles) whose dual Voronoi edges intersect $\mathcal{S}$.

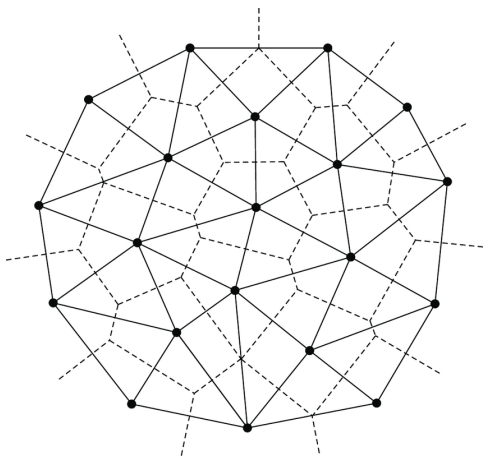


Figure 2.17: Delaunay triangulation (full lines) obtained as the dual of a Voronoi tessellation (dotted lines).

The Delaunay triangulation is shown to enjoy several local and global properties due to its duality with the Voronoi diagram: (i) Delaunay triangles are nearly equilateral; (ii) the minimum angle is maximized; (iii) the triangulation is unique if the points are in a general position, i.e., no four points are cyclic; (iv) if every triangle in a triangulation is non-obtuse, it

is a Delaunay triangulation; (v) any two-dimensional triangulation can be transformed into a Delaunay triangulation by locally flipping of the diagonals of adjacent triangles; (vi) the restricted Delaunay triangulation has optimal approximation properties (both in terms of geometry and topology) when sites are sufficiently dense [Boissonnat and Oudot, 2005]. These properties give some grounds to expect that the grid cells of a Delaunay triangulation are not too deformed, explaining the success of the Delaunay triangulation for mesh generation, as small angles cause numerical problems in finite element methods. In dimension three, however, the solid angle of the triangulation may not be maximized, as degenerated and almost degenerated tetrahedral elements known as *slivers* with zero or very small solid angles can be found in a Delaunay tetrahedrization [Attene and Spagnuolo, 2000]. Hence, methods based on the Delaunay criterion may not be optimal in terms of shape quality for volumetric meshes.

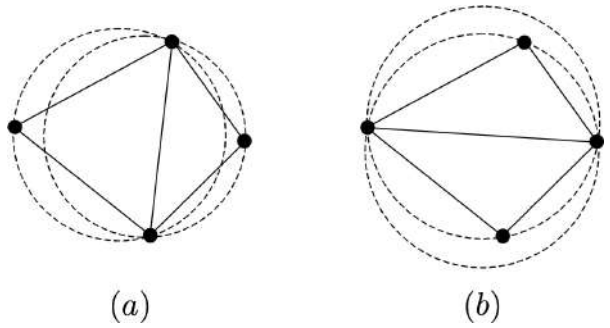


Figure 2.18: Delaunay criterion: since the circumcircles of the triangles in (a) do not contain the other triangle's nodes, the empty circle property is maintained, while in (b) it is not.

Delaunay triangulations satisfy the *Delaunay criterion*, also called the "empty sphere" property, illustrated in Figure 2.18 in the two-dimensional case: the circumsphere of any cell does not enclose any node of the mesh. Although the Delaunay criterion has been known for many years, it was not until the work of Lawson [Lawson, 1977] and Watson [Watson, 1981] that the criterion was utilized for developing algorithms to triangulate a set of nodes. This is because the Delaunay criterion in itself is not an algorithm for generating a mesh. It merely characterizes the connections among a set of points in space, but if we want to mesh a given domain using the Delaunay criterion, it is necessary to devise a method for generating node locations within the geometry and then optimizing the connections to satisfy the Delaunay criterion. Moreover, if an existing boundary discretization is required to be maintained, a Delaunay tessellation conforming to that boundary may not exist at all (e.g., if the boundary discretization violates the Delaunay criterion). For this reason, *boundary-constrained* Delaunay triangulations/tetrahedrizations were introduced [Weatherill and Hassan, 1994].

A common methodology to build a Delaunay mesh of a bounded domain is to first mesh the domain boundary nodes: this step provides an initial set of triangles, or tetrahedra, which satisfy the Delaunay criterion. Once the boundary is meshed, nodes are inserted incrementally

into the existing mesh, redefining the elements locally as each new node is created while keeping the Delaunay criterion. This is typically done with a Bowyer-Watson algorithm [Baker, 1989]. The various methods for Delaunay meshing can be distinguished according to the criteria used for placing new interior nodes:

- The simplest point insertion approach is to define nodes from a regular grid of points covering the domain at a specified nodal density. In order to provide for varying element sizes, a user-specified sizing function can be defined and nodes inserted until the underlying sizing function is satisfied [Weatherill and Hassan, 1994]. Another approach is for nodes to be recursively inserted at triangle or tetrahedral centroids [Ruppert, 1993], or at element circumcircle/sphere centers [Shewchuk, 1996], or with advancing front techniques [Marcum and Weatherill, 1995].

- In the so-called Voronoi-segment method [Rebay, 1993], the new node is introduced at a point along the line segment connecting the circumcircle centers of two adjacent cells, according to a local size criterion. This method tends to generate very structured-looking meshes with six triangles at every internal node.

- Another straightforward method is point insertion along edges [Borouchaki et al., 1995]. A set of candidate nodes is generated by marching along the existing internal edges of the triangulation at a given spacing ratio. Nodes are then inserted incrementally, discarding nodes that would be too close to an existing neighbor. This process is continued recursively until a background sizing function is satisfied.

A Delaunay triangulation of n points can be computed in time $O(n \log n)$ [Preparata and Shamos, 1985]. Based on a sound geometrical concept and the optimality properties, Delaunay triangulation has important applications in many fields, including data visualization, terrain modeling, mesh generation, surface reconstruction, and structural networking for arbitrary point sets. The popularity of Delaunay triangulation is attributed to its nice geometric properties as a dual of Voronoi tessellation, its connection to the concept of *alpha shapes* [Edelsbrunner and Mücke, 1994], and the speed with which it can be constructed in two or higher dimensions with well-developed libraries (e.g., *Triangle* [Shewchuk, 2005] and *Tetgen* [Si, 2015]).

Quadtree-Octree Methods Quadtree-octree meshing is based on the idea of partitioning a domain progressively to produce cells of a size compatible with the node spacing requirement. The use of quadtree and octree decompositions for mesh generation was developed in the 1980s [Yerry and Shephard, 1984, Shephard and Georges, 1991].

In this approach, applied to mesh generation, the n-dimensional domain to be gridded is first enclosed in a bounding root box (an n-dimensional parallelepiped) which is approximated with a collection of disjoint and variably sized cells whose union constitutes the final mesh

of the domain (Figure 2.19). The cells are obtained from a recursive refinement of the root parallelepiped:

- *2-refinement* splits each edge in two, thus obtaining 4 equally sized sub-cells for each adjacent rectangle or 8 equally sized sub-cells for each adjacent hexahedron;
- *3-refinement* splits each edge in three, thus obtaining 9 and 27 sub-cells, respectively.

In both cases, the sequence of splits is encoded in a hierarchical tree structure, which corresponds to a quad/octree for the 2-refinement, and a 9/27-tree for the 3-refinement. However, methods based on 3-refinements are commonly considered to be in the quad/octree class. The stopping criterion used to subdivide a cell can be based on normal similarity [Ito et al., 2009], local thickness [Livesu et al., 2021, Maréchal, 2009], surface approximation [Gao et al., 2019] or a combination of these and other indicators [Bawin et al., 2021]. To ensure element sizes do not change too dramatically, a maximum difference in the tree subdivision level between adjacent cells can be limited to 1. After the refinement stage, smoothing and cleanup operations can be employed to improve element shapes and global connectivity. For instance, the removal of hanging nodes is obtained by substituting elements of the grid with templated topological transitions that locally restore mesh conformity. As the last step, the boundary nodes are projected onto the target geometry. Differently from other approaches, this technique does not match a pre-defined surface mesh, rather surface edges or facets are formed wherever the internal structure intersects the boundary.

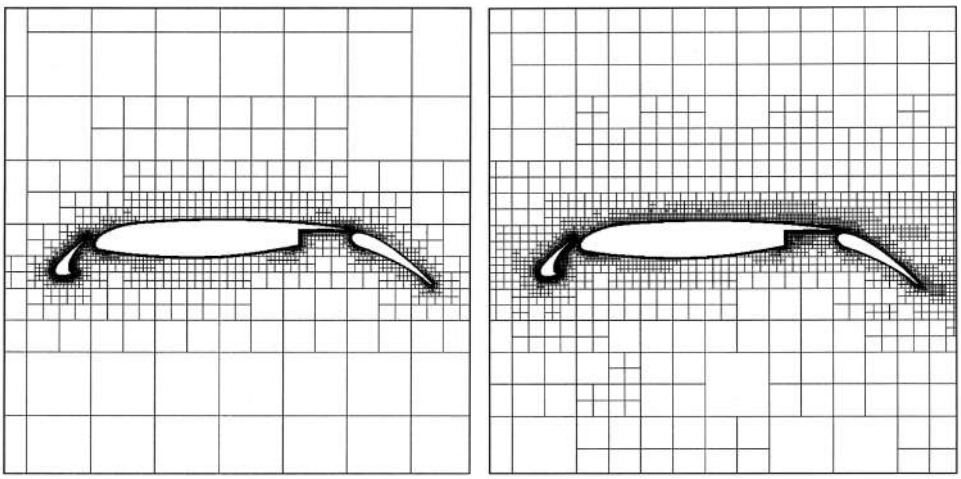


Figure 2.19: Quadtree meshing of an airfoil, with two different refinement levels [Thompson et al., 1998].

When combined with a conversion strategy or a Delaunay-based method (see next paragraph), the quadtree/octree method becomes a powerful method for generating unstructured simplicial complexes. The conversion of a quad/octree grid into a tri/tet-mesh may be useful for two

reasons: first, the accuracy of the boundary representation can be significantly increased by the use of triangular elements. Moreover, simplicial elements allow generating conformal meshes (i.e., meshes without hanging nodes), which is one of the drawbacks of quadtree/octree grids.

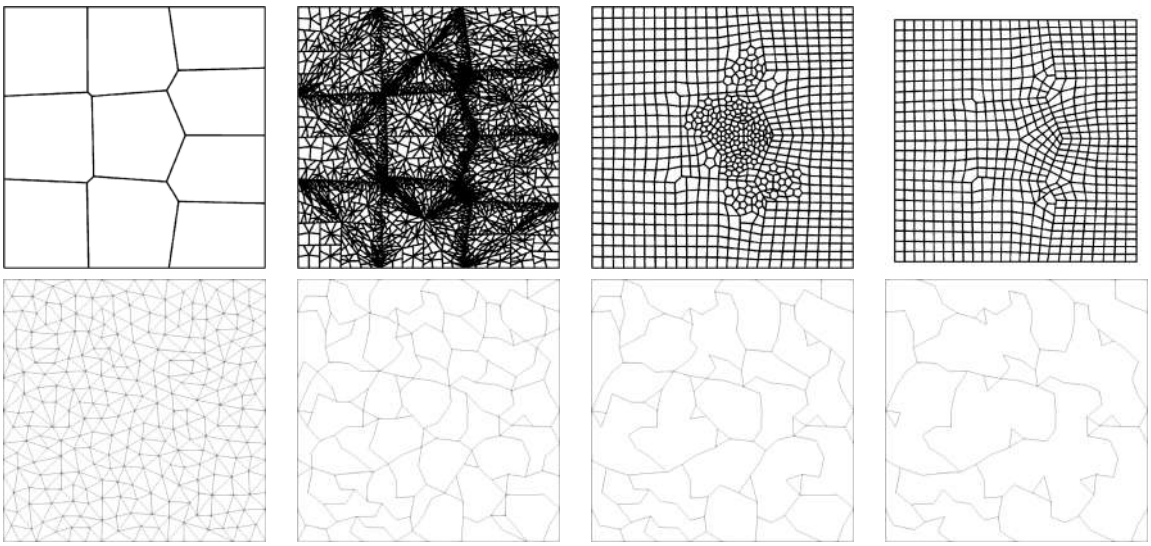
From a mesh quality standpoint, tree-based generation techniques are typically considered inferior to other methods because: (i) the grid is fixed in space and the result depends on the orientation of the model; (ii) the connectivity they generate is intricate and rich of singular edges with high valence; (iii) the resulting meshes are highly unstructured and do not endow a coarse block decomposition. Nevertheless, when compared with alternative options tree-based generation methods really stand out in terms of robustness. To date, they are the only fully automatic methods capable of successfully hex-meshing any input shape, regardless of its geometric or topological complexity [Pietroni et al., 2022]. For this reason, they are the only automatic methods currently implemented in professional software [MeshGems, 2020, CoreForm, 2021, Cubit, 2021].

Mesh Conversion We can imagine generating a mesh starting from a given tessellation and operating a sequence of local operations on its connectivity to convert it into a new discretization with different properties. The reason for such conversion could be, for instance, that we need to work on our mesh with tools that only support particular cell topologies, as frequently happens with numerical methods. Operations on the connectivity of the mesh can be basically the splitting of a cell into a number of smaller elements or the aggregation of a number of cells into a bigger one. In the literature, we can find efficient strategies for splitting and aggregating polygons and polyhedra of any type into any type.

For instance, a triangular mesh can be converted into a quadrangular mesh by merging two original triangles into one quad [Tarini et al., 2010]. An hex-mesh can be generated from a tet-mesh by splitting each tetrahedron into four hexahedra via midpoint refinement [Li et al., 1995]. Such techniques are trivial to implement and always guarantee a correct result, but they produce in general unstructured meshes with overly dense singular structures and highly deformed elements. They are therefore not particularly suited for quad/hex-mesh generation.

A grid-based mesh can be converted into a simplicial mesh following simple refinement rules for quad-to-tri or from hex-to-tet subdivision [Bern and Plassmann, 2000]. However, it is immediate to see how it is not possible to convert a grid into a simplicial complex by elements agglomeration.

A similar operation can be performed starting from any kind of pure mesh and refining or aggregating its elements according to certain rules. The resulting mesh will in general be polygonal/polytopal, and the key ingredient, in this case, is the refinement or agglomeration driving criterion, see Figure 2.20. The driving criterion can be based on some general quality indicator, it can be appositely designed to prevent or remove particular pathological configurations, it can contain any kind of hand-made heuristics [Antonietti et al., 2021b]. Recent works [Antonietti and Manuzzi, 2022] also employ convolutional neural networks.



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

Figure 2.20: (top) Example of mesh refinement [Antonietti and Manuzzi, 2022]: an initial polygonal mesh (a) is multiply refined with the midpoint strategy (b) and with two different CNN-based strategies (c,d). (bottom) Example of mesh coarsening [Antonietti et al., 2021b]: the initial triangular mesh (a) is iteratively agglomerated (b,c,d) according to a custom strategy.

2.3 Mesh Quality

In this section, we turn our attention to the quality of a discretization. The goodness of a tool principally depends on what that tool is used for, therefore a mesh considered "good" for one specific application may not work as fine for another. As we are mainly interested in finite element simulations, the quality of a mesh will be here related to the performance of a numerical scheme over it, in terms of speed and accuracy. It is also important to note that the relation between mesh quality and the quality of a numerical solution of a PDE may heavily depend on the specific PDE, as well as on the solver at hand.

Definition 2.9 (Quality Indicator)**.** *A quality indicator is a function defined over the cells of a mesh, capable of giving insights on the accuracy and the convergence speed of a finite element scheme applied on that mesh, before solving any numerical problem.*

We can distinguish between indicators that measure the quality of a single element (e.g., based on the presence of small edges) and indicators that measure the quality of the mesh altogether (e.g., based on the nodes distribution across the domain). While a common requirement is that all mesh elements are non-degenerate, different numerical schemes may demand the fulfillment of additional requirements. In the literature on PDE solvers, there are plenty of definitions and criteria to measure the quality of elements and meshes: the following review is intended to cover the various flavors proposed, also in relation to the context in which these are defined

and used. This review lays the basis for the definition of the new indicator, which is one of the main contributions of the thesis work.

2.3.1 Element Indicators

Element quality indicators are defined element-wise and then collected into a single mesh quality score in some kind of average. As a consequence, they are often specific to particular types of element typology. One common operator that is used in several contexts is the *Jacobian*. Consider a triangle with vertices x_0, x_1 and x_2. The matrix

$$\mathcal{J}_0 := \begin{bmatrix} x_1 - x_0 & y_1 - y_0 \\ x_2 - x_0 & y_2 - y_0 \end{bmatrix}$$

is called the Jacobian matrix of the triangle relative to the vertex x_0, and analogous matrices can be defined for a tetrahedron. The determinant of the Jacobian matrix for simplicial elements is invariant to the choice of the vertex x_0 [Freitag, 1997], therefore we omit the subscript and note it $\det(\mathcal{J})$, the Jacobian of the element.

The concept can be extended to quads and hexes, but in this case, the invariance is not guaranteed. A possible solution is to consider the different matrices $\mathcal{J}_i$ at the corners and at the center of the element, and define the Jacobian as $\min_i \det(\mathcal{J}_i)$ [Knupp, 2001]. The Jacobian provides useful information on the quality of an element (e.g., skew, length ratio, shape, distortion, volume change, and orientation) [Knupp, 2001]. For instance, if $\det(\mathcal{J}) \leq 0$ (or $\min_i \det(\mathcal{J}_i) \leq 0$), the implied element is said to be *irregular*, and it is considered invalid in the context of finite element methods [Knupp, 2000, Mitchell et al., 1971]. Sometimes a distinction is made between degeneration ($\det(\mathcal{J}) = 0$) and inversion or fold-over ($\det(\mathcal{J}) < 0$). Depending on the concrete setting, such elements may lead to "inaccurate solutions or no solutions at all" [Barrett, 1996], "invalidated" solutions [Roca et al., 2011], or situations in which"calculations cannot be continued" [Salagame and Belegundu, 1994]. Detailed formulations for the majority of the element quality indicators presented in this section can be found in the Verdict library [Stimpson et al., 2007].

Triangles We start our analysis from triangular elements because several indicators are firstly defined over them and then extended to quads, tets, or hexes. There are two main quantities to measure in a triangle: edge lengths and angles, and the quality is typically intended as its deviation from an equilateral triangle. Pathological triangular elements can be either of the type of Figure 2.21(a) or Figure 2.21(b).

Edge lengths, or *aspect regularity*, can be controlled by the ratio between the inradius and the circumradius of the triangle (respectively, the radii of the inscribed and circumscribed circles). The edge ratio instead, is not an interesting quality measure as it does not vanish for all the degenerate cases, for instance, the flattening of Figure 2.21(b).

The *angle regularity* instead, can be devised by measuring the sine of the minimum or maximum angle, or a ratio between these two. Alternatively, it can be formulated as the ratio between the maximum edge and the inradius.

More sophisticated indicators try to measure edges and angles together with a single *shape regularity* function based on the Jacobian matrix. The algebraic shape metric from [Knupp, 2001], also called *mean ratio*, is defined as the ratio of the geometric mean to the arithmetic mean of the eigenvalues of $\mathcal{J}$. An equivalent version is the *Frobenius ratio*, which consists of the inverse of the condition number of $\mathcal{J}$ in the Frobenius norm, but can be also computed as the sum of the edge lengths squared divided by the area.

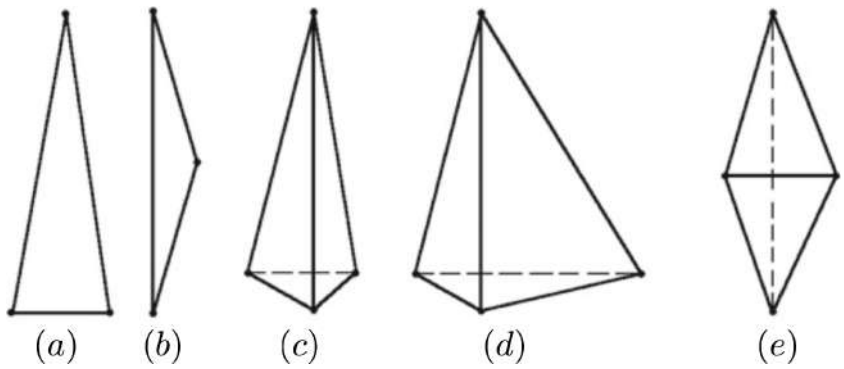


Figure 2.21: Examples of poorly shaped triangles and tetrahedra.

Tets The major difference between a triangle and a tet is that a new entity comes to play in the latter, namely, the solid angle. The solid angle θ_i at vertex $\mathbf{x}_i$ of tetrahedron $T(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4)$ is given by the surface area formed by projecting each point on the face opposite to $\mathbf{x}_i$ to the unit sphere centered at $\mathbf{x}_i$. It measures how large an object appears to an observer looking at it from a point, with a natural geometrical relationship to object visualization. A collection of tetrahedral shape measures is given in [Au et al., 1998], and a systematical comparison between tetrahedral quality measures is provided in [Liu and Joe, 1994].

The *aspect regularity* can be computed by the radius ratio, as for triangles, and it helps avoid situations like Figure 2.21(c),(d) but not (e). We recall that the edge ratio is not an interesting quality measure, as it fails to detect degenerate tets like slivers. Numerous other aspect ratios for tets are proposed in [Stimpson et al., 2007].

An *angle regularity* measure based on the solid angle is defined as the sine of the minimum solid angle. The minimum dihedral angle can be measured as well, but it cannot detect the degenerate case of a needle-shaped tet (Figure 2.21(c)). In fact, the dihedral angles of a tetrahedron remain more or less the same as the triangular face opposite to the pointed node is getting smaller and smaller.

Regarding *shape regularity*, both the mean ratio and the Frobenius ratio defined for triangles

can be easily applied to tets. A simple formula for computing the mean ratio of a tet, involving only the volume and the edge lengths, is provided in [Au et al., 1998].

Quads While quads may be more efficient than triangles from a computational point of view, their properties make them a more difficult primitive to handle compared to triangles. The extra degree of freedom given by the fourth vertex gives birth to many pathological configurations, and even simple operations present a greater challenge.

First of all, a quad is not necessarily flat, and even planar quads may be non-convex. This can be particularly crucial in architectural applications, where quad-meshes featuring this property are often categorized as *Planar-Quad*. Planarity and convexity can be simply imposed as pre-requisites for all the elements in a mesh. The *warping* of a quad measures its distance from being planar, and it is measured through the curvatures of the coordinate surface on which the quad lies (Figure 2.22(d)).

A formula for the *aspect regularity* of a quad can be derived from those for triangles, and it is a measure of the departure of the cell from a rhombus (Figure 2.22(b)). The opposite sides of each quad should have approximately equal length (this can be particularly crucial in contexts like physical simulations), or, for anisotropic approximation, a ratio best for approximation quality. A similar *stretch* measure is given by the ratio between the minimum edge and the maximum diagonal.

Regarding the *angle regularity* (also *skewness* or *orthogonality*), internal angles should be close to 90 degrees, therefore we need to characterize the departure of a quad from a rectangle (Figure 2.22(c)). It can be described through the maximum/minimum angle (or their ratio), or by the angle between the principal axes of the quad. In fact, it is the absolute value of the cosine of the angle between the principal axes.

For the *shape regularity*, we already saw how the Jacobian can be extended by considering the minimum between the values of $\det(\mathcal{J})$ at the vertices and at the center of the quad. Alternative Jacobian-based indicators are the ratio between its minimum and maximum values (weighted Jacobian) or the minimum determinant of $\mathcal{J}_i$ evaluated at each corner and the center of the element, divided by the corresponding edge lengths (scaled Jacobian). Such indicators provide a measure of the variation in the Jacobian across the element, and they can be used in the same formulas as for triangles for a general measure of the quad shape. Note that skewness can be maximized by a rectangle with any aspect ratio, whereas shape can only be maximized by a square. Another possible shape indicator is the Oddy metric [Oddy et al., 1988], which measures the maximum deviation of the metric tensor at the corners of the quad.

Hexes The quality indicators defined for quadrangular elements are straightforwardly extended to hexahedral cells. Warping, stretching, skewness, and aspect regularity of a hex can be defined by the sum, the average, or the minimum/maximum of the respective measures on its faces.

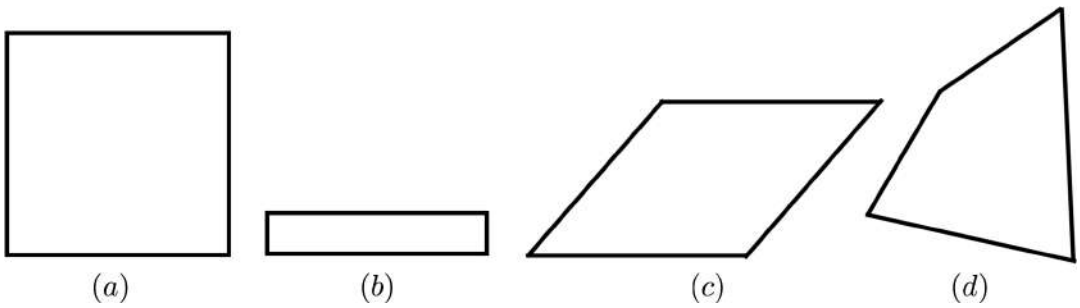


Figure 2.22: Quality of a quadrangular element: (a) ideal element, (b) low aspect ratio, (c) skewed element, (d) warped element.

The *shape regularity* indicators based on the Jacobian matrix are probably the most common in the literature. Shape measures can be the minimum between the $\mathcal{J}_i$ determinants, or the maximum (or the average) between the Frobenius norms of their condition numbers. Weighted and scaled Jacobians (defined as for quads) are also used. The determinant $\det(\mathcal{J})$ quantifies to what extent a hex deviates (in terms of volume distortion) from the unit cube. However, it is blind to angle distortion; therefore it cannot capture skewness. Additional angle-aware measures are thus often taken into account.

Polytopes The concept of the geometric quality of a cell becomes quite vague when the cell is a generic polygon or polyhedron. Most of the quality indicators defined above become meaningless when the elements have an undefined number of vertices, or at least very difficult to extend. For instance, the Jacobian can be extended to generic polygons (similarly to how we did for quads), but the Jacobian matrix in a polyhedron vertex is defined only if such vertex has three incident edges (e.g., a prism is fine, but a pyramid is not).

Trying to delimit the field by excluding unlikely configurations, most mesh generators, and most numerical schemes only allow convex or star-shaped elements. A polytope P is said to be *convex* if, given any two points p_1 and p_2 in P, the line segment connecting p_1 and p_2 is entirely contained in P. Two points p_1 and p_2 in P are said to be *visible* from each other if the segment (p_1, p_2) does not intersect the boundary of P. Last, a polytope P is called *star-shaped* if there exists a sphere from which all the points in P are visible, and the definition of visibility implies that the sphere has non-zero radius. Star-shapedness is weaker than convexity, and it is often used in the literature as many theoretical results in the theory of polynomial approximation in Sobolev spaces rely on this condition [Dupont and Scott, 1980, Scott and Brenner, 2008].

Numerous attempts have been made to measure the quality of a polygon or polyhedron as "how far it is from a regular n-gon or n-hedron" [Chalmeta et al., 2013, Coxeter, 1938, Zunic and Rosin, 2004], because a unitary regular n-gon (or n-hedron) is usually considered to be of high quality in the Euclidean metric. This implies, for instance, considering the ratio between the area (volume) of an element P and the area (volume) of a regular n-gon (n-hedron) with

the same perimeter (surface area) as P, or the difference between the angles of P and those of the ideal element. Unlike triangles, however, n-gons with $n > 3$ generally are not affine similar to a single reference n-gon. Hence the idea of measuring the quality of a polygonal mesh by comparing its elements to regular n-gons through affine mappings does not work in general. To make it work, one needs to build proper mappings connecting arbitrary polygons with the reference ones, or re-define reference polygons of "good quality", or do both [Huang and Wang, 2020]. Another drawback is that the initial assumption is not always true. Polytopal element methods are defined to work over extremely generic elements, therefore there is apparently no reason to believe that the quality of a regular n-gon is the highest possible, or that they are the only configurations able to reach that quality level.

A *shape regularity* indicator for polytopes proposed in [Di Pietro and Droniou, 2019] is the ratio between the inradius and the diameter of the element (the maximum point-to-point distance), see Figure 2.23(a). It is an adaptation of the radius ratio obtained by replacing the circumradius with the diameter, as for generic polytopes the circumradius does not always exist. In other works, the circumradius is replaced by the minimum radius of all disks containing P, called outer-radius. In the same paper, a supplemental condition is presented, which requires looking at the simplicial subdivision of the elements, i.e., the unique decomposition of the cell into triangular or tetrahedral elements obtained connecting its vertices (see [Di Pietro and Droniou, 2019] for a more rigorous definition). The *contact regularity* is the ratio between the diameter of the cell and the maximum edge of the simplex with the highest area, shown in Figure 2.23(b).

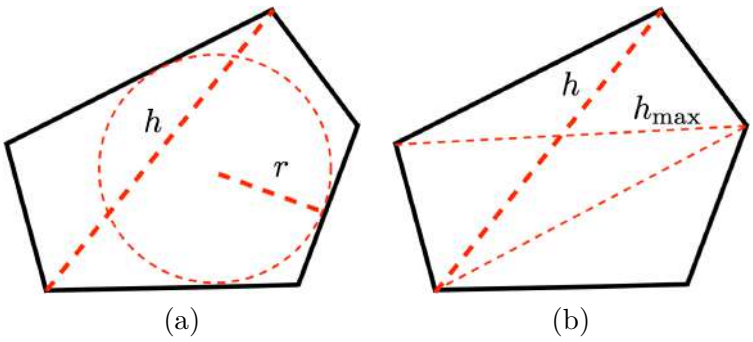


Figure 2.23: Quality indicators for polygonal elements: (a) shape regularity and (b) contact regularity.

Some works on polygonal shape quality also require that the element has no "short edges". Recently, researchers noticed that the no "short edge" requirement can be dropped in the case of virtual element methods [Brenner and Sung, 2018]. But for other numerical methods, the situation is not completely clear and no "short edges" are still required at least in some theoretical analyses [Gillette et al., 2012].

The CVT energy defined in Section 2.2.4 is often used for evaluating the regularity of

Voronoi tessellations [Liu et al., 2009]. However, while this energy accounts for the compactness of the cells, it is a metric that depends both on the number of cells and on the size of the shape. To overcome this problem, in [Wang, 2017] the author proposes to define a quality indicator derived from the second moment (or variance) of the polytope, which measures how far the points inside P are spread out relative to its centroid (which is dimensionless). In this approach, the optimal element is considered to be the hexagon in 2D and the truncated octahedron in 3D.

2.3.2 Mesh Indicators

Mesh quality indicators observe the features of the mesh from a more general perspective. In addition to all the possible functions that we can define collecting element indicators in some sort of average or weighted norm across all the cells, there are indicators related to features that cannot be measured element-wise.

Consistency For the accurate computation of the numerical solution of a PDE defined over the domain, the requirement of consistency with the geometry is indispensable [Liseikin, 2006]. The mesh nodes must adequately approximate the original geometry, that is, the distance between any node of the domain and the nearest mesh node must not be too large, and this distance must approach zero when the number of nodes tends to infinity. This is particularly important for nodes, edges, or faces representing the boundary of the physical domain, as it allows the boundary conditions to be applied more easily and accurately.

Another type of consistency is the one with the solution of the physical problem [Liseikin, 2006]. The distribution of the mesh nodes and the form of the mesh cells should be dependent on the features of the solution, such as preferred directions (e.g., streamlines or vector fields), localized regions of very rapid variations (i.e., regions of high gradients), boundary and interior layers (e.g., in fluid dynamics, combustion, solidification, solid mechanics, and wave propagation), areas of high solution error of the numerical approximation. In all these cases, it may be helpful to subdivide the domain into smaller parts and mesh it locally because the uniform refinement of the entire domain may be very costly for multidimensional computations. Features lines, when present, should be explicitly represented as edge sequences. For example, sharp crease lines in mechanical objects, or lines where some attribute other than normals (e.g., color) varies. Incorporating an input feature network into the mesh is not possible if the connectivity does not align to it, even refining the mesh [Livesu et al., 2020] (see Figure 2.24).

Consistency indicators should measure the distance between the mesh and the domain, or the domain features. A simple example is given by the Hausdorff distance [Taha and Hanbury, 2015] between the mesh and the original geometry.

Structure In a sense, unstructured, semi-structured, block-structured, and structured meshes can be seen as a continuum of cases, with an increasing degree of regularity. Depending on the

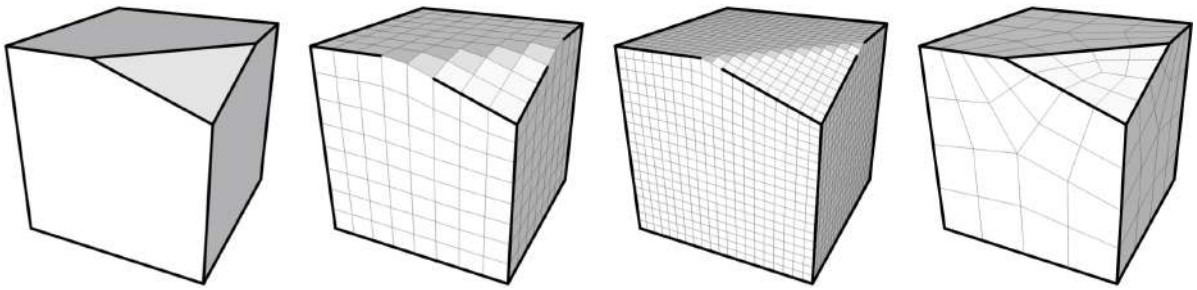


Figure 2.24: Key to feature preservation is the ability to align surface edges to the input network, carefully positioning mesh singularities (image from [Livesu et al., 2020]).

applications, either a lower or higher degree of regularity is required, and this can be measured by the number of singular nodes in the mesh. A low number of singular nodes implies a simpler singular structure, which is more likely to allow for a block-structured mesh. Often, also the positioning of singular nodes is crucial: straight sequences of edges stemming from them (a.k.a. separatrices) should connect them in a graph that is as simple as possible, and they should appear in regions with a strong negative or positive Gaussian curvature (other than where it is needed to change resolution) [Bommes et al., 2013b]. Other works weaken the concept of regular node, considering as regular also nodes with valence close to the fixed value [Aghdaii et al., 2012].

Mesh connectivity is typically assumed to be conforming (i.e., free from T-junctions) and pure (i.e., all polytopes have the same typology). A T-junction is a spot where two elements meet along the edge or face of another element, (see Figure 2.25): this leads to a configuration in which it is not trivial to enforce continuity across the edge or face. The two above requirements can sometimes be loosened, but such topological freedom is not unlimited and may be bounded by the specific application [Pietroni et al., 2022]. Quality indicators, in those cases, are provided by the number of T-junctions, or the number of non-standard elements (for instance, the number of non-quad cells in a quad-dominant mesh).

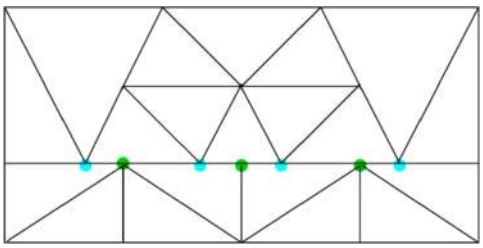


Figure 2.25: Example of non-conforming planar mesh, with T-junctions colored in blue and green.

Distribution Since nodes and elements represent the only contact points between the real world and the approximated copy that we are trying to reproduce through the mesh, the way they are distributed in the domain affects the quality of the final result. If we need a uniform distribution of the information over the domain, the element size (area, volumes, or sum of the faces areas) has to be similar across the mesh. If the mesh is pure, we can measure the balance (or concentration, or density) through simpler quantities like edge lengths or diameters. In particular, in structured meshes, we can measure the change of cell size in a certain direction or along a curve [Liseikin, 2017].

However, it is often beneficial to let the tessellation density vary over the mesh, to adapt to local shape complexity, or to the solution of the numerical problem. In this case, the elements size cannot be required to be uniform across the mesh, but we can at least ask for a smooth transition by imposing a gradual size change (neighboring elements cannot have too different sizes) [Alliez et al., 2005]. Moreover, in order to allow for spatial transition through different levels of resolution, extra singular nodes must be included, unless T-junctions are introduced. Therefore, in many contexts, it is desirable to achieve the right trade-off between adaptivity and connectivity. Moreover, if the physical problem itself is anisotropic, it requires mesh generation to be guided by a prescribed anisotropy field. For instance, in computational fluid dynamics, it is desired to squeeze the elements in the direction normal to the wing of a plane since the most significant physics occurs in the limit layer. If the exact solution is of the type shown in Figure 2.26(a), a discretization like Figure 2.26(c) is likely to perform much better than the one in Figure 2.26(b), despite elements in the latter are more similar to regular polygons.

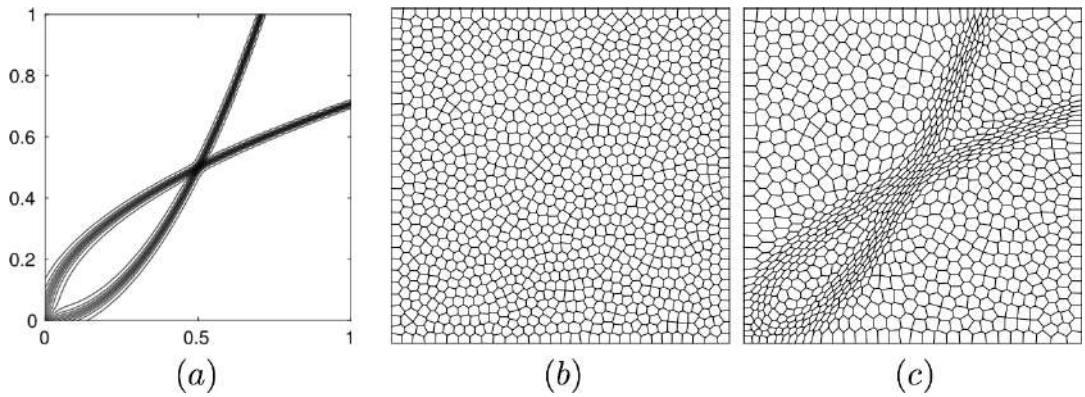


Figure 2.26: Example of anisotropic problem. (a) contour plot of the exact solution, (b,c) two possible discretizations [Huang and Wang, 2020].

2.3.3 Mesh Quality Improvement

The local and global quality indicators presented in Section 2.3.1 and Section 2.3.2 are commonly used to investigate and improve the mesh quality. Some of them can be integrated into the

mesh generation process (e.g., in advancing front methods), so that at the end of the pipeline, the output mesh already reaches a certain quality level. This is particularly true for 2D-meshes, and for indicators concerning properties like structure or connectivity, which may be essential requirements in some applications. The majority of volumetric meshing algorithms instead, employ a two-step process [Pietroni et al., 2022]. The first step generates an initial mesh, which is expected to be dominated by well-shaped elements, but often also contains some poorly-shaped or even degenerate elements. This step is typically followed by an optimization step, whose goal is to maximize the quality of the mesh elements and to remove any degeneracy while keeping the meshed domain boundary intact. As a mesh is defined by the nodal connections of the elements and the coordinates of the nodes, optimization methods can be broadly grouped into two categories [Lo, 2014] - (i) geometrical methods: those that involve a change of element shapes by means of shifting of nodal points; and (ii) topological methods: those that involve a change in the element connections to the nodal points. However, there is no reason why topological and geometrical operations cannot be put together to form even more effective and well-balanced optimization schemes.

Optimization by Geometric Operations Improvement methods that keep the mesh connectivity fixed while changing only the locations of the mesh vertices are commonly referred to as *geometric optimization*, *smoothing*, or *untangling* methods [Owen, 1998, Shepherd and Johnson, 2008]. The shape quality of a mesh can be improved by shifting the interior points within the domain boundary, and usually, boundary points have to be kept intact for compatibility with other meshes or for imposing the required boundary conditions for the finite element analysis.

Before the development of valid shape measures, an early attempt to improve a triangular mesh was to shift each interior node in turn to the centroid of the polygon formed by all the nodes connected to it through an edge of the mesh, and this iteration process is known as *Laplacian smoothing* [Herrmann, 1976]. This kind of heuristic smoothing technique has been successfully employed for tri and quad-meshes [Erten et al., 2009, Xu and Newman, 2006]. Laplacian smoothing is computationally inexpensive and well suited for meshes combining arbitrary element types. However, due to its mesh quality unaware node averaging scheme Laplacian smoothing can lead to mesh quality deterioration and the generation of inverted elements.

Alternatively, based on a shape measure, a cost function can be defined for a mesh, which can be maximized as a local/global optimization problem to improve the overall quality of the mesh [Freitag and Plassmann, 2000, Sastry and Shontz, 2009, Knupp, 2012]. Since this results in a higher computational effort, a combined approach of Laplacian smoothing and local optimization was proposed in [Freitag, 1997], where optimization is only accomplished in problematic regions of the mesh. Theoretical developments on the local optimization problem based on shape measures for triangular and tetrahedral elements were presented by Aiffa and

Flaherty [Aiffa and Flaherty, 2003]. Laplace smoothing and global optimization have been the dominating techniques in mesh optimization based on node shifting in the early days of mesh generation.

A breakthrough was made when Vartziotis et al. [Vartziotis et al., 2008] proposed the Geometric Element Transformation Method (GETMe), which is a purely geometric process to move the nodes of an element to improve its quality. The driving force behind GETMe smoothing is regularizing element transformations which, if applied iteratively, lead to more regular elements (i.e., with a higher mean ratio), see Figure 2.27. Such transformations for polygons with an arbitrary number of nodes have been proposed and analyzed in [Vartziotis and Wipper, 2010], and an extension to hexahedra and other general polyhedral elements is detailed in [Vartziotis and Wipper, 2012].

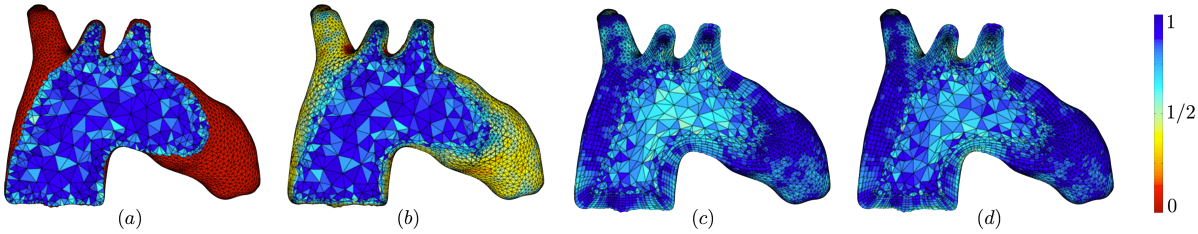


Figure 2.27: Initial and smoothed aorta meshes with elements colored according to their mean ratio [Vartziotis and Wipper, 2012]: (a) initial mesh with prismatic boundary layer; (b) Laplace smoothing; (c) global optimization; (d) GETMe optimization.

Optimization by Topological Operations Local topological operations such as edge/face swaps, elimination of nodes, edges/faces, and elements, are effective means to improve the quality of a mesh. These operations can be carried out based on an appropriate shape measure, and/or according to some topological considerations to create as much as possible a balanced structure in the connection among nodes, edges, faces, and elements. The related literature is wide, and we only report here some examples to give a hint on how shape measures can be combined to mesh optimization (or also *re-meshing*). We address the interested reader to [Alliez et al., 2008] for a more detailed treatment of the subject.

General optimization algorithms, for instance, automatically eliminate valence-three nodes (nodes with three incident edges) from a triangle mesh (Figure 2.28(a)). This is commonly considered a safe operation, as it reduces the number of elements and edges and the quality of the newly created triangle is typically higher than those of the old ones. Similarly, short edges smaller than a certain threshold are eliminated by shrinking the two related triangles to line segments, and small triangles are eliminated by shrinking them to a point (Figure 2.29(b,c)). Analogous automatic operations can be defined for quad, tet, and hex-meshes [Misztal et al., 2009], but only if there is little or no ambiguity on how to adjust the connectivity after an element is removed or modified.

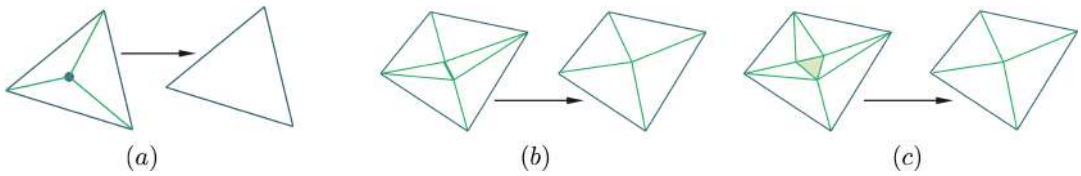


Figure 2.28: Elimination of a valence-three node (a), a short edge (b), and a small triangle (c) from a triangle mesh.

The situation changes if we want to remove a valence-four node from a triangle mesh, as in Figure 2.29(a). In this case, we obtain a quadrilateral element that can be triangulated in two different ways (along the two diagonals), and we need a criterion to establish which one to choose. Quality metrics are employed in topological optimizations every time in which multiple configurations are acceptable, to indicate the best one. Other examples are the elimination of a valence-three node in a quad-mesh and the diagonal swap between couples of adjacent triangles or quads, see Figure 2.29(b,c).

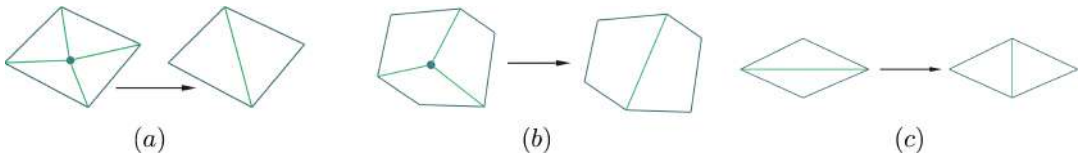


Figure 2.29: Elimination of a valence-four node (a) from a tri-mesh, a short edge (b) from a quad-mesh, and a small triangle (c) from a tri-mesh.

The preferred configurations for local transformations are those whose mean geometric shape qualities are maximized. Although the resulting mesh would certainly be different, the optimization by local transformations can be done with respect to any valid quality indicator. In three dimensions, apart from tetrahedral meshes, optimization of non-simplicial meshes by means of topological operations has not yet been formulated. Pure hexahedral meshes are very often generated by mapping and sweeping methods, which are structures meshed by the nature of their generation, and hence, there is no need to rebuild their topology. A solution for the element connections, in general, is difficult enough to leave any room for topological optimization by a change in element types for which other possible solutions would have already been evaluated right at the spot when the problem first arises. In other words, topological optimization of polyhedral meshes might be possible; however, it is unlikely to be cost-effective as local element swaps can only be applied to very limited locations if any, even after time-consuming rigorous analysis [Lo, 2014].

CHAPTER 3

A Mesh Quality Indicator for the Virtual Element Method

In this chapter, we define a new mesh quality indicator specifically designed for the Virtual Element Method (VEM). In Section 3.1 we formulate the VEM for two and three-dimensional meshes. In Section 3.2 we report the main geometrical assumptions typically made on polygonal and polyhedral meshes in order to ensure optimal convergence rates of the VEM. We report in Section 3.3 a list of reference papers that prove some convergence results for the VEM under different sets of the above geometrical assumptions. Based on these assumptions, in Section 3.4 we present a new mesh quality indicator for polytopal meshes, specifically designed to measure the geometrical quality of a tessellation from the particular point of view of a VEM solver. This chapter is the summary of a collection of articles published in the last years [Sorgente et al., 2021b, Sorgente et al., 2021a, Sorgente et al., 2022b, Sorgente et al., 2022a, Sorgente et al., 2022c].

3.1 The Virtual Element Method

The Virtual Element Method [Beirão da Veiga et al., 2013] is a Galerkin projection method such as the Finite Element Method. The major difference between VEM and FEM is that VEM does not require the explicit evaluation of the basis functions and their gradients, which are integrated into the variational formulation. In the VEM, the basis functions are formally defined as the solutions to suitable partial differential equation problems formulated in every mesh element, and they are dubbed as *virtual* since they are never explicitly evaluated. The resulting schemes can be proved to be consistent with polynomials of a given degree, and this property determines the accuracy of the discretization, while the stability of the method, which implies its well-posedness, is ensured by introducing in the formulation a suitable stabilization term.

This computational approach is extremely powerful and offers indeed several potential advantages with respect to the FEM. In fact, we can easily build approximation spaces that work on very general meshes, including meshes whose elements are generic-shaped polygons in 2D and polyhedra in 3D. An incomplete list of significant applications on general meshes includes, for example, the works of Refs. [Antonietti et al., 2018, Brezzi and Marini, 2013, Beirão da Veiga and Manzini, 2014, Beirão da Veiga and Manzini, 2015, Beirão da Veiga et al., 2019b, Beirão da Veiga et al., 2019a, Beirão da Veiga et al., 2021, Benedetto et al., 2014, Berrone et al., 2018, Benvenuti et al., 2019, Cangiani et al., 2016, Cangiani et al., 2017a, Cangiani et al., 2017b, Certik et al., 2018, Certik et al., 2019, Gardini et al., 2019, Mora et al., 2015, Natarajan et al., 2015, Paulino and Gain, 2015, Perugia et al., 2016, Wriggers et al., 2016]. A detailed description of the state of the art can also be found in the very recent collection of thematic articles [Antonietti et al., 2021a].

We define the virtual element method for two and three-dimensional meshes for the elliptic model problem with Dirichlet boundary conditions. There is no abuse of notation in adopting the same notation for the 2D and the 3D case, as the two cases are totally disjoint.

3.1.1 Notation

We use the standard definition and notation of Sobolev spaces, norms, and seminorms, cf. [Adams and Fournier, 2003]. Let k be a non-negative integer number. The Sobolev space $H^k(\omega)$ consists of all square integrable functions with all square integrable weak derivatives up to order k that are defined on the open, bounded, connected subset ω of $\mathbb{R}^d$, $d = 1, 2, 3$. As usual, if $k = 0$, we prefer the notation $L^2(\omega)$. The norm and seminorm in $H^k(\omega)$ are denoted by $\|\cdot\|_{k,\omega}$ and $|\cdot|_{k,\omega}$, while for the inner product in $L^2(\omega)$ we prefer the integral notation. We denote the space of polynomials of degree less than or equal to $k \geq 0$ on ω by $\mathbb{P}_k(\omega)$ and conventionally assume that $\mathbb{P}_{-1}(\omega) = \{0\}$.

Mesh Generalities The virtual element method is formulated on a mesh family $\mathcal{T} = \{\Omega_h\}_h$, where each mesh Ω_h is a partition of the computational domain Ω into non-overlapping polygonal (polyhedral) elements E, and it is labeled by the mesh size parameter h, defined below.

- In two dimensions, a polygonal element E is a compact subset of $\mathbb{R}^2$ with boundary ∂E, area $|E|$, barycenter $\mathbf{x}_E = (x_E, y_E)^T$, and diameter $h_E = \sup_{\mathbf{x},\mathbf{y}\in E} |\mathbf{x} - \mathbf{y}|$. A mesh edge e is a straight one-dimensional subset of $\mathbb{R}^2$ with length h_e, mid-point $\mathbf{x}_\mathsf{e} = (x_\mathsf{e}, y_\mathsf{e})$, and a local coordinate system s defined on it. A mesh vertex v has a two-dimensional position vector $\mathbf{x}_\mathsf{v}$.

- In three dimensions, a polyhedral element E is a compact subset of $\mathbb{R}^3$ with boundary ∂E, volume $|E|$, barycenter $\mathbf{x}_E = (x_E, y_E, z_E)^T$, and diameter $h_E = \sup_{\mathbf{x},\mathbf{y}\in E} |\mathbf{x} - \mathbf{y}|$. A mesh face f is a planar, two-dimensional subset of $\mathbb{R}^3$ with area $|\mathsf{f}|$, barycenter $\mathbf{x}_\mathsf{f} = (x_\mathsf{f}, y_\mathsf{f}, z_\mathsf{f})^T$,

diameter $h_{\mathsf{f}} = \sup_{\mathbf{x},\mathbf{y}\in\mathsf{f}} |\mathbf{x}-\mathbf{y}|$, and a local coordinate system (ξ,η) defined on it. A mesh edge e is a straight one-dimensional subset of $\mathbb{R}^3$ with length h_{e}, mid-point $\mathbf{x}_{\mathsf{e}} = (x_{\mathsf{e}}, y_{\mathsf{e}}, z_{\mathsf{e}})$, and a local coordinate system s defined on it. A mesh vertex v has a three-dimensional position vector $\mathbf{x}_{\mathsf{v}}$.

We denote the set of mesh faces (only in three dimensions) by $\mathcal{F}_h$, the set of mesh edges by $\mathcal{E}_h$, and the set of mesh vertices by $\mathcal{V}_h$. The set of the mesh elements Ω_h form a finite cover of Ω such that $\Omega = \cup_{E\in\Omega_h} E$, and the mesh size labeling each mesh Ω_h is defined by $h = \max_{E\in\Omega_h} h_E$. We assume that the mesh sizes of the mesh family $\mathcal{T}$ are in a countable subset $\mathcal{H}$ of the segment $(0, h_0)$ for some $h_0 < \infty$, and having 0 as its unique accumulation point.

Monomial Basis On every mesh Ω_h, we denote the space of polyomials of degree k defined on E, f, and e by $\mathbb{P}_k(E)$, $\mathbb{P}_k(\mathsf{f})$ and $\mathbb{P}_k(\mathsf{e})$, respectively, and the space of piecewise discontinuous polynomials of degree k on the whole mesh Ω_h by $\mathbb{P}_k(\Omega_h)$. Accordingly, if $q \in \mathbb{P}_k(\Omega_h)$ then it holds that $q_{|E} \in \mathbb{P}_k(E)$ for all $E \in \Omega_h$.

These polynomial spaces are expressed through an appropriate polynomial basis. In the 2D implementation, we consider the orthogonal basis on every mesh edge through the univariate Legendre polynomials, and inside every mesh cell provided by the Gram-Schmidt algorithm applied to the standard monomial basis. In the 3D implementation, we consider the following basis of $\mathbb{P}_k(E)$ in each element E:

$$m_0^{3\mathrm{D}}(x,y,z) = 1, \quad m_1^{3\mathrm{D}}(x,y,z) = \frac{x-x_E}{h_E}, \quad m_2^{3\mathrm{D}}(x,y,z) = \frac{y-y_E}{h_E},$$
$$m_3^{3\mathrm{D}}(x,y,z) = \frac{z-z_E}{h_E}, \quad m_4^{3\mathrm{D}}(x,y,z) = \left(\frac{x-x_E}{h_E}\right)^2, \quad \dots$$

Similarly, we consider the following basis of $\mathbb{P}_k(\mathsf{f})$ on each face f with center $\mathbf{x}_{\mathsf{f}} = (\xi_{\mathsf{f}}, \eta_{\mathsf{f}})^T$, using the local coordinate system (ξ,η):

$$m_0^{2\mathrm{D}}(\xi,\eta) = 1, \quad m_1^{2\mathrm{D}}(\xi,\eta) = \frac{\xi-\xi_{\mathsf{f}}}{h_{\mathsf{f}}}, \quad m_2^{2\mathrm{D}}(\xi,\eta) = \frac{\eta-\eta_{\mathsf{f}}}{h_{\mathsf{f}}}, \quad m_3^{2\mathrm{D}}(\xi,\eta) = \left(\frac{\xi-\xi_{\mathsf{f}}}{h_{\mathsf{f}}}\right)^2, \quad \dots$$

and the following basis of $\mathbb{P}_k(\mathsf{e})$ on every edge e with center $\mathbf{x}_{\mathsf{e}} = s_{\mathsf{e}}$, using the local coordinate system $s \in [0, h_{\mathsf{e}}]$:

$$m_0^{1\mathrm{D}}(s) = 1, \quad m_1^{1\mathrm{D}}(s) = \frac{s-s_{\mathsf{e}}}{h_{\mathsf{e}}}, \quad m_2^{1\mathrm{D}}(s) = \left(\frac{s-s_{\mathsf{e}}}{h_{\mathsf{e}}}\right)^2, \quad \dots$$

Note that these bases are orthogonal only for $k = 1$, and for $k > 1$ they can be orthogonalized through a Gram-Schmidt procedure. In the one-dimensional case, the orthogonalization provides the Legendre polynomials, up to a scaling factor [Funaro, 1997, Appendix A].

The Projection Operators The definition of the virtual element space requires two particular operators, that are used to approximate the virtual element functions in terms of polynomials. As the virtual element space has not been presented yet, we generically define the operators over the spaces $H^1(E)$ and $L^2(E)$. We will show in Section 3.1.3 how the virtual element space is a subset of $H^1(\Omega)$ and $L^2(\Omega)$, and how we can therefore use and compute these operators.

Let Ω_h be a two or three-dimensional mesh, and $E \in \Omega_h$ a polytopal element. For any $v \in H^1(E)$, the *elliptic projection operator* of order k over E is a function $\Pi_k^{\nabla,E} : H^1(E) \to \mathbb{P}_k(E)$ defined by:

$$\int_E \nabla \Pi_k^{\nabla,E} v \cdot \nabla q \, dV = \int_E \nabla v \cdot \nabla q \, dV \quad \forall q \in \mathbb{P}_k(E) \tag{3.1}$$

$$\int_{\partial E} (\Pi_k^{\nabla,E} v - v) \, dS = 0. \tag{3.2}$$

Equation (3.2) allows us to remove the kernel of the gradient operator from the definition of $\Pi_k^{\nabla,E}$, so that the k-degree polynomial $\Pi_k^{\nabla,E} v$ is uniquely defined for every $v \in H^1(E)$. Moreover, projector $\Pi_k^{\nabla,E}$ is a polynomial-preserving operator, i.e., $\Pi_k^{\nabla,E} q = q$ for every $q \in \mathbb{P}_k(E)$. We can also define a global projection operator $\Pi_k^{\nabla} : H^1(\Omega) \to \mathbb{P}_k(\Omega_h)$, which is such that $(\Pi_k^{\nabla} v)_{|E} = \Pi_k^{\nabla,E}(v_{|E}) \; \forall E \in \Omega_h$.

The *orthogonal projection operator* of order k over E is the function $\Pi_k^{0,E} : L^2(E) \to \mathbb{P}_k(E)$, solution of the variational problem:

$$\int_E \Pi_k^{0,E} v \, q \, dV = \int_E v \, q \, dV \qquad \forall q \in \mathbb{P}_k(E). \tag{3.3}$$

As we have done for the elliptic operator, we can define a global projection operator $\Pi_k^0 : L^2(\Omega) \to \mathbb{P}_k(\Omega_h)$ onto the space of discontinuous polynomials of degree at most k built on mesh Ω_h. This operator is given by taking the elemental L^2-orthogonal projection $\Pi_k^{0,E} v$ in every mesh element E, so that $(\Pi_k^0 v)_{|E} = \Pi_k^{0,E}(v_{|E})$.

3.1.2 The Model Problem

Let $\Omega \subset \mathbb{R}^2$ ($\mathbb{R}^3$) be an open bounded domain with Lipschitz boundary Γ. For exposition's sake, we assume that Ω is a polygonal (polyhedral) domain and its boundary Γ is given by the union of a subset of the edges in $\mathcal{E}_h$ (faces in $\mathcal{F}_h$). We consider the diffusion problem

$$\begin{aligned} -\Delta u &= f \qquad \text{in } \Omega, \\ u &= f \qquad \text{on } \Gamma, \end{aligned}$$

where $f \in L^2(\Omega)$ is the load term and $g \in H^{\frac{1}{2}}(\Gamma)$ the Dirichlet boundary data. The variational form of this problem reads as:

$$\textit{Find } u \in V_g \textit{ such that} \quad a(u,v) = F(v) \qquad \forall v \in V_0, \tag{3.4}$$

where the bilinear form $a(\cdot,\cdot) : H^1(\Omega) \times H^1(\Omega) \to \mathbb{R}$ is given by

$$a(u,v) = \int_\Omega \nabla u \cdot \nabla v \, d\mathbf{x},$$

and the right-hand side linear functional $F : L^2(\Omega) \to \mathbb{R}$ is given by

$$F(v) = \int_\Omega f v \, d\mathbf{x},$$

where $V_g = \{v \in H^1(\Omega) : v_{|\Gamma} = g\}$ and $V_0 = \{v \in H^1(\Omega) : v_{|\Gamma} = 0\}$. The well-posedness of this problem follows from the coercivity and continuity of the bilinear form on the left-hand side of (3.4) and the boundedness on the linear functional of the right-hand side of (3.4). This can be proved by an application of the Lax-Milgram theorem, see [Scott and Brenner, 2008, Theorem 2.7.7].

The Virtual Element Approximation To ease the presentation, we consider the case of homogeneous Dirichlet boundary conditions, i.e., $g = 0$ in (3.4), the extension to the non-homogeneous case being deemed as straightforward. Such a case is also considered in the numerical experiments carried out in this work. The virtual element approximation of equation (3.4) reads as:

$$\textit{Find } u_h \in V_k^h \textit{ such that} \quad a_h(u_h, v_h) = F_h(v_h) \qquad v_h \in V_k^h, \tag{3.5}$$

where

- V_k^h is the virtual element space, containing the virtual element functions v_h that approximate the functions in $H_0^1(\Omega)$;
- the bilinear form $a_h : V_k^h \times V_k^h \to \mathbb{R}$ is the virtual element approximation of the bilinear form $a(\cdot,\cdot)$;
- the linear functional $F_h : V_k^h \to \mathbb{R}$ is the virtual element approximation of the linear functional $F(\cdot)$ using f_h, which is an element of the dual space $(V_k^h)^*$.

We review the construction of these mathematical objects in the next sections.

3.1.3 The Virtual Element Space

There exist various ways of defining the virtual element space V_k^h. In this work, we use the *conforming* space defined through the *enhancement* strategy, following References [Ahmad et al., 2013, Beirão da Veiga et al., 2013].

Two-Dimensional Virtual Element Space Let $E \in \Omega_h$ be a generic element of a two-dimensional mesh. The *conforming virtual element space* V_k^h of order k built on Ω_h is obtained by gluing together the elemental approximation spaces denoted by $V_k^h(E)$, so that

$$V_k^h := \Big\{ v_h \in H_0^1(\Omega) \,:\, v_{h|E} \in V_k^h(E) \quad \forall E \in \Omega_h \Big\}. \tag{3.6}$$

The local virtual element space $V_k^h(E)$ is defined in accordance with the *enhancement strategy* introduced in [Ahmad et al., 2013]:

$$\begin{aligned} V_k^h(E) = \Big\{ v_h \in H^1(E) \,:\, & v_{h|\partial E} \in C^0(\partial E),\, v_{h|\mathsf{e}} \in \mathbb{P}_k(\mathsf{e})\, \forall \mathsf{e} \in \partial E, \\ & \Delta v_h \in \mathbb{P}_k(E), \text{ and} \\ & \int_E (v_h - \Pi_k^{\nabla,E} v_h)\, q\, dV = 0 \;\forall q \in \mathbb{P}_k(E) \backslash \mathbb{P}_{k-2}(E) \Big\}, \end{aligned} \tag{3.7}$$

where the elliptic projection operator $\Pi_k^{\nabla,E}$ defined in (3.2) is here applied to $V_k^h(E) \subset H_0^1(E)$, and $\mathbb{P}_k(E) \backslash \mathbb{P}_{k-2}(E)$ is the space of polynomials of degree equal to $k-1$ and k. We recall that $\mathbb{P}_k(E)$ and $\mathbb{P}_k(\mathsf{e})$ are the linear spaces $\mathbb{P}_k$ respectively defined over an element E or an edge e according to our notation. By definition, the space $V_k^h(E)$ contains $\mathbb{P}_k(E)$ and the global space V_k^h is a conforming subspace of $H_0^1(\Omega)$.

The degrees of freedom (DOFs) of a virtual element function $v_h \in V_k^h(E)$ are given by the following set of values [Beirão da Veiga et al., 2013]:

(D1) for $k \geq 1$, the values of v_h at the vertices of E;

(D2) for $k \geq 2$, the values of v_h at the $k-1$ internal points of the $(k+1)$-point Gauss-Lobatto quadrature rule on every edge $\mathsf{e} \in \partial E$;

(D3) for $k \geq 2$, the cell moments of v_h of order up to $k-2$ on element E:

$$\frac{1}{|E|} \int_E v_h\, q\, dV \quad \forall q \text{ in a basis of } \mathbb{P}_{k-2}(E).$$

These set of values are unisolvent in $V_k^h(E)$, cf. [Beirão da Veiga et al., 2013]; hence, every virtual element function is uniquely identified by them. The degrees of freedom of a virtual element function in the global space V_k^h are given by collecting the elemental degrees of freedom **(D1)**-**(D3)**. Their unisolvence in V_k^h is an immediate consequence of their unisolvence in every elemental space $V_k^h(E)$.

Every projection $\Pi_k^{\nabla,E} v_h$ of a virtual element function $v_h \in V_k^h(E)$ is computable from the degrees of freedom of v_h associated with element E. From the degrees of freedom of a virtual element function $v_h \in V_k^h(E)$ we can also compute the orthogonal projections $\Pi_k^{0,E} v_h \in \mathbb{P}_k(E)$ defined in (3.3), cf. [Ahmad et al., 2013].

Three-Dimensional Virtual Element Space Exactly as for the two-dimensional case, the global virtual element space V_k^h for $\Omega \subset \mathbb{R}^3$ is defined by "gluing" all the elemental virtual element spaces in a conforming way

$$V_k^h := \Big\{ v_h \in H_0^1(\Omega) : \, v_{h|E} \in V_k^h(E) \quad \forall E \in \Omega_h \Big\}. \tag{3.8}$$

The difference, however, is that the three-dimensional conforming elemental spaces $V_k^h(E)$ are built recursively on top of the virtual element spaces defined on the polyhedral faces.

Under the assumptions presented in Section 3.1.1, each polyhedral face f is a two-dimensional planar polygon. The virtual element space $V_k^h(\mathsf{f})$ on f is defined through an intermediate "extended" space $\widetilde{V}_k^h(\mathsf{f})$:

$$\widetilde{V}_k^h(\mathsf{f}) := \Big\{ v_h \in H^1(\mathsf{f}) \cap C^0(\mathsf{f}) : \, v_{h|\mathsf{e}} \in \mathbb{P}_k(\mathsf{e}) \, \forall \mathsf{e} \in \partial \mathsf{f}, \, \Delta v_h \in \mathbb{P}_k(\mathsf{f}) \Big\}. \tag{3.9}$$

The degrees of freedom of the functions in $\widetilde{V}_k^h(\mathsf{f})$ are obtained by replacing the two-dimensional element E in **(D1)-(D3)** with a two-dimensional face f of a three-dimensional element, and they are unisolvent. The elliptic (and the orthogonal) projection operator defined in (3.2) (and (3.3)) can be restricted over $\widetilde{V}_k^h(\mathsf{f})$ exactly as we did for the two-dimensional elements, and it can be computed from the degrees of freedom of the functions $v_h \in \widetilde{V}_k^h(\mathsf{f})$. We can now restrict the space $\widetilde{V}_k^h(\mathsf{f})$, imposing the condition regarding $\Pi_k^{\nabla,\mathsf{f}}$ and get the virtual element space over a face:

$$V_k^h(\mathsf{f}) := \Big\{ v_h \in \widetilde{V}_k^h(\mathsf{f}) : \, \int_{\mathsf{f}} v_h q = \int_{\mathsf{f}} (\Pi_k^{\nabla,\mathsf{f}} v_h) q \quad \forall q \in \mathbb{P}_k(\mathsf{f}) \backslash \mathbb{P}_{k-2}(\mathsf{f}) \Big\}. \tag{3.10}$$

By definition, the $k-$degree polynomials over f are a subspace of $V_k^h(\mathsf{f})$; formally, we can write that $\mathbb{P}_k(\mathsf{f}) \subseteq V_k^h(\mathsf{f}) \subseteq \widetilde{V}_k^h(\mathsf{f})$. The projection $\Pi_k^{\nabla,\mathsf{f}} v_h$ is computable from the degrees of freedom of v_h associated with f, cf. [Beirão da Veiga et al., 2013, Beirão da Veiga et al., 2014].

Once we have defined the virtual element spaces on the faces, we can build the ones for the elements. Let E denote a generic three-dimensional element with boundary ∂E and $\mathsf{f} \in \partial E$ a generic polygonal face of E. We introduce the elemental boundary space

$$B_h(\partial E) := \Big\{ v_h \in C^0(\partial E) : \, v_{h|\mathsf{f}} \in V_k^h(\mathsf{f}) \quad \forall \mathsf{f} \in \partial E \Big\}. \tag{3.11}$$

The functions in $B_h(\partial E)$ are continuous polynomials across the face edges, and their restriction to a given face f belong to the virtual element space $V_k^h(\mathsf{f})$ defined in (3.10). We use (3.11) to introduce a preliminary "extended" space $\widetilde{V}_k^h(E)$:

$$\widetilde{V}_k^h(E) := \Big\{ v_h \in H^1(E) : \, v_{h|\partial E} \in B_h(\partial E), \, \Delta v_h \in \mathbb{P}_k(E) \Big\}. \tag{3.12}$$

The degrees of freedom of the functions in $\widetilde{V}_k^h(E)$ are naturally derived from **(D1)-(D3)**, opportunely replacing edges with faces, and a further set of values over the faces is required:

(D1) for $k \geq 1$, the values of v_h at the vertices of E;

(D2) for $k \geq 2$, the values of v_h at the $k-1$ internal points of the $(k+1)$-point Gauss-Lobatto quadrature rule on every edge $\mathsf{e} \in \partial E$;

(D3) for $k \geq 2$, the face moments of v_h of order up to $k-2$ on face f:

$$\frac{1}{|\mathsf{f}|} \int_{\mathsf{f}} v_h \, q \, dS \quad \forall q \text{ in a basis of } \mathbb{P}_{k-2}(\mathsf{f}).$$

(D4) for $k \geq 2$, the cell moments of v_h of order up to $k-2$ on element E:

$$\frac{1}{|E|} \int_{E} v_h \, q \, dV \quad \forall q \text{ in a basis of } \mathbb{P}_{k-2}(E).$$

They are unisolvent in $\widetilde{V}_k^h(E)$, and allow the computation of elliptic and orthogonal projections $\Pi_k^{\nabla,E} v_h$ and $\Pi_k^{0,E} v_h$ of the functions in this space. Then, as done for (3.10), we define the elemental virtual element space on E:

$$V_k^h(E) := \left\{ v_h \in \widetilde{V}_k^h(E) : \int_E v_h q = \int_E (\Pi_k^{\nabla,E} v_h) q \quad \forall q \in \mathbb{P}_k(E) \backslash \mathbb{P}_{k-2}(E) \right\}. \tag{3.13}$$

It holds that $\mathbb{P}_k(E) \subseteq V_k^h(\mathsf{f}) \subseteq \widetilde{V}_k^h(E)$. The projection $\Pi_k^{\nabla,E} v_h$ of a virtual element function $v_h \in \widetilde{V}_k^h(E)$ only depends on its degrees of freedom, which uniquely characterize any $v_h \in V_k^h(E)$. The unisolvence of the degrees of freedom in $V_k^h(E)$ is proved by using the same arguments as in [Ahmad et al., 2013], and the unisolvence in the global space V_k^h is a consequence of their unisolvence in each elemental space.

Approximation Properties in the Virtual Element Space Under suitable regularity assumptions on the mesh family used, we can prove the following estimates on the projection and interpolation operators:

1. for every s with $1 \leq s \leq k+1$ and for every $w \in H^s(E)$ there exists a $w_\pi \in \mathbb{P}_k(E)$ such that

$$|w - w_\pi|_{0,E} + h_E \, |w - w_\pi|_{1,E} \leq C h_E^s \, |w|_{s,E} \, ;$$

2. for every s with $2 \leq s \leq k+1$, for every h, for all $E \in \Omega_h$ and for every $w \in H^s(E)$ there exists a $w_I \in V_k^h(E)$ such that

$$|w - w_I|_{0,E} + h_E \, |w - w_I|_{1,E} \leq C h_E^s \, |w|_{s,E} \, .$$

In these inequalities, C is a real positive constant depending only on the polynomial degree k and on some mesh regularity constants.

3.1.4 The Virtual Element Functionals

Over the virtual element space, through the elliptic and orthogonal projections, we define the virtual element bilinear form $a_h(\cdot,\cdot) : V_k^h \times V_k^h \to \mathbb{R}$, and the forcing term $F_h : V_k^h \to \mathbb{R}$.

The Bilinear Form Following the "VEM gospel", we write the discrete bilinear form $a_h(\cdot,\cdot)$ as the sum of elemental contributions

$$a_h(u_h, v_h) = \sum_{E\in\Omega_h} a_h^E(u_h, v_h),$$

where every elemental contribution is a bilinear form $a_h^E(\cdot,\cdot) : V_k^h(E) \times V_k^h(E) \to \mathbb{R}$ designed to approximate the corresponding elemental bilinear form $a^E(\cdot,\cdot) : H^1(E) \times H^1(E) \to \mathbb{R}$,

$$a^E(v, w) = \int_E \nabla v \cdot \nabla w \, dV, \quad \forall v, w \in H_0^1(E). \tag{3.14}$$

The bilinear form $a_h^E(\cdot,\cdot)$ on each element E is given by

$$a_h^E(u_h, v_h) = \int_E \nabla\Pi_k^{\nabla,E} u_h \cdot \nabla\Pi_k^{\nabla,E} v_h \, dV + S_h^E\Big((I - \Pi_k^{\nabla,E})u_h, (I - \Pi_k^{\nabla,E})v_h\Big). \tag{3.15}$$

The bilinear form $S_h^E(\cdot,\cdot)$ in the definition of $a_h^E(\cdot,\cdot)$ provides the stability term and can be any computable, symmetric, positive definite bilinear form defined on $V_k^h(E)$ for which there exist two positive constants σ_* and σ^* such that

$$\sigma_* a^E(v_h, v_h) \le S_h^E(v_h, v_h) \le \sigma^* a^E(v_h, v_h) \quad \forall v_h \in V_k^h(E) \cap \ker(\Pi_k^{\nabla,E}). \tag{3.16}$$

The inequalities in (3.16) imply that $S_h^E(\cdot,\cdot)$ scales like $a^E(\cdot,\cdot)$ with respect to h_E, since we assume that σ_* and σ^* are independent on the mesh parameter h_E. Also, the stabilization term in the definition of $a_h^E(\cdot,\cdot)$ is zero if at least one of its two entries is a polynomial of degree (at most) k, since $\Pi_k^{\nabla,E}$ is a polynomial preserving operator. The stabilization term and, in particular, condition (3.16), is designed so that $a_h^E(\cdot,\cdot)$ satisfies the two fundamental properties:

- *k-consistency*: for all $v_h \in V_k^h(E)$ and for all $q \in \mathbb{P}_k(E)$ it holds that

$$a_h^E(v_h, q) = a^E(v_h, q); \tag{3.17}$$

- *stability*: there exist two positive constants α_*, α^*, independent of h and E, such that

$$\alpha_* a^E(v_h, v_h) \le a_h^E(v_h, v_h) \le \alpha^* a^E(v_h, v_h) \quad \forall v_h \in V_k^h(E). \tag{3.18}$$

These properties are necessary for the well-posedness of problem (3.5).

Multiple choices exist in the literature for the stabilization term. In our implementation, we consider the stabilization proposed in [Mascotto, 2018]:

$$S_h^E(v_h, w_h) = \sum_{i=1}^{N^{\text{dofs}}} \sigma_i \text{DOF}_i(v_h)\text{DOF}_i(w_h), \tag{3.19}$$

where $\mathrm{DOF}_i(v_h)$ is the i-th degree of freedom of the basis function v_h, $\sigma_i = \max\{\mathcal{A}^E_{ii}, 1\}$ and $\mathcal{A}^E = (\mathcal{A}^E_{ij})$ is the matrix resulting from the implementation of the first term in the bilinear form $a^E_h(\cdot,\cdot)$. Let ϕ_i be the i-th "canonical" basis functions generating the virtual element space, which is the function in $V^h_k(E)$ whose i-th degree of freedom for $i = 1, \ldots, N^{\mathrm{dofs}}$ (according to a suitable renumbering of the degrees of freedom in **(D1)**, **(D2)**, **(D3)**, and **(D4)** in dimension three), has value equal to 1 and all other degrees of freedom are zero. These basis functions are unknown in the virtual element framework, but their projections $\Pi^{0,E}_{k-1}\nabla\phi_i$ (and $\nabla\Pi^{\nabla,E}_k\phi_i$) are computable from their degrees of freedom. With this notation, the i,j-th entry of matrix $\mathcal{A}^E$ is given by

$$\mathcal{A}^E_{ij} := a^E(\Pi^{\nabla,E}_k\phi_i, \Pi^{\nabla,E}_k\phi_j).$$

The stabilization in (3.19) is sometimes called the "*D-recipe stabilization*" in the virtual element literature, and contains the so called "*dofi-dofi (dd) stabilization*" originally proposed in [Beirão da Veiga et al., 2013] for the 2D case, as the special case with $\mathcal{A}_{ii} = 1$:

$$S^{E,\mathrm{dd}}_h(v_h, w_h) = \sum_{i=1}^{N^{\mathrm{dofs}}} \mathrm{DOF}_i(v_h)\mathrm{DOF}_i(w_h). \tag{3.20}$$

The formulation (3.19) generalizes (3.20) to the $d-$dimensional case, because it automatically considers the proper scaling of $S^E_h(\cdot,\cdot)$ proportional to h^{d-2}_E, with $d = 2, 3$.

The Forcing Term To approximate the right-hand side of (3.5), we split it into the sum of elemental contributions, and every local linear functional is approximated by using the orthogonal projection $\Pi^{0,E}_k v_h$:

$$F_h(v_h) = \sum_{E\in\Omega_h} (f, \Pi^{0,E}_k v_h)_E, \quad \text{where} \quad (f, \Pi^{0,E}_k v_h)_E = \int_E f\,\Pi^{0,E}_k v_h\,dV. \tag{3.21}$$

Estimates of such approximation are found in [Ahmad et al., 2013, Section 5.8].

3.2 Geometrical Assumptions for the VEM

Various geometrical (or *regularity*) assumptions have been proposed in the literature to ensure the convergence of the VEM and optimal estimates of the approximation error with respect to different norms. These assumptions guarantee that all elements of all meshes in a given mesh family in the refinement process are sufficiently regular. In this section, we overview the geometrical assumptions introduced in the VEM literature to guarantee the convergence of the method, reporting the work proposed in [Sorgente et al., 2022b, Sorgente et al., 2022a]. We point out that, differently from the mesh quality indicators of Section 2.3, geometrical assumptions are not meant to provide a quality score of a mesh or an element. Instead, they

simply delimit the class of meshes for which the VEM is ensured to converge with the optimal rate.

We start by reviewing the geometrical assumptions that appeared in the VEM literature since their definition in [Beirão da Veiga et al., 2013]. They are typically first defined for two-dimensional meshes and then properly extended to the three-dimensional case. Note that these assumptions are defined for a single mesh Ω_h, but the conditions contained in them are required to hold independently of the mesh size h. As a consequence, when an assumption is imposed to a mesh family $\mathcal{T} = \{\Omega_h\}_h$, it has to be verified simultaneously by every $\Omega_h \in \mathcal{T}$.

It is well-known from the FEM literature that the approximation properties of a method depend on specific assumptions about the geometry of the mesh elements. Classical examples of geometrical assumptions for a family of triangulations $\{\Omega_h\}_{h\to 0}$, are the ones introduced in [Ciarlet, 2002] and [Zlámal, 1968], respectively:

- *Shape regularity condition:* there exists a real number $\gamma \in (0, 1)$, independent of h, such that for any triangle $E \in \Omega_h$ we have

  $$r_E \geq \gamma h_E, \tag{3.22}$$

  being h_E the longest edge in E and r_E its inradius;

- *Minimum angle condition:* there exists an angle $\alpha_0 > 0$, independent of h, such that for any triangle $E \in \Omega_h$ we have

  $$\alpha_E \geq \alpha_0, \tag{3.23}$$

  being α_E the minimal angle of E.

When we turn our focus on polytopal meshes, a preliminary consideration is needed on the definition of the polytopal elements. It is commonly accepted, even if not always explicitly specified, that a mesh Ω_h has to be made of a finite number of *simple polytopes* [Beirão da Veiga et al., 2013], i.e. open simply connected sets whose boundary is a non-intersecting curve (or surface) made of a finite number of straight line segments (or linear faces). The other regularity assumptions proposed in the VEM literature to ensure approximation properties have been deduced in analogy to the similar conditions developed for the FEMs.

3.2.1 Assumption G1

The main assumption, which systematically recurs in every VEM paper, is the so-called *star-shapedness* of the mesh elements (Figure 3.1), introduced in Section 2.3.1 among the polytopal quality indicators.

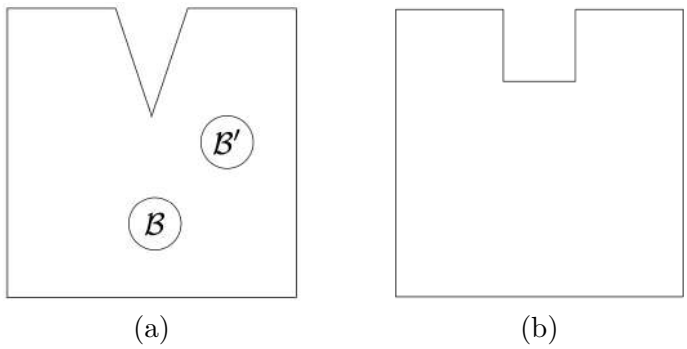


Figure 3.1: The element in (a) is star-shaped with respect to the disk $\mathcal{B}$ but not with respect to the disk $\mathcal{B}'$, while the element in (b) is not star-shaped with respect to any disk [Scott and Brenner, 2008]

Definition 3.1 ($\mathbf{G1^{2D}}$). *There exists a real number $\rho \in (0,1)$, independent of h, such that every polygon $E \in \Omega_h$ is star-shaped with respect to a disc with radius*

$$r_E \geq \rho h_E.$$

We denote r_E the radius of the greatest possible inscribed disk in E and *star center* the center of such disk, where it exists. We stress the fact that, being both ρ and h_E greater than zero, the radius r_E needs to be positive too, in accordance with the definition of star-shapedness. We conventionally say that $r_E = 0$ if E is not star-shaped. Assumption $\mathbf{G1^{2D}}$ is nothing but the polygonal extension of the classical *shape regularity condition* for triangular meshes. In fact, any triangular element E is star-shaped with respect to its maximum inscribed disk (the one with radius r_E), and the diameter h_E coincides with its longest edge. Moreover, $\mathbf{G1^{2D}}$ can also be stated in the following weak form, as specified in [Beirão da Veiga et al., 2013] and more extensively in [Brenner et al., 2017]:

Definition 3.2 ($\mathbf{G1w^{2D}}$). *There exists a real number $\rho \in (0,1)$, independent of h, such that every polygon $E \in \Omega_h$ can be split into a finite number N of disjoint polygonal subcells $E_1, \ldots, E_N$ where, for $j = 1, \ldots, N$,*

- *element E_j is star-shaped with respect to a disc with radius $r_{E_j} \geq \rho h_{E_j}$;*
- *elements E_j and E_{j+1} share a common edge.*

From a practical point of view, assumptions $\mathbf{G1^{2D}}$ and $\mathbf{G1w^{2D}}$ are almost equivalent, and they are treated equivalently in all papers reviewed in Section 3.3.

Assumption $\mathbf{G1^{2D}}$ is naturally extended by imposing the star-shapedness both to the interior of an element and to each of its faces, with the same constant ρ. In the cited works, it is not explicitly considered a weaker piece-wise version of $\mathbf{G1^{3D}}$, but we can easily imagine defining $\mathbf{G1w^{3D}}$ exactly as we did for $\mathbf{G1w^{2D}}$.

Definition 3.3 (**G1$^{\mathbf{3D}}$**). *There exists a real number $\rho \in (0,1)$, independent of h, such that every polyhedron $E \in \Omega_h$ is star-shaped with respect to a ball with radius*

$$r_E \geq \rho h_E,$$

and every face $\mathsf{f} \in \partial E$ is star-shaped with respect to a disc with radius

$$r_{\mathsf{f}} \geq \rho h_{\mathsf{f}}.$$

Assumption **G1$^{\mathbf{2D}}$** (and **G1$^{\mathbf{3D}}$**) plays a key role in most of the theoretical results regarding polytopal methods. It is needed by the Bramble-Hilbert lemma [Scott and Brenner, 2008], an important result on polynomial approximation that is often used for building approximation estimates, and also by the following lemma.

Lemma 3.1. *If a mesh Ω_h satisfies assumption **G1^{2D}** (or **G1^{3D}**), then for all polygons (polyhedra) $E \in \Omega_h$ there exists a mapping $F : \mathcal{B}_1 \to E$, with the Jacobian J of F satisfying*

$$\|J\|_2 \lesssim h, \qquad |\det(J)| \lesssim h^2 \qquad \text{and} \qquad \|J^{-1}\|_2 \lesssim h^{-1},$$

and, for a sufficiently regular u, the following relations hold

$$\|u\|_{0,E} \simeq h_E \|u \circ F\|_{0,\mathcal{B}_1} \qquad \|u\|_{0,\partial E} \simeq h_E^{1/2} \|u \circ F\|_{0,\partial\mathcal{B}_1}$$
$$|u|_{1,E} \simeq |u \circ F|_{1,\mathcal{B}_1} \qquad |u|_{1/2,\partial E} \simeq |u \circ F|_{1/2,\partial\mathcal{B}_1}$$

*where all the implicit constants only depend on the constant ρ from **G1^{2D}** (**G1^{3D}**).*

The symbols $\lesssim$ (and $\simeq$) denote a bound (or an equality) up to a constant that is uniform for all $E \in \Omega_h$. Thanks to the relations in Lemma 3.1, inequalities that we have on the unit ball $\mathcal{B}_1$, such as the Poincaré inequality or the trace inequalities, may be transferred to the polytope E by a "scaling" argument.

3.2.2 Assumption G2

In the very first VEM paper where the method was introduced [Beirão da Veiga et al., 2013], assumption **G1$^{\mathbf{2D}}$** was followed by another condition on the minimum point-to-point distance.

Definition 3.4 (**G2s$^{\mathbf{2D}}$**). *There exists a real number $\rho \in (0,1)$, independent of h, such that for every polygon $E \in \Omega_h$, the distance $d_{i,j}$ between any two vertices v_i, v_j of E satisfies*

$$d_{i,j} \geq \rho h_E.$$

In fact, assumption **G2s$^{\mathbf{2D}}$** (the s stands for "strong") was soon replaced in the following works [Ahmad et al., 2013], [Beirão da Veiga et al., 2017] and [Brenner et al., 2017] by a weaker condition on the length of the elemental edges. This new version allows, for example, the existence of four-sided polygons with equal edges but one diagonal much smaller than the other.

Definition 3.5 (**G2$^{\mathbf{2D}}$**). *There exists a real number $\rho \in (0,1)$, independent of h, such that for every polygon $E \in \Omega_h$, the length h_{e} of every edge $\mathsf{e} \in \partial E$ satisfies*

$$h_{\mathsf{e}} \geq \rho h_E.$$

The Authors consider a single constant ρ for both assumption **G1$^{\mathbf{2D}}$** and **G2$^{\mathbf{2D}}$**, and refer to it as the *mesh regularity constant* or *parameter*. Under assumption **G1w$^{\mathbf{2D}}$** and **G2$^{\mathbf{2D}}$**, it can be proved [Brenner et al., 2017] that the simplicial triangulation of E determined by the star-centers of $E_1, \ldots, E_N$ satisfies the *shape regularity* and the *minimum angle* conditions. The same holds under assumptions **G1$^{\mathbf{2D}}$** and **G2$^{\mathbf{2D}}$**, as a special case of the previous statement. Moreover, assumption **G2$^{\mathbf{2D}}$** implies that for $1 \leq j, k \leq N$ it holds $h_{E_j}/h_{E_k} \leq \rho^{-|j-k|}$.

For a volumetric version of **G2$^{\mathbf{2D}}$**, we set a condition $h_{\mathsf{f}} \geq \rho h_E$ on the element, and substitute it into the analogous condition on the faces $h_{\mathsf{e}} \geq \rho h_{\mathsf{f}}$, obtaining a triple inequality.

Definition 3.6 (**G2$^{\mathbf{3D}}$**). *There exists a real number $\rho \in (0,1)$, independent of h, such that for every polyhedron $E \in \Omega_h$, the length h_{e} of every edge e of every face f satisfies*

$$h_{\mathsf{e}} \geq \rho h_{\mathsf{f}} \geq \rho^2 h_E.$$

As already mentioned in the very first papers, these assumptions are more restrictive than necessary, but at the same time, they allow the VEM to work on very general meshes. For example, Ahmad et al. in [Ahmad et al., 2013] state that:

> *"Actually, we could get away with even more general assumptions, but then it would be long and boring to make precise (among many possible crazy decompositions that nobody will ever use) the ones that are allowed and the ones that are not."*

3.2.3 Assumption G3

In the subsequent papers [Beirão da Veiga et al., 2017] and [Brenner and Sung, 2018], assumption **G1$^{\mathbf{2D}}$** is preserved, but assumption **G2$^{\mathbf{2D}}$** is substituted by the alternative version:

Definition 3.7 (**G3$^{\mathbf{2D}}$**). *There exists a positive integer N, independent of h, such that the number of edges of every polygon $E \in \Omega_h$ is (uniformly) bounded by N.*

The Authors show how assumption **G2$^{\mathbf{2D}}$** implies assumption **G3$^{\mathbf{2D}}$**, but assumption **G3$^{\mathbf{2D}}$** is weaker than assumption **G2$^{\mathbf{2D}}$**, as it allows for edges arbitrarily small with respect to h_E. Both combinations **G1$^{\mathbf{2D}}$**+**G2$^{\mathbf{2D}}$** and **G1$^{\mathbf{2D}}$**+**G3$^{\mathbf{2D}}$** imply that the number of vertices of E and the minimum angle of the simplicial triangulation of E obtained by connecting all the vertices of E to its star-center are controlled by the constant ρ.

Assumption **G3$^{\mathbf{3D}}$** extends the bound from the number of edges to the number of edges and faces of every element.

Definition 3.8 (**G3$^{\mathbf{3D}}$**). *There exists a positive integer N, independent of h, such that the number of edges and faces of every polyhedron $E \in \Omega_h$ is (uniformly) bounded by N.*

Also **G3$^{\mathbf{3D}}$** can be derived from **G1$^{\mathbf{3D}}$** and **G2$^{\mathbf{3D}}$** [Ahmad et al., 2013, Remark 11].

3.2.4 Assumption G4

Another step forward in the refinement of the geometrical assumptions was made by Beirão Da Veiga and Vacca in [Beirão da Veiga and Vacca, 2020]. Besides assuming **G1$^{\mathbf{2D}}$**, the Authors imagine to "unwrap" the boundary ∂E of each element $E \in \Omega_h$ onto an interval of the real line, obtaining a one-dimensional mesh $\mathcal{I}_E$. The mesh $\mathcal{I}_E$ can be subdivided into a number of disjoint sub-meshes $\mathcal{I}_E^1, \ldots, \mathcal{I}_E^N$, corresponding to the edges of E. Then, the following condition is assumed.

Definition 3.9 (**G4$^{\mathbf{2D}}$**). *There exists a real number $\delta > 0$, independent of h, such that for every polygon $E \in \Omega_h$:*

- *the one-dimensional mesh $\mathcal{I}_E$ can be subdivided into a finite number N of disjoint sub-meshes $\mathcal{I}_E^1, \ldots, \mathcal{I}_E^N$;*
- *for each sub-mesh $\mathcal{I}_E^i$, $i = 1, \ldots, N$, it holds that*

$$\frac{\max_{e \in \mathcal{I}_E^i} |e|}{\min_{e \in \mathcal{I}_E^i} |e|} \leq \delta.$$

Each polygon E corresponds to a one-dimensional mesh $\mathcal{I}_E$, but a sub-mesh $\mathcal{I}_E^i \subset \mathcal{I}_E$ might contain more than one edge of E, cf. Figure 3.2. Therefore assumption **G4$^{\mathbf{2D}}$** does not require a uniform bound on the number of edges in each element and does not exclude the presence of small edges. Mesh families created by agglomeration, cracking, gluing, etc.. of existing meshes are admissible according to **G4$^{\mathbf{2D}}$**.

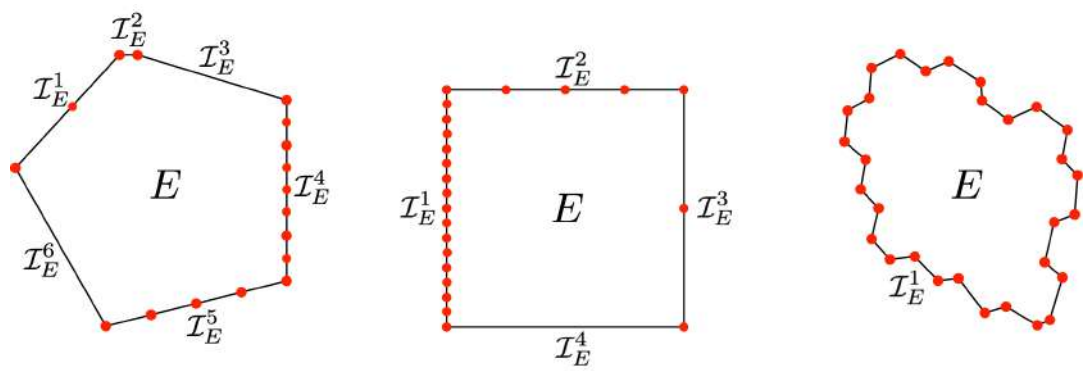


Figure 3.2: Examples of admissible elements according to assumption **G4$^{\mathbf{2D}}$** [Beirão da Veiga and Vacca, 2020]. Red dots indicate the vertices of the element.

A condition similar to $\mathbf{G4^{2D}}$ is presented in [Bertoluzza et al., 2021] for the non-conforming VEM formulation. We do not extend $\mathbf{G4^{2D}}$ to the three-dimensional case because, to the best of our knowledge, such extension has not yet been considered in the analysis of the three-dimensional formulation of the VEM. We point out that there is a fundamental difference between $\mathbf{G1^{2D}}$ ($\mathbf{G1^{3D}}$) on the one hand, and the other assumptions on the other hand. For a fixed h, $\mathbf{G2^{2D}}$, $\mathbf{G3^{2D}}$, and $\mathbf{G4^{2D}}$ (or their respective 3D versions) are always satisfied, as a suitable ρ, N, or δ always exists for any given element. Such assumptions are therefore all about h-independent bounds for sequences of meshes. $\mathbf{G1^{2D}}$ and $\mathbf{G1^{3D}}$, by contrast, can be violated already by an individual non-star-shaped element. This difference will play a role in the definition of the mesh quality indicator in Section 3.4.

3.3 VEM Convergence Results

In this section, we briefly overview the literature on the main results of the convergence analysis of the conforming VEM method, both in two and three dimensions, following the approach proposed in [Sorgente et al., 2021b]. For each article, we explicitly report (if available) the theoretical results and highlight the geometric assumptions used, reporting the abstract energy error, the H^1 error estimate, and the L^2 error. Some of the theoretical results involve a broken H^1-seminorm for functions $v \in H^1(\Omega_h)$, defined as follows:

$$|v|_{h,1} := \left(\sum_{E \in \Omega_h} |\nabla v|^2_{0,E} \right)^{1/2} .$$

For a greater uniformity of the presentation with the rest of the chapter, we have slightly modified some notations and introduced minimal variations to some statements of the theorems.

For the sake of reference, we report below the main convergence results in the L^2-norm and the energy norm for the numerical approximation using the 2D virtual element space (3.6) or the 3D virtual element space (3.8). This result follows from the general convergence theorem that is proved in [Ahmad et al., 2013, Theorem 1]. Let $u \in H^{k+1}(\Omega)$ be the solution to the variational problem (3.4) on a convex domain Ω with $f \in H^k(\Omega)$. Let $u_h \in V_k^h$ be the solution of the virtual element method (3.5) on every mesh of a mesh family $\mathcal{T} = \{\Omega_h\}$ satisfying a suitable set of mesh geometrical assumptions. Then, a strictly positive constant C exists such that

- the H^1-error estimate holds:

$$\|u - u_h\|_{1,\Omega} \leq Ch^k \left(\|u\|_{k+1,\Omega} + |f|_{k,\Omega} \right); \tag{3.24}$$

- the L^2-error estimate holds:

$$\|u - u_h\|_{0,\Omega} \leq Ch^{k+1} \left(\|u\|_{k+1,\Omega} + |f|_{k,\Omega} \right). \tag{3.25}$$

Constant C may depend on the stability constants α_* and α^* from (3.18), on the size of the computational domain $|\Omega_h|$, and on the approximation degree k. More importantly, C depends on the mesh regularity constants ρ, N, and δ of the geometrical assumptions introduced in Section 3.2. Constant C is normally independent of h, but, as for the constant ρ in assumption $\mathbf{G2^{2D}}$, it may depend on the ratio between the longest and shortest edge lengths.

Finally, we note that the approximate solution u_h is not explicitly known inside the elements. Consequently, in the numerical experiments of Section 4.2, we approximate the error in the L^2-norm as follows:

$$||u - u_h||_{0,\Omega} \approx ||u - \Pi_k^0 u_h||_{0,\Omega}, \tag{3.26}$$

where $\Pi_k^0 u_h$ is the global L^2-orthogonal projection of the virtual element approximation u_h to u. In its turn, we approximate the error in the energy norm as follows:

$$|u - u_h|_{1,\Omega} \approx ||\nabla u - \Pi_{k-1}^0 \nabla u_h||_{0,\Omega}, \tag{3.27}$$

where Π_{k-1}^0 is extended component-wisely to the vector fields.

3.3.1 "Basic Principles of Virtual Elements Methods", Beirão Da Veiga et al., 2013

This is the paper in which the VEM for two-dimensional domains was introduced and defined [Beirão da Veiga et al., 2013]. In the original formulation, the paper introduced the so-called *regular conforming virtual element space*. For simplicity and with a small abuse of notation, the regular conforming virtual element spaces are denoted by V_k^h and $V_k^h(E)$ as in Section 3.1.3, even if the elemental space is different from (3.7):

$$\begin{aligned} V_k^h &:= \{v_h \in H^1(\Omega) : v_{h|E} \in V_k^h(E)\ \forall E \in \Omega_h\}, \\ V_k^h(E) &:= \{v_h \in H^1(E) : v_{h|\partial E} \in C^0(\partial E),\ v_{h|\mathsf{e}} \in \mathbb{P}_k(\mathsf{e})\ \forall \mathsf{e} \in \partial E, \\ &\qquad\qquad \text{and}\ \ \Delta v_h \in \mathbb{P}_{k-2}(E)\} \end{aligned} \tag{3.28}$$

and the *dofi-dofi* formulation $S_h^{E,\mathrm{dd}}$ defined in (3.20) is used for the stabilization bilinear form. The authors introduce the concept of *simple polygon* and the geometric regularity assumptions $\mathbf{G1^{2D}}$ and $\mathbf{G2s^{2D}}$. The following statement on the convergence of the VEM in the energy norm is general and largely used in the VEM literature.

Theorem 3.2 (abstract energy error)**.** *Under the k-consistency and stability assumptions* (3.17) *and* (3.18) *on the bilinear form* a_h^E*, the discrete problem* (3.5) *has a unique solution* u_h*. Moreover, for every approximation* $u_I \in V_k^h$ *of* u *and every approximation* u_π *of* u *that is piecewise in* $\mathbb{P}_k(\Omega_h)$*, we have*

$$|u - u_h|_{1,\Omega} \leq C(|u - u_I|_{1,\Omega} + |u - u_\pi|_{h,1} + \mathcal{F}_h),$$

where C is a constant depending only on α_* and α^* from (3.18)*, and, for any h, $\mathcal{F}_h = |f - f_h|_{V_k^{h\prime}}$ is the smallest constant such that*

$$(f, v) - \langle f_h - f, v\rangle \leq \mathcal{F}_h |f|_1, \qquad \forall v \in V_k^h.$$

3.3.2 "Equivalent Projectors for Virtual Element Methods", Ahmad et al., 2013

The work [Ahmad et al., 2013] deals with both the two-dimensional and three-dimensional formulations of the VEM. The *enhanced conforming virtual element space* (3.7), (3.6) is introduced (in the paper it is named "modified VEM space"), as opposed to (3.28), and the *dofi-dofi* stabilization (3.20) is adopted. In the two-dimensional formulation, under the geometrical assumptions $\mathbf{G1^{2D}}$ and $\mathbf{G2^{2D}}$, the paper re-considers Theorem 3.2 also for the three-dimensional case, and presents explicit estimates of the errors using the H^1 and L^2 errors.

Theorem 3.3 (H^1 error estimate)**.** *Assuming $\boldsymbol{G1^{2D}}$, $\boldsymbol{G2^{2D}}$, let the right-hand side f belong to $H^{k-1}(\Omega)$, f_h be defined by $f_h := \Pi^0_{k-2} f$ for $k \geq 2$ and $f_h := \Pi^0_0 f$ for $k = 1$, and the exact solution u belong to $H^{k+1}(\Omega)$. Then, for $u_h \in V_k^h$ defined in* (3.7)*, we have*

$$||u - u_h||_{1,\Omega} \leq C|h|^k |u|_{k+1,\Omega}$$

with C a positive constant independent of h.

Theorem 3.4 (L^2 error estimate)**.** *Assuming $\boldsymbol{G1^{2D}}$, $\boldsymbol{G2^{2D}}$ and with Ω convex, let the right-hand side f belong to $H^k(\Omega)$, f_h be defined by $f_h := \Pi^0_{k-1} f$ for $k \geq 1$, and the exact solution u belong to $H^{k+1}(\Omega)$. Then, for $u_h \in V_k^h$ defined in* (3.7)*, we have*

$$||u - u_h||_{0,\Omega} + |h|\, ||u - u_h||_{1,\Omega} \leq C|h|^{k+1} |u|_{k+1,\Omega},$$

with C a positive constant independent of h.

The authors also show how, in the three-dimensional formulation, analogous results hold if we simply replace $\mathbf{G1^{2D}}$ and $\mathbf{G2^{2D}}$ with $\mathbf{G1^{3D}}$ and $\mathbf{G2^{3D}}$.

3.3.3 "Stability Analysis for the Virtual Element Method", Beirão Da Veiga et al., 2017

The contribution [Beirão da Veiga et al., 2017] is based on the two-dimensional regular virtual element space (3.28) defined in [Beirão da Veiga et al., 2013]. The paper provides a new estimate of the abstract energy error, and an analysis of the H^1 error with respect to two different

stabilization techniques defined later on. Moreover, new analytical assumptions on the bilinear form $a_h(\cdot,\cdot)$ are introduced in place of (3.18):

$$
\begin{aligned}
a_h^E(v_h, v_h) &\leq C_1(E)|||v_h|||_E^2, \text{ for all } v_h \in V_k^h(E); \\
|||q|||_E^2 &\leq C_2(E) a_h^E(q,q), \text{ for all } q \in \mathbb{P}_k(E),
\end{aligned}
\tag{3.29}
$$

being $|||\cdot|||_E$ a discrete semi-norm induced by the stability term, and $C_1(E), C_2(E)$ positive constants which depend on the shape and possibly on the size of E. The second inequality is necessary only for the polynomials $q \in \mathbb{P}_k(E)$, while in the standard analysis of [Beirão da Veiga et al., 2013] an inequality similar to (3.29) is required for every $v_h \in V_k^h(E)$. Thus, even when $C_1(E)$ and $C_2(E)$ can be chosen independent of E, the semi-norm induced on $V_k^h(E)$ by the stabilization term may be stronger than the energy $a_h^E(\cdot,\cdot)^{1/2}$.

Theorem 3.5 (abstract energy error)**.** *Under the stability assumptions* (3.29)*, let the continuous solution u of the problem* (3.5) *satisfy $u_{|E} \in \mathcal{V}_E$ for all $E \in \Omega_h$, where $\mathcal{V}_E \subseteq V_k^h(E)$ is a subspace of sufficiently regular functions. Then, for every $u_I \in V_k^h$ and for every u_π such that $u_{\pi|E} \in \mathbb{P}_k(E)$, the discrete solution u_h satisfies*

$$
|u - u_h|_{1,\Omega} \lesssim C_{err}(h)\ (\mathcal{F}_h + |||u - u_I||| + |||u - u_\pi||| + |u - u_I|_{1,\Omega} + |u - u_\pi|_{h,1}),
$$

where the constant $C_{err}(h)$ is given by

$$
C_{err}(h) = \max\left\{1,\ \tilde{C}(h)C_1(h), \tilde{C}(h)^{3/2}\sqrt{C^*(h)C_1(h)}\right\},
$$

and the following quantities are derived from the constants in (3.29)*:*

$$
\begin{aligned}
\tilde{C}(h) = \max_{E\in\Omega_h}\{1,\ C_2(E)\}, \quad C_1(h) = \max_{E\in\Omega_h}\{C_1(E)\}, \\
C^*(h) = \frac{1}{2}\max_{E\in\Omega_h}\{\min\{1, C_2(E)^{-1}\}\}.
\end{aligned}
$$

The stability term $S_h^E(\cdot,\cdot)$ is considered as the combination of two contributions: the first, $S_h^{\partial E}$, related to the boundary degrees of freedom; the second, S_h^{oE}, related to the internal degrees of freedom. In the following statements, we restrict the analysis to $S_h^{\partial E}$ without losing generality. In this case, $S_h^{\partial E}$ is expressed in the *dofi-dofi* form $S_h^{\partial E,\mathrm{dd}}$ defined in (3.20), or in the *trace* form proposed in [Wriggers et al., 2016] both in the original "H^1-version":

$$
S_h^{\partial E, H^1}(v_h, w_h) = h_E \int_{\partial E} \partial_s v_h \partial_s w_h ds,
\tag{3.30}
$$

where $\partial_s v_h$ denotes the tangential derivative of v_h along ∂E. For completeness, we also report the "L^2-version" of the trace stabilization:

$$
S_h^{\partial E, L^2}(v_h, w_h) = \sum_{\mathsf{e}\in\partial E} h_{\mathsf{e}}^{-1} \int_{\mathsf{e}} v_h w_h ds.
\tag{3.31}
$$

Theorem 3.6 (H^1 error estimate with *dofi-dofi* stabilization)**.** *Assuming* $\boldsymbol{G1^{2D}}$, $\boldsymbol{G3^{2D}}$, *let* $u \in H^s(\Omega)$, $s > 1$, *be the solution of the problem* (3.5) *with* $S_h^E = S_h^{\partial E, dd}$. *Let* u_h *be the solution of the discrete problem, then it holds*

$$\|u - u_h\|_{1,\Omega} \lesssim C(h) h^{s-1} |u|_{s,\Omega} \quad 1 < s \leq k+1,$$

with $C(h) = \max_{E \in \Omega_h}(\log(1 + h_E/h_m(E)))$, *where* $h_m(E)$ *denotes the length of the smallest edge of* E.

The effect of assuming $\mathbf{G3^{2D}}$ instead of $\mathbf{G2^{2D}}$ is a deterioration of the convergence rate proportional to $\log(h)$. This convergence loss is practically negligible in most of the experimental cases.

Corollary 3.7. *Assuming* $\boldsymbol{G1^{2D}}$ *and* $\boldsymbol{G2^{2D}}$ *instead, then* $c(h) \lesssim 1$ *and therefore*

$$\|u - u_h\|_{1,\Omega} \lesssim h^{s-1} |u|_{s,\Omega} \qquad 1 < s \leq k+1.$$

Theorem 3.8 (H^1 error estimate with *trace* stabilization)**.** *Under assumption* $\boldsymbol{G1^{2D}}$, *let* $u \in H^s(\Omega)$, $s > 3/2$ *be the solution of the problem* (3.5) *with* $S_h^E = S_h^{\partial E, H^1}$. *Let* u_h *be the solution of the discrete problem, then it holds*

$$\|u - u_h\|_{1,\Omega} \lesssim h^{s-1} |u|_{s,\Omega} \quad 3/2 < s \leq k+1.$$

3.3.4 "Some Estimates for Virtual Element Methods", Brenner et al., 2017

In the paper [Brenner et al., 2017], the VEM in both two and three dimensions is re-formulated to better suit the convergence analysis when using meshes with small edges and faces. This topic is then considered in the successive paper [Brenner and Sung, 2018], and will be discussed in the next section. The enhanced elemental virtual element space is defined in an equivalent way to (3.13):

$$\begin{aligned} V_k^h(E) := \{ v_h \in H^1(E) : \; & v_{h|\partial E} \in \mathbb{P}_k(\partial E), \\ & \exists\, q_{v_h} (= -\Delta v_h) \in \mathbb{P}_k(E) \text{ such that} \\ & \int_E \nabla v_h \cdot \nabla w_h \, d\mathbf{x} = \int_E q_{v_h} w_h \, d\mathbf{x} \quad \forall w_h \in H_0^1(E), \\ & \text{and } \Pi_k^{0,E} v_h - \Pi_k^{\nabla,E} v_h \in \mathbb{P}_{k-2}(E) \}. \end{aligned} \tag{3.32}$$

The Authors consider different types of stabilization, but the convergence results are independent of the choice of the stabilization term.

Theorem 3.9 (abstract energy error)**.** *Assuming* $\boldsymbol{G1^{2D}}$, $\boldsymbol{G2^{2D}}$, *if* $f \in H^{s-1}(\Omega)$ *for* $1 \leq s \leq k$, *then there exists a positive constant* C *depending only on* k *and* ρ *from* $\boldsymbol{G1^{2D}}$ *such that*

$$|u - u_h|_{1,\Omega} \leq C(\inf_{v \in V_k^h} |u - v|_{1,\Omega} + \inf_{w \in \mathbb{P}_k(\Omega_h)} |u - w|_{h,1} + h^s |f|_{s-1,\Omega}).$$

Theorem 3.10 (H^1 error estimate)**.** *Assuming* $\boldsymbol{G1^{2D}}$, $\boldsymbol{G2^{2D}}$, *if* $u \in H^{s+1}(\Omega)$ *for* $1 \leq s \leq k$, *then there exists positive constants* C_1, C_2 *depending only on* k *and* ρ *from* $\boldsymbol{G1^{2D}}$ *such that*

$$|u - u_h|_{1,\Omega} + |u - \Pi_k^{\nabla} u_h|_{h,1} \leq C_1 h^s |u|_{s+1,\Omega}.$$

Theorem 3.11 (L^2 error estimate)**.** *Assuming* $\boldsymbol{G1^{2D}}$, $\boldsymbol{G2^{2D}}$, *with* Ω *convex, if* $u \in H^{s+1}(\Omega)$ *for* $1 \leq s \leq k$, *then there exists a positive constant* C *depending only on* Ω, k *and* ρ *from* $\boldsymbol{G1^{2D}}$ *such that*

$$\|u - u_h\|_{0,\Omega} \leq C h^{s+1} |u|_{s+1,\Omega}.$$

Afterward, the author state that these results can be extended to three dimensions through identical arguments, based on assumptions $\mathbf{G1^{3D}}$ and $\mathbf{G2^{3D}}$.

3.3.5 "Virtual Element Methods on Meshes with Small Edges or Faces", Brenner and Sung, 2018

In the paper [Brenner and Sung, 2018], error estimates of the VEM are yield for polygonal or polyhedral meshes possibly equipped with small edges (in 2D) or faces (in 3D). The virtual element space is formulated as (3.32) (i.e., the enhanced space). The local stabilizing bilinear form is considered in both the *dofi-dofi* formulation $S_h^{E,\mathrm{dd}}$ (3.20) and in the *trace* formulation $S_h^{E,\mathrm{tr}}$ (3.30). The following constants are defined:

$$\mathcal{H} := \sup_{E \in \Omega_h} \left(\frac{\max_{\mathsf{e} \in \partial E} h_{\mathsf{e}}}{\min_{\mathsf{e} \in \partial E} h_{\mathsf{e}}} \right), \qquad \alpha_h := \begin{cases} \ln(1 + \mathcal{H}) & \text{with } S_h^{E,\mathrm{dd}} \\ 1 & \text{with } S_h^{E,\mathrm{tr}} \end{cases} \tag{3.33}$$

Moreover, a mesh-dependent energy norm $\|\cdot\|_h := \sqrt{a_h(\cdot,\cdot)}$ and a functional $\Xi_h : V_k^h \to \mathbb{P}_k(\Omega_h)$ are introduced. The function Ξ_h is defined as:

$$\Xi_h = \begin{cases} \Pi_1^0 & \text{if } k = 1, 2 \\ \Pi_{k-1}^0 & \text{if } k \geq 3. \end{cases}$$

In two dimensions, the geometrical assumptions considered are $\mathbf{G1^{2D}}$ and $\mathbf{G3^{2D}}$.

Theorem 3.12 (abstract energy error)**.** *Assuming* $\boldsymbol{G1^{2D}}$, $\boldsymbol{G3^{2D}}$, *let* u *and* u_h *be the solutions of the continuous and discrete problems* (3.4), (3.5)*. We have:*

$$\|u - u_h\|_h \lesssim \inf_{w \in V_k^h} \|u - w\|_h + \|u - \Pi_k^{\nabla} u\|_h + \sqrt{\alpha_h} \left(\|u - \Pi_k^{\nabla} u\|_{h,1} + \sup_{w \in V_k^h} \frac{(f, w - \Xi_h w)}{|w|_{1,\Omega}} \right).$$

Theorem 3.13 (H^1 error estimate)**.** *Assuming* ***G1^{2D}***, ***G3^{2D}***, *if the solution u belongs to $H^{s+1}(\Omega)$ for some $1 \leq s \leq k$, we have:*

$$\|u - u_h\|_h \lesssim \sqrt{\alpha_h} h^s |u|_{s+1,\Omega}, \text{ and}$$
$$|u - u_h|_{1,\Omega} + \sqrt{\alpha_h} \left[|u - \Pi_k^{\nabla} u_h|_{h,1} + |u - \Pi_k^0 u|_{h,1} \right] \lesssim \alpha_h h^s |u|_{s+1,\Omega}.$$

Theorem 3.14 (L^2 error estimate)**.** *Assuming* ***G1^{2D}***, ***G3^{2D}***, *if the solution u belongs to $H^{s+1}(\Omega)$ for some $1 \leq s \leq k$, we have:*

$$\|u - u_h\|_{0,\Omega} \leq C \ \alpha_h h^{s+1} |u|_{s+1,\Omega}.$$

In three dimensions, the Authors only consider stabilization $S_h^{E,\mathrm{dd}}$, and prove analogous results under assumption **G1^{3D}** and **G3^{3D}**, replacing the constant α_h with its volumetric extension β_h:

$$\beta_h := \ln \left(1 + \sup_{E \in \Omega_h} \left(\frac{\max_{\mathsf{f} \in \partial E} h_{\mathsf{f}}}{\min_{\mathsf{f} \in \partial E} h_{\mathsf{f}}} \right) \right).$$

3.3.6 "Sharper Error Estimates for Virtual Elements and a Bubble-Enriched Version", Beirão Da Veiga and Vacca, 2020

The paper [Beirão da Veiga and Vacca, 2020] focuses on the 2D case, showing that the H^1 interpolation error $|u - u_I|_{1,E}$ on each element E can be split into two parts: a boundary and a bulk contribution. The intuition behind this work is that it is possible to decouple the polynomial order on the boundary and in the bulk of the element. Let k_o and k_∂ be two positive integers with $k_o \geq k_\partial$ and let $\mathbf{k} = (k_o, k_\partial)$. For any $E \in \Omega_h$, the *generalized virtual element space* is defined as follows:

$$\begin{aligned} V_{\mathbf{k}}^h &:= \{v_h \in H_0^1(\Omega) : v_{h|E} \in V_{\mathbf{k}}^h(E), \ \forall E \in \Omega_h\}, \\ V_{\mathbf{k}}^h(E) &:= \{v_h \in H_0^1(E) : v_{h|\partial E} \in C^0(\partial E), \ v_{h|\mathsf{e}} \in \mathbb{P}_{k_\partial}(\mathsf{e}) \ \forall \mathsf{e} \in \partial E, \\ &\qquad \text{and} \ \ \Delta v_h \in \mathbb{P}_{k_o - 2}(E)\}. \end{aligned} \tag{3.34}$$

For $k_o = k_\partial$, the space $V_{\mathbf{k}}^h(E)$ coincides with the regular virtual element space in (3.28). In addition, given a function $v \in H_0^1 \cap H^s(\Omega_h)$, on each element $E \in \Omega_h$ the interpolant function $\mathcal{I}_h v$ is defined as the solution of an elliptic problem as follows:

$$\begin{cases} \Delta \mathcal{I}_h v = \Pi_{k_o-2}^{0,E} \Delta v & \text{in } E \\ \mathcal{I}_h v = v_b & \text{on } \partial E, \end{cases}$$

where v_b is the standard 1D piecewise polynomial interpolation of $v|_{\partial E}$.

Theorem 3.15 (abstract energy error)**.** *Under assumption* $\boldsymbol{G1^{2D}}$*, let* $u \in H_0^1(\Omega_h) \cap H^s(\Omega_h)$ *with* $s > 1$ *be the solution of the continuous problem* (3.4) *and* $u_h \in V_{\mathbf{k}}^h$ *be the solution of the discrete problem* (3.5)*. Consider the functions*

$$e_h = u_h - \mathcal{I}_h u,\ e_{\mathcal{I}} = u - \mathcal{I}_h u,\ e_\pi = u - u_\pi,\ e_u = u_\pi - \mathcal{I}_h u,$$

where $u_\pi \in \mathbb{P}_{k_o}(\Omega_h)$ *is the piecewise polynomial approximation of* u *defined in Bramble-Hilbert Lemma. Then it holds that*

$$|u - u_h|_{1,\Omega}^2 + \alpha\ a_h(e_h, e_h) \lesssim \alpha^2 \sum_{E \in \Omega_h} h_E^2 \|f - f_h\|_{0,E}^2 + \alpha^2 |e_\pi|_{1,\Omega_h}^2 + \alpha |e_{\mathcal{I}}|_{1,\Omega}^2 + \alpha \sum_{E \in \Omega_h} \sigma^E$$

where α *is the coercivity constant from* (3.14) *and* $\sigma^E := S_h^E((I - \Pi_{k_0}^{\nabla,E})e_u, (I - \Pi_{k_0}^{\nabla,E})e_u)$*.*

Theorem 3.16 (H^1 error estimate with *dofi-dofi* stabilization)**.** *Assuming* $\boldsymbol{G1^{2D}}$*,* $\boldsymbol{G4^{2D}}$*, let* $u \in H_0^1(\Omega_h)$ *be the solution of the continuous problem* (3.4) *and* $u_h \in V_{\mathbf{k}}^h$ *be the solution of the discrete problem* (3.5) *obtained with the dofi-dofi stabilization. Assume moreover that* $u \in H^{\bar{k}}(\Omega_h)$ *with* $\bar{k} = \max\{k_o + 1,\ k_\partial + 2\}$ *and* $f \in H^{k_o - 1}$*. Then it holds that*

$$|u - u_h|_{1,\Omega}^2 \lesssim \alpha \sum_{E \in \Omega_h} \left((\alpha + \mathcal{N}_E)^{1/2} h_E^{k_o} + h_{\partial E}^{k_\partial} \right)^2,$$

where $h_{\partial E}$ *denotes the maximum edge length,* α *is the constant defined in* (3.33)*, and* $\mathcal{N}_E$ *is the number of edges in* E*.*

Theorem 3.17 (H^1 error estimate with *trace* stabilization)**.** *Under assumption* $\boldsymbol{G1^{2D}}$*, let* $u \in H_0^1(\Omega_h)$ *be the solution of the continuous problem* (3.4) *and* $u_h \in V_{\mathbf{k}}^h$ *be the solution of the discrete problem* (3.5) *obtained with the trace stabilization. Assume moreover that* $u \in H^{\bar{k}}(\Omega_h)$ *with* $\bar{k} = \max\{k_o + 1,\ k_\partial + 2\}$ *and* $f \in H^{k_o - 1}$*. Then it holds that*

$$|u - u_h|_{1,\Omega}^2 \lesssim \sum_{E \in \Omega_h} \left(h_E^{k_o} + h_{\partial E}^{k_\partial} \right)^2.$$

3.4 Mesh Quality Indicator

We define our mesh quality indicator, that is, a scalar function capable of providing insights on the behavior of the VEM over a particular sequence of meshes, before actually computing the approximated solutions. The quality indicator for two-dimensional meshes has been introduced in [Sorgente et al., 2022b], and its three-dimensional version in [Sorgente et al., 2022a]. We start from the geometrical assumptions defined in Section 3.2. The driving idea is that, instead of imposing an absolute condition that a mesh can only satisfy or violate, we want to measure

how much the mesh satisfies that condition. This approach is more accurate, as it captures small quality differences between meshes, and it does not exclude a priori all the particular cases of meshes only slightly outside the geometrical assumptions.

From each geometrical assumption for two-dimensional meshes $\mathbf{Gi^{2D}}$, $\mathbf{i} = 1, \ldots, 4$, we derive a scalar function $\varrho_i^{2\mathrm{D}} : \{E \subset \Omega_h\} \to [0, 1]$ defined element-wise, which measures how well a polytope $E \in \Omega_h$ meets the requirements of $\mathbf{Gi^{2D}}$. We set $\varrho^{2\mathrm{D}} = 0$ if E does not respect $\mathbf{Gi^{2D}}$, and the higher $\varrho^{2\mathrm{D}}$ the better E is shaped with respect to $\mathbf{Gi^{2D}}$. Similar scalar functions are derived from the assumptions $\mathbf{Gi^{3D}}$, $\mathbf{i} = 1, \ldots, 3$ for three-dimensional meshes. In this case, $\varrho_i^{3\mathrm{D}} : \{E \subset \Omega_h\} \to [0, 1]$ measures both the quality of the interior of E (through a new volumetric operator), and the quality of its faces ∂E using the indicators $\varrho_i^{2\mathrm{D}}$.

3.4.1 The Kernel of a Polytope

We preliminarily define the notion of *kernel* of a polytope, that will be pivotal in the construction of the quality indicators. Indeed, the concept of kernel is strictly connected to the one of star-shapedness, already mentioned in Section 3.2.

A polytope P is said to be *convex* if, given any two points p_1 and p_2 in P, the line segment connecting p_1 and p_2 is entirely contained in P. It can be shown that the intersection of convex polytopes is a convex polytope [Preparata and Shamos, 1985].

Definition 3.10 (Kernel)**.** *The kernel of a polytope E, noted $k(E)$, is the set of points in the interior of E from which all the points in E are visible. Two points p_1 and p_2 in E are said to be visible from each other if the segment (p_1, p_2) does not intersect the boundary of E.*

The first obvious consideration is that the kernel of a polytope is a convex polytope. If P is convex, its kernel coincides with its interior, because any two points inside a convex polytope are visible from each other. A polytope may also not have a kernel at all; in this case, we say that its kernel is *empty*. Last, as already mentioned in Section 3.2 when introducing assumption $\mathbf{G1^{2D}}$, a polytope E is called *star-shaped* if there exists a sphere, with a non-zero radius, completely contained in its kernel. A polytope is star-shaped if and only if its kernel is not empty, therefore star-shapedness can be thought of as an indicator of the existence of a kernel. Star-shapedness is weaker than convexity, and it is often used in the literature as many theoretical results in the theory of polynomial approximation in Sobolev spaces rely on this condition [Scott and Brenner, 2008, Dupont and Scott, 1980]. A visual example of these concepts is presented in Figure 3.3.

In the two-dimensional scenario, that is when the shape is a polygon, the standard way of computing the kernel is by intersecting appropriate half-planes generated from its edges, following a *geometric* approach. This problem has been tackled since the 70s, when Shamos and Hoey [Shamos and Hoey, 1976] presented an algorithm able to perform the kernel computation in $O(e \log e)$ operations, being e the number of edges of a polygon, as the intersection of e

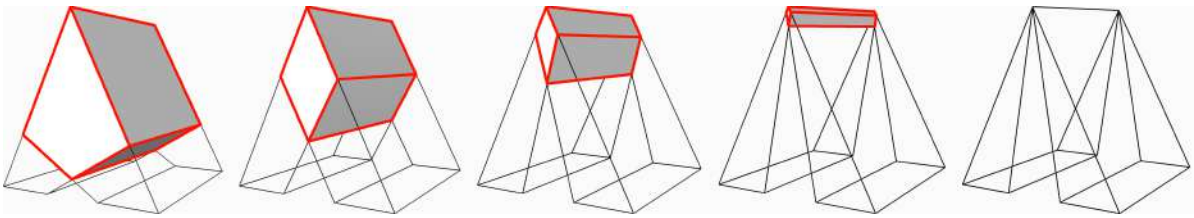


Figure 3.3: A sequence of polyhedra whose kernel is progressively shrinking. The kernel is the polyhedron delimited by the red edges [Sorgente et al., 2022c].

half-edges. After that, an algorithm able to run in $O(n)$ operations over an $n-$sided polygon, has been proposed by Lee and Preparata [Lee and Preparata, 1979], which also proved its optimality. Famous computational tools and libraries like *Boost* [Boost, 2021], *Geogram* [Lévy and Filbois, 2015], *CGAL* [Fabri and Pion, 2009], or *Libigl* [Jacobson and Panozzo, 2017] currently implement routines to compute intersections between polygons and planes, which can be used to estimate the kernel.

The first attempts to solve the volumetric version of the problem date back to the 80s [Preparata and Shamos, 1985]. The natural approach has been to extend the 2D method, which is well studied and documented, to the higher dimension from a theoretical point of view. The problem with the 3D case is that, whereas two convex polygons with respectively n_1 and n_2 vertices can be intersected in time $O(n)$, being $n = n_1 + n_2$, two convex polyhedra with the same parameters are intersected in time $O(n \log n)$, thus the generalization of the two-dimensional instance would yield an $O(n \log^2 n)$ algorithm [Preparata and Shamos, 1985]. This is not optimal with respect to the result obtained in the same work, which established a lower bound for the intersection of convex polyhedra at $O(n \log n)$. The geometric approach to the 3D problem was therefore soon dismissed as unattractive, and alternative ways have been explored. A new algorithm was formulated [Preparata and Shamos, 1985, Section 7.3.2], based on the so-called "double duality trick", which makes use of linear algebra (for solving a linear system) and homogeneous coordinates. Thanks to this *algebraic* approach, it is possible to compute the intersection of n half-spaces in time $O(n \log n)$. Recently, Hong and Elber [Hong and Elber, 2022] formulated the inequality constraints that locate the interior of the kernel domain as multivariates and extended the algebraic approach to the computation of the kernel of free-form curves and free-form surfaces.

Algorithms based on the algebraic approach are state-of-the-art for computing 3D kernels, and they are currently implemented by libraries like CGAL. In [Sorgente et al., 2021a, Sorgente et al., 2022c] we presented an algorithm for the computation of the kernel of a volumetric model (and, in particular, of a polyhedron), based on the extension to the 3D case of the geometric approach commonly adopted in two dimensions. The geometric approach showed up to be significantly faster than the algebraic one when dealing with models satisfying at least one of the following conditions:

1. the size of the model is small;
2. the model contains a significant number of co-planar faces;
3. the kernel is empty.

The detailed algorithm and experimental results are reported in Appendix B.

3.4.2 The G1-based Indicator

From assumption $\mathbf{G1^{2D}}$ we derive the indicator $\varrho_1^{2\mathrm{D}}$, defined as the ratio between the area of the kernel and the total area of the element:

$$\varrho_1^{2\mathrm{D}}(E) = \frac{|k(E)|}{|E|}.$$

We have $\varrho_1^{2\mathrm{D}}(E) = 1$ if E is convex, $\varrho_1^{2\mathrm{D}}(E) = 0$ if E is not star-shaped and $\varrho_1^{2\mathrm{D}}(E) \in (0, 1)$ if E is concave but star-shaped. Therefore, $\varrho_1^{2\mathrm{D}}$ can be interpreted as an estimate of the value of the constant ρ from $\mathbf{G1^{2D}}$ on the polygon E. We stress the fact that $\varrho_1^{2\mathrm{D}}$ is constant 0 for non-star-shaped cells, regardless of how far they are from being star-shaped, as the regularity constant ρ that we are approximating is zero in all those cases. It could be interesting to also measure the entity of the eventual non-star-shapedness of an element, but this information would have no relation with ρ. This function can be found in other works under the name of *kernel ratio* [Attene et al., 2021].

From assumption $\mathbf{G1^{3D}}$ instead, we derive the three-dimensional counterpart $\varrho_1^{3\mathrm{D}}$ of $\varrho_1^{2\mathrm{D}}$. The volumetric component is expressed through a kernel ratio that involves the volume of the kernel, computed with our geometric approach [Sorgente et al., 2022c], and the elemental volume. We want to have $\varrho_1^{3\mathrm{D}}(E) = 1$ if E and all its faces are convex, $\varrho_1^{3\mathrm{D}}(E) = 0$ if E or one of its faces are not star-shaped, and $\varrho_1^{3\mathrm{D}}(E) \in (0, 1)$ if E and all its faces are concave and star-shaped.

$$\varrho_1^{3\mathrm{D}}(E) = \frac{|k(E)|}{|E|} \prod_{\mathsf{f} \in \partial E} \varrho_1^{2\mathrm{D}}(\mathsf{f}).$$

The volumetric and the boundary components (the kernel ratio of each face) are multiplied so that, even if only one of them is zero (if a single face is not star-shaped), the whole product vanishes.

3.4.3 The G2-based Indicator

The function $\varrho_2^{2\mathrm{D}}$ returns an estimate of the constant ρ introduced in $\mathbf{G2^{2D}}$. It is expressed through the ratio $|\mathsf{e}| / h_E$, with the insertion of the quantity $\sqrt{|E|}$ in order to avoid pathological situations. Note that this indicator was formulated differently in [Sorgente et al., 2021b, Sorgente

et al., 2022b], the denominator being $\max(\sqrt{|E|}, h_E)$. In fact, the max operator was useless, because the diameter of a polygon is always greater or equal to the root of its area.

$$\varrho_2^{2\mathrm{D}}(E) = \frac{\min(\sqrt{|E|},\ \min_{\mathsf{e}\in\partial E} |\mathsf{e}|)}{h_E}.$$

If E is an equilateral triangle, we have that $\sqrt{|E|} = \frac{\sqrt[4]{3}}{2}|\mathsf{e}|$ and $\min_{\mathsf{e}\in\partial E}|\mathsf{e}| = h_E = |\mathsf{e}|$, that lead to $\varrho_2^{2\mathrm{D}}(E) = \frac{\sqrt[4]{3}}{2} \sim 0.65$. Likewise, in the case of a square, we have $\sqrt{|E|} = \min_{\mathsf{e}\in\partial E}|\mathsf{e}| = |\mathsf{e}|$ and $h_E = \sqrt{2}\,|\mathsf{e}|$, obtaining $\varrho_2^{2\mathrm{D}}(E) = \frac{1}{\sqrt{2}} \sim 0.7$. Therefore, from the point of view of this indicator, the quality of a squared element is slightly higher than that of an equilateral triangle. On the other side, $\varrho_2^{2\mathrm{D}}(E) \to 0$ if the length of at least one edge of E approaches 0 while the diameter does not vanishes.

The function $\varrho_2^{3\mathrm{D}}$ is an average of the volumetric constant ρ from $\mathbf{G2^{3D}}$, expressed through the ratio h_{f}/h_E, and all the boundary constants represented by the two-dimensional indicators $\varrho_2^{2\mathrm{D}}$.

$$\varrho_2^{3\mathrm{D}}(E) = \frac{1}{2}\left[\frac{\min(\sqrt[3]{|E|},\ \min_{\mathsf{f}\in\partial E} h_{\mathsf{f}})}{h_E} + \frac{1}{\#\{\mathsf{f}\in\partial E\}}\sum_{\mathsf{f}\in\partial E}\varrho_2^{2\mathrm{D}}(\mathsf{f})\right].$$

Again, this formulation is different from the one presented in [Sorgente et al., 2022a], because we simplified the denominator of the first term by removing the useless max operator.

3.4.4 The G3-based Indicator

Function $\varrho_3^{2\mathrm{D}}$ is a simple counter of the number of edges of the polygon, which penalizes elements with numerous edges as required by $\mathbf{G3^{2D}}$:

$$\varrho_3^{2\mathrm{D}}(E) = \frac{3}{\#\{\mathsf{e}\in\partial E\}}.$$

It returns 1 if E is a triangle, which is therefore the optimal shape for this indicator, and it tends to 0 as the number of edges increases.

Function $\varrho_3^{3\mathrm{D}}$ measures the number of edges and faces of a polyhedron, penalizing elements with numerous edges and faces as required by $\mathbf{G3^{3D}}$. In analogy to $\varrho_3^{2\mathrm{D}}$, it returns 1 if E is a tetrahedron, and it tends to 0 the number of faces and edges increases.

$$\varrho_3^{3\mathrm{D}}(E) = \frac{1}{2}\left[\frac{4}{\#\{\mathsf{f}\in\partial E\}} + \frac{1}{\#\{\mathsf{f}\in\partial E\}}\sum_{\mathsf{f}\in\partial E}\varrho_3^{2\mathrm{D}}(\mathsf{f})\right].$$

3.4.5 The G4-based Indicator

Last, we recall from the definition of $\mathbf{G4^{2D}}$ in Section 3.2 that the boundary of a polygon E can be considered as a 1-dimensional mesh $\mathcal{I}_E$, which can be subdivided into disjoint sub-meshes

$\mathcal{I}_E^1, \ldots, \mathcal{I}_E^N$, each one containing possibly more than one edge of E. In practice, we consider as a sub-mesh the collection (or *sequence*) of all edges whose vertices lie on the same line. An example is shown in Figure 3.4, where the boundary of element E in a polygonal mesh is represented by the 1-dimensional mesh $\mathcal{I}_E = \{\mathcal{I}_E^1, \mathcal{I}_E^2, \mathcal{I}_E^3, \mathcal{I}_E^4\}$. The sub-meshes $\mathcal{I}_E^1, \mathcal{I}_E^2$ and $\mathcal{I}_E^3$ contain, respectively, the left, top and right edge of E, while $\mathcal{I}_E^4$ contains the sequence of all the aligned edges in the bottom of E.

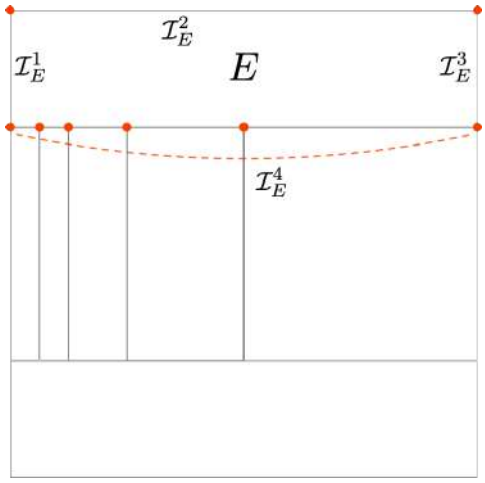


Figure 3.4: 1-dimensional mesh $\mathcal{I}_E = \{\mathcal{I}_E^1, \mathcal{I}_E^2, \mathcal{I}_E^3, \mathcal{I}_E^4\}$ relative to element E of a polygonal mesh [Sorgente et al., 2022b].

The indicator $\varrho_4^{2\mathrm{D}}$ returns the minimum ratio between the smallest and the largest element in every $\mathcal{I}_E$, which is a measure of the quasi-uniformity of $\mathcal{I}_E$ imposed by $\mathbf{G4^{2D}}$:

$$\varrho_4^{2\mathrm{D}}(E) = \min_i \frac{\min_{\mathsf{e} \in \mathcal{I}_E^i} |\mathsf{e}|}{\max_{\mathsf{e} \in \mathcal{I}_E^i} |\mathsf{e}|}.$$

If an element E does not present aligned edges, every sub-mesh $\mathcal{I}_E^i$ contains exactly one edge, and $\varrho_4^{2\mathrm{D}}$ is constantly equal to 1. At the presence of at least one sequence of aligned edges, $\varrho_4^{2\mathrm{D}}(E) = 1$ if all the edges in all the sequences have equal lengths. Last, $\varrho_4^{2\mathrm{D}}(E) \to 0$ as the edges lengths in even one single sequence (as we are considering the minimum among all the sequences) become more and more unbalanced.

We do not consider $\varrho_4^{3\mathrm{D}}$, since assumption $\mathbf{G4^{2D}}$ has not been extended to the three-dimensional scenario.

3.4.6 The Global Indicator

Combining together $\varrho_1^{2\mathrm{D}}$, $\varrho_2^{2\mathrm{D}}$, $\varrho_3^{2\mathrm{D}}$ and $\varrho_4^{2\mathrm{D}}$, we define a global function $\varrho^{2\mathrm{D}} : \{\Omega_h\}_h \to [0, 1]$, which measures the overall quality of a polygonal mesh Ω_h. In particular, we combine the indicators with the formula $\varrho_1^{2\mathrm{D}}\varrho_2^{2\mathrm{D}} + \varrho_1^{2\mathrm{D}}\varrho_3^{2\mathrm{D}} + \varrho_1^{2\mathrm{D}}\varrho_4^{2\mathrm{D}}$, as it reflects the way in which these

assumptions are typically imposed: $\mathbf{G1^{2D}}$ and $\mathbf{G2^{2D}}$, $\mathbf{G1^{2D}}$ and $\mathbf{G3^{2D}}$ or $\mathbf{G1^{2D}}$ and $\mathbf{G4^{2D}}$ (but not, for instance, $\mathbf{G2^{2D}}$ and $\mathbf{G3^{2D}}$ simultaneously):

$$\varrho^{2\mathrm{D}}(\Omega_h) = \sqrt{\frac{1}{\#\{E \in \Omega_h\}} \sum_{E \in \Omega_h} \frac{\varrho_1^{2\mathrm{D}}(E)\varrho_2^{2\mathrm{D}}(E) + \varrho_1^{2\mathrm{D}}(E)\varrho_3^{2\mathrm{D}}(E) + \varrho_1^{2\mathrm{D}}(E)\varrho_4^{2\mathrm{D}}(E)}{3}}\,. \tag{3.35}$$

We also tried alternative formulations for $\varrho^{2\mathrm{D}}$, for instance considering only $\varrho_1^{2\mathrm{D}}, \varrho_3^{2\mathrm{D}}$ and $\varrho_4^{2\mathrm{D}}$ (as, technically, $\mathbf{G3^{2D}}$ is meant to replace $\mathbf{G2^{2D}}$), adding weights to give more importance to a specific indicator, or multiplying all the indicators instead of summing them. Among all these possible variants, however, the formulation (3.35) was the one providing the highest correlation between the quality score of a mesh and the numerical results of the VEM over it.

Following the same philosophy, we can define a global function $\varrho^{3\mathrm{D}} : \{\Omega_h\}_h \to [0, 1]$ which measures the overall quality of a polyhedral mesh Ω_h. The formula for combining $\varrho_1^{3\mathrm{D}}$, $\varrho_2^{3\mathrm{D}}$ and $\varrho_3^{3\mathrm{D}}$ is derived straightforwardly from (3.35), considering the fact that we do not have a 3D version of $\mathbf{G4^{2D}}$:

$$\varrho^{3\mathrm{D}}(\Omega_h) = \sqrt{\frac{1}{\#\{E \in \Omega_h\}} \sum_{E \in \Omega_h} \frac{\varrho_1^{3\mathrm{D}}(E)\varrho_2^{3\mathrm{D}}(E) + \varrho_1^{3\mathrm{D}}(E)\varrho_3^{3\mathrm{D}}(E)}{2}}\,. \tag{3.36}$$

We have $\varrho^{2\mathrm{D}}(\Omega_h) \to 1$ if Ω_h contains only perfectly-shaped elements like equilateral triangles or squares, $\varrho^{2\mathrm{D}}(\Omega_h) = 0$ if and only if Ω_h contains only non star-shaped polygons, and $0 < \varrho^{2\mathrm{D}}(\Omega_h) < 1$ otherwise. Likewise, we have $\varrho^{3\mathrm{D}}(\Omega_h) \to 1$ if Ω_h is made only of well-shaped polyhedra (e.g., equilateral tets or cubes), $\varrho^{3\mathrm{D}}(\Omega_h) = 0$ if and only if Ω_h is made only of non star-shaped polyhedrons (or with non star-shaped faces), and $0 < \varrho^{3\mathrm{D}}(\Omega_h) < 1$ otherwise.

We point out that the indicators $\varrho^{2\mathrm{D}}$ and $\varrho^{3\mathrm{D}}$ only depend on the geometrical properties of the mesh elements (because the same holds for all the $\varrho_i^{2\mathrm{D}}$ and $\varrho_i^{3\mathrm{D}}$), and they are problem-independent (but $\varrho_2^{2\mathrm{D}}$ and $\varrho_2^{3\mathrm{D}}$ penalize anisotropic configurations). Therefore, they can be computed before applying the VEM or any other numerical scheme. Given a dataset $\mathcal{D}$, we can study the behavior of $\varrho^{2\mathrm{D}}(\Omega_h)$ (or $\varrho^{3\mathrm{D}}(\Omega_h)$) for $h \to 0$ and determine the quality of the meshes through the refinement process without having to solve the numerical problem from time to time.

The construction of $\varrho^{2\mathrm{D}}$ and $\varrho^{3\mathrm{D}}$ is easily upgradeable to future developments. Whenever new assumptions on the features of a mesh should come up, one simply needs to introduce a new function $\varrho_i^{2\mathrm{D}}$ or $\varrho_i^{3\mathrm{D}}$ that measures the fulfillment of the new assumption and opportunely insert it into equation (3.35) or (3.36). This could be done, for instance, if a valid extension of $\mathbf{G4^{2D}}$ to the three-dimensional case should appear in the literature. Similarly, the global indicators are easily extendable to other numerical schemes by substituting the assumptions designed for the VEM with the assumptions on the new scheme, and defining new relative indicators $\varrho_i^{2\mathrm{D}}$ and $\varrho_i^{3\mathrm{D}}$.

3.4.7 The Elemental Indicator

In the next sections, we will also make use of the *elemental quality indicator*, i.e., the indicator restricted to a single element $E \in \Omega_h$:

$$\begin{aligned} \varrho^{2\mathrm{D}}(E) &:= \varrho^{2\mathrm{D}}(\Omega_h)_{|E} = \sqrt{\frac{\varrho_1^{2\mathrm{D}}(E)\varrho_2^{2\mathrm{D}}(E) + \varrho_1^{2\mathrm{D}}(E)\varrho_3^{2\mathrm{D}}(E) + \varrho_1^{2\mathrm{D}}(E)\varrho_4^{2\mathrm{D}}(E)}{3}}\,; \\ \varrho^{3\mathrm{D}}(E) &:= \varrho^{3\mathrm{D}}(\Omega_h)_{|E} = \sqrt{\frac{\varrho_1^{3\mathrm{D}}(E)\varrho_2^{3\mathrm{D}}(E) + \varrho_1^{3\mathrm{D}}(E)\varrho_3^{3\mathrm{D}}(E)}{2}}\,. \end{aligned} \tag{3.37}$$

Note that, in the above definition, $\varrho^{2\mathrm{D}}(E)$ is defined for elements in a two-dimensional domain, while $\varrho^{3\mathrm{D}}(E)$ is defined for elements in a three-dimensional domain. The elemental quality indicators have all the good properties of their global versions, but allow us to investigate the local quality of the elements of the mesh.

Related Publications

- T. Sorgente, D. Prada, D. Cabiddu, S. Biasotti, G. Patané, M. Pennacchio, S. Bertoluzza, G. Manzini, and M. Spagnuolo. VEM and the mesh. In *SEMA SIMAI Springer Series*, vol. 31(1), pages 1–54, Springer, 2021.
- T. Sorgente, S. Biasotti, G. Manzini, and M. Spagnuolo. The role of mesh quality and mesh quality indicators in the virtual element method. In *Advances in Computational Mathematics*, vol. 48(1) pages 1–34, 2022.
- T. Sorgente, S. Biasotti, G. Manzini, and M. Spagnuolo. Polyhedral mesh quality indicator. In *Computers and Mathematics with Applications*, vol. 114, pages 151-160, Pergamon, 2022.
- T. Sorgente, S. Biasotti, and M. Spagnuolo. A geometric approach for computing the kernel of a polyhedron. In *Smart Tools and Apps for Graphics - Eurographics Italian Chapter Conference*, pages 11–19, P. Frosini, D. Giorgi, S. Melzi, E. Rodolà editors, 2021.
- T. Sorgente, S. Biasotti, and M. Spagnuolo. Polyhedron kernel computation using a geometric approach. In *Computers & Graphics*, vol. 105, pages 94-104, 2022.

CHAPTER 4

Verification of the Quality Indicator

In this chapter, we test the mesh quality indicators ϱ^{2D} and ϱ^{3D} defined in Section 3.4. To this goal, in Section 4.1 we define a collection of both two and three-dimensional mesh "datasets". Then, in Section 4.2, we evaluate the indicators over the datasets and solve the elliptic problem (3.4) with the VEM (3.5) defined in Section 3.1 over each of them. We compare the quality score provided by the indicator for a dataset to the real performance of the VEM over it, looking for correlations. We, therefore, present the first possible employment of the indicators as a tool to classify two and three-dimensional meshes according to their quality. In Section 4.3 we describe a second usage of the indicators, as a driving criterion for a mesh coarsening algorithm. In the described algorithm, adjacent elements in a mesh are agglomerated whenever their merging preserves the quality of the mesh, measured by ϱ^{2D} or ϱ^{3D}. The resulting mesh contains a significantly smaller number of elements, being therefore computationally cheaper to deal with. Section 4.4 concludes the chapter with ongoing work on an application of the agglomeration algorithm to the context of Discrete Fracture Networks.

4.1 Generation of the Datasets

The quality indicators are defined for a single mesh, and they can therefore give an indication of the accuracy of the VEM solution over that mesh. In addition to this, we are interested in studying the indicator and the VEM over a sequence of meshes, in order to understand if the indicator is also able to capture the asymptotic behavior of the VEM. We call *dataset* a collection $\mathcal{D} := \{\Omega_n\}_{n=0,\dots,N}$ of meshes Ω_n covering the domain $\Omega \subset \mathbb{R}^d$, $d = 2, 3$, such that

- the mesh Ω_{n+1} has smaller mesh size than Ω_n for every $n = 0, \dots, N-1$;

- the meshes Ω_n are built with the same technique, so that they contain similar elements organized in similar configurations.

Note that each mesh Ω_n is uniquely identified by its size as Ω_h, therefore we can consider a dataset $\mathcal{D}$ as a subset of a mesh family: $\mathcal{D} = \{\Omega_h\}_{h \in \mathcal{H}'} \subset \mathcal{T}$ where $\mathcal{H}'$ is a finite subset of $\mathcal{H}$. The domain Ω of the datasets is the unit square $(0,1)^2$ in the 2D case and the unit cube $(0,1)^3$ in the 3D case.

Two and three-dimensional datasets are generated through different processes, and with different goals. In the 2D case, we aim at generating extremely pathological meshes, in order to stress the VEM to its limits. Accordingly, we appositely build datasets that do not fulfill any set of geometrical assumptions required by the VEM convergence analysis found in the literature (see Section 3.2). In the 3D case instead, we are more interested in generating ordinary meshes. The theoretical analysis for the three-dimensional formulation of the VEM is less developed than the 2D one. In fact, results on the VEM convergence over volumetric meshes are typically derived from their 2D analogues without investigating too deeply over 3D-specific pathologies. In addition, tools and libraries for generating and operating over generic polyhedral meshes are still hard to find in the literature. Well-developed software exists related to tetrahedral and hexahedral meshes [Livesu, 2019], but they support a very limited number of operations for generic polyhedral meshes (e.g., standard mesh file formats do not exist). Three-dimensional datasets will be therefore used to test the VEM at a smaller scale, in the attempt to understand if the quality indicators are able to spot small quality differences between datasets.

4.1.1 Generation of the 2D Datasets

Experimental studies already showed how the VEM is able to converge at optimal rates also on polygonal meshes that violate the geometrical assumptions [Attene et al., 2021, Sorgente et al., 2021b]. For this reason, we aim at generating extremely pathological meshes, with an high number of elements that violate one or more assumptions, to understand if the quality indicator is able to identify such complicated tessellations, while also testing the stability of the VEM in extreme situations. In addition to the violation of the geometrical assumptions, we are also interested in the behavior of the VEM when the measures of mesh elements and edges scale in a nonuniform way in the refinement process. To this end, for each mesh $\Omega_n \in \mathcal{D}$ we define the following quantities and study their trend for $n \to N$:

$$A_n = \frac{\max_{E \in \Omega_n} |E|}{\min_{E \in \Omega_n} |E|} \quad \text{and} \quad e_n = \frac{\max_{\mathsf{e} \in \Omega_n} |\mathsf{e}|}{\min_{\mathsf{e} \in \Omega_n} |\mathsf{e}|}. \tag{4.1}$$

We specifically designed our datasets in order to consider several possible combinations of the geometrical assumptions of Section 3.2 and the scaling indicators A_n and e_n. For each of them, we describe how they are built, which geometrical assumptions they fulfill or violate, and how the indicators A_n and e_n depend on n in the limit for $n \to N$. Each dataset is built around

(and often named after) a particular polygonal element contained in it, which is meant to stress one or more assumptions or indicators. Their construction was originally presented in [Sorgente et al., 2021b]. The detailed construction algorithms, together with the explicit computations of A_n and e_n for all datasets, can be found in Appendix B. All the meshes presented in this section are publicly accessible at `https://github.com/TommasoSorgente/vem-quality-dataset`.

4.1.1.1 Hybrid Datasets.

We start by considering hybrid datasets, i.e., datasets whose meshes contain mainly triangular elements, and occasionally some other polygons. The first one, $\mathcal{D}_{\text{Triangle}}$, contains only triangular meshes that are built by inserting a number of points in the domain through the *Poisson Disk Sampling* algorithm [Bridson, 2007], and connecting them in a Delaunay triangulation. The refinement is obtained by increasing the number of points generated by the Poisson algorithm. The meshes in this dataset do not violate any of the geometrical assumptions, and the indicators A_n and e_n are almost constant. We use $\mathcal{D}_{\text{Triangle}}$ as the reference dataset to evaluate the other datasets by comparing the performance of the VEM over them.

Next, we consider some datasets characterized by a progressive insertion in Ω of one or more identical polygonal elements (called the *initial polygons*, see Figure 4.1), the rest of the domain being tessellated by triangles. These triangles are created by the library *Triangle* [Shewchuk, 2005], bounding the area of the triangular elements with the area of the initial polygons. Steiner points [Shewchuk, 2005] can be added, and the edges of the initial polygons are split when necessary by the insertion of new nodes. The refinement is iterative, with parameters to indicate the size, shape, and number of the initial polygons; details on this process are provided in Appendix B. The top and bottom panels of Figure 4.3 respectively show the datasets $\mathcal{D}_{\text{Maze}}$ and $\mathcal{D}_{\text{Star}}$, which we selected as they violate different geometrical assumptions. Other choices for the initial polygons are possible, for instance considering the ones in the benchmark published in [Attene et al., 2021].

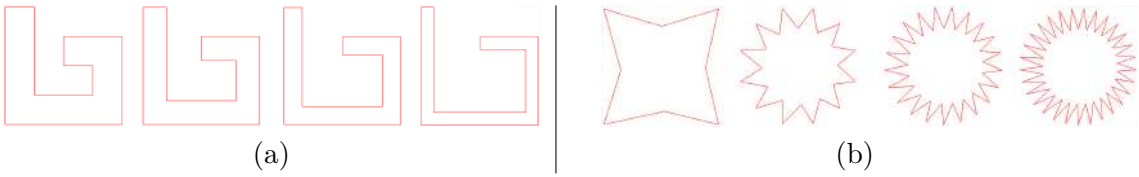


Figure 4.1: Evolution of the initial polygons in datasets $\mathcal{D}_{\text{Maze}}$(a) and $\mathcal{D}_{\text{Star}}$(b), as $n \to N$.

A "maze" is a 10-sided polygonal element E spiraling around an external point. Progressively, each mesh in $\mathcal{D}_{\text{Maze}}$ contains an increasing number of mazes E with decreasing thickness as $n \to N$. Every E is obviously *not* star-shaped, challenging assumption $\mathbf{G1^{2D}}$. Moreover, the length of the shortest edge e of E decreases faster than the diameter h_E of the polygon. This fact implies, on one side, that the ratio $|e|/h_E$ of assumption $\mathbf{G2^{2D}}$ cannot be bounded from

below by a constant ρ that is independent of h, and, on the other side, that assumption $\mathbf{G1w^{2D}}$ also fails. Indeed, even splitting E into a finite number of rectangles, it is not possible to define a global radius ρ, independent of h, with respect to which the union of these rectangles is star-shaped according to $\mathbf{G1^{3D}}$, if the shortest edge of E is constantly decreasing. Concerning the scaling indicators, we have $A_n \sim a^n$ for a constant $e < a < 3$ and $e_n \sim n \log(n)$.

Dataset $\mathcal{D}_{\text{Star}}$ is built by inserting star-like polygonal elements, still denoted by E. As $n \to N$, the number of spikes of each E increases, and the inner vertices of the star move towards the barycenter of the element. In this case, assumption $\mathbf{G3^{2D}}$ is not satisfied because the number of spikes in each E increases from mesh to mesh. Therefore, the total number of vertices and edges in a single element cannot be bounded uniformly. Each star E is star-shaped with respect to the maximum circle inscribed in it. However, as shown in Figure 4.2, the radius r of such circle decreases faster than the elemental diameter h_E, therefore it is not possible to define a global $\rho > 0$ able to uniformly bound from below the quantity r/h_E: this violates assumption $\mathbf{G1^{2D}}$. In order to satisfy assumption $\mathbf{G1w^{2D}}$, we should split each E into a number of sub-polygons that are star-shaped according to $\mathbf{G1^{2D}}$. Independently of the way we partition E, the number of sub-polygons would always be bigger than or equal to the number of spikes in E, which is constantly increasing. So, the number of sub-polygons would tend to infinity violating condition $\mathbf{G1w^{2D}}$. Last, both A_n and e_n scale linearly.

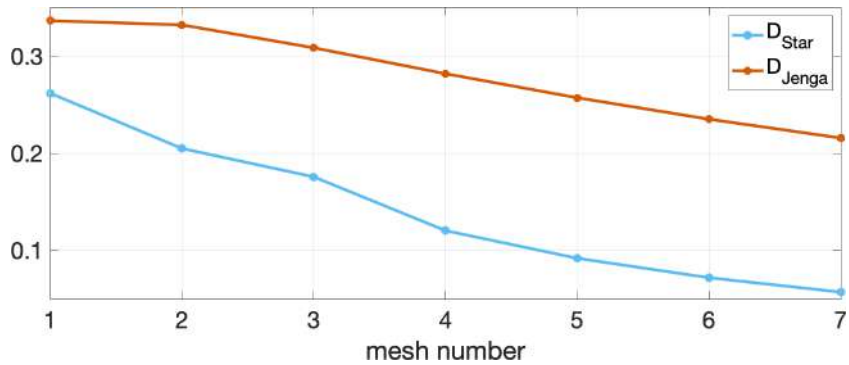


Figure 4.2: Ratio r/h_E for datasets $\mathcal{D}_{\text{Star}}$ and $\mathcal{D}_{\text{Jenga}}$.

4.1.1.2 Mirroring Datasets.

Another possible strategy to build a sequence of meshes whose elements are progressively smaller is to adopt a *mirroring* technique. In practice, we start with the first base mesh $\widehat{\Omega}_0$, which coincides with the first computational mesh Ω_0. At every step $n \geq 1$, we build a new base mesh $\widehat{\Omega}_n$ from the previous base mesh $\widehat{\Omega}_{n-1}$. The computational mesh Ω_n is then obtained by mirroring $\widehat{\Omega}_n$ 4^n times and resizing everything to fit the domain Ω. This construction allows us to obtain a number of nodes and degrees of freedom in each mesh that is comparable to that of the meshes at the same refinement level in datasets $\mathcal{D}_{\text{Maze}}$ and $\mathcal{D}_{\text{Star}}$. Examples of meshes from

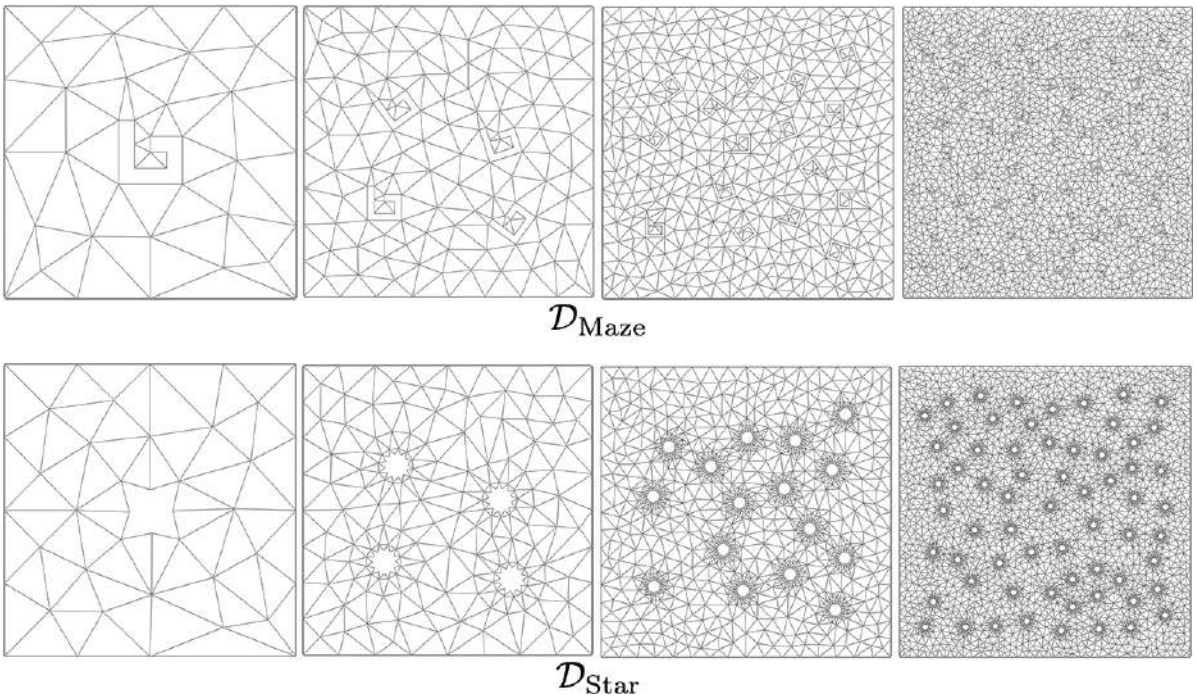


Figure 4.3: Meshes $\Omega_0, \Omega_2, \Omega_4, \Omega_6$ from datasets $\mathcal{D}_{\text{Maze}}$ (top) and $\mathcal{D}_{\text{Star}}$ (bottom).

mirrored datasets are presented in Figure 4.4. Algorithms for the construction of the following datasets, together with the mirroring algorithm are detailed in Appendix B.

In the case of the dataset $\mathcal{D}_{\text{Jenga}}$, we build the n-th base mesh $\widehat{\Omega}_n$ as follows. We start by drawing two horizontal edges that split the domain $(0,1)^2$ into three horizontal rectangles with area equal to 1/4, 1/2, and 1/4 respectively. Then, we split the rectangle with area 1/2 vertically, into two equally-sized rectangles with area 1/4. This provides us with base mesh $\widehat{\Omega}_0$, which coincides with mesh Ω_0. At each next refinement step $n \geq 1$, we split the left-most rectangle in the middle of the base mesh $\widehat{\Omega}_{n-1}$ by adding a new vertical edge, and apply the mirroring technique to obtain Ω_n. This process is shown in the top panels of Figure 4.4. This mesh family breaks all assumptions $\mathbf{G1^{2D}}$ (and $\mathbf{G1w^{2D}}$), $\mathbf{G2^{2D}}$, $\mathbf{G3^{2D}}$, and $\mathbf{G4^{2D}}$. In fact, the length of the radius r of the biggest possible disc inscribed into a rectangle is equal to 1/2 of its shortest edge e. As shown in Figure 4.2, the ratio $|\mathsf{e}|/h_E$, decreases unboundedly in the left rectangle E every time we split it, and consequently r/h_E decreases at a similar rate. This implies that a lower bound with a uniform constant ρ independent of h cannot exist for these ratios, thus breaking assumptions $\mathbf{G1^{2D}}$, $\mathbf{G1w^{2D}}$ and $\mathbf{G2^{2D}}$. In addition, the number of edges of the top and bottom rectangular elements also grows unboundedly, against assumption $\mathbf{G3^{2D}}$. Last, the one-dimensional mesh of assumption $\mathbf{G4^{2D}}$, which is built on the elemental boundary of the top and bottom rectangular elements, cannot be subdivided into a finite number of quasi-uniform sub-meshes. In fact, either we have infinite sub-meshes or an infinite edge ratio.

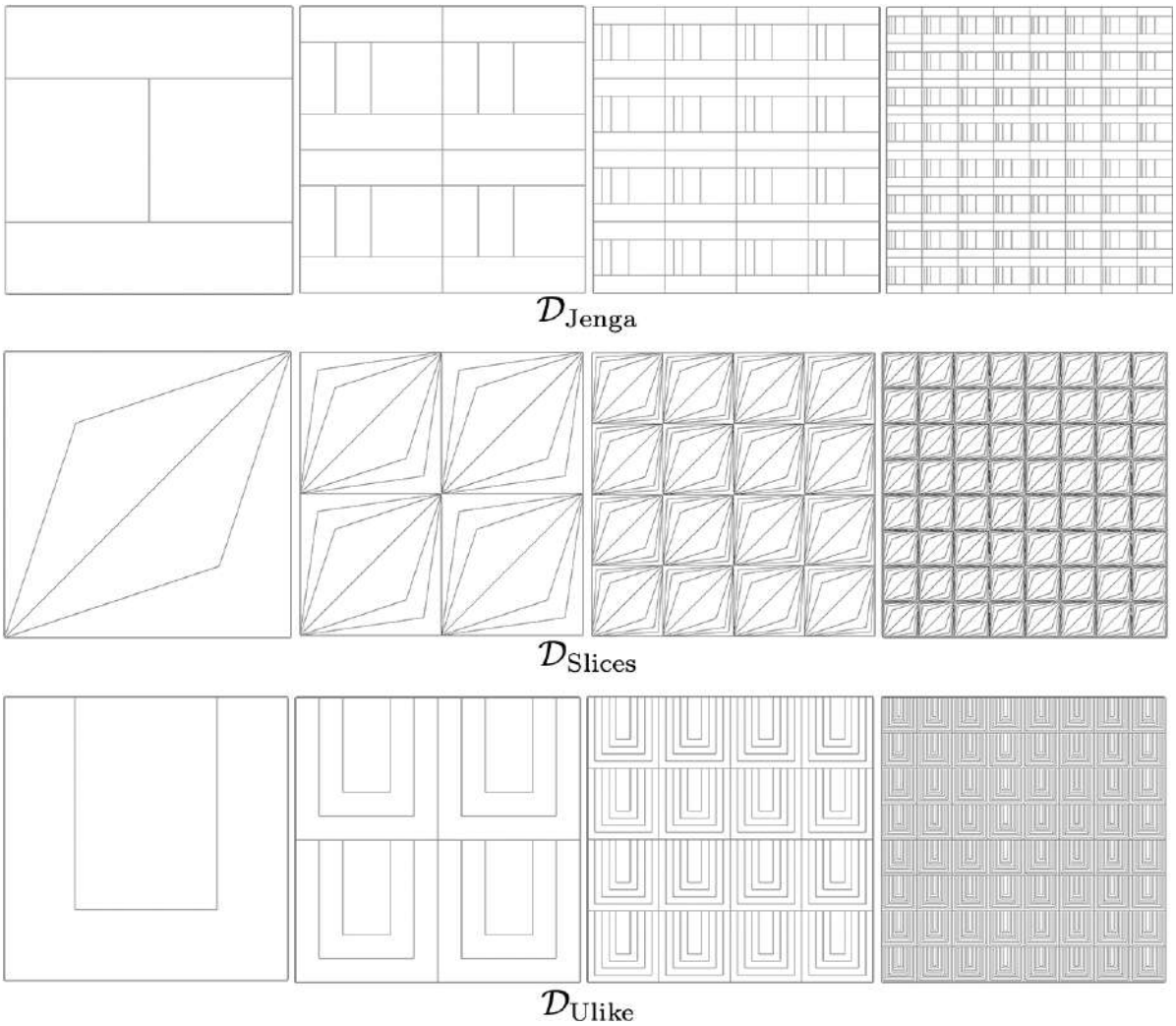


Figure 4.4: Meshes $\Omega_0, \Omega_1, \Omega_2, \Omega_3$ from datasets $\mathcal{D}_{\text{Jenga}}$ (top), $\mathcal{D}_{\text{Slices}}$ (middle), and $\mathcal{D}_{\text{Ulike}}$ (bottom).

Finally, we note that both A_n and e_n scale like 2^n.

In the case of the dataset $\mathcal{D}_{\text{Slices}}$ (Figure 4.4, middle), we build the n-th base mesh $\widehat{\Omega}_n$ as follows. First, we sample a collection of points along the diagonal (the one connecting the points with coordinates $(0,1)$ and $(1,0)$) of the reference square $[0,1]^2$, and connect them to the points $(0,0)$ and $(1,1)$. In particular, at each step $n \geq 0$, the base mesh $\widehat{\Omega}_n$ contains the points $(0,0)$ and $(1,1)$, plus the points with coordinates $(2^{-i}, 1-2^{-i})$ and $(1-2^{-i}, 2^{-i})$ for $i = 1, \ldots, n+2$. Then, we apply the mirroring technique. The dataset $\mathcal{D}_{\text{Slices}}$ violates assumptions $\mathbf{G1^{2D}}$ and $\mathbf{G1w^{2D}}$. In fact, up to a multiplicative scaling factor depending on h, the length of the radius of the biggest inscribed disc in every element E is decreasing faster than the diameter of the element, which is constantly equal to $\sqrt{2}$ times the same scaling factor, thus violating $\mathbf{G1^{2D}}$. Furthermore, the dataset also breaks assumption $\mathbf{G1w^{2D}}$ because any finite subdivisions of its

elements would suffer the same issue. Instead, the other geometrical assumptions are satisfied. Since no edge is split, we find that $e_n \sim c$, while $A_n \sim 2^n$.

In $\mathcal{D}_{\text{Ulike}}$ (Figure 4.4, bottom), we build $\widehat{\Omega}_n$ at each step $n \geq 0$ by inserting 2^n equispaced U-shaped continuous polylines inside the domain, creating as many U-like polygons. Then, we apply the mirroring technique. For arguments similar to the ones brought for $\mathcal{D}_{\text{Maze}}$, $\mathcal{D}_{\text{Ulike}}$ does not satisfy assumptions $\mathbf{G1^{2D}}$, $\mathbf{G1w^{2D}}$ and $\mathbf{G2^{2D}}$. For connectivity reasons, the lower side of the outer U-shaped polygon of every base mesh must be split into smaller segments when we apply the mirroring technique. Therefore, the number of edges of such cells cannot be limited from above, contradicting assumption $\mathbf{G3^{2D}}$. Nonetheless, assumption $\mathbf{G4^{2D}}$ is satisfied because this subdivision is uniform. Last, edge lengths scale exponentially and areas scale uniformly, i.e., $e_n \sim 2^n$, $A_n \sim c$.

4.1.1.3 Multiple Mirroring Datasets.

As a final test, we modified datasets $\mathcal{D}_{\text{Jenga}}$, $\mathcal{D}_{\text{Slices}}$, and $\mathcal{D}_{\text{Ulike}}$ in order to stress the indicators A_n and e_n harder. This is easily obtained by inserting four new elements at each step instead of one, as explained in Appendix B. The resulting datasets, $\mathcal{D}_{\text{Jenga4}}$, $\mathcal{D}_{\text{Slices4}}$, and $\mathcal{D}_{\text{Ulike4}}$, are qualitatively similar to the mirroring datasets above. These datasets fulfill the same assumptions as their respective original versions, but the number of elements at each refinement step now increases four times faster. The indicators A_n and e_n change from 2^n to 2^{4n}, but A_n remains constant for $\mathcal{D}_{\text{Ulike4}}$, and e_n remains constant for $\mathcal{D}_{\text{Slices4}}$ (see Table 4.1).

Table 4.1: Summary of the geometrical assumptions violated and the asymptotic trend of the indices A_n and e_n for two-dimensional datasets (a is a constant such that $e < a < 3$). Assumption $\mathbf{G1w^{2D}}$ is not explicitly reported because all the considered datasets that violate $\mathbf{G1^{2D}}$, also violate $\mathbf{G1w^{2D}}$.

	$\mathcal{D}_{\text{Triangle}}$	$\mathcal{D}_{\text{Maze}}$	$\mathcal{D}_{\text{Star}}$	$\mathcal{D}_{\text{Jenga}}$	$\mathcal{D}_{\text{Slices}}$	$\mathcal{D}_{\text{Ulike}}$	$\mathcal{D}_{\text{Jenga4}}$	$\mathcal{D}_{\text{Slices4}}$	$\mathcal{D}_{\text{Ulike4}}$
$\mathbf{G1^{2D}}$		$\times$	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$
$\mathbf{G2^{2D}}$		$\times$		$\times$		$\times$	$\times$		$\times$
$\mathbf{G3^{2D}}$			$\times$	$\times$		$\times$	$\times$		$\times$
$\mathbf{G4^{2D}}$				$\times$			$\times$		
A_n	c	a^n	n	2^n	2^n	c	2^{4n}	2^{4n}	c
e_n	c	$n\log(n)$	n	2^n	c	2^n	2^{4n}	c	2^{4n}

4.1.2 Generation of the 3D Datasets

Three-dimensional datasets are characterized by a *sampling strategy* for generating a number of points inside the unit cube Ω, and a *meshing technique* to connect them. Their construction was originally presented in [Sorgente et al., 2022a]. Algorithms for generating all the samplings

strategies and the meshing techniques are built over the *cinolib* library [Livesu, 2019]; all the generated datasets are publicly available at: `https://github.com/TommasoSorgente/vem-indicator-3D-dataset`

4.1.2.1 Sampling Strategies

We defined six different sampling strategies, summarized in Figure 4.5. In order to create multiple refinements for each dataset, each sampling takes an integer parameter t in input, which determines the number of points to be generated, and consequently the size of the induced mesh.

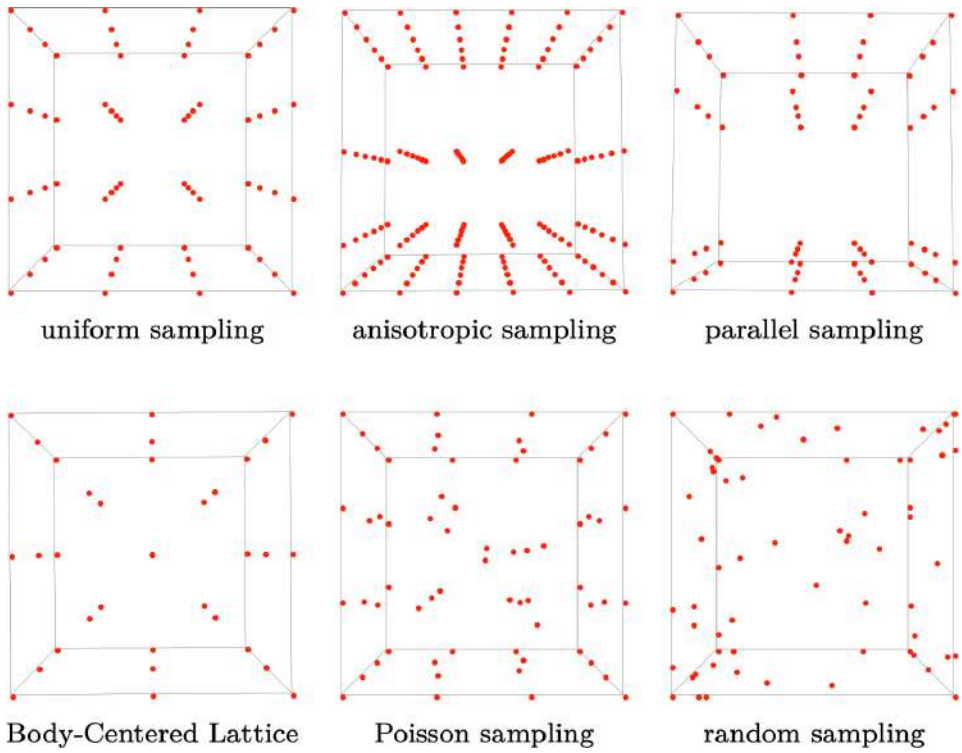


Figure 4.5: Summary of the sampling strategies.

The sampling strategies are as follows:

- *Uniform sampling.* The points are disposed along a uniform equispaced grid of size $1/t$.
- *Anisotropic sampling.* A regular grid in which the distance between the points is fixed at $1/t$ along two directions while it linearly increases ($1/t$, $2/t$, $3/t$, ...) along the third axis, leading to anisotropic configurations.
- *Parallel sampling.* This sampling is obtained from a uniform sampling with parameter t by randomly moving all the points belonging to a certain plane p_1 to another plane p_2, parallel to p_1. In practice, we pick all the points sharing the same $x-$coordinate and

randomly change x by the same quantity; then, we repeat this operation for the y and the $z-$coordinates.

- *Body-Centered Lattice (BCL).* A uniform equispaced grid of size $1/t$ is generated and one more point is added to the center of each cubic cell, using the implementation proposed in `https://github.com/csverma610/CrystalLattice`. This sampling produces equilateral tetrahedra when combined with tetrahedral meshing and truncated octahedra when combined with Voronoi meshing (defined below).

- *Poisson sampling.* The points are generated following the Poisson Disk Sampling algorithm [Bridson, 2007]. First, we apply the algorithm to the cube $(1/t, 1-1/t)^3$ and the square $(1/t, 1-1/t)^2$ with radius $1/t$, to generate points inside Ω and on its boundary. Then, to cover the domain more uniformly, we add an equispaced sampling with distance $1/t$ on each edge of Ω. The result is a collection of points disposed randomly, but at a fixed distance $(1/t)$ apart.

- *Random sampling.* The points are randomly placed inside Ω. In order to guarantee a decent distribution, given the input parameter t we generate t points along each edge of Ω, t^2 points on each face and t^3 points inside Ω.

We point out that, because of how the samplings are defined, the number of points generated with the same parameter t in the different strategies may vary.

4.1.2.2 Meshing Techniques

We create meshes by connecting a set of points with different techniques: we considered the three most common types of mesh connectivity found in the literature (tetrahedral, hexahedral, and Voronoi) plus a generic polyhedral one. As soon as sampling points are connected into a mesh, we call them *nodes*.

- *Tetrahedral meshing.* Points are connected in tetrahedral elements with *TetGen* library [Si, 2015], enforcing two quality constraints on tetrahedra: a maximum radius-edge ratio bound and a minimum dihedral angle bound.

- *Voronoi meshing.* Points are considered as centroids for the construction of a Voronoi lattice using library *Voro++* [Rycroft, 2009]. In this case, the sampled points will not appear as nodes in the final mesh, and the number of nodes does not depend uniquely on the number of points.

- *Hexahedral meshing.* Points are connected to form hexahedral elements. For ease of implementation, the mesh is still generated as a Voronoi lattice, but we make sure that points are placed in such a way that the final result is a pure hexahedral mesh.

- *Polyhedral meshing.* We start from a tetrahedral mesh and we aggregate 20% of the elements to generate possibly non-convex polyhedra. To avoid numerical problems, we select the elements with the greatest volume and aggregate them with the neighboring element sharing the widest face. Note that we only aggregate couples of elements and merge eventual coplanar faces, therefore we do not remove or modify any node of the original tetrahedral mesh.

It would also be possible to generate polyhedral meshes starting from hexahedral or Voronoi ones, but we limit our tests to aggregations of the tetrahedral datasets for ease of implementation.

4.1.2.3 Verification Datasets

Each combination of sampling strategy and meshing technique gives a dataset. As not all combinations are possible or meaningful, we selected the most diversified datasets, presented in Figure 4.6 and Figure 4.7.

We generated datasets composed of five meshes each, with increasing number of nodes (and, therefore, decreasing mesh size): they contain 60, 500, 4000, 32000, and 120000 nodes approximately. For each n, the mesh Ω_{n+1} has about eight times more nodes than Ω_n, except for the last mesh which only has four times more nodes than the previous, for computational reasons. We determine the number of nodes by opportunely setting the sampling parameter t, and we have no constraints on the number of edges, faces, or elements. We label each dataset to indicate the meshing technique and the sampling strategy. For example, $\mathcal{D}_{\text{tet-uniform}}$ is the dataset that is built by combining the tetrahedral meshing and the uniform sampling.

- *Tetrahedral datasets.* We combined the tetrahedral meshing with all the considered sampling methods, creating six tetrahedral datasets: $\mathcal{D}_{\text{tet-uniform}}$, $\mathcal{D}_{\text{tet-anisotropic}}$, $\mathcal{D}_{\text{tet-parallel}}$, $\mathcal{D}_{\text{tet-bcl}}$, $\mathcal{D}_{\text{tet-poisson}}$ and $\mathcal{D}_{\text{tet-random}}$;

- *Hexahedral datasets.* Among the six considered samplings, only the first three provide regular grids suitable for the generation of hexahedral elements. Therefore our hexahedral datasets are $\mathcal{D}_{\text{hex-uniform}}$, $\mathcal{D}_{\text{hex-anisotropic}}$, and $\mathcal{D}_{\text{hex-parallel}}$;

- *Voronoi datasets.* The last three samplings instead have been used to generate Voronoi datasets: $\mathcal{D}_{\text{voro-bcl}}$, $\mathcal{D}_{\text{voro-poisson}}$, and $\mathcal{D}_{\text{voro-random}}$;

- *Polyhedral datasets.* Finally, any of the tetrahedral datasets could have been modified to obtain polyhedral meshes, but we observed that aggregating elements from $\mathcal{D}_{\text{tet-uniform}}$, $\mathcal{D}_{\text{tet-anisotropic}}$, or $\mathcal{D}_{\text{tet-bcl}}$ would still generate convex elements, not so different from the original ones. We, therefore, chose to consider only $\mathcal{D}_{\text{poly-parallel}}$, $\mathcal{D}_{\text{poly-poisson}}$ and $\mathcal{D}_{\text{poly-random}}$.

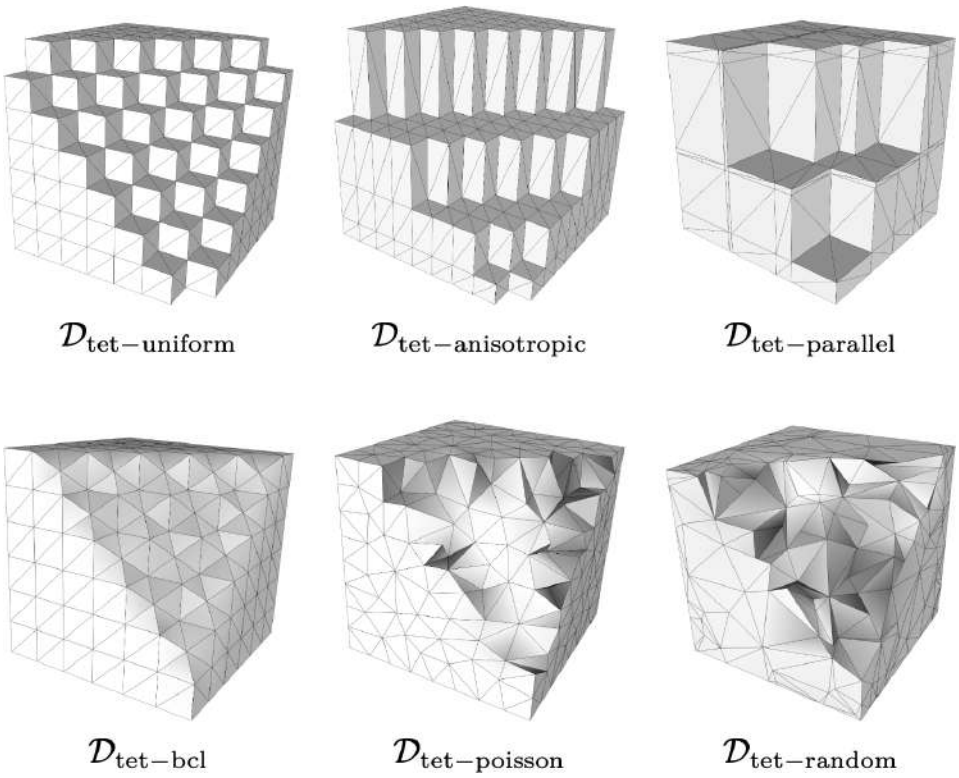


Figure 4.6: Summary of the tetrahedral datasets

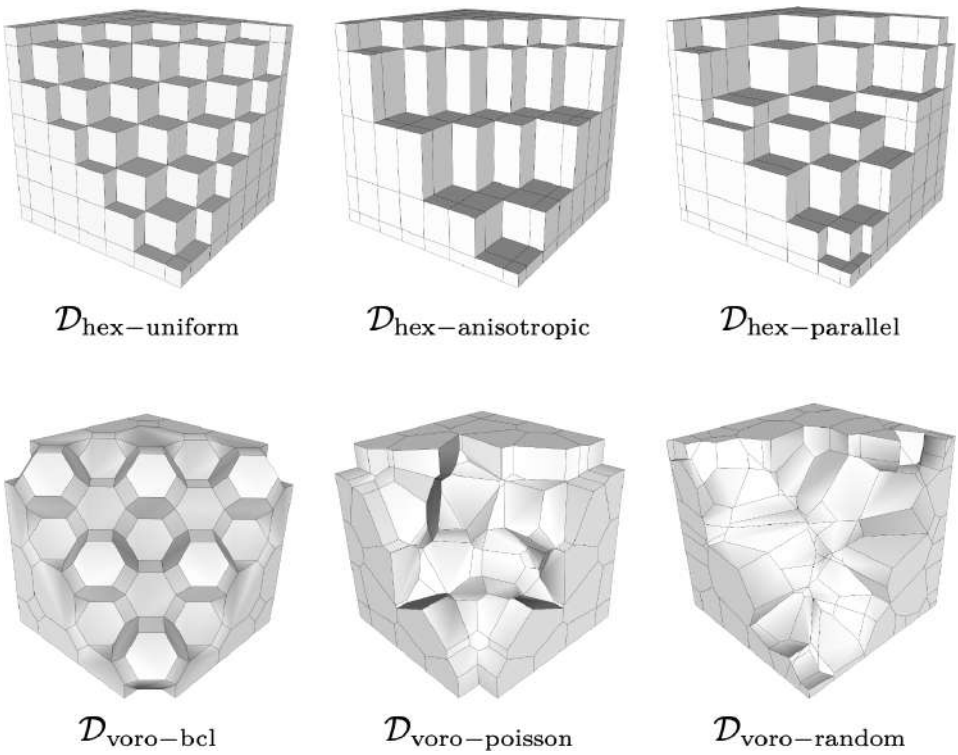


Figure 4.7: Summary of the hexahedral and Voronoi datasets

We point out that $\mathcal{D}_{\text{tet-uniform}}$ serves as a reference for the other 3D datasets, similarly to $\mathcal{D}_{\text{Triangle}}$ in the two-dimensional case. Comparing to the datasets defined over the unit cube in [Beirão da Veiga et al., 2017], we could say that dataset $\mathcal{D}_{\text{voro-random}}$ is analogous to the "Random" discretization, dataset $\mathcal{D}_{\text{hex-uniform}}$ is equivalent to their "Structured" meshes and dataset $\mathcal{D}_{\text{voro-bcl}}$ can be considered as a particular case of "CVT" discretization.

In Table 4.2 we report some considerations about the datasets and the geometrical assumptions from Section 3.2. First, datasets originating from uniform and BCL samplings obviously satisfy all three assumptions, independently of the meshing technique, and the same holds for $\mathcal{D}_{\text{tet-poisson}}$ and $\mathcal{D}_{\text{poly-poisson}}$ (elements in $\mathcal{D}_{\text{poly-poisson}}$ are not all convex, but still star-shaped, being the union of two tetrahedra). $\mathcal{D}_{\text{voro-poisson}}$ instead, is not guaranteed to satisfy $\mathbf{G2^{3D}}$: the Poisson distribution of the points generates elements with bounded diameters, while it is not possible to estimate in advance the length of the shortest edge in a Voronoi cell. Datasets obtained through anisotropic sampling are guaranteed to violate $\mathbf{G1^{3D}}$ (despite containing only convex elements) and $\mathbf{G2^{3D}}$. In fact, a generic element E of a mesh Ω_n belonging to $\mathcal{D}_{\text{hex-anisotropic}}$, is a square cuboid with basis $1/t \times 1/t$, height n/t, and therefore diagonal $\sqrt{2+n^2}/t$ (we assume that E is not on the boundary of Ω_n, and orient it as in Figure 4.7). The radius of the maximum inscribed ball is $r_E = 1/2t$, the smallest edge $h_{\mathsf{e}} = 1/t$, and the diameter $h_E = \sqrt{2+n^2}/t$, hence the ratios relative to assumptions $\mathbf{G1^{3D}}$ and $\mathbf{G2^{3D}}$ are, respectively, $\frac{1}{2\sqrt{2+n^2}}$ and $\frac{1}{\sqrt{2+n^2}}$, that both go to zero as n grows. Similar reasoning holds for meshes in $\mathcal{D}_{\text{tet-anisotropic}}$. Datasets originating from parallel and random samplings have a random nature, which makes it impossible to precisely estimate the above quantities. Obviously, for any finite sequence of such meshes, it is possible to find a constant ρ such that it satisfies $\mathbf{G1^{3D}}$ and/or $\mathbf{G2^{3D}}$, no matter how close the nodes. However, such a constant cannot be defined a-priori, but only computed afterward. Similarly, we cannot set an a-priori bound on the number of faces and edges of the Voronoi cells in $\mathcal{D}_{\text{voro-random}}$, which therefore potentially violates assumption $\mathbf{G3^{3D}}$.

We recall from Section 3.2 that, to ensure the optimal behavior of the method, either assumptions $\mathbf{G1^{3D}}$ and $\mathbf{G2^{3D}}$ or assumptions $\mathbf{G1^{3D}}$ and $\mathbf{G3^{3D}}$ need to be satisfied. Therefore, we are allowed to expect optimal convergence rates only over $\mathcal{D}_{\text{tet-uniform}}$, $\mathcal{D}_{\text{tet-bcl}}$, $\mathcal{D}_{\text{tet-poisson}}$, $\mathcal{D}_{\text{hex-uniform}}$, $\mathcal{D}_{\text{voro-bcl}}$, and $\mathcal{D}_{\text{poly-poisson}}$. On the other datasets, the VEM is not guaranteed to converge properly, but we can expect different results according to the entity of the violation of the assumptions.

4.2 Correlations Between the Quality and the Performance

In this section, we analyze the behavior of the quality indicators $\varrho^{2\text{D}}$ and $\varrho^{3\text{D}}$ defined in Section 3.4. First, we evaluate them over the meshes of the datasets from Section 4.1, obtaining some quality scores. Then, we solve the Poisson problem (3.5) with the VEM described in

Table 4.2: Summary of the geometrical assumptions violated by each three-dimensional dataset.

dataset	$\mathbf{G1^{3D}}$	$\mathbf{G2^{3D}}$	$\mathbf{G3^{3D}}$
$\mathcal{D}_{\text{tet-uniform}}$			
$\mathcal{D}_{\text{tet-anisotropic}}$	×	×	
$\mathcal{D}_{\text{tet-parallel}}$	×	×	
$\mathcal{D}_{\text{tet-bcl}}$			
$\mathcal{D}_{\text{tet-poisson}}$			
$\mathcal{D}_{\text{tet-random}}$	×	×	

dataset	$\mathbf{G1^{3D}}$	$\mathbf{G2^{3D}}$	$\mathbf{G3^{3D}}$
$\mathcal{D}_{\text{hex-uniform}}$			
$\mathcal{D}_{\text{hex-anisotropic}}$	×	×	
$\mathcal{D}_{\text{hex-parallel}}$	×	×	
$\mathcal{D}_{\text{voro-bcl}}$			
$\mathcal{D}_{\text{voro-poisson}}$		×	
$\mathcal{D}_{\text{voro-random}}$	×	×	×
$\mathcal{D}_{\text{poly-parallel}}$	×	×	
$\mathcal{D}_{\text{poly-poisson}}$			
$\mathcal{D}_{\text{poly-random}}$	×	×	

Section 3.1 over each mesh, and we study the relative H^1-seminorm and L^2-norm, as defined in (3.26), (3.27):

$$|u - u_h|_{1,\Omega}/|u|_{1,\Omega}, \qquad ||u - u_h||_{0,\Omega}/||u||_{0,\Omega},$$

and the relative L^∞-norm

$$||u - u_h||_\infty/||u||_\infty, \text{ where } ||u||_\infty = \text{ess sup}_{x\in\Omega}|u(x)|,$$

of the approximation error $u - u_h$ as the number of degrees of freedom increases. The optimal convergence rate of the method, provided by the estimates (3.24) and (3.25), is indicated for each k by the slope of the reference triangle. In the case of the L^∞-norm, we do not have such theoretical results.

The exercise we propose is to first analyze the values of $\varrho^{2\text{D}}$ ($\varrho^{3\text{D}}$) on a dataset, computed before solving the problem and make some predictions on the behavior of the VEM over it in terms of convergence rate and error magnitude. Then, looking at the approximation errors actually produced by that dataset, search for correspondences between $\varrho^{2\text{D}}$ ($\varrho^{3\text{D}}$) and the errors, checking the accuracy of the prediction. Clearly, as $\varrho^{2\text{D}}$ ($\varrho^{3\text{D}}$) does not depend on the polynomial degree k nor on the type of norm used, we will compare it to an average of the plots for the different k values and for the different norms. The mesh quality values are plotted against the number of nodes in the mesh, while the approximation errors are plotted against the degrees of freedom (DOFs). In the case $k = 1$, these quantities coincide.

We preliminarily observe that, for an ideal dataset made by meshes containing only perfectly-shaped elements (e.g., equilateral triangles/tetrahedra), $\varrho^{2\text{D}}$ and $\varrho^{3\text{D}}$ would be almost constant, and very close to 1. We assume this value as a reference for the other datasets: the closer is $\varrho^{2\text{D}}$ ($\varrho^{3\text{D}}$) on a dataset to the line $y = 1$, the smaller is the approximation error that we expect that dataset to produce. Moreover, the $\varrho^{2\text{D}}$ ($\varrho^{3\text{D}}$) slope is indicative of the convergence rate.

Since an ideal dataset would produce a horizontal line, the more negative is the slope, the worse the convergence rate that we expect. A positive trend instead should indicate a convergence rate higher than the one obtained with an equilateral triangular (tetrahedral) mesh, that is, higher than the theoretical estimates.

4.2.1 Analysis of the 2D Dataset

For the analysis of the two-dimensional datasets, we solved the discrete Poisson problem (3.4) with the VEM (3.5) for $k = 1, 2, 3$ over each mesh of each of the datasets defined in Section 4.1.1, using as groundtruth the function

$$u(x, y) = \frac{\sin(\pi x)\sin(\pi y)}{2\pi^2}, \quad (x, y) \in \Omega = (0, 1)^2. \tag{4.2}$$

This function has homogeneous Dirichlet boundary conditions, and this choice was appositely made to prevent the boundary treatment from having an influence on the approximation error.

We also consider the condition numbers of matrices $\mathbf{G}$ and $\mathbf{H}$ (with the notation adopted in [Beirão da Veiga et al., 2014]) as numerical indicators of the good behavior of the method, and identities $|\mathbf{\Pi}_k^\nabla \mathbf{D} - \mathbf{I}| = 0$ and $|\mathbf{\Pi}_k^0 \mathbf{D} - \mathbf{I}| = 0$ as an estimate of the approximation error produced by projectors Π_k^∇ and Π_k^0, represented by matrices $\mathbf{\Pi}_k^\nabla$ and $\mathbf{\Pi}_k^0$, respectively. The computation of the projectors is obviously affected by the condition numbers of $\mathbf{G}$ and $\mathbf{H}$, but the two indicators are not necessarily related. All of these quantities are computed element-wise and the maximum value among all elements of the mesh is selected. Condition numbers and identity values for $k = 1, 2, 3$ are reported in Table 4.3 (for $k < 3$ we have $\mathbf{\Pi}_k^0 = \mathbf{\Pi}_k^\nabla$).

4.2.1.1 Hybrid Datasets

Predictions Looking at the quality plot in Figure 4.8 (the leftmost), we note how the $\varrho^{2\mathrm{D}}$ values for meshes belonging to dataset $\mathcal{D}_{\mathrm{Triangle}}$ are almost constant and very close to 1. According to this information, we would say that the VEM should converge with the optimal rate and small errors over dataset $\mathcal{D}_{\mathrm{Triangle}}$. The quality of datasets $\mathcal{D}_{\mathrm{Maze}}$ and $\mathcal{D}_{\mathrm{Star}}$ are very close to 1 as well, even if the latter decreases in the last couple of meshes. Note that the differences between the three are minimal, i.e., in the range of 10^{-2}. Therefore, we expect $\mathcal{D}_{\mathrm{Maze}}$ and $\mathcal{D}_{\mathrm{Star}}$ to behave similarly to $\mathcal{D}_{\mathrm{Triangle}}$ in the first meshes, and $\mathcal{D}_{\mathrm{Star}}$ to perform worse in the last ones.

Verification The performance of the VEM over the hybrid datasets is shown in the second, third, and fourth plots of Figure 4.8. First, the reference dataset $\mathcal{D}_{\mathrm{Triangle}}$ performs perfectly, according to the theoretical results, both in L^2 and in H^1 norms for all k values, maintaining reasonable condition numbers and optimal errors on the projectors Π_k^0 and Π_k^∇ (see Table 4.3). For datasets $\mathcal{D}_{\mathrm{Star}}$ and $\mathcal{D}_{\mathrm{Maze}}$, errors decrease at the correct rate for most of the meshes and only start deflecting for very high numbers of degrees of freedom and very complicated meshes.

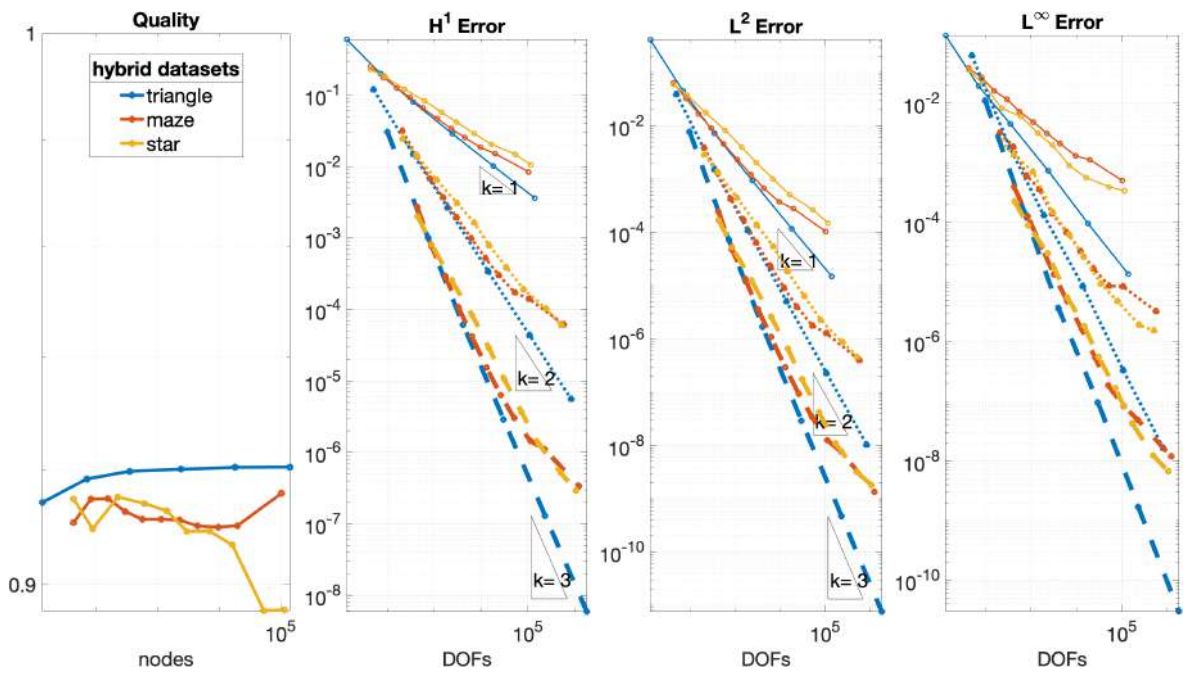


Figure 4.8: Mesh quality indicator ϱ^{2D}, H^1-seminorm, L^2-norm and L^∞-norm of the approximation errors relative to the hybrid datasets. Solid, dashed and dotted lines relate to $k = 1, 2, 3$ respectively.

These deflections are not due to numerical problems, as in both datasets we have cond($\mathbf{G}$) $< 10^6$ and cond($\mathbf{H}$) $< 10^9$, which are still reasonable values. Projectors seem to work properly: $|\mathbf{\Pi}_k^\nabla \mathbf{D} - \mathbf{I}|$ remains below 10^{-8} and $|\mathbf{\Pi}_k^0 \mathbf{D} - \mathbf{I}|$ below 10^{-7}. In a preliminary stage of this work, we obtained similar plots (not reported here) using other hybrid datasets built in the same way, with polygons surrounded by triangles. In particular, we did not see big differences when starting with the other initial polygons of Benchmark [Attene et al., 2021], cf. the construction discussed in "Hybrid datasets" in Section 4.1.1.

4.2.1.2 Mirroring Datasets

Predictions In Figure 4.9(left), the plot relative to dataset $\mathcal{D}_{\text{Jenga}}$ predicts a perfect convergence rate but greater error values with respect to the hybrid datasets seen before. The curve relative to $\mathcal{D}_{\text{Slices}}$ is quite distant from the ideal value of 1, and it keeps decreasing from mesh to mesh. However, the plot allows us to assume that it may flatten within a couple more meshes. We, therefore, expect it to produce higher errors than $\mathcal{D}_{\text{Jenga}}$, but still a decent convergence rate. Last, the ϱ^{2D} values for $\mathcal{D}_{\text{Ulike}}$ are significantly lower than the others, and the plot is not flattening at all. Hence we expect huge errors and a bad convergence rate.

Verification On the meshes from mirroring datasets, A_n or e_n may scale non-uniformly, as reported in Table 4.1 (indeed, they can scale exponentially). This reflects to cond($\mathbf{G}$)

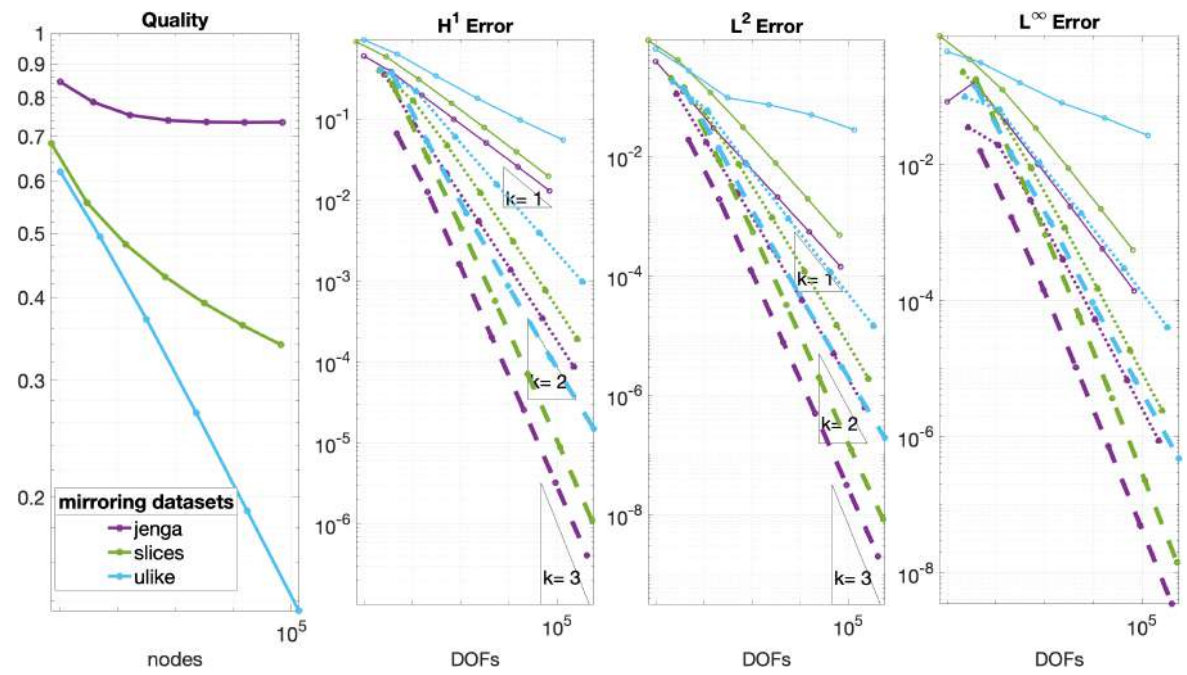


Figure 4.9: Mesh quality indicator ϱ^{2D}, H^1-seminorm, L^2-norm and L^∞-norm of the approximation errors relative to the mirroring datasets. Solid, dashed and dotted lines relate to $k = 1, 2, 3$ respectively.

and cond($\mathbf{H}$), which grow up to 10^{10} and 10^{14} for $\mathcal{D}_{\text{Jenga}}$ in the case $k = 3$. Nonetheless, the discrepancy of the projectors identities remains below 10^{-5}, which is not far from what happened with $\mathcal{D}_{\text{Maze}}$ and $\mathcal{D}_{\text{Star}}$. Dataset $\mathcal{D}_{\text{Jenga}}$ exhibits an almost perfect convergence rate, even though the errors in the different norms are bigger in magnitude than the ones measured for hybrid datasets (for instance, in the case $k = 1$ with the H^1 norm, $\mathcal{D}_{\text{Jenga}}$ remains above 10^{-2}, while the hybrid datasets were all below). $\mathcal{D}_{\text{Slices}}$ shows even bigger errors but still an optimal convergence rate, and $\mathcal{D}_{\text{Ulike}}$ is the dataset with the poorest performance, exhibiting incorrect convergence rates for all values of k and all types of norm (see in particular the case $k = 1$ with H^1 or L^∞ errors).

4.2.1.3 Multiple Mirroring Datasets

Predictions As far as multiply mirrored datasets are concerned, we notice that the ϱ^{2D} plots in Figure 4.10 have similar trends to the ones obtained for the single-mirrored datasets in Figure 4.9. We should therefore expect $\mathcal{D}_{\text{Jenga4}}$, $\mathcal{D}_{\text{Slices4}}$ and $\mathcal{D}_{\text{Ulike4}}$ to perform similarly to $\mathcal{D}_{\text{Jenga}}$, $\mathcal{D}_{\text{Slices}}$ and $\mathcal{D}_{\text{Ulike}}$. However, dataset $\mathcal{D}_{\text{Slices4}}$ decreases faster than $\mathcal{D}_{\text{Slices}}$, reaching a ϱ^{2D} value of ~ 0.2 instead of ~ 0.34 (these are the ϱ^{2D} values produced by the last meshes of $\mathcal{D}_{\text{Slices4}}$ and $\mathcal{D}_{\text{Slices}}$ in Figure 4.10 and Figure 4.9, respectively). Last, the ϱ^{2D} plot of $\mathcal{D}_{\text{Ulike4}}$ is significantly worse than the one of $\mathcal{D}_{\text{Ulike}}$ (and than any other), both in terms of distance from $y = 1$ and slope.

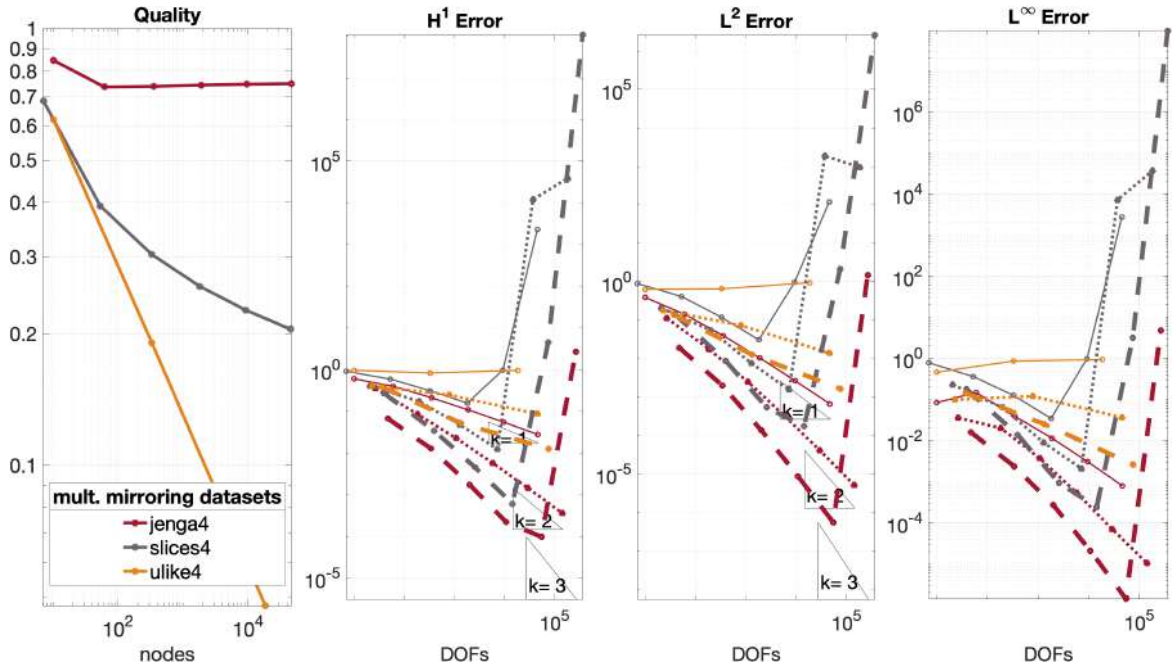


Figure 4.10: Mesh quality indicator $\varrho^{2\mathrm{D}}$, H^1-seminorm, L^2-norm and L^∞-norm of the approximation errors relative to the multiple mirroring datasets. Solid, dashed and dotted lines relate to $k = 1, 2, 3$ respectively.

Verification In this setting, all datasets diverge badly, and this is principally due to very poor conditioning in the matrices involved in the calculations (see Table 4.3). Dataset $\mathcal{D}_{\mathrm{Jenga4}}$ and $\mathcal{D}_{\mathrm{Slices4}}$ maintain a similar trend to the ones in Figure 4.9 until numerical problems cause $\mathrm{cond}(\mathbf{G})$ and $\mathrm{cond}(\mathbf{H})$ to explode up to over 10^{30} for $\mathcal{D}_{\mathrm{Jenga4}}$ and 10^{18} for $\mathcal{D}_{\mathrm{Slices4}}$. In these conditions, projection matrices $\mathbf{\Pi}_k^\nabla$ and $\mathbf{\Pi}_k^0$ become meaningless, and the method diverges. The situation slightly improves for $\mathcal{D}_{\mathrm{Ulike4}}$: $\mathrm{cond}(\mathbf{H})$ is still 10^{16}, but the discrepancy of $\mathbf{\Pi}_k^\nabla$ and $\mathbf{\Pi}_k^0$ remain acceptable. As a result, $\mathcal{D}_{\mathrm{Ulike4}}$ does not properly explode, but the approximation error and the convergence rate are much worse than those seen in Figure 4.9. We notice that, since it only depends on the geometry of the elements, $\varrho^{2\mathrm{D}}$ is not affected by numerical errors. As a consequence, its predictions remain reliable as long as $\mathrm{cond}(\mathbf{G})$ and $\mathrm{cond}(\mathbf{H})$ remain low. Dataset $\mathcal{D}_{\mathrm{Jenga4}}$ converges as expected, i.e., with optimal rate, until the last mesh for $k = 3$, when numerical problems appear which $\varrho^{2\mathrm{D}}$ is not able to predict. As above, $\mathcal{D}_{\mathrm{Slices4}}$ performs similarly to $\mathcal{D}_{\mathrm{Slices}}$ until condition numbers explode, in the last two meshes for every value of k. Last, we can observe how, even if $\mathcal{D}_{\mathrm{Ulike4}}$ does not properly explode (as it suffers less from numerical problems, cf. Table 4.3), the approximation error and the convergence rate are the worse among all the considered datasets.

Table 4.3: Summary of numerical performance for two-dimensional datasets. We report the $\log_{10}$ of the original values for the condition number of $\mathbf{G}$ and $\mathbf{H}$ and the discrepancy of projection matrices $\mathbf{\Pi}_k^{\nabla}$ and $\mathbf{\Pi}_k^{0}$. Note that for $k < 3$ we have $\mathbf{\Pi}_k^{0} = \mathbf{\Pi}_k^{\nabla}$.

dataset	$\mathcal{D}_{\text{Triangle}}$			$\mathcal{D}_{\text{Maze}}$			$\mathcal{D}_{\text{Star}}$			$\mathcal{D}_{\text{Jenga}}$			$\mathcal{D}_{\text{Slices}}$			$\mathcal{D}_{\text{Ulike}}$		
k	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3
cond($\mathbf{G}$)	0	2	5	2	3	6	1	3	6	1	5	10	2	4	6	1	4	7
cond($\mathbf{H}$)	2	5	7	2	5	8	3	6	9	4	9	14	2	8	10	3	7	10
$\lvert\mathbf{\Pi}_k^{\nabla}\mathbf{D} - \mathbf{I}\rvert$	-13	-11	-9	-12	-10	-8	-12	-10	-8	-12	-8	-5	-12	-10	-9	-13	-10	-8
$\lvert\mathbf{\Pi}_k^{0}\mathbf{D} - \mathbf{I}\rvert$			-10			-8			-7			-5			-5			-7

dataset	$\mathcal{D}_{\text{Jenga4}}$			$\mathcal{D}_{\text{Slices4}}$			$\mathcal{D}_{\text{Ulike4}}$		
k	1	2	3	1	2	3	1	2	3
cond($\mathbf{G}$)	6	18	31	6	8	10	2	6	11
cond($\mathbf{H}$)	13	26	39	2	15	18	5	10	16
$\lvert\mathbf{\Pi}_k^{\nabla}\mathbf{D} - \mathbf{I}\rvert$	-9	3	13	-8	-6	-5	-13	-8	-5
$\lvert\mathbf{\Pi}_k^{0}\mathbf{D} - \mathbf{I}\rvert$			20			8			-4

4.2.1.4 Analysis of the Stability Term

In order to investigate the impact of the choice of the stability term on the performance of the method, we compare the results obtained with four different formulations of $S_h^E(\cdot,\cdot)$. The results shown in Figure 4.8, 4.9, and 4.10 were relative to the D-recipe stabilization (3.19). In addition to this, we consider the particular case of the dofi-dofi stabilization (3.20), as well as the H^1 and L^2 trace forms (3.30), (3.31). In Figure 4.11 we exhibit two representative cases relative to $\mathcal{D}_{\text{Slices}}$ and $\mathcal{D}_{\text{Maze}}$ datasets, the others being very similar.

The stability terms, at least the considered ones, seem to have a very small impact on the convergence of the method as long as the convergence rate remains optimal. This is the case for $\mathcal{D}_{\text{Slices}}$ (Figure 4.11(a)) and for the first meshes of $\mathcal{D}_{\text{Maze}}$ (Figure 4.11(b)). On the other side, when the method does not work properly, all types of stabilization lead to similar misbehavior, as happens for the last meshes of $\mathcal{D}_{\text{Maze}}$. Analogous results were obtained for all datasets: whenever the method works, all stabilizations lead to equally accurate approximations; when the method does not work, there is not a particular stabilization providing better results than the others. For example, in the case of $\mathcal{D}_{\text{Slices}}$ the L^2 trace stabilization seems to be more problematic than the others, while for $\mathcal{D}_{\text{Maze}}$ the H^1 trace seems the least reliable, and in general, the worst stabilization varies from datasets to dataset.

4.2.1.5 Error Localization

As a further investigation, we analyze the distribution of the approximation errors across the elements of a mesh. In Figure 4.12 we consider two examples from datasets $\mathcal{D}_{\text{Maze}}$ and $\mathcal{D}_{\text{Jenga}}$: we only report the H^1 error because the L^2 and the L^∞ errors produce very similar results. First, we color each element of the mesh with respect to the value of the elemental quality

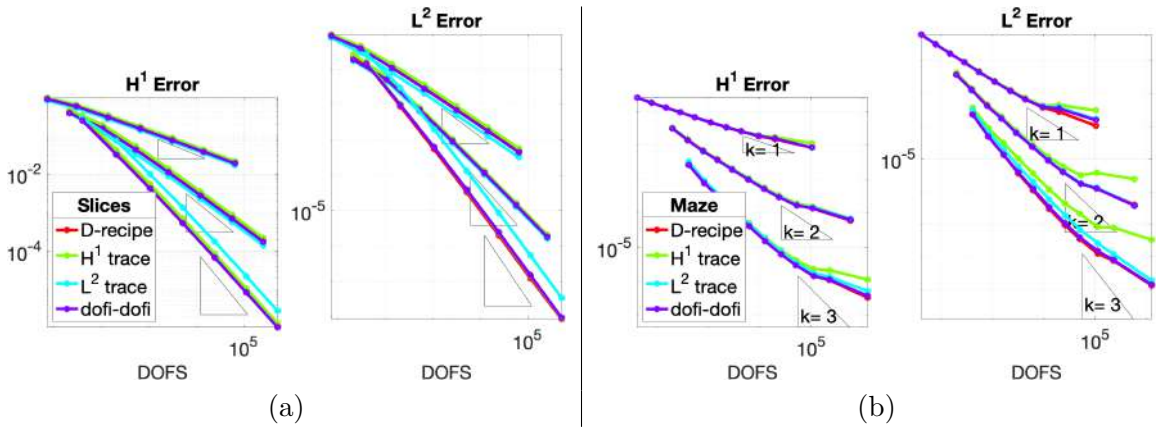


Figure 4.11: H^1-seminorm and L^2-norm of the approximation errors relative to $\mathcal{D}_{\text{Slices}}$ (a) and $\mathcal{D}_{\text{Maze}}$ (b) datasets, with different stability terms.

indicator $\varrho^{2\text{D}}(E) := \varrho^{2\text{D}}{}_{|E}(\Omega_h)$, defined in Section 3.4. Then, we visually compare this colored mesh with the same mesh colored with respect to the H^1 error, for $k = 1, 2, 3$, produced by the VEM on it:

$$\epsilon(E) = -\log \frac{|u - u_h|_{1,E}}{|u|_{1,\Omega}}, \quad \forall E \in \Omega_h,$$

where the negative sign is introduced so that high error values correspond in color to low-quality elements (remember that $\varrho^{2\text{D}}(E)$ is 1 if E is an equilateral triangle and 0 if E is not star-shaped). Moreover, ϵ values are re-scaled in the range $(\min_{E\in\Omega_h} \epsilon(E), \max_{E\in\Omega_h} \epsilon(E))$ in order to highlight differences between the elements. In particular, this means that there is no relationship between a certain color in the figure for $k = 1$ and the same color in the case $k = 2$ or 3.

In the first column of Figure 4.12 we can observe how the quality indicator perfectly "recognizes" the pathological elements, assigning them a deep blue color. Regarding the error, we can see similar color patterns across the columns, which depend on the function we are approximating. Being the ground-truth (4.2) a sinusoidal function, the error is naturally distributed along "waves" which vary with the order of the method. Besides this, it is still appreciable how poor quality elements produce higher errors than their neighbors, highlighting once again a correlation between the quality indicator and the performance of the VEM.

4.2.2 Analysis of the 3D Dataset

For the analysis of the three-dimensional datasets, we solve the discrete Poisson problem (3.4) with the VEM (3.5) over each mesh of each of the datasets defined in Section 4.1.2, using as groundtruth the function

$$u(x, y, z) = x^3y^2z + x\sin(2\pi xy)\sin(2\pi yz)\sin(2\pi z), \ (x, y, z) \in \Omega. \tag{4.3}$$

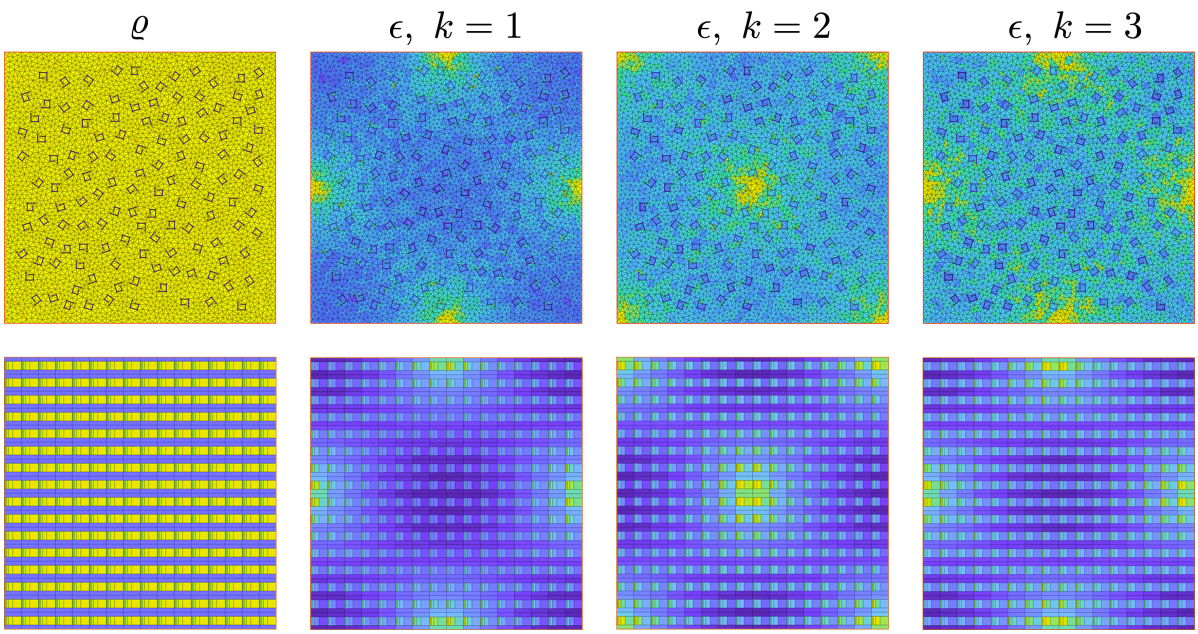


Figure 4.12: Localization of the quality indicator ϱ^{2D} and H^1 error ϵ in meshes from $\mathcal{D}_{\text{Maze}}$ (top) and $\mathcal{D}_{\text{Jenga}}$ (bottom) datasets.

Differently from Section 4.2.1, we limit our analysis to the case $k = 1$. We focus on the linear VEM for computational simplicity, because we think that this is the most interesting choice in three-dimensional practical engineering applications. In fact, as stated in Section 4.1, the 3D datasets were defined with the goal of generating ordinary meshes, to understand if the quality indicators are able to spot small quality differences between them. However, in the two-dimensional case (Section 4.2.1), the lowest-order and higher-order VEM performance was seen to be optimal or, at least, similar on the same mesh families. Based on such experience, we expect the information given by the indicator to remain valid also for higher-order VEM formulations.

4.2.2.1 Tetrahedral Datasets

Predictions Looking at the quality plot in Figure 4.13 (the leftmost) we would say that the VEM should converge with the optimal rate over almost all tetrahedral datasets, as their ϱ^{3D} tend to get horizontal. One exception is $\mathcal{D}_{\text{tet-anisotropic}}$, which has a negative trend and therefore is not expected to converge properly. We can also observe how the slope of $\mathcal{D}_{\text{tet-parallel}}$ becomes positive in the last mesh: this should indicate a more than optimal convergence rate. Regarding the error magnitude, represented by the overall distance from the top of the plot, we can predict that $\mathcal{D}_{\text{tet-bcl}}$ will produce the smallest errors, being the one with the highest quality. This is reasonable because this dataset is composed mainly of equilateral tetrahedra. We can then order decreasingly the other datasets according to their quality: $\mathcal{D}_{\text{tet-poisson}}$, $\mathcal{D}_{\text{tet-uniform}}$, $\mathcal{D}_{\text{tet-random}}$,

$\mathcal{D}_{\text{tet-parallel}}$, and $\mathcal{D}_{\text{tet-anisotropic}}$, and we expect the errors' magnitudes to behave accordingly.

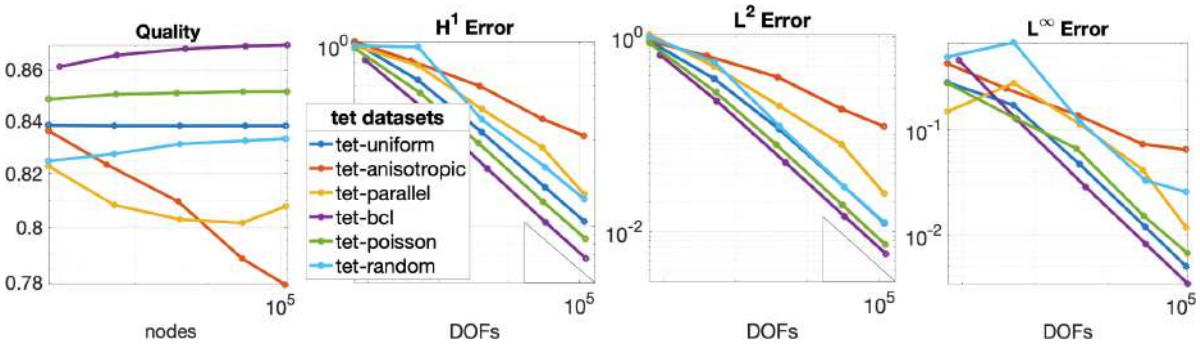


Figure 4.13: Mesh quality indicator ϱ^{3D}, H^1-seminorm, L^2-norm and L^∞-norm of the approximation errors relative to the tetrahedral datasets.

Verification The convergence rates of the tetrahedral datasets in the error plots of Figure 4.13 faithfully respect all the above considerations. In both H^1 and L^2-norms, the method converges with the optimal rate (the one suggested by the reference triangle) over almost all the datasets, with $\mathcal{D}_{\text{tet-parallel}}$ even improving the rate in its last mesh. The only exception, as expected, is dataset $\mathcal{D}_{\text{tet-anisotropic}}$. We checked the condition numbers for the linear systems that are built on the meshes of $\mathcal{D}_{\text{tet-anisotropic}}$ and we verified that their values are reasonably small, i.e., in the range $[1, 10^6]$. These values are comparable to the condition numbers seen in the other datasets. In the L^∞-norm the situation is similar, even if $\mathcal{D}_{\text{tet-random}}$ has an unexpected peak in the last mesh. The error magnitudes, i.e. the distance of the line from the $y-$axis, perfectly follow the ordering suggested by the quality plot. The dataset which produces the smallest errors is $\mathcal{D}_{\text{tet-bcl}}$. After that, in the H^1 and L^2 plots we have $\mathcal{D}_{\text{tet-poisson}}$, $\mathcal{D}_{\text{tet-uniform}}$ and $\mathcal{D}_{\text{tet-random}}$, which tend to become very close, then $\mathcal{D}_{\text{tet-parallel}}$ and last $\mathcal{D}_{\text{tet-anisotropic}}$. The situation slightly changes if we look at the L^∞ error: in this case, $\mathcal{D}_{\text{tet-uniform}}$ performs better than $\mathcal{D}_{\text{tet-poisson}}$, probably due to a bunch of poor quality elements which do not particularly affect the overall accuracy of the method.

4.2.2.2 Hexahedral Datasets

Predictions Results for the hexahedral datasets are shown in Figure 4.14. Similar to what happened for the tetrahedral datasets, the meshes produced by the anisotropic sampling have very poor quality. While $\mathcal{D}_{\text{hex-uniform}}$ and $\mathcal{D}_{\text{hex-parallel}}$ tend to flatten, with the second one increasing in the last refinement, the ϱ^{3D} value for the meshes of $\mathcal{D}_{\text{hex-anisotropic}}$ keeps decreasing. Our prediction is therefore to have optimal convergence on $\mathcal{D}_{\text{hex-uniform}}$ and $\mathcal{D}_{\text{hex-parallel}}$ and bad results with $\mathcal{D}_{\text{hex-anisotropic}}$. In addition, $\mathcal{D}_{\text{hex-uniform}}$ is expected to produce smaller errors than $\mathcal{D}_{\text{hex-parallel}}$.

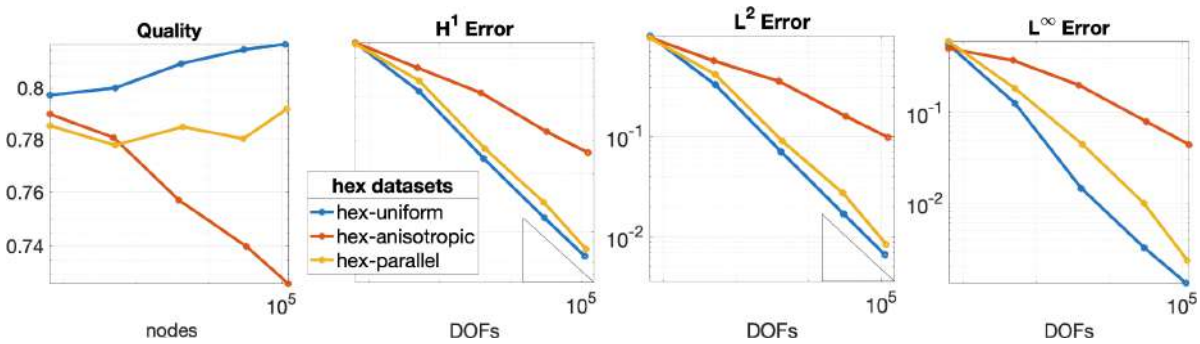


Figure 4.14: Mesh quality indicator ϱ^{3D}, H^1-seminorm, L^2-norm and L^∞-norm of the approximation errors relative to the hexahedral datasets.

Verification In the error plots of Figure 4.14, all the predictions are confirmed. The VEM converges perfectly over $\mathcal{D}_{\text{hex-uniform}}$ and $\mathcal{D}_{\text{hex-parallel}}$, with the second one producing higher errors than the first one and improving its convergence rate in the last refinement. Instead, $\mathcal{D}_{\text{hex-anisotropic}}$ does not produce a correct convergence rate in the H^1 and L^2 plots, and also in the L^∞ plot exhibits a significantly slower rate with respect to the other datasets. Also, in this case, the condition numbers for the linear systems are not particularly bigger than the ones of the other datasets.

4.2.2.3 Voronoi Datasets

Predictions In Figure 4.15, results relative to the Voronoi datasets are shown. The quality of all three datasets tends to stabilize to a constant value, and this makes us presume a correct convergence rate for all of them. We can expect $\mathcal{D}_{\text{voro-bcl}}$ to produce smaller errors than the other two, and $\mathcal{D}_{\text{voro-random}}$ to be the less accurate.

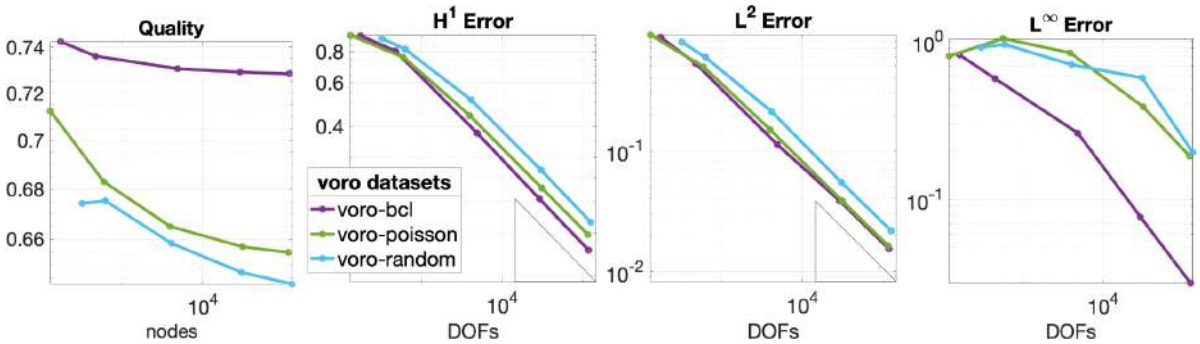


Figure 4.15: Mesh quality indicator ϱ^{3D}, H^1-seminorm, L^2-norm and L^∞-norm of the approximation errors relative to the Voronoi datasets.

Verification Looking at the H^1 and L^2 error plots we notice how all datasets converge properly, and the accuracy of the approximation follows the order foreseen by the indicator:

$\mathcal{D}_{\text{voro-bcl}}$, $\mathcal{D}_{\text{voro-poisson}}$, and $\mathcal{D}_{\text{voro-random}}$. The L^∞ plot is less similar to the other two in this case, but still, we can recognize a common pattern.

4.2.2.4 Polyhedral Datasets

Predictions Last, in Figure 4.16 we report the analysis of the polyhedral datasets. The indicator $\varrho^{3\text{D}}$ suggests that $\mathcal{D}_{\text{poly-poisson}}$ and $\mathcal{D}_{\text{poly-random}}$ converge perfectly. Regarding $\mathcal{D}_{\text{poly-parallel}}$, the indicator seems to flatten and then increase in the last refinement. The convergence should therefore be optimal for the first meshes and more than optimal for the last one. The most accurate dataset should be $\mathcal{D}_{\text{poly-poisson}}$ and the least accurate $\mathcal{D}_{\text{poly-parallel}}$.

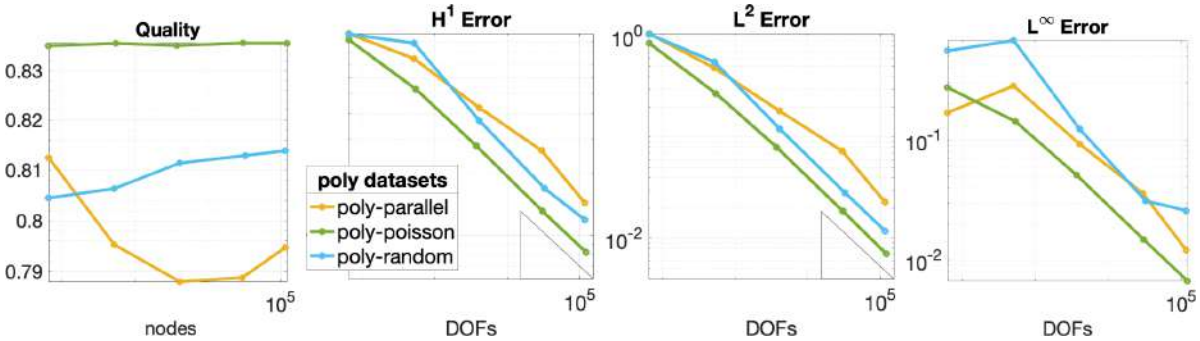


Figure 4.16: Mesh quality indicator $\varrho^{3\text{D}}$, H^1-seminorm, L^2-norm and L^∞-norm of the approximation errors relative to the polyhedral datasets.

Verification The method performs essentially as expected. All datasets produce optimal rates and $\mathcal{D}_{\text{poly-parallel}}$ converges even faster than the reference in the last refinement. Dataset $\mathcal{D}_{\text{poly-random}}$ has a peak in the last mesh with the L^∞ error: this is similar to what happened with $\mathcal{D}_{\text{tet-random}}$ and it is probably due to the same bad-shaped element (remember that tetrahedral and polyhedral meshes differ only for the 20% of their elements). Concerning the errors' magnitude, as foreseen by the indicator, $\mathcal{D}_{\text{poly-poisson}}$ is the most accurate, and then we have $\mathcal{D}_{\text{poly-random}}$ and $\mathcal{D}_{\text{poly-parallel}}$.

4.2.3 Discussion

We conclude with some more general considerations on the results obtained in Section 4.2.1 and Section 4.2.2.

When a sufficient subset of the geometrical assumptions defined in Section 3.2 is respected, that is, with $\mathcal{D}_{\text{Triangle}}$ or with all datasets from uniform sampling and BCL, the VEM performance is obviously optimal. We also built examples of datasets, like $\mathcal{D}_{\text{Ulike}}$ or $\mathcal{D}_{\text{tet-anisotropic}}$, which violate several regularity assumptions, and over which the VEM shows a sub-optimal convergence rate, or diverges. However, the first important consideration is that we experimentally verified

how the VEM works, with a good convergence rate, also on datasets clearly outside the assumptions. The most significant examples of this are probably $\mathcal{D}_{\text{Jenga}}$ and $\mathcal{D}_{\text{Slices}}$ in 2D, or $\mathcal{D}_{\text{voro-random}}$ in 3D.

These results are not unexpected, as the geometrical assumptions are only *sufficient* conditions for the convergence of the method, but confirm the suspicion that the current restrictions on the meshes are probably more severe than necessary. For instance, dataset $\mathcal{D}_{\text{Jenga}}$ violates all the 2D assumptions and still converges better than $\mathcal{D}_{\text{Maze}}$, $\mathcal{D}_{\text{Star}}$, and $\mathcal{D}_{\text{Slices}}$ which only violate a couple of them. Among the 3D datasets, the VEM seems not to particularly suffer the violation of the assumptions due to random point distributions. Datasets from parallel and random samplings produce correct convergence rates, even if the approximation errors are higher than those relative to uniform sampling or BCL. In conclusion, the relationship between the geometrical assumptions respected by a dataset and the performance of the VEM on it is not so obvious, especially when we try to violate at least one of them.

In those situations, our mesh quality indicator turns out to be particularly useful in predicting the result of the numerical approximation. Its effectiveness lies in the ability to capture a qualitative measure of the fulfillment of each assumption. The mesh quality indicator has been able to properly predict the behavior of the VEM, both in terms of convergence rate and error magnitude, up to a certain precision. It showed up to be particularly accurate when compared to the H^1 and the L^2-norms of the error, while it did not always manage to capture the oscillations of the L^∞-norm. The prediction may be inaccurate in presence of very similar performance (the case of $\mathcal{D}_{\text{Maze}}$ and $\mathcal{D}_{\text{Star}}$), or in extreme situations in which the numerical problems become so significant to overcome any influence that the geometrical features of the mesh could have on the performance (the last meshes of $\mathcal{D}_{\text{Jenga4}}$ and $\mathcal{D}_{\text{Slices4}}$).

The results obtained are encouraging, showing a satisfactory correlation between the errors and this indicator. Consequently, our approach provides an experimental score that is able to predict if a tessellation of a domain can be critical for the VEM. Moreover, it is enough accurate to be employed in a classification process, where a collection of meshes, or datasets, have to be ordered according to their quality. For instance, it could be used to choose among different tessellations of the same domain which one is likely to produce the most accurate results when combined with the VEM.

4.3 Mesh Quality Agglomeration

Alongside the classification purposes, the mesh quality indicator can be exploited to optimize the quality of a mesh, integrating it into adaptation or *agglomeration* algorithms. By agglomeration, we mean a process in which some elements of a mesh get agglomerated to form bigger elements. With *quality agglomeration*, we stress the fact that the purpose of the process is to optimize the global mesh quality. In this sense, it differs from the concept of mesh *coarsening*, in which the

goal is simply to increase the average mesh size. In absence of particular constraints, quality agglomeration also results in mesh coarsening. However, if the mesh contains nodes, edges, faces, or elements which have to be preserved, or if the domain presents non-trivial geometrical features, the effect of quality agglomeration can be localized in particular regions or limited to a small number of elements. In particular, we are interested in operating over the two following types of cells:

1. Elements located in areas of the domain that are poor of significant details or features. In this case, the size of the elements can be increased without significantly impacting the accuracy of the simulation;

2. Bad-shaped or pathological elements. The quality of such elements can be often improved if we merge them with some neighboring cells.

We let the indicator "drive" the agglomeration process, indicating which elements in a mesh should be merged in order to maximize its quality. In this sense, we define an energy functional over the mesh, based on the quality indicator, that we want to minimize. In addition to the driver, we need a "car", i.e., a tool that allows us to navigate the mesh and move efficiently from one element to the other. We interpret the mesh as a graph, where each element represents a node, and any two nodes corresponding to neighboring elements are connected by an arc. The car will be then represented by the *graph-cut* technique, which gives us an optimal order for scouting all the possible combinations of element pairings and reaching a minimum of the energy functional. Since the discussion will be independent of the dimension of the domain, we generically note the quality indicator as ϱ meaning the elemental quality indicators $\varrho^{2\mathrm{D}}(E)$ and $\varrho^{3\mathrm{D}}(E)$ defined in Section 3.4 for every element E in a mesh.

4.3.1 Energy Functional

The energy functional $\mathcal{E}$ that we want to minimize is defined as:

$$\mathcal{E} := \sum_{E,E' \in \Omega_h} dc(E,E') + \lambda \sum_{E,E' \in \Omega_h} sc(E,E'), \tag{4.4}$$

where dc and sc are two *cost functions*, both defined from the product space $\Omega_h \times \Omega_h$ to the unit interval $[0,1]$, and $\lambda \in [0,1]$ is an *agglomeration parameter* that balances the importance of the two terms. Empirically, high values of λ lead to more aggressive agglomerations. The cost functions are defined as follows:

- the *data cost* represents the cost of agglomerating two elements $E, E' \in \Omega_h$, and measures

the potential quality of the element $E \cup E'$. It is defined as:

$$dc(E, E') := \begin{cases} 0 & \text{if } E = E' \\ 1 - \varrho(E \cup E') & \text{if } E \text{ and } E' \text{ are adjacent} \\ 1 & \text{otherwise} \end{cases} \tag{4.5}$$

where $E \cup E'$ is the boolean union of the two neighboring elements, and ϱ is the elemental quality indicator.

- the *smoothness cost* encodes information on the structure of the mesh, assigning zero weight to the non-adjacent elements:

$$sc(E, E') := \begin{cases} 1 & \text{if } E \text{ and } E' \text{ are adjacent and } E \neq E' \\ 0 & \text{otherwise} \end{cases} \tag{4.6}$$

For implementation reasons, both dc and sc are multiplied by the number of elements in Ω_h and then rounded off, in order to obtain integer values.

4.3.2 Graph-cut

The graph-cut (also known as maximum-flow, or minimum-cut [Goldberg and Tarjan, 1988]) is an efficient graph-based technique aimed at segmenting a graph into two or more parts, minimizing a certain energy. It has been successfully employed for images [Boykov et al., 2001, Kolmogorov and Zabin, 2004, Boykov and Kolmogorov, 2004] and to tackle problems like image segmentation, object co-segmentation, and other problems that can be formulated in terms of energy minimization [Yi and Moon, 2012]. There are also contributions towards the extension of graph-cut to 3D data, mainly aimed at segmentation, reconstruction, and generation: for example in [Liu et al., 2015] graph-cut is used to segment triangulated models. We used the implementation provided by the *Multi-label Optimization* algorithm, available at `https://vision.cs.uwaterloo.ca/code/`.

In our context, each node of the graph represents one element E of the mesh Ω_h, and two nodes are connected if their relative elements are adjacent (i.e., if they share an edge in 2D or a face in 3D). Moreover, each node is equipped with a *label*: an integer number that provides information about the node. We note L the set of all possible labels, and a labeling will be a map $\mathcal{L} : \Omega_h \to L$ that assigns a label $l \in L$ to each node $E \in \Omega_h$. What graph-cut essentially does, is to opportunely re-label the nodes: after one iteration, some nodes may share the same label, and this will mean that we want to agglomerate the corresponding elements.

In the Multi-label Optimization algorithm, the cost functions are defined in the form $\widetilde{dc} : \Omega_h \times L \to [0, 1]$ and $\widetilde{sc} : L \times L \to [0, 1]$. To align to this notation, we set $L := \{1, \ldots, \#\Omega_h\}$, and define the trivial labeling $\widetilde{\mathcal{L}} : \Omega_h \to L$ that bijectively maps each element in the mesh to its *index*, i.e., its position in the array of the mesh data structure containing all the elements. Then,

we opportunely compose the cost functions (4.5) and (4.6) with the inverse map $\widetilde{\mathcal{L}}^{-1} : L \to \Omega_h$, which, therefore, connects a label $l \in L$ to the (unique) element $E \in \Omega_h$ with index l:

$$\widetilde{dc}(E, l) := dc(E, \widetilde{\mathcal{L}}^{-1}(l)) = dc(E, E')$$
$$\widetilde{sc}(l_1, l_2) := sc(\widetilde{\mathcal{L}}^{-1}(l_1), \widetilde{\mathcal{L}}^{-1}(l_2)) = sc(E_1, E_2),$$

being E', E_1, and E_2 the nodes whose relative elements have indices l, l_1, and l_2, respectively. Substituting $\widetilde{dc}$ and $\widetilde{sc}$ into (4.4) we obtain a new energy functional, that depends on the particular labeling:

$$\widetilde{\mathcal{E}}(\mathcal{L}) := \sum_{E \in \Omega_h} \widetilde{dc}(E, l_E) + \lambda \sum_{E, E' \in \Omega_h} \widetilde{sc}(l_E, l_{E'}).$$

The minimization problem is then defined as follows, given $\mathcal{P}$ the set of all the possible labelings:

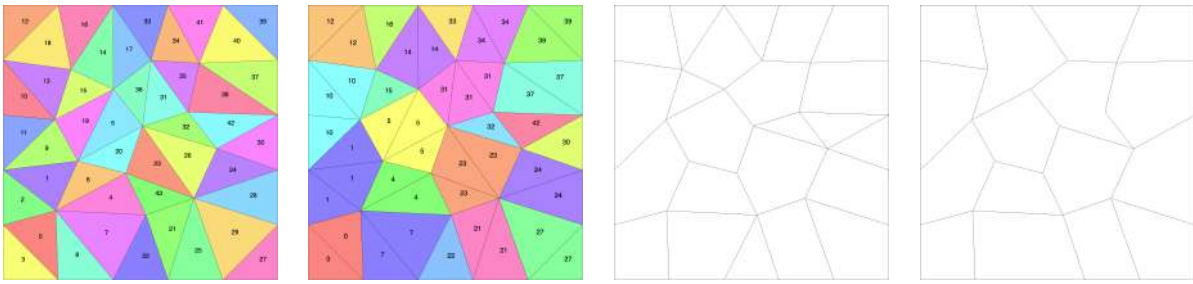
$$\min_{\mathcal{L} \in \mathcal{P}} \widetilde{\mathcal{E}}(\mathcal{L}). \tag{4.7}$$

We solve this problem with the *alpha-beta swap* algorithm [Boykov et al., 2001]. This technique iteratively segments the nodes labeled with a given label α with respect to those with another label β. The two given labels change after each iteration, scouting all the possible combinations. Other algorithms exist in the literature, for example, the so-called *alpha-expansion* algorithm, which requires the function sc to be a metric (i.e., to respect the triangular inequality). The results obtained with *alpha-expansion* are less useful in our context because it generally leads to uneven distributions of the labels. Indeed, this algorithm tries to expand each label as much as possible, generating large portions of the domain with the same label and small isolated areas with different ones, which leads to meshes with unbalanced elements.

4.3.3 Quality Agglomeration Algorithm

Based on the mesh quality indicator and the graph-cut algorithm, we can define our quality agglomeration algorithm. Given an input mesh, we initialize the node labels with the trivial labeling $\widetilde{\mathcal{L}}$ (Figure 4.17(a)). Then, we run the graph-cut algorithm with a certain value of the parameter λ, and find new labels for the elements (Figure 4.17(b)). We run the graph-cut algorithm until convergence, i.e. until the energy term $\widetilde{\mathcal{E}}$ does not decrease anymore: this typically takes less than 5 iterations. Last, all elements sharing the same label are merged (Figure 4.17(c)), also merging aligned edges in the newly generated elements. At this stage, we make sure to preserve eventual constraints found on nodes, edges, faces, or elements of the initial mesh. We can compute different agglomerated versions of the same mesh by choosing different values of the parameter λ (Figure 4.17(c,d)). A limitation of this approach is that it only compares elements pairwise. Given an element E with neighboring elements E' and E'', the algorithm will compute separately the potential quality of $E \cup E'$ and that of $E \cup E''$. In the case both qualities are considered good, E' and E'' will be assigned the same label of E,

without having checked the potential quality of $E \cup E' \cup E''$. This problem could be partially controlled (but with an higher computational cost) by merging the elements with the same label after each graph-cut iteration, instead of agglomerating only after the graph-cut convergence.



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

Figure 4.17: (a) Initial mesh, colored w.r.t. element labels (in black), (b) mesh after graph-cut with $\lambda = 0.25$, colored w.r.t. element labels (in black), (c) agglomerated mesh with $\lambda = 0.25$, (d) agglomerated mesh with $\lambda = 1.0$.

In the following, we test the effects of the quality agglomeration algorithm over a selection of datasets. We solve the discrete Poisson problem (3.4) with the two- or three-dimensional VEM (3.5) (according to the context), on both the original and agglomerated datasets. Each agglomerated mesh contains a smaller number of elements, faces, nodes, and therefore degrees of freedom of the VEM space, with respect to its original version, thus requiring a smaller computational cost for the VEM. At the same time, we expect the errors produced by the agglomerated meshes to be slightly higher, as for every removed degree of freedom we have less information on the numerical problem that we want to solve. We are interested in understanding how this reduction of degrees of freedom impacts the accuracy of the VEM, and also if it produces any effect on the convergence rate of the method.

Test 1 The first context in which we test the quality agglomeration algorithm is on the two-dimensional dataset $\mathcal{D}_{\text{Maze}}$ from Section 4.1.1. For computational reasons, while the original dataset contained 11 meshes, we here reduce it to its first 7 meshes. We generate two agglomerated datasets: $\mathcal{D}^{\lambda_1}_{\text{Maze}}$ with parameter $\lambda_1 = 0.25$ and $\mathcal{D}^{\lambda_2}_{\text{Maze}}$ with parameter $\lambda_2 = 1.0$, presented in Figure 4.18. After the agglomeration, the number of nodes, edges, and elements in each mesh significantly decreases, and consequently also the number of degrees of freedom, as shown in Figure 4.19. While for $k = 1$ the degrees of freedom coincide with the mesh nodes, for a higher order of the method they also depend on edges and elements. The agglomeration process operates mainly on edges and elements, therefore the reduction of degrees of freedom is particularly appreciable for $k > 1$. In Table 4.4 we report the differences between the degrees of freedom of the three datasets in the case $k = 3$.

In Figure 4.20 we repeat the VEM analysis presented in Section 4.2.1, measuring the quality

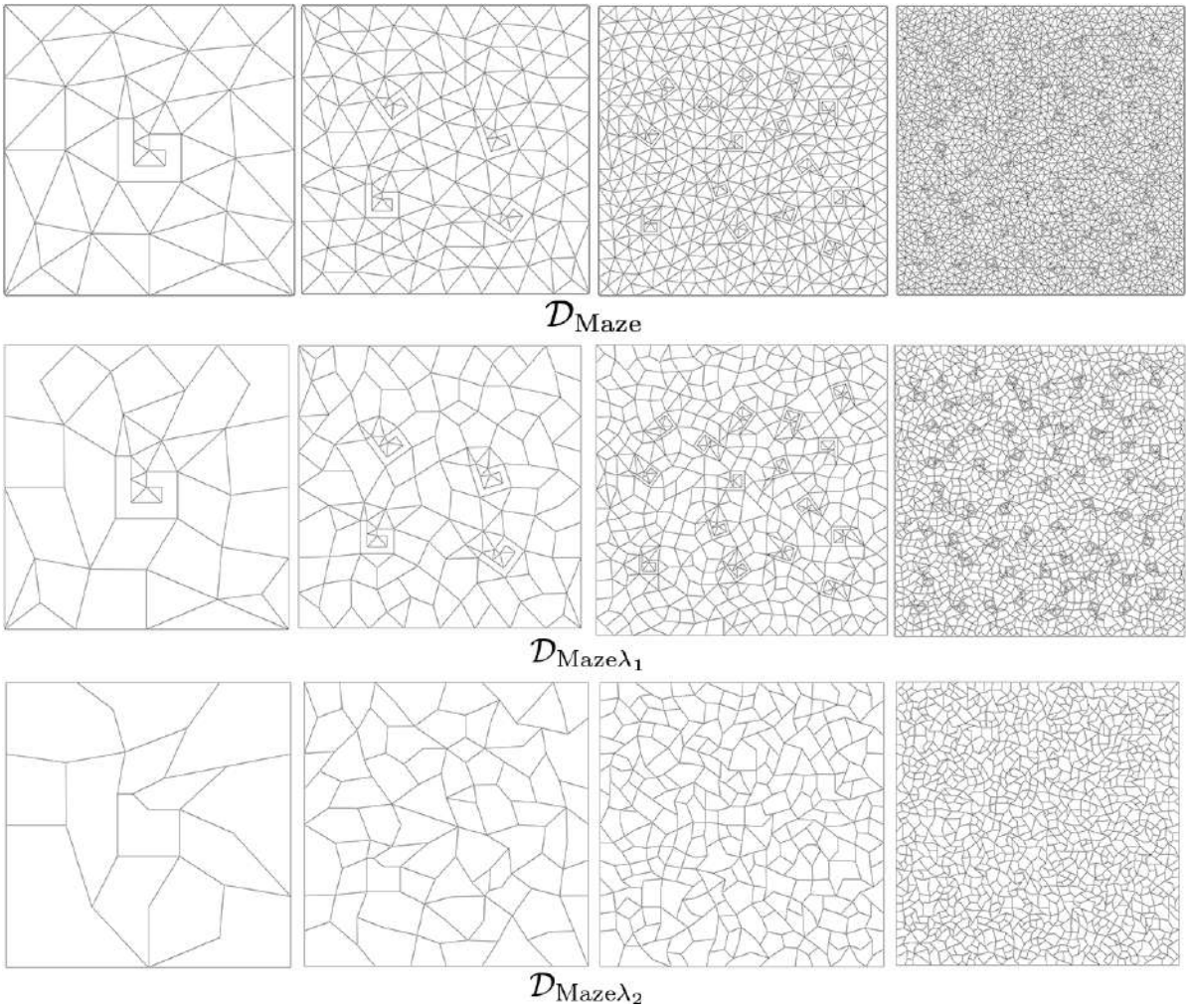


Figure 4.18: Meshes $\Omega_0, \Omega_2, \Omega_4, \Omega_6$ from datasets $\mathcal{D}_{\mathrm{Maze}}$ (top), $\mathcal{D}^{\lambda_1}_{\mathrm{Maze}}$ (middle), and $\mathcal{D}^{\lambda_2}_{\mathrm{Maze}}$ (bottom), with $\lambda_1 = 0.25$ and $\lambda_2 = 1.0$.

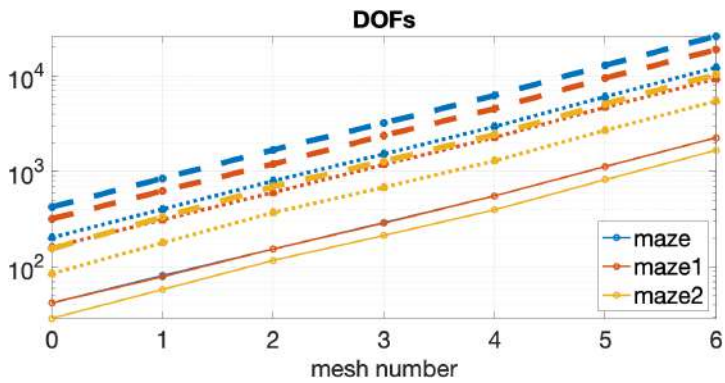


Figure 4.19: Degrees of freedom of datasets $\mathcal{D}_{\mathrm{Maze}}$, $\mathcal{D}^{\lambda_1}_{\mathrm{Maze}}$, and $\mathcal{D}^{\lambda_2}_{\mathrm{Maze}}$, for $k = 1, 2, 3$. Solid, dashed and dotted lines relate to $k = 1, 2, 3$ respectively.

Table 4.4: Degrees of freedom for the meshes in $\mathcal{D}_{\text{Maze}}$, $\mathcal{D}^{\lambda_1}_{\text{Maze}}$, and $\mathcal{D}^{\lambda_2}_{\text{Maze}}$ in the case $k = 3$.

$k = 3$	Ω_0	Ω_1	Ω_2	Ω_3	Ω_4	Ω_5	Ω_6
$\mathcal{D}_{\text{Maze}}$	424	846	1680	3216	6258	12921	26099
$\mathcal{D}^{\lambda_1}_{\text{Maze}}$	319	625	1195	2377	4553	9508	18811
$\mathcal{D}^{\lambda_2}_{\text{Maze}}$	155	332	694	1280	2429	5103	10338

indicator and the H^1, L^2, and L^∞ errors over each mesh. We notice how the mesh quality preserves its rate in the agglomerated datasets, even if the quality score is lower over $\mathcal{D}^{\lambda_1}_{\text{Maze}}$ and $\mathcal{D}^{\lambda_2}_{\text{Maze}}$. The lower quality levels are mainly due to the impact of indicator $\varrho_3^{2\text{D}}$, because agglomeration generates elements with a higher number of edges. Concerning the errors, the convergence rates of $\mathcal{D}^{\lambda_1}_{\text{Maze}}$ and $\mathcal{D}^{\lambda_2}_{\text{Maze}}$ remain optimal (as foreseen by the quality indicator). Moreover, there is almost no difference between the accuracy of the original dataset and those of the agglomerated ones. This is particularly interesting if combined with the information from Figure 4.19, i.e., the reduction of the number of degrees of freedom. In practice, after the agglomeration process, we obtain meshes that produce a performance similar to those of the original mesh, but with a lower computational cost. In particular, for $k = 3$, the number of degrees of freedom in meshes from $\mathcal{D}^{\lambda_2}_{\text{Maze}}$ is less than halved with respect to that of meshes from $\mathcal{D}_{\text{Maze}}$ (see Table 4.4), and the error plots in the three norms almost coincide.

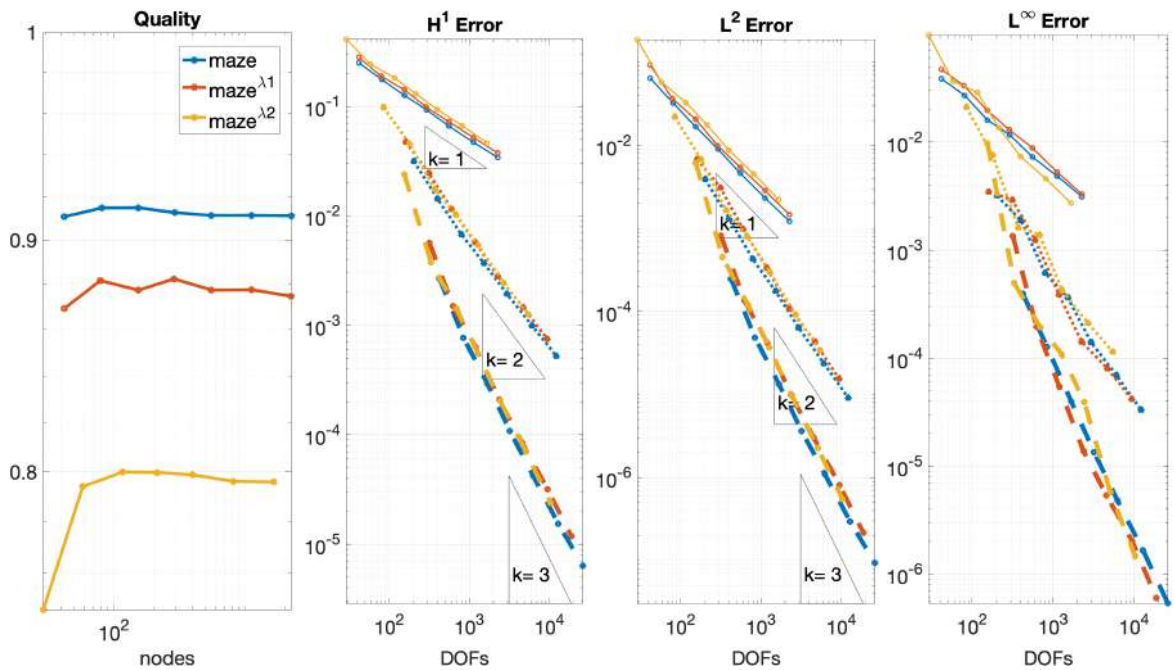


Figure 4.20: Mesh quality indicator $\varrho^{2\text{D}}$, H^1-seminorm, L^2-norm and L^∞-norm of the approximation errors relative to the datasets $\mathcal{D}_{\text{Maze}}$, $\mathcal{D}^{\lambda_1}_{\text{Maze}}$ and $\mathcal{D}^{\lambda_2}_{\text{Maze}}$. Solid, dashed and dotted lines relate to $k = 1, 2, 3$ respectively.

Test 2 As the second setting, we consider the four planar meshes defined in [Antonietti and Manuzzi, 2022], obtained by recursive application of the midpoint strategy (we thank the authors for kindly sharing these meshes). We denote them $\mathcal{MP}_1$, $\mathcal{MP}_2$, $\mathcal{MP}_3$, and $\mathcal{MP}_4$, and agglomerate them with parameter $\lambda = 0.5$, see Figure 4.21. These meshes are particularly complex, as they contain high numbers of skewed and deformed polygons.

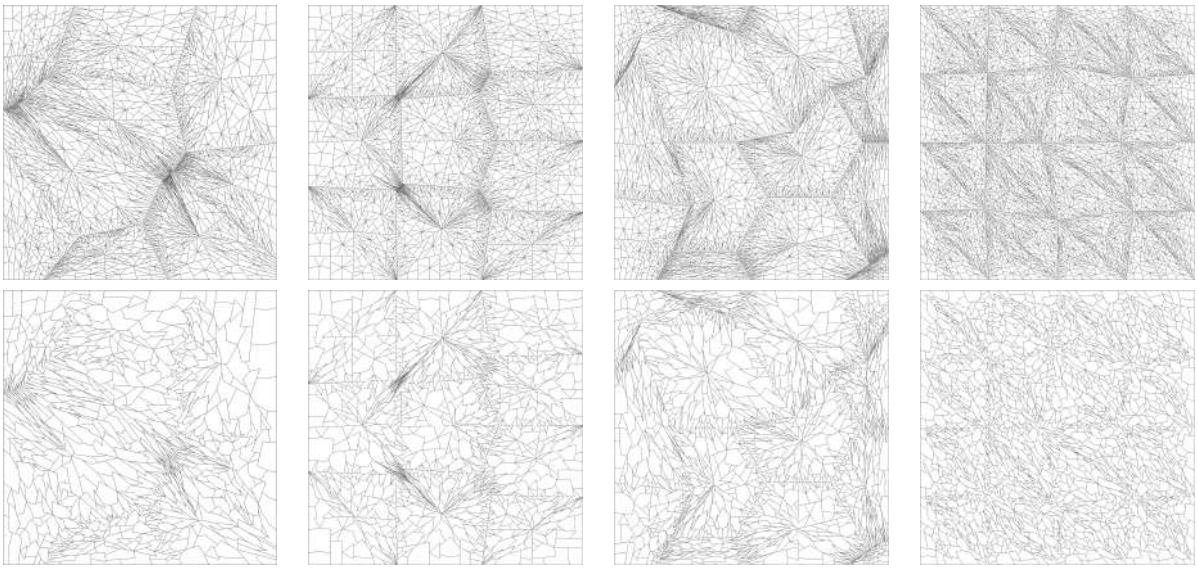


Figure 4.21: Midpoint refined meshes (top) and their agglomerated version (bottom), with parameter $\lambda = 0.5$. From left to right, $\mathcal{MP}_1$, $\mathcal{MP}_2$, $\mathcal{MP}_3$, and $\mathcal{MP}_4$.

In this case, we compare the quality and the errors of the original mesh $\mathcal{MP}$ with those relative to the agglomerated mesh $\mathcal{MP}^\lambda$. In Figure 4.22 we inverted the $x-$axis, so that for each plot we have the original mesh on the left and its agglomeration on the right. We report the quality plot for completeness, even if, in this case, it is not particularly interesting as we do not have a sequence of refinements. We can only notice how, as for dataset $\mathcal{D}_{\text{Maze}}$, the quality score of the single mesh decreases after the agglomeration. Concerning the errors, we cannot talk about convergence because we are only considering two meshes per time. However, we can make some considerations according to the general error estimates (3.24) and (3.25). If the original mesh $\mathcal{MP}$ has mesh size h, then (3.24) states that the order-k VEM should produce an approximation error ε_{H^1} proportional to h^k and to the term $S := ||u||_{k+1,\Omega} + |f|_{k,\Omega}$. If we consider a mesh $\mathcal{MP}^\lambda$, similar to $\mathcal{MP}$ (i.e., with similar term S) but with mesh size $h_\lambda = \alpha h$, we should have an approximation error $\varepsilon_{H^1\lambda}$ proportional to $\alpha^k \varepsilon_{H^1}$. The same holds for the L^2 error, using (3.25) and $k+1$, and we can translate these relationships in terms of the degrees of freedom instead of the mesh size. In conclusion, we can draw the reference triangles in Figure 4.22 for the different values of k. Note that, as we inverted the $x-$axis, the triangles are flipped with respect to the reference triangles in the previous plots. We observe how the H^1 and L^2 errors scale properly over all the midpoint meshes after the agglomeration. We are

significantly reducing the number of degrees of freedom (they are at least halving in each case) and the approximation error consequently increases, but we maintain optimal error values. Last, we do not have a theoretical reference for the L^∞ error values, hence we only observe that it scales consistently with the previous two.

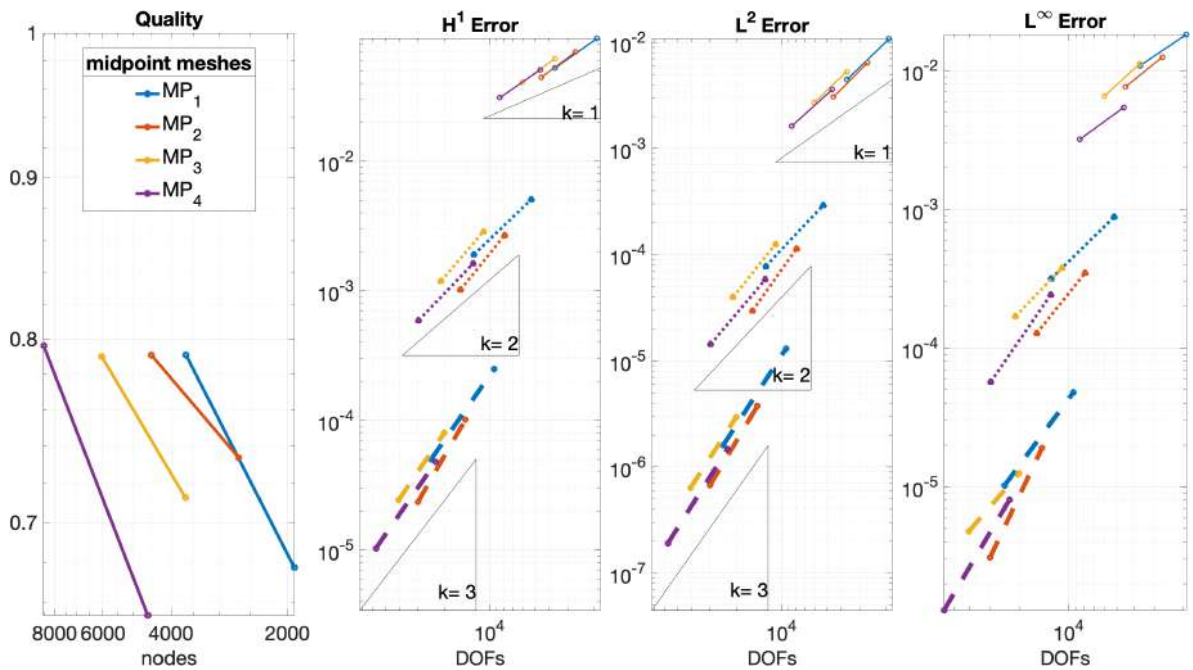
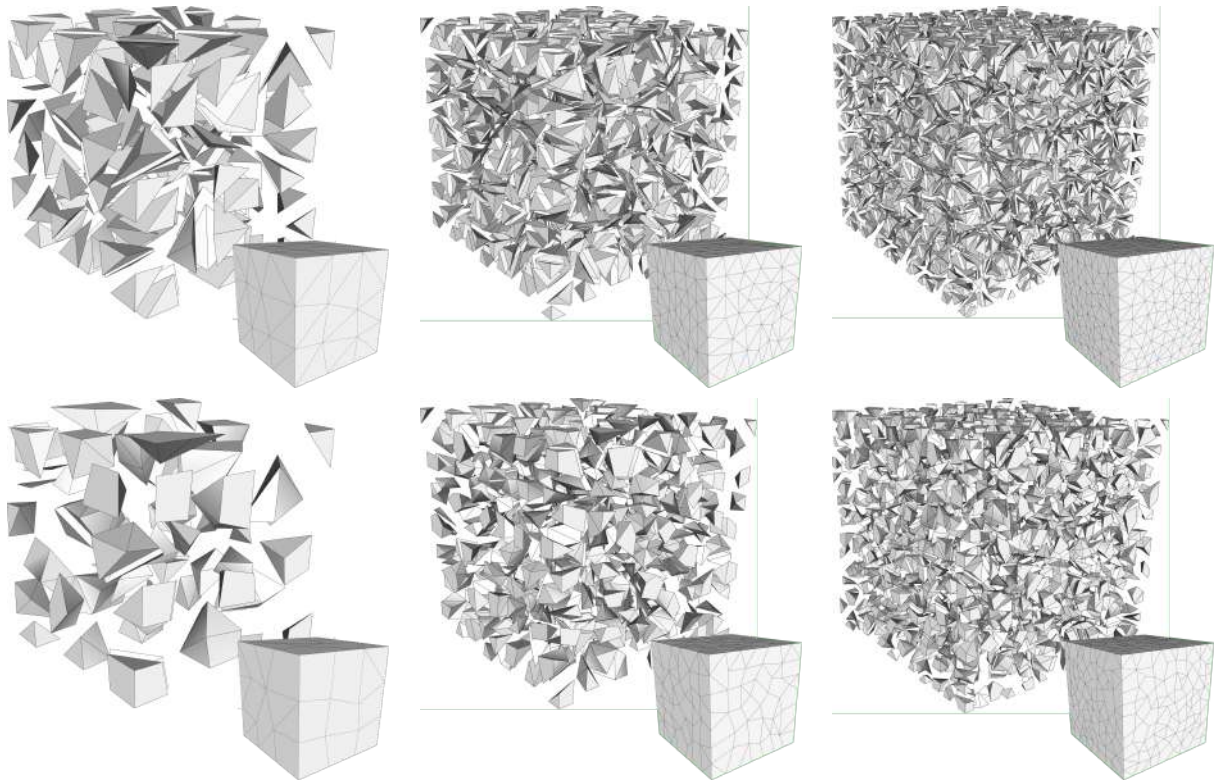


Figure 4.22: Mesh quality indicator ϱ^{2D}, H^1-seminorm, L^2-norm and L^∞-norm of the approximation errors relative to the meshes $\mathcal{MP}_1$, $\mathcal{MP}_2$, $\mathcal{MP}_3$, and $\mathcal{MP}_4$ and their agglomerated verions. Solid, dashed and dotted lines relate to $k = 1, 2, 3$ respectively, and the $x-$axis is inverted.

Test 3 For the three-dimensional case, we generated a dataset similar to $\mathcal{D}_{\text{tet-poisson}}$ from Section 4.1.2. In particular, we adopted the same generation strategy but with lower input parameters t, see Figure 4.23, for generating a sequence of five meshes $\Omega_0, \dots \Omega_4$, with a number of nodes that doubles at every refinement. We apply the agglomeration algorithm with parameter $\lambda = 0.5$, obtaining a dataset $\mathcal{D}^\lambda_{\text{tet-poisson}}$ of five agglomerated meshes. For three-dimensional meshes, the major difference between the original and the agglomerated meshes is in the number of faces and elements, while the total number of nodes does not decrease significantly. In Table 4.5 we report statistics about the number of elements in the original and the agglomerated meshes.

In Figure 4.24 we repeat the VEM analysis presented in Section 4.2.2. The quality of $\mathcal{D}^\lambda_{\text{tet-poisson}}$ is lower than that of $\mathcal{D}_{\text{tet-poisson}}$, in accordance with the previous results. The approximation errors produced by the two datasets in the three norms are very similar, both in terms of convergence rate and error magnitude. In particular, the VEM converges with an



(a) (b) (c)

Figure 4.23: Meshes Ω_0 (a), Ω_2 (a), and Ω_4 (a) from datasets $\mathcal{D}_{\text{tet-poisson}}$ (top), and $\mathcal{D}^{\lambda}_{\text{tet-poisson}}$ (bottom), with $\lambda = 0.5$.

Table 4.5: Summary of the number of elements and computational time T for $\mathcal{D}^{\lambda}_{\text{tet-poisson}}$ and $\mathcal{D}_{\text{tet-poisson}}$ (in brackets).

	#elements	#faces	#edges	#nodes	T
Ω_0	64 (175)	241 (404)	240 (292)	64 (64)	$2.047 \cdot 10^{-2}$ ($2.887 \cdot 10^{-2}$)
Ω_1	240 (651)	888 (1434)	830 (965)	183 (183)	$8.318 \cdot 10^{-2}$ ($1.031 \cdot 10^{-1}$)
Ω_2	568 (1575)	2090 (3384)	1902 (2189)	381 (381)	$1.795 \cdot 10^{-1}$ ($2.492 \cdot 10^{-1}$)
Ω_3	1155 (3143)	4173 (6640)	3703 (4182)	686 (686)	$3.601 \cdot 10^{-1}$ ($5.015 \cdot 10^{-1}$)
Ω_4	2039 (5751)	7536 (12036)	6684 (7472)	1188 (1188)	$6.041 \cdot 10^{-1}$ ($9.547 \cdot 10^{-1}$)

optimal rate over the agglomerated dataset, producing a H^1 error lower than the one relative to the original meshes. In the last column of Table 4.5 we report the computational time T required to assemble and solve the problem for the meshes of the two datasets. Clearly, as the number of elements, faces, edges, and nodes in $\mathcal{D}^{\lambda}_{\text{tet-poisson}}$ is lower than that of $\mathcal{D}_{\text{tet-poisson}}$, the computational time is also lower. This further enforces the above results: after the agglomeration process we obtain meshes with fewer elements, cheaper to handle, and that produce very similar results with respect to the original ones, sometimes even more accurate.

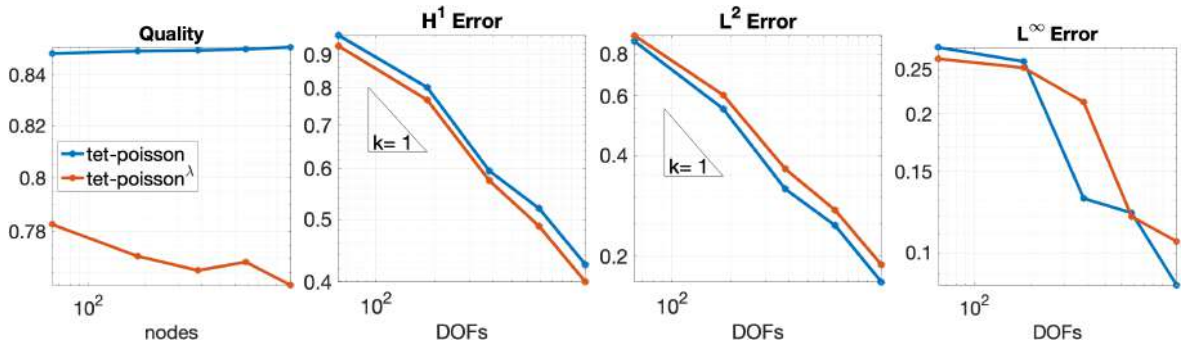


Figure 4.24: Mesh quality indicator $\varrho^{3\text{D}}$, H^1-seminorm, L^2-norm and L^∞-norm of the approximation errors relative to the datasets $\mathcal{D}_{\text{tet-poisson}}$ and $\mathcal{D}^{\lambda}_{\text{tet-poisson}}$.

4.4 Application to Discrete Fracture Networks

This section presents an ongoing work with Drs. F. Vicini, S. Berrone (*Dipartimento di Matematica, Politecnico di Torino*), S. Biasotti, G. Manzini, and M. Spagnuolo (*Istituto di Matematica Applicata e Tecnologie Informatiche 'E. Magenes', CNR*). We test the mesh quality indicator and the quality agglomeration algorithm in the practical application context of Discrete Fracture Networks (DFN) [Adler and Thovert, 1999, Fidelibus et al., 2009]. DFN flow simulations are characterized by complex geometries generated by the use of random probability distributions to create the computational domain in geological applications. They are used in a wide range of applications such as pollutant percolation, gas recovery, aquifers, and reservoir analysis, among others [Decroux and Gosselin, 2013, Panfili and Cominelli, 2014]. The generation of a conforming mesh discretization in real DFNs with classic tools often leads to a large number of degrees of freedom to guarantee conformity on fracture intersections. To tackle this problem, the use of VEM has been proved to be successful in the treatment of tessellations with very generally shaped elements, and therefore highly versatile in the admissible meshes [Benedetto et al., 2016, Berrone et al., 2017].

We investigate the application of the agglomeration algorithm on two different DFNs of increasing complexity. A DFN, or simply *network*, is a collection of planar *domains* (representing fractures), that intersect each other along *interfaces*. Each domain is initially discretized with

a triangular mesh, according to a given mesh size h. Given the initial discretization, the algorithm navigates the network and agglomerates neighboring cells trying to optimize the global quality while respecting conformity constraints along boundaries and interfaces. The parameter λ regulates the entity of the optimization of each domain, and we assume that the value $\lambda = 0$ corresponds to the original non-agglomerated network. The distinct domains that compose the network are discretized and agglomerated independently and in parallel, which significantly speeds up the process. Then, we solve the elliptic problem (3.4) on the original and the agglomerated networks, imposing some boundary conditions and the continuity of the solution across interfaces. We use the VEM formulation adopted in [Berrone et al., 2017] (which is equivalent to the one presented in Section 3.1) for $k = 1, 2, 3$, and compare the performance.

4.4.1 Simulation on a Simple Network

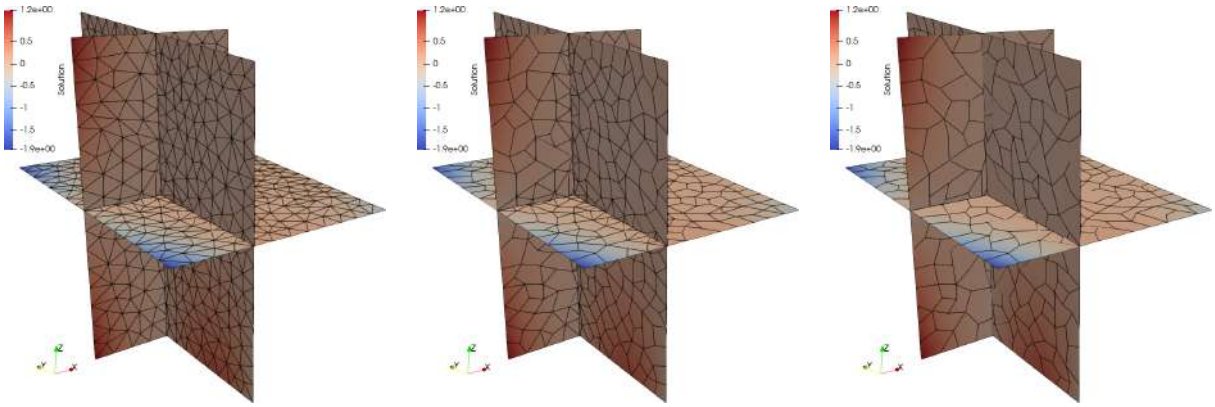
The first example, denoted *Network1*, is a network constituted by three planar domains, parallel to the coordinate axis, meeting along two interfaces. The three domains are meshed with three different mesh sizes, $h_0 = 10^{-1}$, $h_1 = 10^{-2}$, and $h_2 = 10^{-3}$, and then agglomerated with parameters $\lambda_1 = 0.25$ and $\lambda_2 = 1.0$ (the original mesh is conventionally indicated with $\lambda_0 = 0.0$). We note Ω_i^j, for $i, j = 0, 1, 2$, the network with mesh size h_i and parameter λ_j, e.g., Ω_1^2 is the network with mesh size $h_1 = 10^{-2}$, agglomerated with parameter $\lambda_2 = 1.0$.

For a fixed parameter value λ_j, the networks obtained for the three different values of h constitute a dataset $\mathcal{D}^j = \{\Omega_0^j, \Omega_1^j, \Omega_2^j\}$, with decreasing mesh size. We analyze the convergence rate of the VEM and the error magnitude over the three datasets $\mathcal{D}^0, \mathcal{D}^1, \mathcal{D}^2$, similarly to what was done in Section 4.2, to check if the agglomerated networks $\mathcal{D}^1, \mathcal{D}^2$ produce different results from the original $\mathcal{D}^0$. Vice-versa, for a fixed mesh size h_i, we have three networks $\Omega_i^0, \Omega_i^1, \Omega_i^2$ with the same size but increasing agglomeration parameter: see the case Ω_1^j in Figure 4.25 for an example. In this case, we analyze how the number of cells and degrees of freedom varies for the different λ values, and how this variation impacts the numerical error.

We solve the elliptic problem (3.4) on each Ω_i^j, using as ground-truth on the three domains the functions

$$
\begin{aligned}
&f_0(x,y) := -\frac{1}{10}\left(\frac{1}{2}+x\right)\left[x^3 + 8xy\left(x^2+y^2\right)\operatorname{atan2}(y,x)\right], \quad (x,y) \in \Omega_i^j;\\
&f_1(x,z) := -\frac{1}{10}\left(\frac{1}{2}+x\right)x^3 + \pi\frac{4}{5}\left(\frac{1}{2}+x\right)x^3|z|, \quad (x,z) \in \Omega_i^j;\\
&f_2(y,z) := y(y-1)(y+1)z(z-1), \quad (y,z) \in \Omega_i^j,
\end{aligned}
\tag{4.8}
$$

which all have homogeneous Dirichlet boundary conditions. In Figure 4.26 we present the results produced by the VEM over *Network1*: the different colors correspond to the different λ values, while the solid and dashed lines indicate the H^1 and L^2 errors, respectively. In other words, blue lines correspond to dataset $\mathcal{D}^0$, red lines to dataset $\mathcal{D}^1$, and green lines to dataset $\mathcal{D}^2$.



(a) (b) (c)

Figure 4.25: Quality agglomeration algorithm over *Network1* with mesh size $h_1 = 10^{-2}$, colored with respect to the exact solution (4.8). (a) original network Ω_1^0; (b) network Ω_1^1 agglomerated with $\lambda_1 = 0.25$; (c) network Ω_1^2 agglomerated with $\lambda_2 = 1.0$.

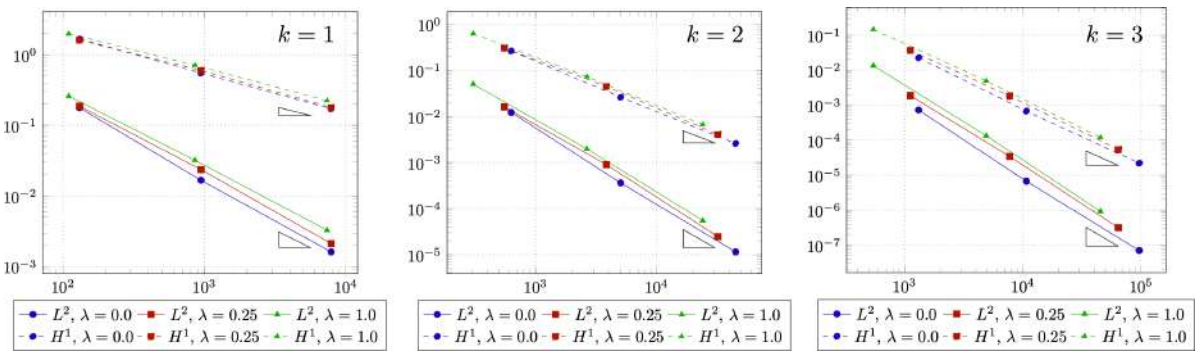


Figure 4.26: Errors in the H^1 and L^2 norms (solid and dashed lines, respectively) relative to *Network1*, for $k = 1, 2, 3$.

We note how the VEM converges with very similar rates and errors on any dataset, for $k = 1, 2, 3$; in particular, the plots in the H^1-seminorm are barely distinguishable. However, as λ increases, the dots indicating the single networks inside each dataset are increasingly shifted towards the left of the plane, and this effect is amplified by higher k values. This effect indicates the significant reduction of the degrees of freedom caused by the quality agglomeration, whose level depends on the agglomeration parameter and whose effect is more appreciable for higher k values.

A quantitative measure of the above observations is deducible from Table 4.6, which summarizes the number of cells, degrees of freedom, and numerical errors in each network. It is split into three sub-tables, concerning the different values of k. We report the total number of cells in the network because, as verified in Section 4.3, the major difference between the original and the agglomerated meshes is in the number of cells, while the number of nodes does not decrease significantly. Note that the number of cells does not depend on k, and it is therefore

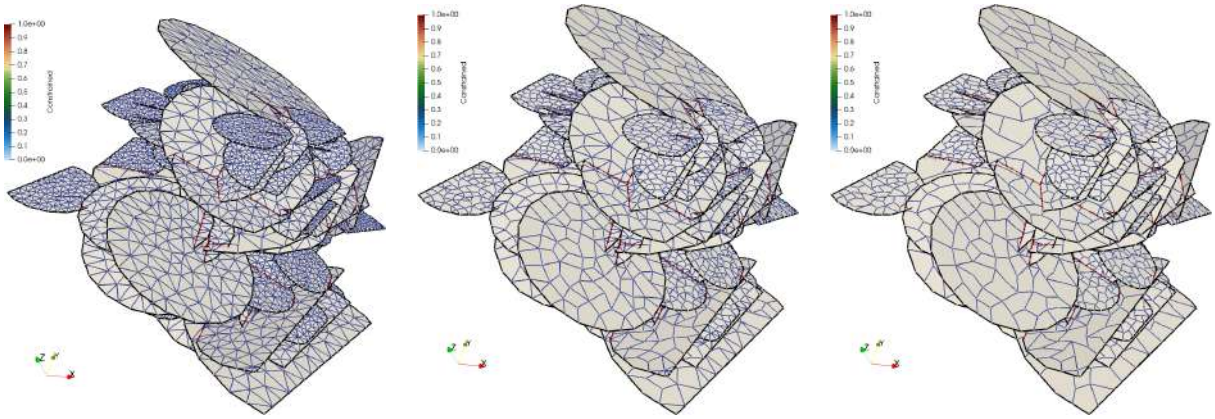
Table 4.6: Results of the quality agglomeration algorithm over *Network1*. We report the number of cells and DOFs, and the L^2 and H^1 norms of the computed solution.

	network	#cells	#DOFs	$\|u - u_h\|_{0,\Omega}$	$\|u - u_h\|_{1,\Omega}$
$k = 1$	Ω_0^0	184	131	1.7758e-01	1.6816e+00
	Ω_0^1	143	131	1.9084e-01	1.6207e+00
	Ω_0^2	52	92	2.8219e-01	1.9004e+00
	Ω_1^0	1,571	947	1.6651e-02	5.4935e-01
	Ω_1^1	972	947	2.3599e-02	6.0055e-01
	Ω_1^2	484	788	3.8770e-02	7.1998e-01
	Ω_2^0	14,548	7,865	1.6124e-03	1.7183e-01
	Ω_2^1	8,112	7,865	2.1136e-03	1.7904e-01
	Ω_2^2	4,687	7,065	3.8983e-03	2.2326e-01
$k = 2$	Ω_0^0	184	629	1.2251e-02	2.6309e-01
	Ω_0^1	143	547	1.6465e-02	3.1200e-01
	Ω_0^2	52	287	4.4063e-02	5.5645e-01
	Ω_1^0	1,571	5,035	3.6072e-04	2.6098e-02
	Ω_1^1	972	3,837	8.9759e-04	4.4846e-02
	Ω_1^2	484	2,543	1.7383e-03	6.8635e-02
	Ω_2^0	14,548	44,825	1.1458e-05	2.5864e-03
	Ω_2^1	8,112	31,953	2.4618e-05	4.0638e-03
	Ω_2^2	4,687	23,503	5.4295e-05	6.6139e-03
$k = 3$	Ω_0^0	184	1,311	7.2477e-04	2.2584e-02
	Ω_0^1	143	1,106	1.8690e-03	3.7121e-02
	Ω_0^2	52	534	1.1745e-02	1.2751e-01
	Ω_1^0	1,571	10,694	6.6740e-06	6.7445e-04
	Ω_1^1	972	7,699	3.4226e-05	1.8136e-03
	Ω_1^2	484	4,782	8.4998e-05	3.6409e-03
	Ω_2^0	14,548	96,333	6.9926e-08	2.1951e-05
	Ω_2^1	8,112	64,153	3.1432e-07	5.3694e-05
	Ω_2^2	4,687	44,628	8.6702e-07	1.1447e-04

identical across the three sub-tables, but it leads to a different number of DOFs, according to the order of the VEM. For $k = 1$, agglomeration with λ_1 does not affect the degrees of freedom: networks Ω_i^0 and Ω_i^1 have the same number of DOFs for every i. This is because agglomerating with a small parameter does not remove mesh nodes, while using λ_2 the effects start to become appreciable. For $k = 2, 3$ the differences are huge: agglomerating the network Ω_2 with λ_2 leads to a reduction of the number of DOFs by a factor of 2.5 (from Ω_2^0 to Ω_2^2).

4.4.2 Simulation on a Complex Network

The second example, denoted *Network2*, contains 86 circular domains, with 100 triangular cells each, that meet along 159 interfaces. The circular domains are agglomerated with parameters $\lambda_1 = 0.25$ and $\lambda_2 = 1.0$, obtaining three networks Ω^0, Ω^1, and Ω^2, presented in Figure 4.27.



(a) (b) (c)

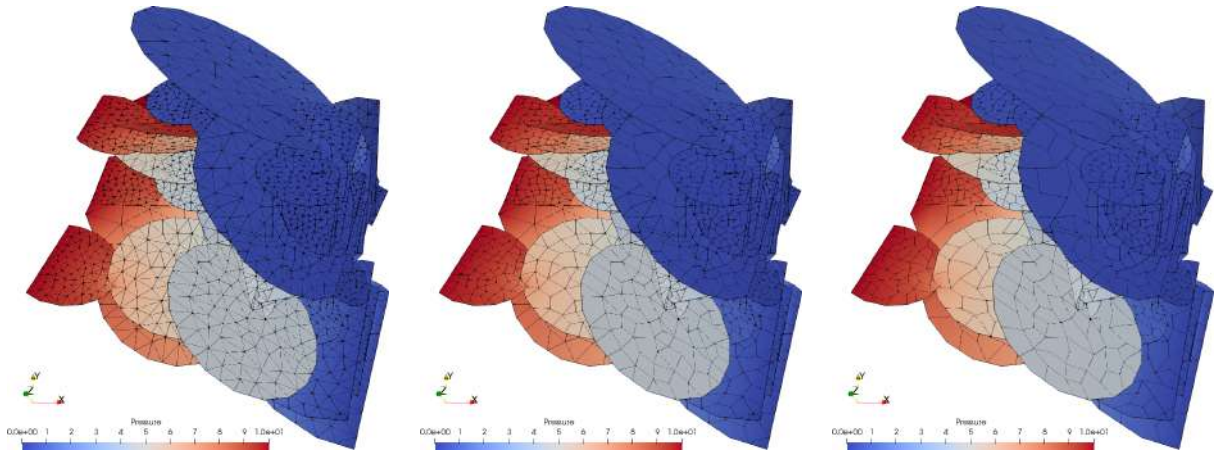
Figure 4.27: Quality agglomeration algorithm over *Network2*: (a) original network Ω^0; (b) network Ω^1 agglomerated with $\lambda_1 = 0.25$; (c) network Ω^2 agglomerated with $\lambda_2 = 1.0$.

We solve the elliptic problem (3.4), imposing Dirichlet boundary conditions on the left and the right side of the 3D space in which the network is embedded. In particular, we set a value of 10 for the boundary points on the plane $x = 0$, representing a pressure applied to the volume, and a value of 0 on the boundary points on the plane $x = 1$, see Figure 4.28. On the boundaries of the domains which do not intersect such two planes, we set homogeneous Neumann conditions. In this case, we do not have an exact solution, hence we compare the H^1-seminorm and the L^2-norm of the computed solution u_h, and check how much they differ from each other for the different λ values.

Statistics on the number of cells and DOFs for each domain are reported in Table 4.7, together with the H^1 and L^2 norms of the computed solution. As for *Network1*, we notice that the number of cells significantly decreases for increasing values of λ, leading to a small reduction of the degrees of freedom for $k = 1$ and a huge drop (up to 50%) for $k = 3$. The solutions computed over the different networks, however, differ for a very small factor: around 0.01% in the H^1-seminorm and 1% in the L^2-norm.

4.4.3 Discussion

The obtained results are encouraging: through the agglomeration algorithm, we are able to reduce the number of degrees of freedom of a network up to 50% (in particular, when working with high order formulations of the method), while preserving optimal convergence rate and at a very small cost in terms of approximation error. This translates into a significant gain in



(a) (b) (c)

Figure 4.28: Computed solution over *Network2*: (a) original network Ω^0; (b) network Ω^1 agglomerated with $\lambda_1 = 0.25$; (c) network Ω^2 agglomerated with $\lambda_2 = 1.0$.

Table 4.7: Results of the quality agglomeration algorithm over *Network2*. We report the number of cells and DOFs, and the L^2 and H^1 norms of the computed solution.

	network	#cells	#DOFs	$\|u_h\|_{0,\Omega}$	$\|u_h\|_{1,\Omega}$
$k = 1$	Ω^0	16,976	11,852	2.3335e+04	1.8707e+01
	Ω^1	11,959	11,808	2.3333e+04	1.8632e+01
	Ω^2	6,007	10,134	2.3321e+04	1.8447e+01
$k = 2$	Ω^0	16,976	57,698	2.3331e+04	1.8493e+01
	Ω^1	11,959	47,576	2.3331e+04	1.8481e+01
	Ω^2	6,007	32,323	2.3326e+04	1.8431e+01
$k = 3$	Ω^0	16,976	120,520	2.3332e+04	1.8492e+01
	Ω^1	11,959	95,303	2.3333e+04	1.8487e+01
	Ω^2	6,007	60,519	2.3331e+04	1.8463e+01

terms of computational cost. In particular, this process is strongly effective in the computation of several solutions over the same network with a parameter changing at every iteration, e.g., time-dependent problems or model reduction problems.

Related Publications

- T. Sorgente, D. Prada, D. Cabiddu, S. Biasotti, G. Patané, M. Pennacchio, S. Bertoluzza, M. Manzini, and M. Spagnuolo. VEM and the mesh. In *SEMA SIMAI Springer Series*, vol. 31(1), pages 1–54, Springer, 2021.

- T. Sorgente, S. Biasotti, M. Manzini, and M. Spagnuolo. The role of mesh quality and mesh quality indicators in the virtual element method. In *Advances in Computational*

Mathematics, vol. 48(1) pages 1–34, 2022.

- T. Sorgente, S. Biasotti, M. Manzini, and M. Spagnuolo. Polyhedral mesh quality indicator. In *Computers and Mathematics with Applications*, vol. 114, pages 151-160, Pergamon, 2022.

- T. Sorgente, F. Vicini, S. Berrone, S. Biasotti, G. Manzini, and M. Spagnuolo. Quality-based agglomeration of discrete fracture networks for the VEM. *Ongoing work.*

Chapter 5

Conclusions

In this work, we analyzed the concept of mesh quality across the existing literature. We realized that there is some general confusion around this topic, and that, despite many efforts in the last decades, a comprehensive understanding of how and how much the accuracy of a numerical solution depends on the mesh is still lacking. Two major strategies are currently adopted for generating meshes that match a target level of accuracy: (i) to drastically decrease the mesh size (increase the number of nodes) and generate a huge number of tiny identical cells, taking advantage of the increasing computational power, or (ii) to analyze the domain features and generate a small number of cells with higher quality. We decided to deepen the second approach, restricting our focus to generic polytopal meshes defined for numerical simulations and, in particular, for the Virtual Element Method.

We defined a mesh quality indicator, i.e., a mathematical instrument to investigate the local and global quality of a polytopal mesh, directly deduced by the theoretical results on the convergence of the VEM. We tested the indicator through a series of different experiments, that showed a significant level of accuracy and reliability. It was shown to be capable of identifying pathological meshes before the computation of the VEM solution and classifying groups of meshes according to the quality of the approximation that they will produce. Moreover, when applied to sequences of mesh refinements, the indicator was able to make accurate predictions on the convergence rate of the VEM. It is therefore a practical tool to rapidly determine the behavior of the VEM over a mesh before actually computing the solution.

Based on the mesh quality indicator, we then built a quality agglomeration algorithm, that modifies an existing mesh in order to maximize its quality. The efficacy of the agglomeration algorithm has been tested in several scenarios, including the practical problem of solving a PDE over a discrete fracture network (DFN). After the agglomeration process, we obtained

meshes with a significantly smaller number of elements, faces, edges, and nodes with respect to the input mesh, that are therefore computationally cheaper to handle. At the same time, the performance of the VEM over the agglomerated meshes was comparable, and sometimes even better, than over the input ones.

The routines developed for the implementation of the above concepts have been presented and made publicly available, together with all the meshes built for the testing phase. In particular, we defined an algorithm for computing the *kernel* of a polyhedron, as this information was needed by the quality indicator. Our algorithm is optimized for the efficient computation of the kernel of small polyhedra, and, in this, outperforms the other methods available in the literature.

5.1 Future Work

The analysis of the concept of mesh quality related to the VEM has been satisfactorily carried out, as proven by the results obtained by the mesh quality indicators. We observe that the theoretical analysis for the three-dimensional formulation of the VEM is still unripe, and we hope this work will encourage research on this aspect. An extension of $\mathbf{G4^{2D}}$ to volumetric elements, or indications on the relative importance of the quality of the faces with respect to the quality of the interior, remain open issues. A further step forward could be the inclusion into the indicator of information on the nature of the physical problem that we want to solve. The current indicator is problem-independent, but it is intrinsically defined for isotropic problems because the contribution given by $\varrho_2^{2\mathrm{D}}$ and $\varrho_2^{3\mathrm{D}}$ penalizes anisotropic configurations. An extension to anisotropic problems would therefore lead to problem-specific indicators, with a consequent loss of generality.

Regarding the quality agglomeration algorithm, experimental results suggest that the error in the H^1 norm benefits more than the others from the agglomeration process. This seems an interesting observation, that deserves more accurate investigations. A possible improvement of the algorithm could be the integration into the process of an optimization step that modifies the position of the mesh nodes after the agglomeration. This would allow for alignment and collapsing of multiple edges shared by two adjacent cells. However, the main limitation is the algorithm computational cost. Despite the use of graph-cut and our efficient method for the kernel computation, the analysis of dense meshes (especially in 3D) may take significant time because we need to explore a high number of possible combinations. In such cases, it could be useful to subdivide the domain into smaller regions, in order to run the algorithm in parallel, e.g., as done in Section 4.4 for Discrete Fracture Networks examples. However, the proposed framework is not graph-cut-dependent, and alternative methods for the optimization of the energy induced by the quality indicator could be considered.

It would be interesting to adapt the quality indicator, and consequently, the quality

agglomeration, to other numerical schemes different from the VEM, such as the Discontinuous Galerkin method [Cockburn et al., 2012] or schemes defined over elements with curvilinear edges [Beirão da Veiga et al., 2020, Anand et al., 2020], by opportunely defining a new set of scalar functions based on appropriate geometrical assumptions.

As already noted in the Introduction 1, the agglomeration process cannot be properly considered a mesh generation method. The most challenging problem remains the definition of a method for generating quality polytopal meshes from scratch, and this will surely be a major focus of our future research. Meanwhile, we plan to employ the quality indicator in a new quality refinement strategy, with a similar philosophy to the one adopted for the quality agglomeration. The integration of the refinement and the agglomeration methods will result in a pipeline really capable of generating adaptive meshes. The first step in this direction will be the comparative analysis of the results produced by current refinement schemes, from the point of view of the mesh quality indicator.

Last, we plan to continue sharing our routines on GitHub, so as to create a proper library over time. We will also collect all the meshes generated in these years into a wide database of meshes, to be used by practitioners as a reference to test polytopal numerical schemes and other quality measures.

APPENDIX A

Algorithms for the Computation of the Kernel of a Polyhedron

In this section, we illustrate our method for computing the kernel of a polyhedron with a geometric approach, announced in Section 3.4.1. A first version of the method was published in [Sorgente et al., 2021a], while the algorithms here reported have been described in [Sorgente et al., 2022c]. It has a modular structure composed of four nested algorithms, each one calling the next one in its core part (see Figure A.1). It is modular in the sense that each algorithm can be entirely replaced by another one performing the same operation(s). This property is particularly useful for making comparisons: one could, for instance, use different strategies for computing the intersection between a polygon and a plane and simply replace Algorithm 3, measuring the efficiency from time to time.

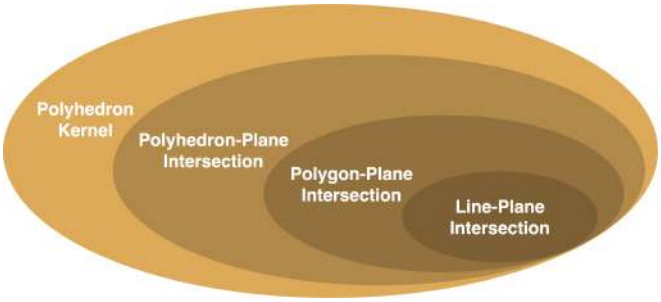


Figure A.1: A visual scheme of the whole algorithm.

A.1 Data Structure

We adopt the following data structure inherited by the *cinolib* library [Livesu, 2019], in which the code has been written:

- `Points`: Array of unordered 3D points.
- `Face`: Array of unsigned integers associated to a `Points` array, representing the indices of the vertices of a face ordered counter-clockwise.
- `Polyhedron`: Struct composed by a field `verts` of type `Points` containing the vertices and a field `faces` of type `array`< `Face` > containing the faces of a polyhedron.
- `Plane`: Class defining a plane in the Hessian form, composed by a 3D point n indicating the unit normal of the plane (i.e., the a, b, c coefficients of the plane equation) and a number d indicating the distance of the plane from the origin (i.e., the d coefficient of the plane equation). The plane class also contains three additional points p_1, p_2, p_3 lying on the plane, useful for the Shewchuck predicates [Shewchuk, 1997].
- `Sign`: Array of labels (BELOW, ABOVE, or INTER) used to store information on the position of the elements of a polyhedron (points, edges, or faces) with respect to an unspecified plane.

Given a plane p, the elements of a polyhedron are classified as follows:

- A point v is labelled as BELOW, ABOVE or INTER provided that the function *orient3d-fast*$(p.p_1, p.p_2, p.p_3, v)$ in the Shewchuck predicates library [Shewchuk, 1997] is negative, positive or zero, within a tolerance of 10^{-8};
- An edge e is labeled as BELOW (resp. ABOVE) if both its endpoints are BELOW or INTER (resp. ABOVE or INTER), and as INTER if one point is ABOVE and the other one is BELOW;
- A face f is labeled as BELOW if all of its points are BELOW, as ABOVE if none of its points is BELOW, and as INTER otherwise.

The classification of points is computed at the top level of the algorithm with *orient3d-fast*. For edges and faces instead, we define a function *classify(S)* which determines the classification of an edge or a face from an array S of type `Sign` containing the classification of its points.

A.2 Polyhedron Kernel

With Algorithm 1 we tackle the general problem: given a polyhedron P, we want to find the polyhedron K representing its kernel. In addition to P, we also need as input an array containing the outwards normals of its faces, as it is not always possible to determine the orientation of a face only from its vertices (for example, with non-convex faces). We require the face normals explicitly, and not simply a boolean indicating the faces orientation because these points will be used to define the planes containing the faces of P.

Algorithm 1 Polyhedron Kernel

Input: `Polyhedron` P, `Points` N (faces normals);
Output: `Polyhedron` K

```
K := AABB of P;
[optional] shuffle P.faces;
for Face f in P.faces do
    Plane p := plane containing f with normal −N(f);
    Sign S := orient3d-fast(p.p1, p.p2, p.p3, K.verts);
    K := Polyhedron-Plane Intersection(K, S, p);
    if size(K.faces) < 3 then return NULL;
    end if
end for
return K;
```

We initialize K with the axis-aligned bounding box (AABB) of P, i.e., the box with the smallest volume within which all the vertices of P lie, aligned with the axes of the coordinate system. Then we recursively "slice" K with a number of planes, generating a sequence of convex polyhedra K_i, $i = 1, \ldots, \#P.\texttt{faces}$, such that $K_i \subseteq K_{i-1}$. For each face f of P, we define the plane p containing its vertices and with normal vector given by the opposite of the face normal $N(f)$, that is to say, $p.n := -N(f)$. We consider the plane together with the direction indicated by $p.n$, which is equivalent to considering the half-space originating in p and containing its normal vector. Given this plane, in a `Sign` array S we store the classification of all the points in $K.\texttt{verts}$ according to their position with respect to p. For improving the efficiency, we can set the sign of all points belonging to f to INTER without evaluating their position. In general, p will separate K into two polyhedra, and between those two we keep the one containing the vector $p.n$, which therefore points towards the interior of the element. This operation is performed by Algorithm 2 (*Polyhedron-Plane Intersection*), which replaces K with the new polyhedron. The whole pipeline is illustrated in Figure A.2.

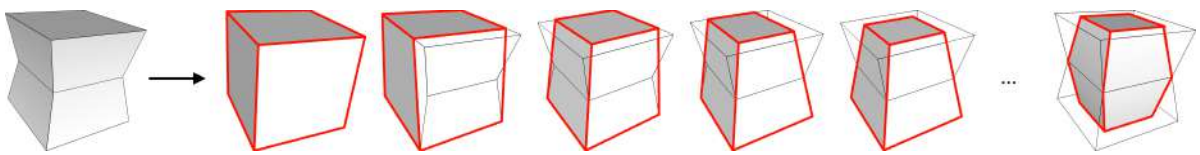


Figure A.2: Pipeline of the kernel computation for a polyhedron. At first step, we compute the Axis Aligned Bounding Box (AABB) of the polyhedron; then, we iterate on each face f of the polyhedron (black edges) and cut AABB with the plane induced by f (red edges).

The order in which we consider the faces is not relevant from a theoretical point of view but turns out to have a huge impact on the performance. For instance, if we imagine computing the kernel of a non-simply connected object, which is obviously empty, visiting the faces in the order they are stored may take a very long time, especially if the tessellation of the object is fine.

This is because, generally, the faces of a tessellated model are numbered somehow coherently with their neighbors. For this reason, we optionally propose to visit the faces in random order, or to *shuffle* P.`faces`, and to return an empty polyhedron if, after a "slice", K has less than three faces. In this way, the empty kernel of a non-simply connected object with thousands of faces could be detected in just three or four iterations. When this command is turned on, we say the algorithm is run in *shuffle* mode.

We point out that cutting a convex polyhedron with a plane will always generate two convex polyhedra, and since we start from the bounding box (which is convex), we are guaranteed that K will always be a convex polyhedron. No matter how weird the initial element P is, from this point on we will only be dealing with convex polyhedra and convex faces. Last, we could as well start by considering the polyhedron's convex hull instead, but it would be less efficient because the convex hull costs in general $O(n \log n)$ while the AABB is only $O(n)$, and we would still need to intersect the polyhedron with each of its faces.

A.3 Polyhedron-Plane Intersection

With Algorithm 2, we intersect a polyhedron P with a plane p, given in S the position of the vertices of P with respect to p. The intersection will in general determine two polyhedra, and between these two we are interested in the one containing the normal vector of p (conventionally called the one "above" the plane and indicated with A). This algorithm is inspired from [Ahn and Shashkov, 2008], where the authors define an algorithm for the intersection of a convex polyhedron with a half-space.

The first part of Algorithm 2 is called the "clipping" part (recalling the terminology from [Ahn and Shashkov, 2008]) and consists in clipping each face of P with the plane p, see Figure A.3(a). It corresponds to the *for* loop: we iterate on P.`faces`, each time extracting from S the labels f_s of the vertices f_v of the current face and using the *classify* function. Faces classified as BELOW are discarded, ABOVE faces are added to A together with their vertices, and INTER faces are split by Algorithm 3 (*Polygon-Plane Intersection*). While we visit every face only once, the same does not hold for vertices, therefore we check if a vertex is already in A.`verts` before adding it.

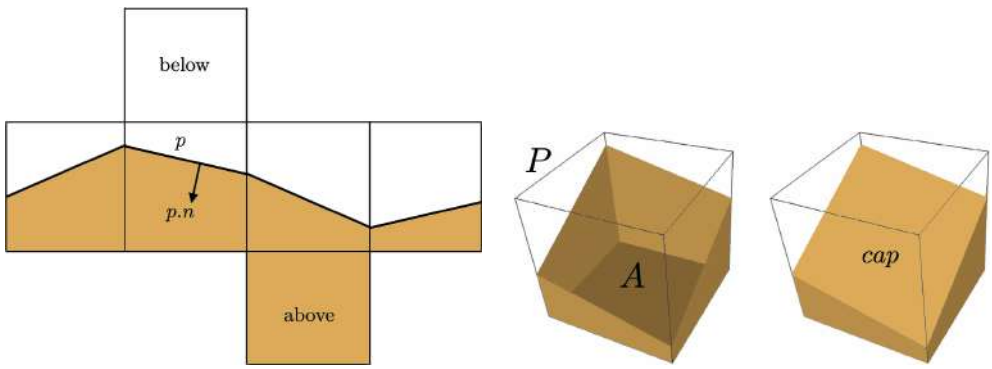
This simple idea of checking in advance the faces classification resolves several implementation issues, and in some cases significantly improves the efficiency of the algorithm. By doing this, we make sure that only the faces properly intersected by the plane are passed to Algorithm 3, so that we do not need to implement all the particular cases of intersections in a single point, or along an edge, or of faces contained in the plane. In addition, for every face not passed to Algorithm 3 we have an efficiency improvement, and this happens frequently in models with many co-planar faces.

If at the end of this step, A contains at least three INTER points, given that A and all its

Algorithm 2 Polyhedron-Plane Intersection

Input: `Polyhedron` P, `Sign` S, `Plane` p
Output: `Polyhedron` A

```
for Face f in P.faces do
    Points f_v := vertices in P.verts relative to f;
    Sign f_s := S|_{f_v}
    switch classify(f_s) do
        case BELOW break;
        case ABOVE
            A.verts ← f_v, A.faces ← f;
        case INTER
            (V,F):=Polygon-Plane Intersection(f_v, f, f_s, p);
            A.verts ← V, A.faces ← F;
end for
Points capV := vertices in A.verts which are INTER;
if size(capV) < 3 then return A;
end if
Face capF := indices of the capV vertices ordered CCW;
if capF ∉ A.faces then A.faces ← capF;
end if
return A;
```



(a) (b)

Figure A.3: Intersection of a polyhedron with a plane: (a) clipping and (b) capping of a cube.

faces are convex, these vertices will define a "cap" face of A completely contained in p, see Figure A.3(b). We can optimize the algorithm by storing in a `Sign` array the classification of the vertices in A, updating it with the sign of every vertex added in the switch loop. Note that in this case, we do not need to use *orient3d-fast*: we already know the sign of the old vertices and the new vertices will obviously be of type INTER. In our data structure, the vertices of the faces are ordered counter-clockwise (CCW): to sort the points contained in $capV$ we project them onto a plane, drop one coordinate, and apply the algorithm proposed in [Baeldung, 2021] for 2D points. Note that if the cap face was not convex it would make no sense to order its vertices, but the intersection between a plane and a convex polyhedron will always generate convex faces. Last, we need to check that this new face is not already present in A: for example, if p was tangent to P along a face, this face could be added to A both as an ABOVE face and as a cap face. If this is not the case, we add $capF$ to A.`faces` but we do not need to add any vertex from $capV$, as we can assume they are all already present in A.`verts`.

A.4 Polygon-Plane Intersection

Algorithm 3 describes the intersection of a polygon (representing a face of the polyhedron), defined by an array of 3D points $polyV$ and an array of indices $polyF$, with a plane p. As before, we also require as input an array `polys` containing the position of the vertices of $polyV$ with respect to p. In analogy to Algorithm 2, the intersection will in general determine two polygons and we are only interested in the one above the plane, see Figure A.4(a), defined by points $aboveV$ and indexes $aboveF$. We generically say that a vertex v is added to *above* meaning that v is added to $aboveV$ and its index id_v is added to $aboveF$.

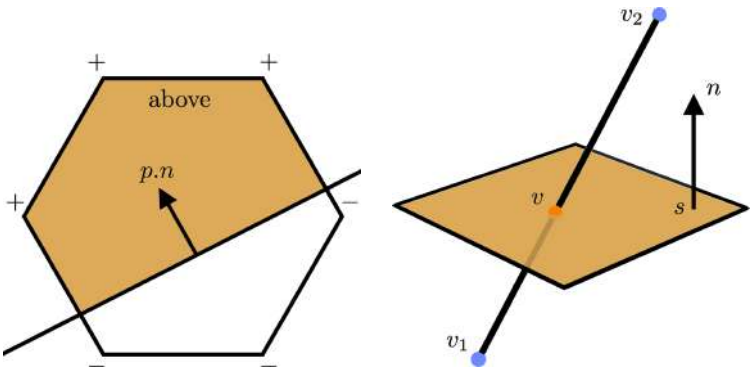
This time we iterate on the edges of $polyF$, extract the signs s_1, s_2 of the edge endpoints, and switch between the three possible classifications of the edge. In order to avoid duplicates, for each couple of consecutive points v_1, v_2, we only accept to add to *above* the second point v_2 or the intersection point v, but never v_1. We are here taking advantage of the fact that all faces are oriented coherently.

If the edge is of type BELOW, we ignore it unless v_2 is INTER (i.e. it lies exactly on the plane), in which case we add it to *above*. In case of ABOVE edges, we add v_2 to *above*. For edges of type INTER, we perform Algorithm 4 (*Line-Plane Intersection*) and find a new point v. Its index id_v will be equal to the maximum value in $polyF$ plus one, just to make sure that we are not using the index of an existing point. We always add v to *above*, and if v_1 is BELOW we also add v_2. As already noted in the previous section, treating separately the weak intersections (the BELOW and ABOVE cases) makes the code simpler and more efficient.

Algorithm 3 Polygon-Plane Intersection

Input: `Points` $polyV$, `Face` $polyF$, `Sign` `polys`, `Plane` p.
Output: `Points` $aboveV$, `Face` $aboveF$.

```
for i = 1 : size(polyF) do
    id_1 := polyF(i), id_2 := polyF(i + 1);
    v_1 := polyV(id_1), v_2 := polyV(id_2);
    s_1 := polys(id_1), s_2 := polys(id_2);
    switch classify(s_1, s_2) do
        case BELOW
            if v_2 is INTER then
                aboveV ← v_2, aboveF ← id_2;
            end if
        case ABOVE
            aboveV ← v_2, aboveF ← id_2;
        case INTER
            v := Line-Plane Intersection(v_1, v_2, p);
            id_v := max(polyF)+1;
            aboveV ← v, aboveF ← id_v;
            if v_1 is BELOW then
                aboveV ← v_2, aboveF ← id_2;
            end if
end for
return aboveV, aboveF;
```



(a) (b)

Figure A.4: (a) Intersection between a polygon and a plane, with the above part colored in green. (b) Intersection between a line and a plane.

A.5 Line-Plane Intersection

This last algorithm computes the intersection point between a line, given as a couple of vertices, and a plane. We use a very simple and well-known procedure, and we report it here only for completeness.

Algorithm 4 Line-Plane Intersection

Input: vertices v_1, v_2, `Plane` p.
Output: vertex v.

1: $N := (p.n) \cdot (v_1 - p.p_1)$;
2: $D := (p.n) \cdot (v_2 - v_1)$;
3: `assert`$(D! = 0)$
4: $t := -N/D$;
5: **return** $v := v_1 + t\ (v_2 - v_1)$;

The intersection vertex v is defined as the linear combination of vertices v_1 and v_2, with a coefficient t which may also fall outside the standard range $[0, 1]$. The coefficient t is found as the negative ratio between two scalar products involving the plane normal n and another generic point on the plane s, other than v_1 and v_2 see Fig A.4(b). The normal is $p.n$, and we can use one of the three points on the plane $p.p_1$, $p.p_2$, or $p.p_3$ as s. If the denominator D vanishes, it means either that the line is contained in the plane (if $N = 0$ as well), or that the line does not intersect the plane. We treat these exceptions as errors because in Algorithm 3 we only call this algorithm after checking that the edge (v_1, v_2) properly intersects the plane p.

A.6 Tests and discussions

We test our method in different settings, comparing its performance to the results obtained by our implementation of the algebraic method in CGAL. Experiments have been performed on a MacBook Pro equipped with a 2,3 GHz Intel Core i5 processor with four CPUs and 16GB of RAM. Source code is written in C++ and it is accessible at `https://github.com/TommasoSorgente/polyhedron_kernel`, together with all datasets.

In the following subsections, we will present some plots and tables: we point out here some notation remarks. Regarding plots, we color the CGAL computational time in blue and ours in red, both on a logarithmic scale. On the $x-$axis, depending on the context, we have the number of elements in the mesh or the number of vertices of the single model. In the several tables presented, we first report the number of elements or vertices of the mesh. Since all the considered objects have genus zero, and their surface is purely triangular, Euler's formula states that the number of faces is approximately equal to twice the number of vertices. Therefore, we only indicate the number of vertices, but the number of faces is easily computable. Then the computational times (in seconds) are shown, and the ratio between CGAL time and ours. Note that ratios are computed from the original time values, while in the tables we indicate

truncated times; therefore they do not exactly correspond to the division between the values in the previous columns.

A.6.1 Polyhedral meshes

We first test our algorithm in the setting it was developed for, i.e., the computation of the kernels of elements in a 3D tessellation. To do so, we use datasets $\mathcal{D}_{\text{poly-parallel}}$, $\mathcal{D}_{\text{poly-poisson}}$, and $\mathcal{D}_{\text{poly-random}}$ from Section 4.1.2. The meshes contained in these datasets are typical examples of tessellations that can be found in numerical analysis for the approximation of a PDE. Each of them contains five tessellations of the unit cube with decreasing mesh size, from 100 to 100K vertices. The resulting meshes contain between 100 and 600K elements, most of which are tetrahedra, and 20% of them are generic polyhedra (in blue in Figure A.6) obtained by the union of two tetrahedra. Non-tetrahedral elements are generated by the agglomeration of two tetrahedral elements, therefore they may also be non-convex, see Figure A.5.

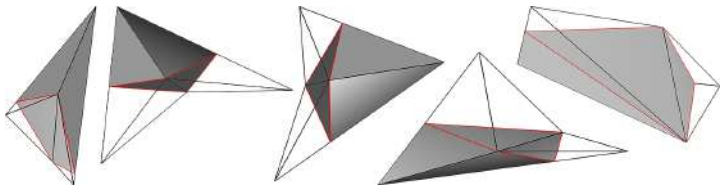


Figure A.5: Examples of non-convex elements found in polyhedral meshes from $\mathcal{D}_{\text{poly-parallel}}$, $\mathcal{D}_{\text{poly-poisson}}$, and $\mathcal{D}_{\text{poly-random}}$, and relative kernels.

In Table A.1 we report, for each mesh of each dataset, the number of elements, the computational times for both methods, and the ratio between the CGAL time and ours. Moreover, at the bottom of Figure A.6, we plot the times against the number of elements in the mesh. It is visible how both methods scale linearly with respect to the number of elements since the kernel of the elements is computed separately and independently for each element. Our method performs 8 to 11 times faster than CGAL, which approximately means one order of magnitude. As the elements in these meshes have either 4 faces, if they are tetrahedra or 6 faces, if they are the union of two tetrahedra, computing their kernel in a geometrical way results much faster than solving a linear problem. In this case, we did not use the *shuffle* mode, as the number of faces was so small that the visiting order resulted not relevant.

A.6.2 Refinements

As the second setting for our tests, instead of increasing the number of elements, we measure the asymptotic behavior of the method as the number of faces and vertices of a single element explodes. We selected two polyhedra from the dataset *Thingi10K* [Zhou and Jacobson, 2016]: the so-called *spiral* (ThingiID: 60246) and *vase* (ThingiID: 85580). These models are given in the form of a surface triangular mesh, but we treat them as single volumetric cells, analyzing the

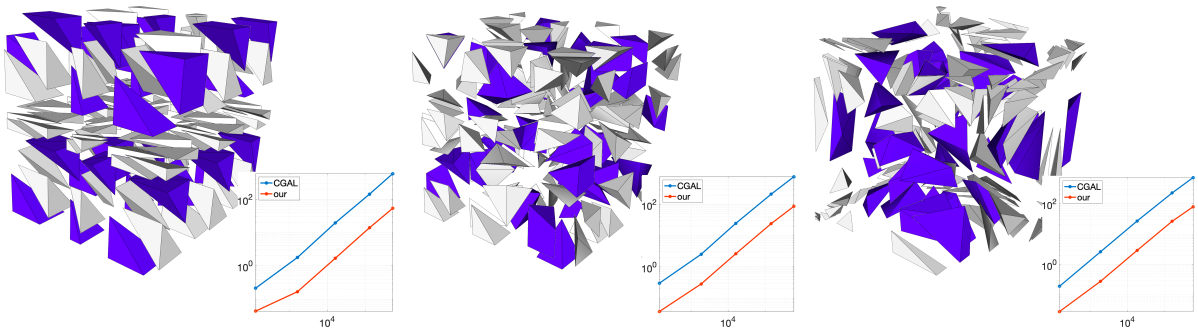


Figure A.6: Polyhedral meshes and time plots from datasets $\mathcal{D}_{\text{poly-parallel}}$, $\mathcal{D}_{\text{poly-poisson}}$, and $\mathcal{D}_{\text{poly-random}}$, with non-tetrahedral elements highlighted in blue.

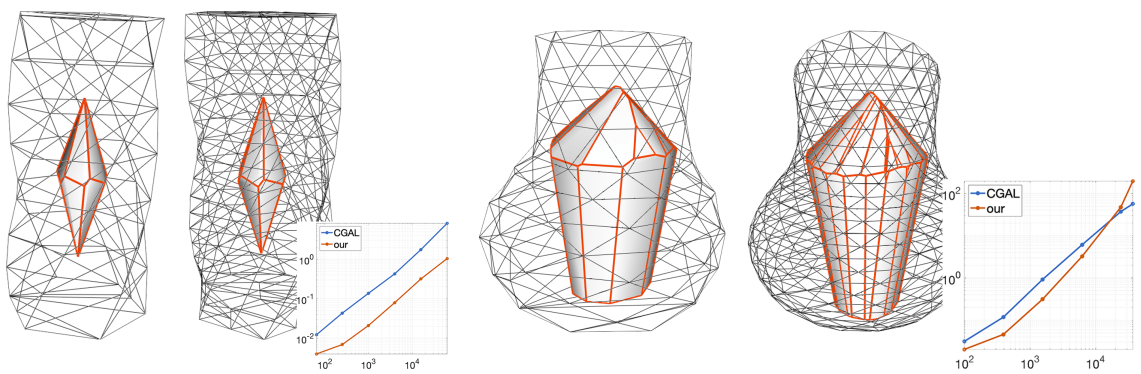
Table A.1: Computational times for polyhedral meshes.

dataset	mesh	#elements	our	CGAL	ratio
$\mathcal{D}_{\text{poly-parallel}}$	Ω_0	130	0.04	0.21	4.89
	Ω_1	1647	0.17	1.79	10.69
	Ω_2	16200	1.68	19.4	11.55
	Ω_3	129600	13.94	142.36	10.21
	Ω_4	530842	53.47	588.43	11
$\mathcal{D}_{\text{poly-poisson}}$	Ω_0	140	0.04	0.3	8.05
	Ω_1	1876	0.29	2.54	8.91
	Ω_2	16188	2.64	24.79	9.38
	Ω_3	146283	24.24	212.23	8.75
	Ω_4	601393	86.77	770.66	8.88
$\mathcal{D}_{\text{poly-random}}$	Ω_0	147	0.03	0.19	6.8
	Ω_1	1883	0.28	2.62	9.41
	Ω_2	18289	2.91	27.11	9.33
	Ω_3	161512	26.51	228.08	8.6
	Ω_4	598699	80.15	735.55	9.18

performance of both algorithms as we refine them. In Table A.2 we report the computational times and the ratio for each refinement.

The *spiral* model is refined through a midpoint strategy: each face is subdivided by connecting its barycenter to its other vertices. As a consequence, the planes induced by its faces remain the same and the kernels of the refined models are all equal (Figure A.7). In this example, our method performs on average 5.77 times better than the algebraic method (see Table A.2), and the computational time scales with a constant rate (see the plot in Figure A.7). Our implementation takes advantage of the fact that Algorithm 2 recognizes the several co-planar faces and always performs Algorithm 3 the same number of times, independently of the number of faces.

The *vase* model is more complex, as it presents a curved surface that generates a lot of different planes defining the kernel. Moreover, we refined this model using Loop's algorithm, and this generated faces lying on completely new planes. This explains the difference between



(a) (b)

Figure A.7: (a) Original *spiral* model and its first refinement, with identical kernels. (b) Original *vase* model and its first refinement: small perturbations in the faces lead to slightly different kernels.

the two kernels in Figure A.7: the general shape is similar, but the more faces we add to our model the more faces we find on the resulting kernel. Our geometric method improves the performance of the algebraic one by a factor of around 2 in the first refinements, but in the last two meshes the complexity increases drastically and CGAL results faster (see Table. A.2). Even the *shuffle* mode did not particularly improve the performance, being the object star-shaped.

Table A.2: Computational times for the *spiral* and *vase* refinements.

dataset	mesh	#vertices	our	CGAL	ratio
spiral	Ω_0	64	0.004	0.01	3.07
	Ω_1	250	0.007	0.04	6.27
	Ω_2	994	0.02	0.14	6.56
	Ω_3	3970	0.08	0.43	5.38
	Ω_4	15874	0.32	1.77	5.54
	Ω_5	63490	1.05	8.24	7.86
vase	Ω_0	99	0.02	0.03	1.55
	Ω_1	390	0.04	0.12	2.56
	Ω_2	1554	0.31	0.92	2.94
	Ω_3	6210	3.24	6.1	1.88
	Ω_4	24261	47.49	37.24	0.78
	Ω_5	36988	196.7	56.75	0.29

A.6.3 Complex models

Last, we try to compute the kernel of some more complex models, taken again from the dataset *Thingi10K* and treated as single volumetric cells. Even if our method is designed for dealing with polyhedra of relatively small size, we already saw in Section A.6.2 how our algorithm is still able to compute the kernel of objects with thousands of vertices and faces. We filtered the *Thingi10K* dataset, selecting only "meaningful" models: objects with one connected component,

genus zero, Euler characteristic greater than zero, closed, not degenerate, and of size smaller than 1MB. Note that, even applying these filters, the majority of the models are not star-shaped, i.e., with empty kernel. We discarded a few models for computational and stability reasons; for instance because one of the two algorithms failed to process them. The final collection, which we will call the *Thingi dataset*, contains exactly 1806 distinct volumetric models.

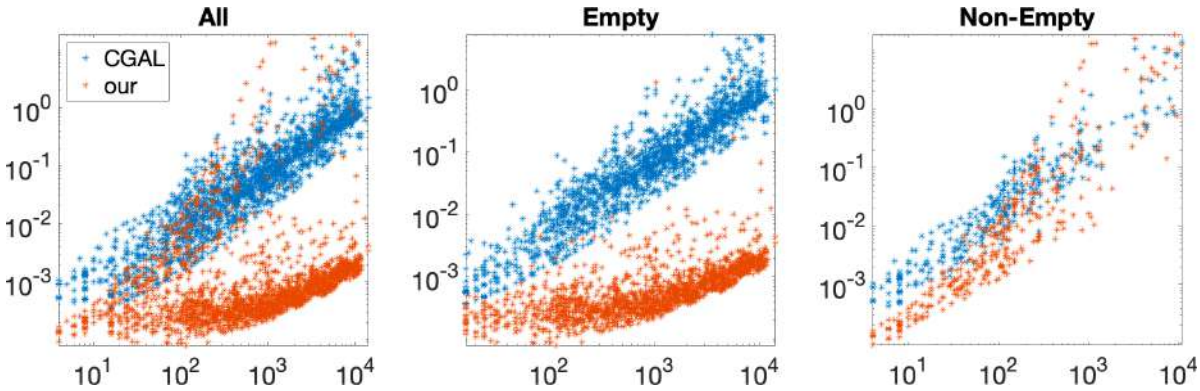


Figure A.8: *Thingi dataset* computational times. From left to right: all *Thingi dataset*, models with empty kernel, models with non-empty kernel.

In Figure A.8 we show the times distribution for the whole *Thingi dataset*, with a particular focus on the difference between models with empty kernels and models with non-empty kernels. Globally, the overall cost of computing all kernels is 173 seconds with our method against 518 seconds with CGAL, for an improvement of 3 times. When the kernel is empty, our algorithm is always faster than CGAL: the main reason for this is the usage of the *shuffle* mode, which makes it extremely cheap to recognize non-star-shaped elements. When the model is star-shaped, the distinction between the two methods is not always clear, as the result mainly depends on the shape and size of the object.

To further investigate this point, in Figure A.9 we present the kernel computation of 10 selected star-shaped examples from this dataset. In the top row we have models on which the geometric method is by far more efficient: *plus* (ThingiID: 1120761), *star* (ThingiID: 313883), *flex* (ThingiID: 827640) and *cross* (ThingiID: 313882). In the middle row, models for which the performance are similar: *part* (ThingiID: 472063), *super-ellipse* (ThingiID: 40172), *bot-eye* (ThingiID: 37276) and *button* (ThingiID: 1329185). Then, in the bottom row, we show models on which the algebraic method is preferable: *rt4-arm* (ThingiID: 39353), *ball* (ThingiID: 58238), *acorn* (ThingiID: 815480), *muffin* (ThingiID: 101636). The computational times, together with the ones relative to the whole dataset, are reported in Table A.3.

Once again, we notice that the size of the model impacts the performance of our method. Looking at Figure A.8, we can see how the models for which our method performs worse than CGAL are all in the right part of the plane (the one relative to models with a high number of vertices). At the same time, the number of vertices of the element, by itself, is not sufficient to

Table A.3: Computational times for complex models. The first number relative to *Thingi* dataset indicates the number of models instead of the number of vertices.

mesh	#vertices	our-shuffle	CGAL	ratio
plus	448	0.004	0.09	22.75
star	9633	0.4	5.15	12.93
flex	834	0.02	0.27	12.76
cross	3914	0.19	2.1	11.12
part	5382	2.58	6.94	2.69
super-ellipse	290	0.02	0.04	2.05
bot-eye	453	0.03	0.03	0.96
button	1227	0.1	0.08	0.75
rt4-arm	655	0.13	0.09	0.67
ball	660	0.24	0.04	0.15
acorn	4114	4.35	0.55	0.13
muffin	8972	11.73	0.54	0.04
Thingi dataset	1806	172.88	518.2	2.99

justify the supremacy of one method over the other. For example, models *star* and *flex* have very different sizes and times, but their ratio is quite similar; the same holds for models *parts* and *super-ellipse* or *ball* and *acorn*.

The shape of the object also plays an important role: over models with numerous adjacent co-planar faces like *plus*, *star* (whose bottom is completely flat), and *cross*, our method is preferable even when the size grows. As already seen in Section A.6.2, the presence of co-planar faces significantly improves the performance of our method. Vice-versa, over elements with significant curvatures like *rt4-arm*, *acorn*, or *muffin*, the algebraic method performs similarly or better than ours, even on relatively small models like *bot-eye*. Over these models, it is still possible to compute a correct kernel with the geometric approach, but the ratio between CGAL time and ours is in the order of 10^{-1}, or even 10^{-2}.

Related Publications

- T. Sorgente, S. Biasotti, and M. Spagnuolo. A geometric approach for computing the kernel of a polyhedron. In *Smart Tools and Apps for Graphics - Eurographics Italian Chapter Conference*, pages 11–19, P. Frosini, D. Giorgi, S. Melzi, E. Rodolà editors, 2021.

- T. Sorgente, S. Biasotti, and M. Spagnuolo. Polyhedron kernel computation using a geometric approach. In *Computers & Graphics*, vol. 105, pages 94-104, 2022.

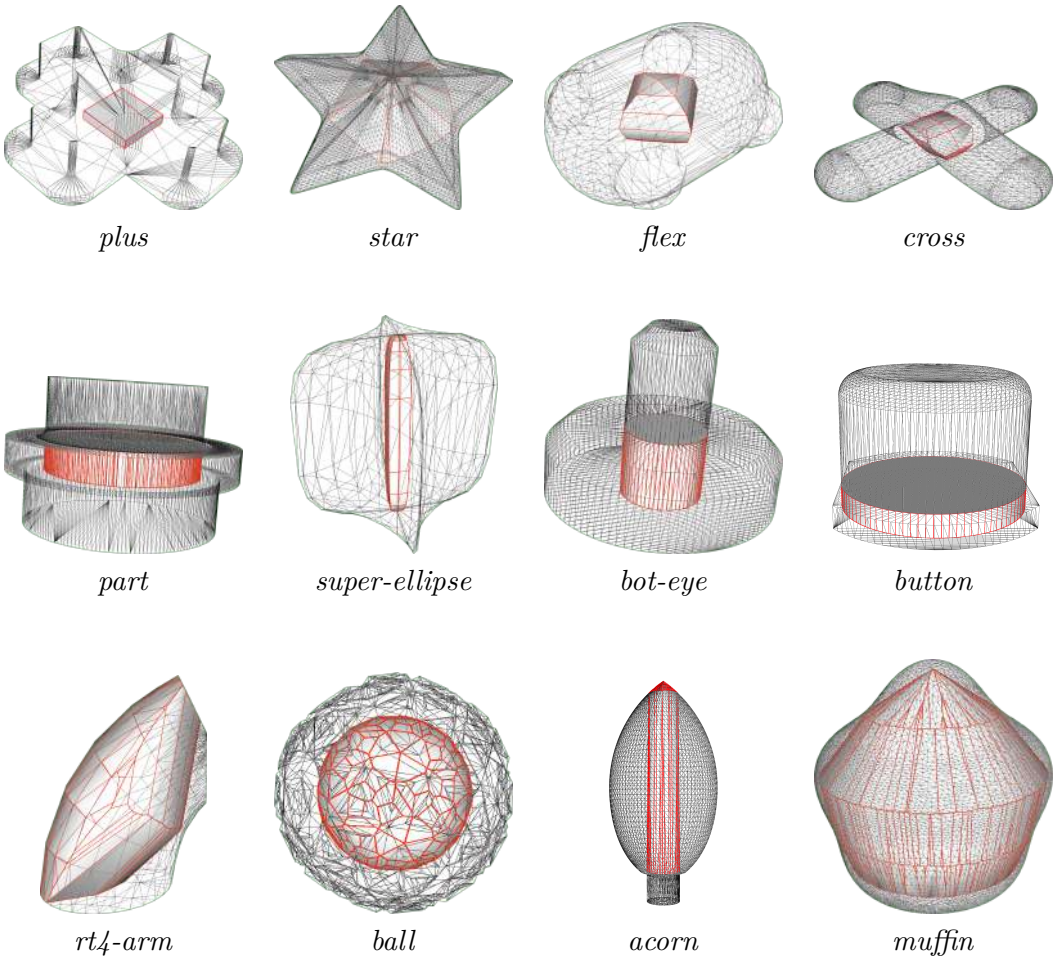


Figure A.9: Examples of our kernel evaluation for complex models. In the top row, models on which the geometric method is more efficient; in the middle, models for which the performance are similar; in the bottom row, models on which the algebraic method is preferable.

APPENDIX B

Algorithms for the Generation of the 2D Datasets

We take a closer look at how the datasets presented in Section 4.1.1 are built, and we precisely compute the quantities A_n and e_n defined in (4.1). All algorithms have been written using the *cinolib* library [Livesu, 2019]. We recall that a dataset is a finite mesh sequence $\mathcal{D} = \{\Omega_n\}_{n=0,\dots,N}$, ordered decreasingly with respect to the mesh size, and that the domain Ω is the unit square $(0,1)^2$.

Each of the following datasets is built around (and often named after) a particular polygonal element contained in it, which is meant to stress one or more assumptions or indicators. Their construction was originally presented in [Sorgente et al., 2021b]. All the meshes generated in this section are publicly accessible at `https://github.com/TommasoSorgente/vem-quality-dataset`.

B.1 Hybrid Datasets

The first dataset, $\mathcal{D}_{\text{Triangle}}$, contains only triangular meshes that are built by inserting a number of points in the domain and connecting them in a Delaunay triangulation. The point set is defined through the *Poisson Disk Sampling* algorithm proposed in [Bridson, 2007], empirically adjusting the distance between points (called "radius" in the original paper) in order to generate meshes with the desired number of vertices. Points are then connected in a Delaunay triangulation using the well-known *Triangle* library [Shewchuk, 2005], with the default parameters configuration. In $\mathcal{D}_{\text{Triangle}}$, A_n and e_n are almost constant, as no constraints are imposed on the triangulation process.

The construction of datasets $\mathcal{D}_{\text{Maze}}$ and $\mathcal{D}_{\text{Star}}$ is characterized by the insertion in Ω of a number of polygonal elements and by a tessellation algorithm. They are built around (and

named after) an *initial polygon* $E = E(t_n)$ depending on a *deformation parameter* $t_n \in [0, 1)$, which is used to deform E. This parameter directly depends on the mesh number (i.e. $t_n \to 1$ as $n \to N$), and it can be adjusted to improve or worsen the quality of the polygon E (the higher, the worse). At refinement step n, mesh Ω_n is created by inserting a number of identical copies of the deformed polygon $E(t_n)$ (opportunely resized) in the domain Ω, and tessellating the rest of Ω using *Triangle*. Note that, a whole family of other datasets may be generated by simply defining a new initial polygon. More examples can be found in [Attene et al., 2021].

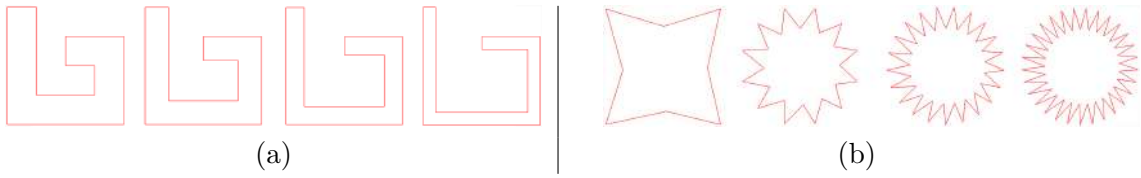


Figure B.1: Initial polygons $E(t_0)$, $E(t_2)$, $E(t_4)$, $E(t_6)$ from datasets $\mathcal{D}_{\text{Maze}}$(a) and $\mathcal{D}_{\text{Star}}$(b).

- The initial polygon $E(t_n)$ for dataset $\mathcal{D}_{\text{Maze}}$ is the 10-sided element shown in Figure B.1(a), with vertices:

$$(0,1), (0,0), (1,0), (1,0.75), (0.5,0.75), \left(0.5, 0.5 + \frac{t_n}{4}\right), \left(0.75 + \frac{t_n}{4}, 0.5 + \frac{t_n}{4}\right),$$
$$\left(0.75 + \frac{t_n}{4}, 0.25 - \frac{t_n}{4}\right), \left(0.25 - \frac{t_n}{4}, 0.25 - \frac{t_n}{4}\right), \left(0.25 - \frac{t_n}{4}, 1\right).$$

  As $t_n \to 1$, the length of the shortest edge (the one with vertices $(0, 1)$ and $(0.25 - t_n/4, 1)$) goes to zero, and so does the area of $E(t_n)$.

- For building the initial polygon $E(t_n)$ of dataset $\mathcal{D}_{\text{Star}}$ (Figure B.1(b)), we first build a $\bar{i}$-sided regular polygon, with $\bar{i} = 8(1 + \lfloor 10t_n \rfloor)$ and vertices $v_0 = (1, 0)$, $v_i = \sigma(v_{i-1})$ for $i = 1, \ldots, \bar{i}$, being $\sigma(v)$ the rotation centered at $(0, 0)$ of vertex v by an angle of $2\pi/\bar{i}$. Then we project every odd-indexed vertex towards the barycenter of $E(t_n)$: $v'_{2j+1} = s\ v_{2j+1}$, for $j = 0, \ldots, \frac{\bar{i}-1}{2}$, where the projection factor $s \in (0, 1)$ is gradually decreased until the angles at the even-indexed vertices become smaller than $(1 - t_n)\pi/3$. As $t_n \to 1$ we have an increasing number of edges (from 8 to almost 90), the minimum angle and the area decrease to zero while the length of every edge increases.

Once we defined the initial polygon $E(t_n)$, we can build the corresponding dataset through Algorithm 5. We have some initial parameters, which are set a priori and remain untouched: the number of meshes in the dataset N, the area of the initial polygon at the first step d_0 and the deformation range $T = [t_{\min}, t_{\max}]$. In this work we set $N = 10$, $d_0 = 0.03$, which corresponds to 3% of the domain, and $T = [0, 0.95]$. Then we have three main parameters, $e_n \in \mathbb{N}$, $t_n \in T$ and $d_n \in (0, d_0)$, which respectively regulate the number of initial polygons inserted, the deformation of these polygons and their area. In particular, e_n increases inversely

to d_n (Ω_{n+1} has twice as polygons as Ω_n, with halved areas), so that the percentage of the domain covered by polygons (not triangles) is preserved all across the dataset. Due to the complicated shapes of some initial polygons, it may be hard to ask for exactly $|E(t_n)| = d_n$, therefore we only impose $|E(t_n)| \leq d_n$.

Several options are possible for setting e_n, t_n and d_n, and the speed at which these quantities vary strongly affects the geometrical qualities of the meshes in the dataset. In our datasets, e_n increases exponentially, t_n increases linearly inside T and d_n decreases exponentially. The exponential increase of the number of initial polygons inserted in the domain may lead to intersections between them, or with the domain boundaries. To avoid this phenomenon, we inserted a *while* loop in Algorithm 5 which decreases d_n until no intersections occur: this ensures stability to the algorithm, but in practice, it activates only for very dense meshes and it typically runs only a few iterations.

Last, when all polygons have been inserted in Ω, the *Triangle* algorithm is used to generate a Delaunay triangulation. The already inserted polygons are considered as holes in the domain, and we set no limitations on the number of Steiner points that may appear in the triangulation process. We also impose to have no angles smaller than 20 degrees and set a maximum triangle area constraint equal to d_n. Due to the freedom left to the *Triangle* algorithm, it is not possible to estimate A_n and e_n precisely; hence, the relative values reported in Table 4.1 have been measured a posteriori.

Algorithm 5 Hybrid datasets generation

Input: initial polygon $E(t)$, initial parameters N, d_0, T;
Output: dataset $\mathcal{D}$

```
for n = 0, ..., N do
    set the main parameters: e_n := 2^n, t_n := n (t_max − t_min)/N, d_n := d_0/2^n;
    use Poisson Disk Sampling with r = 1/√(2e_n) to find e_n points {p_i}_{i=1,...,e_n} in Ω;
    generate polygon E(t_n) with |E(t_n)| ≤ d_n;
    insert a copy of E(t_n) centered around every p_i;
    while polygon E(t_n) intersects other polygons or the boundary of Ω do
        d_n := d_n − ε;
        generate a polygon E(t_n) with |E(t_n)| ≤ d_n;
        insert a copy of E(t_n) centered around every p_i;
    end while
    mesh Ω_n := Triangle(Ω), considering polygons E(t_n) as holes;
    D ← Ω_n;
end for
return D;
```

B.2 Mirroring Datasets

The construction of $\mathcal{D}_{\text{Jenga}}$, $\mathcal{D}_{\text{Slices}}$ and $\mathcal{D}_{\text{Ulike}}$, at every step $n \geq 1$, consists in a first algorithm for iteratively generating a *base mesh* $\widehat{\Omega}_n$ from the previous base mesh $\widehat{\Omega}_{n-1}$, followed by a *mirroring technique* which returns the computational mesh Ω_n. The base mesh generation algorithm is different for each dataset (Algorithms 6, 7 and 8), while the mirroring algorithm (Algorithm 9) is common to all three datasets. Algorithms 6, 7 and 8 depend on two initial parameters: N indicates the number of meshes in the dataset and N_{el} indicates the number of elements to insert at each step. For mirroring datasets we set $N_{el} = 1$, while for multiple mirroring datasets (described in the next section) we will set $N_{el} = 4$.

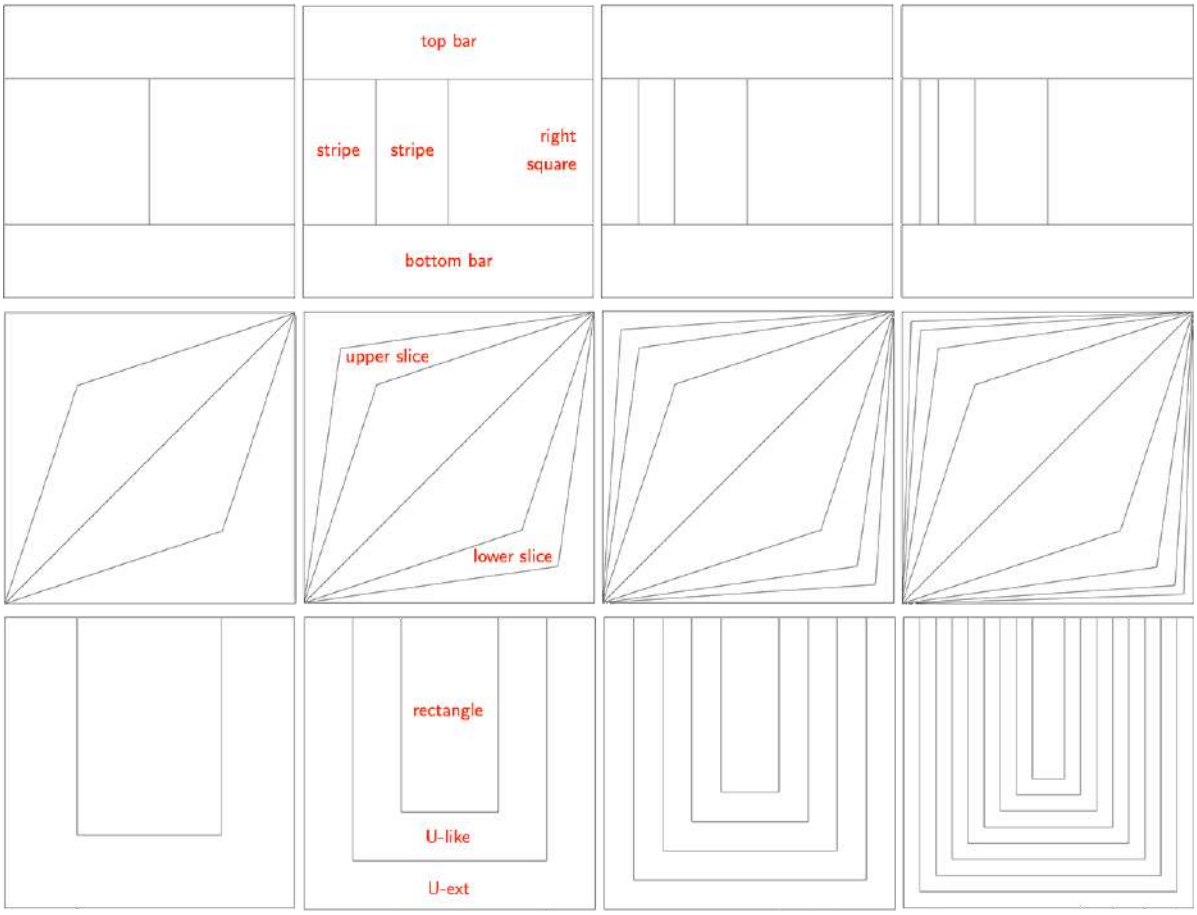


Figure B.2: Non-mirrored base meshes $\widehat{\Omega}_0, \widehat{\Omega}_1, \widehat{\Omega}_2, \widehat{\Omega}_3$ from datasets $\mathcal{D}_{\text{Jenga}}$(top), $\mathcal{D}_{\text{Slices}}$(middle), and $\mathcal{D}_{\text{Ulike}}$(bottom).

In the $\mathcal{D}_{\text{Jenga}}$ base mesh shown in Figure B.2(top) we have a top bar, a bottom bar and a right square which are fixed independently of n, and $n+1$ stripe elements in the left part of the domain. At each refinement step $n \geq 1$, a new rectangular element is created by splitting in two equal parts the leftmost stripe in the previous base mesh, and consequently updating

top bar and bottom bar with new vertices and edges. Therefore, all elements in $\widehat{\Omega}_n$, except for top bar and bottom bar, are rectangles with height equal to $1/2$ and basis ranging from $1/2$ to $1/2^{n+1}$. Once the base mesh $\widehat{\Omega}_n$ is generated, the mirroring algorithm *Mirror* is recursively applied for n times to generate the computational mesh Ω_n, as described in Algorithm 6. When computing A_n and e_n, we can restrict our calculations to the base mesh, because these ratios are not affected by the mirroring algorithm. In particular, the longest edge in the base mesh is the upper edge of top bar, which is never split, while the shortest edge is the basis of the leftmost stripe, which halves at each step: this causes $e_n \sim 2^n$. The top bar is also the element with the greatest area (together with bottom bar and right square), which is constantly equal to $1/4$, while the leftmost stripe has area $1/2 \cdot 1/2^{n+1} = 1/2^{n+2}$, therefore $A_n \sim 2^n$.

Algorithm 6 $\mathcal{D}_{\text{Jenga}}$ dataset generation

Input: number of meshes N, number of elements N_{el}
Output: dataset $\mathcal{D}_{\text{Jenga}}$

```
for n = 0, ..., N do
    top bar := {(0, 0.75), (1, 0.75), (1, 1), (0, 1)};
    bottom bar := {(0, 0), (1, 0), (1, 0.25), (0, 0.25)};
    right square := {(0.5, 0.25), (1, 0.25), (1, 0.75), (0.5, 0.75)};
    vector b := sample nN_el equally spaced points inside interval (0, 0.5);
    for i = 1, ..., size(b) do
        stripe[i] := {(b[i − 1], 0.25), (b[i], 0.25), (b[i], 0.75), (b[i − 1], 0.75)};
        top bar ← (b[i], 0.75);
        bottom bar ← (b[i], 0.25);
    end for
    mesh Ω̂_n := {top bar, bottom bar, right square, stripe};
    for i = 1, ..., n do
        mesh Ω̂_n := Mirror(Ω̂_n);
    end for
    D_Jenga ← Ω̂_n;
end for
return D_Jenga;
```

In the $\mathcal{D}_{\text{Slices}}$ base meshes shown in Figure B.2(middle), at each step $n \geq 0$, we add the vertices with coordinates $(2^{-i}, 1 - 2^{-i})$ and $(1 - 2^{-i}, 2^{-i})$ for $i = 1, \ldots, n+2$, and we connect them to the vertices $(0, 0)$ and $(1, 1)$. As a result, at each iteration, we create a couple of new polygons, called upper slice and lower slice, symmetrical with respect to the diagonal, and we add them to the base mesh. The area of the two inner triangles (the biggest polygons in the base mesh) is constantly equal to $1/4$. For evaluating the area of the two most external polygons, we consider them as the union of the two identical triangles obtained by splitting the polygons along the diagonal (the one connecting the vertices with coordinates $(0, 1)$ and $(1, 0)$). Then the smallest area in the base mesh is the sum of the areas of two equal triangles with basis $\sqrt{2}/2$ and height $2^{-n}/\sqrt{2}$, and simple calculations lead to $A_n \sim 2^n$. Last, we notice that all the edges

in the base mesh have lengths between 1 and $\sqrt{2}$, because no edge is ever split, hence $e_n \sim c$.

Algorithm 7 $\mathcal{D}_{\text{Slices}}$ dataset generation

Input: number of meshes N, number of elements N_{el}
Output: dataset $\mathcal{D}_{\text{Slices}}$

```
for n = 0, ..., N do
    vector b := [2^{-1}, 2^{-2}, ..., 2^{-n N_el}];
    for i = 1, ..., size(b) do
        upper slice[i] := {(0,0), (b[i], 1 - b[i]), (1,1), (b[i+1], 1 - b[i+1])};
        lower slice[i] := {(0,0), (1 - b[i], b[i]), (1,1), (1 - b[i+1], b[i+1])};
    end for
    mesh Ω̂_n := {upper slice, lower slice};
    for i = 1, ..., n do
        mesh Ω̂_n := Mirror(Ω̂_n);
    end for
    D_Slices ← Ω̂_n;
end for
return D_Slices;
```

In the $\mathcal{D}_{\text{Ulike}}$ base meshes shown in Figure B.2(bottom), at each step $n \geq 0$ we insert 2^n U-shaped continuous polylines inside the domain. We have an internal rectangle and a sequence of concentric equispaced polygons U-like culminating with an external polygon U-ext. This last element is not different from the polygons U-like, but is created separately because we need to split its lower edge in order to match the base mesh that will appear below it during the mirroring algorithm. In every base mesh, the shortest edge is the one corresponding to the width of each U-like, which measures $2^{-(n+1)}$, and the longest edges are the left and right boundaries of the domain. This causes $e_n \sim 2^n$. Said e the shortest edge, the smallest area is the one of rectangle, equal to $2e(1/2 + e)$, and the biggest area is the one relative to U-ext, equal to $3e - 2e^2$. We have

$$A_n = \frac{3 - 2e}{1 + 2e} = \frac{3 - 2(2^{-(n+1)})}{1 + 2(2^{-(n+1)})} = \frac{3 - 2^{-n}}{1 + 2^{-n}} \sim c.$$

B.3 Multiple Mirroring Datasets

Multiple mirroring datasets are built with the exact same algorithms as the mirroring datasets, changing the parameter N_{el}. This parameter regulates the number of elements generated in each base mesh of the dataset. In particular, datasets $\mathcal{D}_{\text{Jenga4}}$, $\mathcal{D}_{\text{Slices4}}$ and $\mathcal{D}_{\text{Ulike4}}$ are defined by setting $N_{el} = 4$. An example of a multiple mirroring dataset with $N_{el} = 4$ is shown in Figure B.3, where the first three base meshes of $\mathcal{D}_{\text{Ulike4}}$ are presented. The N_{el} value influences ratios A_n and e_n: if $A_n, e_n \sim 2^n$ for $N_{el} = 1$, these quantities become asymptotic to 2^{4n} when $N_{el} = 4$, except for the cases in which the ratios were constant (see Table 4.1).

Algorithm 8 $\mathcal{D}_{\text{Ulike}}$ dataset generation

Input: number of meshes N, number of elements N_{el}

Output: dataset $\mathcal{D}_{\text{Ulike}}$

for $n = 0, \ldots, N$ **do**
  vector $\mathbf{b}$:= sample $2^{nN_{el}}$ equally spaced points inside interval $(0, 0.5)$;
  for $i = 1, \ldots, \text{size}(\mathbf{b})$ **do**
    $\mathsf{U\text{-}like}[i] := \{(\mathbf{b}[i], 1), (\mathbf{b}[i], \mathbf{b}[i]), (1 - \mathbf{b}[i], \mathbf{b}[i]), (1 - \mathbf{b}[i], 1), (1 - \mathbf{b}[i+1], 1),$
      $(1 - \mathbf{b}[i+1], \mathbf{b}[i+1]), (\mathbf{b}[i+1], \mathbf{b}[i+1]), (\mathbf{b}[i+1], 1)\}$;
  end for
  $\mathsf{rectangle} := \{(\mathbf{b}[end], 1), (\mathbf{b}[end], \mathbf{b}[end]), (1 - \mathbf{b}[end], \mathbf{b}[end]), (1 - \mathbf{b}[end], 1)\}$;
  $\mathsf{U\text{-}ext} := \{(0, 1), (0, 0), (1, 0), (1, 1), (1 - \mathbf{b}[0], 1), (1 - \mathbf{b}[0], \mathbf{b}[0]), (\mathbf{b}[0], \mathbf{b}[0]), (\mathbf{b}[0], 1)\}$;
  for $b \in \mathbf{b}$ **do**
    $\mathsf{U\text{-}ext} \leftarrow \{(b, 0), (1 - b, 0)\}$;
  end for
  mesh $\widehat{\Omega}_n := \{\mathsf{U\text{-}ext}, \mathsf{U\text{-}like}, \mathsf{rectangle}\}$;
  for $i = 1, \ldots, n$ **do**
    mesh $\widehat{\Omega}_n = \mathit{Mirror}(\widehat{\Omega}_n)$;
  end for
  $\mathcal{D}_{\text{Ulike}} \leftarrow \widehat{\Omega}_n$;
end for
return $\mathcal{D}_{\text{Ulike}}$;

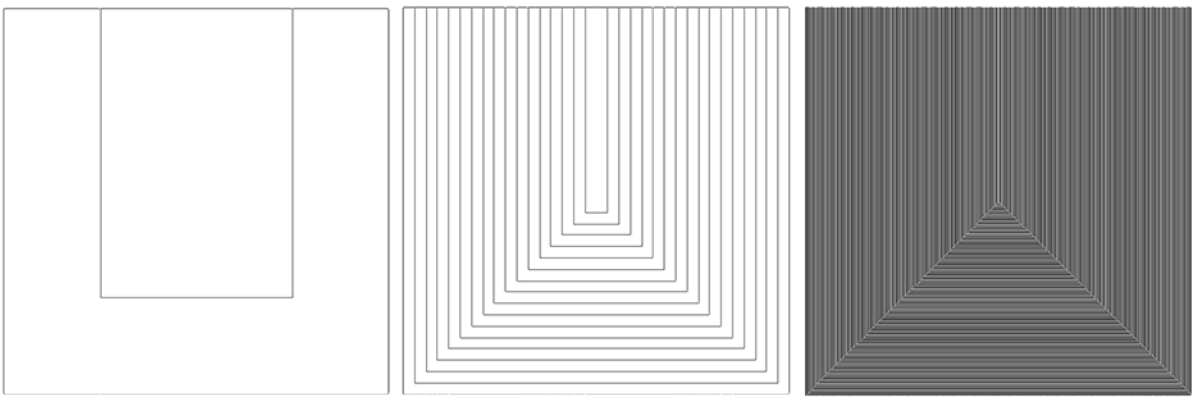


Figure B.3: Non-mirrored base meshes $\widehat{\Omega}_0, \widehat{\Omega}_1$ and $\widehat{\Omega}_2$ from dataset $\mathcal{D}_{\text{Ulike4}}$.

B.4 The Mirroring Algorithm

The *mirroring* algorithm (Algorithm 9) generates four adjacent copies of any polygonal mesh $\mathcal{M}$ defined over the domain $\Omega = (0,1)^2$. In *cinolib*, a polygonal mesh can be defined by a vector verts containing all its vertices and a vector polys containing all its polygons. The result of the algorithm is therefore a polygonal mesh $\mathcal{M}'$, generated by some vectors new-verts and new-polys, containing four times the number of vertices and polygons of $\mathcal{M}$. When iterated a sufficient number of times, this construction allows us to obtain a number of vertices and degrees of freedom in each mesh of the mirroring datasets that is comparable to that of the meshes at the same refinement level in hybrid datasets. Vector new-verts contains all vertices $v \in$ verts copied four times and translated by vectors $(0,0)$, $(1,0)$, $(1,1)$ and $(0,1)$ respectively. The coordinates of all vertices in new-verts are divided by 2 so that all new points lie in the same domain as the input mesh. Vector new-polys is simply vector polys repeated four times. A final cleaning step is required to remove duplicate vertices and edges that may arise in the mirroring process, for example, if the initial mesh $\mathcal{M}$ has vertices along its boundary.

Algorithm 9 Mesh mirroring

Input: base mesh $\mathcal{M}$
Output: mirrored mesh $\mathcal{M}'$

```
verts := vertices of M;  polys := polygons of M;
new-verts := verts;
for vertex v ∈ verts do new-verts ← v + (1,0);
end for
for vertex v ∈ verts do new-verts ← v + (1,1);
end for
for vertex v ∈ verts do new-verts ← v + (0,1);
end for
for vertex v ∈ new-verts do v := v/2;
end for
new-polys := [polys, polys, polys, polys];
M' := {new-verts, new-polys};
remove duplicated vertices and edges from M';
return M';
```

Related Publications

- T. Sorgente, D. Prada, D. Cabiddu, S. Biasotti, G. Patané, M. Pennacchio, S. Bertoluzza, M. Manzini, and M. Spagnuolo. VEM and the mesh. In *SEMA SIMAI Springer Series*, vol. 31(1), pages 1–54, Springer, 2021.

- T. Sorgente, S. Biasotti, M. Manzini, and M. Spagnuolo. The role of mesh quality and mesh quality indicators in the virtual element method. In *Advances in Computational*

Mathematics, vol. 48(1) pages 1–34, 2022.

Bibliography

[Abdelkader et al., 2020] Abdelkader, A., Bajaj, C. L., Ebeida, M. S., Mahmoud, A. H., Mitchell, S. A., Owens, J. D., and Rushdi, A. A. (2020).
Vorocrust: Voronoi meshing without clipping.
ACM Transactions on Graphics (TOG), 39(3):1–16.

[Adams and Fournier, 2003] Adams, R. A. and Fournier, J. J. F. (2003).
Sobolev spaces.
Pure and Applied Mathematics. Academic Press, Amsterdam, 2 edition.

[Adler and Thovert, 1999] Adler, P. M. and Thovert, J.-F. (1999).
Fractures and fracture networks, volume 15.
Springer Science & Business Media.

[Aghdaii et al., 2012] Aghdaii, N., Younesy, H., and Zhang, H. (2012).
5–6–7 meshes: Remeshing and analysis.
Computers & Graphics, 36(8):1072–1083.

[Ahmad et al., 2013] Ahmad, B., Alsaedi, A., Brezzi, F., Marini, L. D., and Russo, A. (2013).
Equivalent projectors for virtual element methods.
Computers & Mathematics with Applications, 66:376–391.

[Ahn and Shashkov, 2008] Ahn, H. T. and Shashkov, M. (2008).
Geometric algorithms for 3D interface reconstruction.
In *Proceedings of the 16th international meshing roundtable*, pages 405–422. Springer.

[Aiffa and Flaherty, 2003] Aiffa, M. and Flaherty, J. (2003).
A geometrical approach to mesh smoothing.
Computer methods in applied mechanics and engineering, 192(39-40):4497–4514.

[Alliez et al., 2003] Alliez, P., Cohen-Steiner, D., Devillers, O., Lévy, B., and Desbrun, M. (2003).
Anisotropic polygonal remeshing.
In *ACM SIGGRAPH 2003 Papers*, pages 485–493. -.

[Alliez et al., 2005] Alliez, P., Cohen-Steiner, D., Yvinec, M., and Desbrun, M. (2005).
Variational tetrahedral meshing.

In *ACM SIGGRAPH 2005 Papers*, SIGGRAPH '05, pages 617–625, New York, NY, USA. Association for Computing Machinery.

[Alliez et al., 2008] Alliez, P., Ucelli, G., Gotsman, C., and Attene, M. (2008).
Recent advances in remeshing of surfaces.
Shape analysis and structuring, pages 53–82.

[Anand et al., 2020] Anand, A., Ovall, J. S., Reynolds, S. E., and Weiser, S. (2020).
Trefftz finite elements on curvilinear polygons.
SIAM Journal on Scientific Computing, 42(2):A1289–A1316.

[Antonietti et al., 2021a] Antonietti, P. F., Beirão da Veiga, L., and Manzini, G., editors (2021a).
The Virtual Element Method and its Applications.
SEMA-SIMAI Series. Springer.
(to appear).

[Antonietti et al., 2021b] Antonietti, P. F., Berrone, S., Busetto, M., and Verani, M. (2021b).
Agglomeration-based geometric multigrid schemes for the virtual element method.
arXiv preprint arXiv:2112.11080.

[Antonietti et al., 2016] Antonietti, P. F., Cangiani, A., Collis, J., Dong, Z., Georgoulis, E. H., Giani, S., and Houston, P. (2016).
Review of discontinuous galerkin finite element methods for partial differential equations on complicated domains.
In *Building bridges: connections and challenges in modern approaches to numerical partial differential equations*, pages 281–310. Springer.

[Antonietti and Manuzzi, 2022] Antonietti, P. F. and Manuzzi, E. (2022).
Refinement of polygonal grids using convolutional neural networks with applications to polygonal discontinuous galerkin and virtual element methods.
Journal of Computational Physics, 452:110900.

[Antonietti et al., 2018] Antonietti, P. F., Manzini, G., and Verani, M. (2018).
The fully nonconforming Virtual Element method for biharmonic problems.
M3AS Math. Models Methods Appl. Sci., 28(2).

[Armstrong et al., 2015] Armstrong, C. G., Fogg, H. J., Tierney, C. M., and Robinson, T. T. (2015).
Common themes in multi-block structured quad/hex mesh generation.
Procedia Engineering, 124:70–82.

[Attene et al., 2021] Attene, M., Biasotti, S., Bertoluzza, S., Cabiddu, D., Livesu, M., Patané, G., Pennacchio, M., Prada, D., and Spagnuolo, M. (2021).
Benchmarking the geometrical robustness of a virtual element Poisson solver.

Mathematics and Computers in Simulation, 190:1392–1414.

[Attene and Spagnuolo, 2000] Attene, M. and Spagnuolo, M. (2000).
Automatic surface reconstruction from point sets in space.
In *Computer Graphics Forum*, volume 19-3, pages 457–465. Wiley Online Library.

[Au et al., 1998] Au, P., Dompierre, J., Labbé, P., Labb, P., Guibault, F., Guibault, F., and Camarero, R. (1998).
Proposal of benchmarks for 3d unstructured tetrahedral mesh optimization.
In *In Proceedings of the 7th International Meshing RoundTable'98*. Citeseer.

[Aurenhammer, 1987] Aurenhammer, F. (1987).
Power diagrams: properties, algorithms and applications.
SIAM Journal on Computing, 16(1):78–96.

[Aurenhammer, 1991] Aurenhammer, F. (1991).
Voronoi diagrams—a survey of a fundamental geometric data structure.
ACM Computing Surveys (CSUR), 23(3):345–405.

[Baeldung, 2021] Baeldung (2021).
Baeldung guides and courses.
`https://www.baeldung.com/cs/sort-points-clockwise`.

[Baker, 1989] Baker, T. J. (1989).
Automatic mesh generation for complex three-dimensional regions using a constrained delaunay triangulation.
Engineering with Computers, 5(3):161–175.

[Baker, 2005] Baker, T. J. (2005).
Mesh generation: Art or science?
Progress in Aerospace Sciences, 41(1):29–63.

[Barrett, 1996] Barrett, K. (1996).
Jacobians for isoparametric finite elements.
Communications in numerical methods in engineering, 12(11):755–766.

[Bawin et al., 2021] Bawin, A., Henrotte, F., and Remacle, J.-F. (2021).
Automatic feature-preserving size field for three-dimensional mesh generation.
International Journal for Numerical Methods in Engineering, 122(18):4825–4847.

[Beall and Shephard, 1997] Beall, M. W. and Shephard, M. S. (1997).
A general topology-based mesh data structure.
International Journal for Numerical Methods in Engineering, 40(9):1573–1596.

[Beirão da Veiga et al., 2013] Beirão da Veiga, L., Brezzi, F., Cangiani, A., Manzini, G., Marini, L. D., and Russo, A. (2013).
Basic principles of virtual element methods.

Mathematical Models & Methods in Applied Sciences, 23:119–214.

[Beirão da Veiga et al., 2014] Beirão da Veiga, L., Brezzi, F., Marini, L. D., and Russo, A. (2014).
The Hitchhiker's guide to the virtual element method.
Mathematical Models and Methods in Applied Sciences, 24(8):1541–1573.

[Beirão da Veiga et al., 2021] Beirão da Veiga, L., Dassi, F., Manzini, G., and Mascotto, L. (2021).
Virtual elements for Maxwell's equations.
arXiv preprints, arXiv: 2102.00950.

[Beirão da Veiga et al., 2017] Beirão da Veiga, L., Dassi, F., and Russo, A. (2017).
High-order virtual element method on polyhedral meshes.
Computers & Mathematics with Applications, 74:1110–1122.

[Beirão da Veiga and Manzini, 2014] Beirão da Veiga, L. and Manzini, G. (2014).
A virtual element method with arbitrary regularity.
IMA Journal on Numerical Analysis, 34(2):782–799.
DOI: 10.1093/imanum/drt018, (first published online 2013).

[Beirão da Veiga and Manzini, 2015] Beirão da Veiga, L. and Manzini, G. (2015).
Residual a posteriori error estimation for the virtual element method for elliptic problems.
ESAIM: Mathematical Modelling and Numerical Analysis, 49:577–599.

[Beirão da Veiga et al., 2019a] Beirão da Veiga, L., Manzini, G., and Mascotto, L. (2019a).
A posteriori error estimation and adaptivity in hp virtual elements.
Numer. Math., 143:139–175.

[Beirão da Veiga et al., 2019b] Beirão da Veiga, L., Mora, D., and Vacca, G. (2019b).
The Stokes complex for virtual elements with application to Navier–Stokes flows.
J. Sci. Comput., 81:990–1018.

[Beirão da Veiga and Vacca, 2020] Beirão da Veiga, L. and Vacca, G. (2020).
Sharper error estimates for virtual elements and a bubble-enriched version.
arXiv preprint arXiv:2005.12009.

[Beirão da Veiga et al., 2020] Beirão da Veiga, L., Brezzi, F., Marini, L., and Russo, A. (2020).
Polynomial preserving virtual elements with curved edges.
Mathematical Models and Methods in Applied Sciences, 30(08):1555–1590.

[Beirão da Veiga et al., 2014] Beirão da Veiga, L., Lipnikov, K., and Manzini, G. (2014).
The mimetic finite difference method for elliptic problems, volume 11.
Springer.

[Beirão da Veiga et al., 2017] Beirão da Veiga, L., Lovadina, C., and Russo, A. (2017).
Stability analysis for the virtual element method.

Mathematical Models and Methods in Applied Sciences, 27(13):2557–2594.

[Benedetto et al., 2014] Benedetto, M. F., Berrone, S., Pieraccini, S., and Scialò, S. (2014).
The virtual element method for discrete fracture network simulations.
Computer Methods in Applied Mechanics and Engineering, 280(0):135 – 156.

[Benedetto et al., 2016] Benedetto, M. F., Berrone, S., and Scialò, S. (2016).
A globally conforming method for solving flow in discrete fracture networks using the virtual element method.
Finite Elements in Analysis and Design, 109:23–36.

[Benvenuti et al., 2019] Benvenuti, E., Chiozzi, A., Manzini, G., and Sukumar, N. (2019).
Extended virtual element method for the Laplace problem with singularities and discontinuities.
Computer Methods in Applied Mechanics and Engineering, 356:571 – 597.

[Benzley et al., 1995] Benzley, S. E., Perry, E., Merkley, K., Clark, B., and Sjaardama, G. (1995).
A comparison of all hexagonal and all tetrahedral finite element meshes for elastic and elasto-plastic analysis.
In *Proceedings, 4th international meshing roundtable*, volume 17, pages 179–191. Citeseer.

[Bern and Eppstein, 1992] Bern, M. and Eppstein, D. (1992).
Mesh generation and optimal triangulation.
Computing in Euclidean geometry, 1:23–90.

[Bern and Plassmann, 2000] Bern, M. W. and Plassmann, P. E. (2000).
Mesh generation.
Handbook of computational geometry, 38.

[Berrone et al., 2017] Berrone, S., Benedetto, M., Borio, A., Pieraccini, S., and Scialò, S. (2017).
The virtual element method for the transport of passive scalars in discrete fracture networks.
In *European Conference on Numerical Mathematics and Advanced Applications*, pages 501–508. Springer.

[Berrone et al., 2018] Berrone, S., Borio, A., and Manzini (2018).
SUPG stabilization for the nonconforming virtual element method for advection–diffusion–reaction equations.
Computer Methods in Applied Mechanics and Engineering, 340:500–529.

[Bertoluzza et al., 2021] Bertoluzza, S., Manzini, G., Pennacchio, M., and Prada, D. (2021).
Stabilization of the nonconforming virtual element method.
Computers & Mathematics with Applications.

[Biasotti et al., 2014] Biasotti, S., Falcidieno, B., Giorgi, D., and Spagnuolo, M. (2014).
Mathematical tools for shape analysis and description.

Synthesis Lectures on Computer Graphics and Animation, 6(2):1–138.

[Blacker and Stephenson, 1991] Blacker, T. D. and Stephenson, M. B. (1991).
Paving: A new approach to automated quadrilateral mesh generation.
International journal for numerical methods in engineering, 32(4):811–847.

[Boier-Martin et al., 2004] Boier-Martin, I., Rushmeier, H., and Jin, J. (2004).
Parameterization of triangle meshes over quadrilateral domains.
In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 193–203.

[Boissonnat, 1984] Boissonnat, J.-D. (1984).
Geometric structures for three-dimensional shape representation.
ACM Transactions on Graphics (TOG), 3(4):266–286.

[Boissonnat et al., 2000] Boissonnat, J.-D., Devillers, O., Teillaud, M., and Yvinec, M. (2000).
Triangulations in cgal.
In *Proceedings of the sixteenth annual symposium on Computational geometry*, pages 11–18.

[Boissonnat and Oudot, 2005] Boissonnat, J.-D. and Oudot, S. (2005).
Provably good sampling and meshing of surfaces.
Graphical Models, 67(5):405–451.

[Boissonnat and Yvinec, 1998] Boissonnat, J.-D. and Yvinec, M. (1998).
Algorithmic geometry.
Cambridge university press.

[Bommes et al., 2013a] Bommes, D., Campen, M., Ebke, H.-C., Alliez, P., and Kobbelt, L. (2013a).
Integer-grid maps for reliable quad meshing.
ACM Transactions on Graphics (TOG), 32(4):1–12.

[Bommes et al., 2013b] Bommes, D., Lévy, B., Pietroni, N., Puppo, E., Silva, C., Tarini, M., and Zorin, D. (2013b).
Quad-mesh generation and processing: A survey.
In *Computer Graphics Forum*, volume 32-6, pages 51–76. Wiley Online Library.

[Bommes et al., 2009] Bommes, D., Zimmer, H., and Kobbelt, L. (2009).
Mixed-integer quadrangulation.
ACM Transactions On Graphics (TOG), 28(3):1–10.

[Boost, 2021] Boost (2021).
Boost C++ Libraries.
`http://www.boost.org/`.

[Boots et al., 2009] Boots, B., Sugihara, K., Chiu, S. N., and Okabe, A. (2009).
Spatial tessellations: concepts and applications of Voronoi diagrams.

John Wiley & Sons.

[Borouchaki et al., 1995] Borouchaki, H., Hecht, F., Saltel, E., and George, P. (1995).
Reasonably efficient delaunay based mesh generator in 3 dimensions.
In *Proceedings 4th International Meshing Roundtable*, pages 3–14. Citeseer.

[Botsch et al., 2010] Botsch, M., Kobbelt, L., Pauly, M., Alliez, P., and Lévy, B. (2010).
Polygon mesh processing.
CRC press.

[Boykov and Kolmogorov, 2004] Boykov, Y. and Kolmogorov, V. (2004).
An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision.
IEEE transactions on pattern analysis and machine intelligence, 26(9):1124–1137.

[Boykov et al., 2001] Boykov, Y., Veksler, O., and Zabih, R. (2001).
Fast approximate energy minimization via graph cuts.
IEEE Transactions on pattern analysis and machine intelligence, 23(11):1222–1239.

[Brenner et al., 2017] Brenner, S. C., Guan, Q., and Sung, L.-Y. (2017).
Some estimates for virtual element methods.
Computational Methods in Applied Mathematics, 17(4):553–574.

[Brenner and Sung, 2018] Brenner, S. C. and Sung, L.-Y. (2018).
Virtual element methods on meshes with small edges or faces.
Mathematical Models and Methods in Applied Sciences, 28(07):1291–1336.

[Brezzi et al., 2005] Brezzi, F., Lipnikov, K., and Shashkov, M. (2005).
Convergence of the mimetic finite difference method for diffusion problems on polyhedral meshes.
SIAM Journal on Numerical Analysis, 43(5):1872–1896.

[Brezzi and Marini, 2013] Brezzi, F. and Marini, L. D. (2013).
Virtual element methods for plate bending problems.
Computer Methods in Applied Mechanics and Engineering, 253:455–462.

[Bridson, 2007] Bridson, R. (2007).
Fast Poisson disk sampling in arbitrary dimensions.
SIGGRAPH sketches, 10:1.

[Brückler et al., 2022] Brückler, H., Gupta, O., Mandad, M., and Campen, M. (2022).
The 3D Motorcycle Complex for Structured Volume Decomposition.
Computer Graphics Forum.

[Campen et al., 2016] Campen, M., Silva, C. T., and Zorin, D. (2016).
Bijective maps from simplicial foliations.
ACM Transactions on Graphics (TOG), 35(4):1–15.

[Cangiani et al., 2014] Cangiani, A., Georgoulis, E. H., and Houston, P. (2014).
hp-version discontinuous galerkin methods on polygonal and polyhedral meshes.
Mathematical Models and Methods in Applied Sciences, 24(10):2009–2041.

[Cangiani et al., 2017a] Cangiani, A., Georgoulis, E. H., Pryer, T., and Sutton, O. J. (2017a).
A posteriori error estimates for the virtual element method.
Numerische Mathematik, pages 1–37.

[Cangiani et al., 2016] Cangiani, A., Gyrya, V., and Manzini, G. (2016).
The non-conforming virtual element method for the Stokes equations.
SIAM Journal on Numerical Analysis, 54(6):3411–3435.

[Cangiani et al., 2017b] Cangiani, A., Manzini, G., and Sutton, O. (2017b).
Conforming and nonconforming virtual element methods for elliptic problems.
IMA Journal on Numerical Analysis, 37:1317–1354.

[Carr et al., 2006] Carr, N. A., Hoberock, J., Crane, K., and Hart, J. C. (2006).
Rectangular multi-chart geometry images.
In *Proceedings of the fourth Eurographics symposium on Geometry processing*.

[Certik et al., 2019] Certik, O., Gardini, F., Manzini, G., Mascotto, L., and Vacca, G. (2019).
The p- and hp-versions of the virtual element method for elliptic eigenvalue problems.
Computers & Mathematics with Applications.
published online: 31 October 2019.

[Certik et al., 2018] Certik, O., Gardini, F., Manzini, G., and Vacca, G. (2018).
The virtual element method for eigenvalue problems with potential terms on polytopic meshes.
Applications of Mathematics, 63(3):333–365.

[Chalmeta et al., 2013] Chalmeta, R., Hurtado, F., Sacristán, V., and Saumell, M. (2013).
Measuring regularity of convex polygons.
Computer-Aided Design, 45(2):93–104.

[Chen et al., 2019] Chen, W., Zheng, X., Ke, J., Lei, N., Luo, Z., and Gu, X. (2019).
Quadrilateral mesh generation i: Metric based method.
Computer Methods in Applied Mechanics and Engineering, 356:652–668.

[Cherchi et al., 2016] Cherchi, G., Livesu, M., and Scateni, R. (2016).
Polycube simplification for coarse layouts of surfaces and volumes.
In *Computer Graphics Forum*, volume 35-5, pages 11–20. Wiley Online Library.

[Ciarlet, 2002] Ciarlet, P. G. (2002).
The finite element method for elliptic problems.
SIAM.

[Cignoni et al., 2003] Cignoni, P., Montani, C., Rocchini, C., and Scopigno, R. (2003).

External memory management and simplification of huge meshes.
IEEE Transactions on Visualization and Computer Graphics, 9(4):525–537.

[Cockburn et al., 2008] Cockburn, B., Dong, B., and Guzmán, J. (2008).
A superconvergent ldg-hybridizable galerkin method for second-order elliptic problems.
Mathematics of Computation, 77(264):1887–1916.

[Cockburn et al., 2012] Cockburn, B., Karniadakis, G. E., and Shu, C.-W. (2012).
Discontinuous Galerkin methods: theory, computation and applications, volume 11.
Springer Science & Business Media.

[Cordova and Barth, 1988] Cordova, J. and Barth, T. (1988).
Grid generation for general 2-d regions using hyperbolic equations.
In *26th Aerospace Sciences Meeting*, page 520.

[CoreForm, 2021] CoreForm (2021).
Coreform.
`https://coreform.com/products/coreform-cubit/government/`.

[Coxeter, 1938] Coxeter, H. S. M. (1938).
Regular skew polyhedra in three and four dimension, and their topological analogues.
Proceedings of the London Mathematical Society, 2(1):33–62.

[Coxeter, 1973] Coxeter, H. S. M. (1973).
Regular polytopes.
Courier Corporation.

[Cubit, 2021] Cubit (2021).
Cubit.
`https://cubit.sandia.gov`.

[De Floriani et al., 1985] De Floriani, L., Falcidieno, B., and Pienovi, C. (1985).
Delaunay-based representation of surfaces defined over arbitrarily shaped domains.
Computer Vision, Graphics, and Image Processing, 32(1):127–140.

[Decroux and Gosselin, 2013] Decroux, B. and Gosselin, O. (2013).
Computation of effective dynamic properties of naturally fractured reservoirs: Comparison and validation of methods.
In *EAGE Annual Conference & Exhibition incorporating SPE Europec.* OnePetro.

[Dey, 2006] Dey, T. K. (2006).
Curve and surface reconstruction: algorithms with mathematical analysis, volume 23.
Cambridge University Press.

[Dey and Sun, 2006] Dey, T. K. and Sun, J. (2006).
Defining and computing curve-skeletons with medial geodesic function.
In *Symposium on geometry processing*, volume 6, pages 143–152.

[Di Pietro and Droniou, 2019] Di Pietro, D. A. and Droniou, J. (2019).
The Hybrid High-Order method for polytopal meshes, volume 19.
Springer.

[Dong et al., 2006] Dong, S., Bremer, P.-T., Garland, M., Pascucci, V., and Hart, J. C. (2006).
Spectral surface quadrangulation.
In *ACM SIGGRAPH 2006 Papers*, pages 1057–1066. .

[Du et al., 1999] Du, Q., Faber, V., and Gunzburger, M. (1999).
Centroidal voronoi tessellations: Applications and algorithms.
SIAM review, 41(4):637–676.

[Dupont and Scott, 1980] Dupont, T. and Scott, R. (1980).
Polynomial approximation of functions in sobolev spaces.
Mathematics of Computation, 34(150):441–463.

[Edelsbrunner et al., 2001] Edelsbrunner, H. et al. (2001).
Geometry and topology for mesh generation.
Cambridge University Press.

[Edelsbrunner and Mücke, 1994] Edelsbrunner, H. and Mücke, E. P. (1994).
Three-dimensional alpha shapes.
ACM Transactions on Graphics (TOG), 13(1):43–72.

[Edelsbrunner and Shah, 1994] Edelsbrunner, H. and Shah, N. R. (1994).
Triangulating topological spaces.
In *Proceedings of the tenth annual symposium on Computational geometry*, pages 285–292.

[Erten et al., 2009] Erten, H., Üngör, A., and Zhao, C. (2009).
Mesh smoothing algorithms for complex geometric domains.
In *Proceedings of the 18th international meshing roundtable*, pages 175–193. Springer.

[Fabri and Pion, 2009] Fabri, A. and Pion, S. (2009).
Cgal: The computational geometry algorithms library.
In *Proceedings of the 17th ACM SIGSPATIAL international conference on advances in geographic information systems*, pages 538–539.

[Fang et al., 2016] Fang, X., Xu, W., Bao, H., and Huang, J. (2016).
All-hex meshing using closed-form induced polycube.
ACM Transactions on Graphics (TOG), 35(4):1–9.

[Fidelibus et al., 2009] Fidelibus, C., Cammarata, G., and Cravero, M. (2009).
Hydraulic characterization of fractured rocks.

[Fortune, 1995] Fortune, S. (1995).
Voronoi diagrams and delaunay triangulations.
Computing in Euclidean geometry, pages 225–265.

[Freitag, 1997] Freitag, L. A. (1997).
On combining laplacian and optimization-based mesh smoothing techniques.
Technical report, Argonne National Lab., IL (United States).

[Freitag and Plassmann, 2000] Freitag, L. A. and Plassmann, P. (2000).
Local optimization-based simplicial mesh untangling and improvement.
International Journal for Numerical Methods in Engineering, 49(1-2):109–125.

[Frey and George, 2007] Frey, P. J. and George, P.-L. (2007).
Mesh generation: application to finite elements.
Iste.

[Fu et al., 2016] Fu, X.-M., Bai, C.-Y., and Liu, Y. (2016).
Efficient volumetric polycube-map construction.
In *Computer Graphics Forum*, volume 35-7, pages 97–106. Wiley Online Library.

[Funaro, 1997] Funaro, D. (1997).
Spectral elements for transport-dominated equations, volume 1.
Springer Science & Business Media.

[Gao et al., 2017] Gao, X., Jakob, W., Tarini, M., and Panozzo, D. (2017).
Robust hex-dominant mesh generation using field-guided polyhedral agglomeration.
ACM Transactions on Graphics (TOG), 36(4):1–13.

[Gao et al., 2019] Gao, X., Shen, H., and Panozzo, D. (2019).
Feature preserving octree-based hexahedral meshing.
In *Computer graphics forum*, volume 38-5, pages 135–149. Wiley Online Library.

[Gardini et al., 2019] Gardini, F., Manzini, G., and Vacca, G. (2019).
The nonconforming virtual element method for eigenvalue problems.
ESAIM: Mathematical Modelling and Numerical Analysis, 53:749–774.

[Garimella, 2002] Garimella, R. V. (2002).
Mesh data structure selection for mesh generation and fea applications.
International journal for numerical methods in engineering, 55(4):451–478.

[Geuzaine and Remacle, 2009] Geuzaine, C. and Remacle, J.-F. (2009).
Gmsh: A 3-d finite element mesh generator with built-in pre-and post-processing facilities.
International journal for numerical methods in engineering, 79(11):1309–1331.

[Gillette et al., 2012] Gillette, A., Rand, A., and Bajaj, C. (2012).
Error estimates for generalized barycentric interpolation.
Advances in computational mathematics, 37(3):417–439.

[Goldberg and Tarjan, 1988] Goldberg, A. V. and Tarjan, R. E. (1988).
A new approach to the maximum-flow problem.
Journal of the ACM (JACM), 35(4):921–940.

[Gordon and Hall, 1973] Gordon, W. J. and Hall, C. A. (1973).
Transfinite element methods: blending-function interpolation over arbitrary curved element domains.
Numerische Mathematik, 21(2):109–129.

[Gordon and Thiel, 1982] Gordon, W. J. and Thiel, L. C. (1982).
Transfinite mappings and their application to grid generation.
Applied Mathematics and Computation, 10:171–233.

[Gregson et al., 2011] Gregson, J., Sheffer, A., and Zhang, E. (2011).
All-hex mesh generation via volumetric polycube deformation.
In *Computer graphics forum*, volume 30-5, pages 1407–1416. Wiley Online Library.

[Hatcher, 2002] Hatcher, A. (2002).
Algebraic Topology.
Cambridge University Press.

[Herrmann, 1976] Herrmann, L. R. (1976).
Laplacian-isoparametric grid generation scheme.
Journal of the Engineering Mechanics Division, 102(5):749–756.

[Ho-Le, 1988] Ho-Le, K. (1988).
Finite element mesh generation methods: a review and classification.
Computer-aided design, 20(1):27–38.

[Hong and Elber, 2022] Hong, Q. Y. and Elber, G. (2022).
Detection and computation of conservative kernels of models consisting of freeform curves and surfaces, using inequality constraints.
Computer Aided Geometric Design, page 102075.

[Huang and Wang, 2020] Huang, W. and Wang, Y. (2020).
Anisotropic mesh quality measures and adaptation for polygonal meshes.
Journal of Computational Physics, 410:109368.

[Hughes et al., 2005] Hughes, T. J., Cottrell, J. A., and Bazilevs, Y. (2005).
Isogeometric analysis: Cad, finite elements, nurbs, exact geometry and mesh refinement.
Computer methods in applied mechanics and engineering, 194(39-41):4135–4195.

[Ito et al., 2009] Ito, Y., Shih, A. M., and Soni, B. K. (2009).
Octree-based reasonable-quality hexahedral mesh generation using a new set of refinement templates.
International Journal for Numerical Methods in Engineering, 77(13):1809–1833.

[Jacobson and Panozzo, 2017] Jacobson, A. and Panozzo, D. (2017).
Libigl: prototyping geometry processing research in c++.
In *SIGGRAPH Asia 2017 courses*, pages 1–172. -.

[Kälberer et al., 2007] Kälberer, F., Nieser, M., and Polthier, K. (2007).
Quadcover-surface parameterization using branched coverings.
In *Computer graphics forum*, volume 26-3, pages 375–384. Wiley Online Library.

[Knupp, 2012] Knupp, P. (2012).
Introducing the target-matrix paradigm for mesh optimization via node-movement.
Engineering with Computers, 28(4):419–429.

[Knupp, 2000] Knupp, P. M. (2000).
Achieving finite element mesh quality via optimization of the jacobian matrix norm and associated quantities. part ii—a framework for volume mesh optimization and the condition number of the jacobian matrix.
International Journal for numerical methods in engineering, 48(8):1165–1185.

[Knupp, 2001] Knupp, P. M. (2001).
Algebraic mesh quality metrics.
SIAM journal on scientific computing, 23(1):193–218.

[Kolmogorov and Zabin, 2004] Kolmogorov, V. and Zabin, R. (2004).
What energy functions can be minimized via graph cuts?
IEEE transactions on pattern analysis and machine intelligence, 26(2):147–159.

[Kong and Rosenfeld, 1989] Kong, T. and Rosenfeld, A. (1989).
Digital topology: Introduction and survey.
Computer Vision, Graphics, and Image Processing, 48(3):357–393.

[Kovalevsky, 1989] Kovalevsky, V. (1989).
Finite topology as applied to image analysis.
Computer Vision, Graphics, and Image Processing, 46(2):141–161.

[Lawson, 1977] Lawson, C. L. (1977).
Software for c1 surface interpolation.
In *Mathematical software*, pages 161–194. Elsevier.

[Lee and Preparata, 1979] Lee, D.-T. and Preparata, F. P. (1979).
An optimal algorithm for finding the kernel of a polygon.
J. ACM, 26(3):415–421.

[Lee et al., 1980] Lee, K., Huang, M., Yu, N., and Rubbert, P. (1980).
Grid generation for general three-dimensional configurations.
NASA. Langley Research Center Numerical Grid Generation Tech.

[Lévy, 2014] Lévy, B. (2014).
Restricted voronoi diagrams for (re)-meshing surfaces and volumes.
In *8th International Conference on Curves and Surfaces*, volume 6, page 14.

[Lévy and Filbois, 2015] Lévy, B. and Filbois, A. (2015).

Geogram: a library for geometric algorithms.

[Lévy and Liu, 2010] Lévy, B. and Liu, Y. (2010).
Lp centroidal voronoi tessellation and its applications.
ACM Transactions on Graphics (TOG), 29(4):1–11.

[Li et al., 1995] Li, T., McKeag, R., and Armstrong, C. (1995).
Hexahedral meshing using midpoint subdivision and integer programming.
Computer methods in applied mechanics and engineering, 124(1-2):171–193.

[Liseikin, 2006] Liseikin, V. D. (2006).
A computational differential geometry approach to grid generation.
Springer Science & Business Media.

[Liseikin, 2017] Liseikin, V. D. (2017).
Grid generation methods, volume 1.
Springer.

[Liu and Joe, 1994] Liu, A. and Joe, B. (1994).
Relationship between tetrahedron shape measures.
BIT Numerical Mathematics, 34(2):268–287.

[Liu et al., 2015] Liu, L., Sheng, Y., Zhang, G., and Ugail, H. (2015).
Graph cut based mesh segmentation using feature points and geodesic distance.
In *2015 International Conference on Cyberworlds (CW)*, pages 115–120. IEEE.

[Liu et al., 2009] Liu, Y., Wang, W., Lévy, B., Sun, F., Yan, D.-M., Lu, L., and Yang, C. (2009).
On centroidal voronoi tessellation—energy smoothness and fast computation.
ACM Transactions on Graphics (ToG), 28(4):1–17.

[Livesu, 2019] Livesu, M. (2019).
cinolib: a generic programming header only C++ library for processing polygonal and polyhedral meshes.
Transactions on Computational Science XXXIV.
https://github.com/mlivesu/cinolib/.

[Livesu et al., 2016] Livesu, M., Muntoni, A., Puppo, E., and Scateni, R. (2016).
Skeleton-driven adaptive hexahedral meshing of tubular shapes.
In *Computer Graphics Forum*, volume 35-7, pages 237–246. Wiley Online Library.

[Livesu et al., 2020] Livesu, M., Pietroni, N., Puppo, E., Sheffer, A., and Cignoni, P. (2020).
Loopycuts: Practical feature-preserving block decomposition for strongly hex-dominant meshing.
ACM Transactions on Graphics (TOG), 39(4):121–1.

[Livesu et al., 2021] Livesu, M., Pitzalis, L., and Cherchi, G. (2021).
Optimal dual schemes for adaptive grid based hexmeshing.

ACM Transactions on Graphics (TOG), 41(2):1–14.

[Livesu et al., 2013] Livesu, M., Vining, N., Sheffer, A., Gregson, J., and Scateni, R. (2013).
Polycut: Monotone graph-cuts for polycube base-complex construction.
ACM Transactions on Graphics (TOG), 32(6):1–12.

[Lo, 1985] Lo, D. S. (1985).
A new mesh generation scheme for arbitrary planar domains.
International journal for numerical methods in engineering, 21(8):1403–1426.

[Lo, 2013] Lo, D. S. (2013).
Dynamic grid for mesh generation by the advancing front method.
Computers & Structures, 123:15–27.

[Lo, 2014] Lo, D. S. (2014).
Finite element mesh generation.
CRC Press.

[Löhner, 1988] Löhner, R. (1988).
Some useful data structures for the generation of unstructured grids.
Communications in Applied Numerical Methods, 4(1):123–135.

[Lyon et al., 2016] Lyon, M., Bommes, D., and Kobbelt, L. (2016).
Hexex: Robust hexahedral mesh extraction.
ACM Transactions on Graphics (TOG), 35(4):1–11.

[Mäntylä, 1987] Mäntylä, M. (1987).
An introduction to solid modeling.
Computer Science Press, Inc.

[Marcum and Weatherill, 1995] Marcum, D. L. and Weatherill, N. P. (1995).
Unstructured grid generation using iterative point insertion and local reconnection.
AIAA journal, 33(9):1619–1625.

[Maréchal, 2009] Maréchal, L. (2009).
Advances in octree-based all-hexahedral mesh generation: handling sharp features.
In *Proceedings of the 18th international meshing roundtable*, pages 65–84. Springer.

[Martin et al., 2009] Martin, T., Cohen, E., and Kirby, R. M. (2009).
Volumetric parameterization and trivariate b-spline fitting using harmonic functions.
Computer aided geometric design, 26(6):648–664.

[Mascotto, 2018] Mascotto, L. (2018).
Ill-conditioning in the virtual element method: stabilizations and bases.
Numer. Methods Partial Differential Equations, 34(4):1258–1281.

[MeshGems, 2020] MeshGems (2020).

Distene sas.
`http://www.meshgems.com/volume-meshing-meshgems-hexa.html`.

[Meyer et al., 2003] Meyer, M., Desbrun, M., Schröder, P., and Barr, A. H. (2003).
Discrete differential-geometry operators for triangulated 2-manifolds.
In Hege, H.-C. and Polthier, K., editors, *Visualization and Mathematics III*, pages 35–57, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Misztal et al., 2009] Misztal, M. K., Bærentzen, J. A., Anton, F., and Erleben, K. (2009).
Tetrahedral mesh improvement using multi-face retriangulation.
In *Proceedings of the 18th international meshing roundtable*, pages 539–555. Springer.

[Mitchell et al., 1971] Mitchell, A., Phillips, G., and Wachspress, E. (1971).
Forbidden shapes in the finite element method.
IMA Journal of Applied Mathematics, 8(2):260–269.

[Mora et al., 2015] Mora, D., Rivera, G., and Rodríguez, R. (2015).
A virtual element method for the Steklov eigenvalue problem.
Mathematical Models and Methods in Applied Sciences, 25(08):1421–1445.

[Moriguchi and Sugihara, 2006] Moriguchi, M. and Sugihara, K. (2006).
A new initialization method for constructing centroidal voronoi tessellations on surface meshes.
In *2006 3rd International Symposium on Voronoi Diagrams in Science and Engineering*, pages 159–165. IEEE.

[Munkres, 2000] Munkres, J. R. (2000).
Topology.

[Myles et al., 2010] Myles, A., Pietroni, N., Kovacs, D., and Zorin, D. (2010).
Feature-aligned t-meshes.
ACM Transactions on Graphics (TOG), 29(4):1–11.

[Nakahashi, 1987] Nakahashi, K. (1987).
Viscous flow computations using a composite grid.
In *8th Computational Fluid Dynamics Conference*, page 1128.

[Nakamura, 1982] Nakamura, S. (1982).
Marching grid generation using parabolic partial differential equations.
In *Numerical grid generation; Symposium on Numerical Generation of Curvilinear Coordinate Systems and Their Use in the Numerical Solution of Partial Differential Equations.*

[Natarajan et al., 2015] Natarajan, S., Bordas, P. A., and Ooi, E. T. (2015).
Virtual and smoothed finite elements: a connection and its application to polygonal/polyhedral finite element methods.
International Journal on Numerical Methods in Engineering, 104(13):1173–1199.

[Nieser et al., 2011] Nieser, M., Reitebuch, U., and Polthier, K. (2011).
Cubecover–parameterization of 3d volumes.
In *Computer graphics forum*, volume 30-5, pages 1397–1406. Wiley Online Library.

[Oddy et al., 1988] Oddy, A., Goldak, J., McDill, M., and Bibby, M. (1988).
A distortion metric for isoparametric finite elements.
Transactions of the Canadian Society for Mechanical Engineering, 12(4):213–217.

[Owen, 1998] Owen, S. J. (1998).
A survey of unstructured mesh generation technology.
IMR, 239:267.

[Palmer et al., 2020] Palmer, D., Bommes, D., and Solomon, J. (2020).
Algebraic representations for volumetric frame fields.
ACM Transactions on Graphics (TOG), 39(2):1–17.

[Panfili and Cominelli, 2014] Panfili, P. and Cominelli, A. (2014).
Simulation of miscible gas injection in a fractured carbonate reservoir using an embedded discrete fracture model.
In *Abu Dhabi International Petroleum Exhibition and Conference*. OnePetro.

[Park and Washam, 1979] Park, S. and Washam, C. (1979).
Drag method as a finite element mesh generation scheme.
Computers & Structures, 10(1-2):343–346.

[Paulino and Gain, 2015] Paulino, G. H. and Gain, A. L. (2015).
Bridging art and engineering using Escher-based virtual elements.
Structures and Multidisciplinary Optimization, 51(4):867–883.

[Perugia et al., 2016] Perugia, I., Pietra, P., and Russo, A. (2016).
A plane wave virtual element method for the Helmholtz problem.
ESAIM: Mathematical Modelling and Numerical Analysis, 50(3):783–808.

[Pietroni et al., 2022] Pietroni, N., Campen, M., Sheffer, A., Cherchi, G., Bommes, D., Gao, X., Scateni, R., Ledoux, F., Remacle, J.-F., and Livesu, M. (2022).
Hex-mesh generation and processing: a survey.
arXiv preprint arXiv:2202.12670.

[Preparata and Shamos, 1985] Preparata, F. P. and Shamos, M. I. (1985).
Computational Geometry: An Introduction.
Springer-Verlag, Berlin, Heidelberg.

[Preparata and Shamos, 2012] Preparata, F. P. and Shamos, M. I. (2012).
Computational geometry: an introduction.
Springer Science & Business Media.

[Quinn et al., 2012] Quinn, J., Sun, F., Langbein, F. C., Lai, Y.-K., Wang, W., and Martin, R. R. (2012).
Improved initialisation for centroidal voronoi tessellation and optimal delaunay triangulation.
Computer-Aided Design, 44(11):1062–1071.

[Ray et al., 2009] Ray, N., Vallet, B., Alonso, L., and Levy, B. (2009).
Geometry-aware direction field processing.
ACM Transactions on Graphics (TOG), 29(1):1–11.

[Rebay, 1993] Rebay, S. (1993).
Efficient unstructured mesh generation by means of delaunay triangulation and bowyer-watson algorithm.
Journal of computational physics, 106(1):125–138.

[Richardson, 1922] Richardson, L. F. (1922).
Weather prediction by numerical process.
Cambridge university press.

[Richeson, 2012] Richeson, D. S. (2012).
Euler's gem.
In *Euler's Gem*. Princeton University Press.

[Rjasanow and Weiser, 2012] Rjasanow, S. and Weiser, S. (2012).
Higher order bem-based fem on polygonal meshes.
SIAM Journal on Numerical Analysis, 50(5):2357–2378.

[Roca et al., 2011] Roca, X., Gargallo-Peiró, A., and Sarrate, J. (2011).
Defining quality measures for high-order planar triangles and curved mesh generation.
In *Proceedings of the 20th International Meshing Roundtable*, pages 365–383. Springer.

[Ruppert, 1993] Ruppert, J. (1993).
A new and simple algorithm for quality 2-dimensional mesh generation.
In *SODA*, volume 93, pages 83–92.

[Rycroft, 2009] Rycroft, C. (2009).
Voro++: A three-dimensional Voronoi cell library in C++.
Technical report, Lawrence Berkeley National Lab.(LBNL), Berkeley, CA (United States).

[Salagame and Belegundu, 1994] Salagame, R. R. and Belegundu, A. D. (1994).
Distortion, degeneracy and rezoning in finite elements—a survey.
Sadhana, 19(2):311–335.

[Sander et al., 2003] Sander, P. V., Wood, Z. J., Gortler, S., Snyder, J., and Hoppe, H. (2003).
Multi-chart geometry images.
ACM Symposium on Geometry Processing.

[Sastry and Shontz, 2009] Sastry, S. P. and Shontz, S. M. (2009).

A comparison of gradient-and hessian-based optimization methods for tetrahedral mesh quality improvement.
In *Proceedings of the 18th International Meshing Roundtable*, pages 631–648. Springer.

[Schneider et al., 2019] Schneider, T., Dumas, J., Gao, X., Botsch, M., Panozzo, D., and Zorin, D. (2019).
Poly-spline finite-element method.
ACM Transactions on Graphics (TOG), 38(3):1–16.

[Schneider et al., 2022] Schneider, T., Hu, Y., Gao, X., Dumas, J., Zorin, D., and Panozzo, D. (2022).
A large-scale comparison of tetrahedral and hexahedral elements for solving elliptic pdes with the finite element method.
ACM Trans. Graph., 41(3).

[Scott and Brenner, 2008] Scott, L. R. and Brenner, S. C. (2008).
The mathematical theory of finite element methods.
Texts in applied mathematics 15. Springer-Verlag, New York, 3 edition.

[Shamos and Hoey, 1976] Shamos, M. I. and Hoey, D. (1976).
Geometric intersection problems.
In *17th Annual Symposium on Foundations of Computer Science (sfcs 1976)*, pages 208–215.

[Shephard and Georges, 1991] Shephard, M. S. and Georges, M. K. (1991).
Automatic three-dimensional mesh generation by the finite octree technique.
International Journal for Numerical methods in engineering, 32(4):709–749.

[Shepherd and Johnson, 2008] Shepherd, J. F. and Johnson, C. R. (2008).
Hexahedral mesh generation constraints.
Engineering with Computers, 24(3):195–213.

[Shepherd et al., 2000] Shepherd, J. F., Mitchell, S. A., Knupp, P., and White, D. R. (2000).
Methods for multisweep automation.
Technical report, Sandia National Labs., Albuquerque, NM, and Livermore, CA (US).

[Shewchuk, 1996] Shewchuk, J. R. (1996).
Triangle: Engineering a 2d quality mesh generator and delaunay triangulator.
In *Workshop on applied computational geometry*, pages 203–222. Springer.

[Shewchuk, 1997] Shewchuk, J. R. (1997).
Adaptive precision floating-point arithmetic and fast robust geometric predicates.
Discrete & Computational Geometry, 18(3):305–363.

[Shewchuk, 2005] Shewchuk, J. R. (2005).
Triangle library.
https://www.cs.cmu.edu/quake/triangle.html.

[Si, 2015] Si, H. (2015).
TetGen, a Delaunay-based quality tetrahedral mesh generator.
ACM Transactions on Mathematical Software (TOMS), 41(2):1–36.

[Sokolov et al., 2016] Sokolov, D., Ray, N., Untereiner, L., and Lévy, B. (2016).
Hexahedral-dominant meshing.
ACM Transactions on Graphics (TOG), 35(5):1–23.

[Sorgente et al., 2018] Sorgente, T., Biasotti, S., Livesu, M., and Spagnuolo, M. (2018).
Topology-driven shape chartification.
Computer Aided Geometric Design, 65:13–28.

[Sorgente et al., 2022a] Sorgente, T., Biasotti, S., Manzini, G., and Spagnuolo, M. (2022a).
Polyhedral mesh quality indicator for the virtual element method.
Computers & Mathematics with Applications, 114:151–160.

[Sorgente et al., 2022b] Sorgente, T., Biasotti, S., Manzini, G., and Spagnuolo, M. (2022b).
The role of mesh quality and mesh quality indicators in the virtual element method.
Advances in Computational Mathematics, 48(1):1–34.

[Sorgente et al., 2021a] Sorgente, T., Biasotti, S., and Spagnuolo, M. (2021a).
A Geometric Approach for Computing the Kernel of a Polyhedron.
In Frosini, P., Giorgi, D., Melzi, S., and Rodolà, E., editors, *Smart Tools and Apps for Graphics - Eurographics Italian Chapter Conference*, pages 11–19, online. The Eurographics Association.

[Sorgente et al., 2022c] Sorgente, T., Biasotti, S., and Spagnuolo, M. (2022c).
Polyhedron kernel computation using a geometric approach.
Computers & Graphics, 105:94–104.

[Sorgente et al., 2021b] Sorgente, T., Prada, D., Cabiddu, D., Biasotti, S., Patane, G., Pennacchio, M., Bertoluzza, S., Manzini, G., and Spagnuolo, M. (2021b).
VEM and the Mesh, volume 31 of *SEMA SIMAI Springer series*, chapter 1, pages 1–54.
Springer.
ISBN: 978-3-030-95318-8.

[Staten et al., 2010] Staten, M. L., Kerr, R. A., Owen, S. J., Blacker, T. D., Stupazzini, M., and Shimada, K. (2010).
Unconstrained plastering-hexahedral mesh generation via advancing-front geometry decomposition.
International journal for numerical methods in engineering, 81(2):135–171.

[Stimpson et al., 2007] Stimpson, C., Ernst, C., Knupp, P., Pébay, P., and Thompson, D. (2007).
The verdict library reference manual.

Sandia National Laboratories Technical Report, 9(6).

[Stojmenovic, 1997] Stojmenovic, I. (1997).
Honeycomb networks: Topological properties and communication algorithms.
IEEE Transactions on parallel and distributed systems, 8(10):1036–1042.

[Sukumar and Tabarraei, 2004] Sukumar, N. and Tabarraei, A. (2004).
Conforming polygonal finite elements.
International Journal for Numerical Methods in Engineering, 61(12):2045–2066.

[Tadepalli et al., 2011] Tadepalli, S. C., Erdemir, A., and Cavanagh, P. R. (2011).
Comparison of hexahedral and tetrahedral elements in finite element analysis of the foot and footwear.
Journal of biomechanics, 44(12):2337–2343.

[Taha and Hanbury, 2015] Taha, A. A. and Hanbury, A. (2015).
An efficient algorithm for calculating the exact hausdorff distance.
IEEE transactions on pattern analysis and machine intelligence, 37(11):2153–2163.

[Tarini et al., 2004] Tarini, M., Hormann, K., Cignoni, P., and Montani, C. (2004).
Polycube-maps.
ACM transactions on graphics (TOG), 23(3):853–860.

[Tarini et al., 2010] Tarini, M., Pietroni, N., Cignoni, P., Panozzo, D., and Puppo, E. (2010).
Practical quad mesh simplification.
In *Computer Graphics Forum*, volume 29-2, pages 407–418. Wiley Online Library.

[Tautges, 2004] Tautges, T. J. (2004).
Moab-sd: integrated structured and unstructured mesh representation.
Engineering With Computers, 20(3):286–293.

[Thompson, 1982] Thompson, J. F. (1982).
Elliptic grid generation.
Applied Mathematics and Computation, 10:79–105.

[Thompson et al., 1998] Thompson, J. F., Soni, B. K., and Weatherill, N. P. (1998).
Handbook of grid generation.
CRC press.

[Thompson et al., 1985] Thompson, J. F., Warsi, Z. U., and Mastin, C. W. (1985).
Numerical grid generation: foundations and applications.
Elsevier North-Holland, Inc.

[Tong et al., 2006] Tong, Y., Alliez, P., Cohen-Steiner, D., and Desbrun, M. (2006).
Designing quadrangulations with discrete harmonic forms.
In *Eurographics symposium on geometry processing.*

[Vartziotis et al., 2008] Vartziotis, D., Athanasiadis, T., Goudas, I., and Wipper, J. (2008).
Mesh smoothing using the geometric element transformation method.
Computer Methods in Applied Mechanics and Engineering, 197(45-48):3760–3767.

[Vartziotis and Wipper, 2010] Vartziotis, D. and Wipper, J. (2010).
Characteristic parameter sets and limits of circulant hermitian polygon transformations.
Linear algebra and its applications, 433(5):945–955.

[Vartziotis and Wipper, 2012] Vartziotis, D. and Wipper, J. (2012).
Fast smoothing of mixed volume meshes based on the effective geometric element transformation method.
Computer methods in applied mechanics and engineering, 201:65–81.

[Viertel et al., 2016] Viertel, R., Staten, M. L., and Ledoux, F. (2016).
Analysis of non-meshable automatically generated frame fields.
Technical report, Sandia National Lab.(SNL-NM), Albuquerque, NM (United States).

[Wang and Ye, 2013] Wang, J. and Ye, X. (2013).
A weak galerkin finite element method for second-order elliptic problems.
Journal of Computational and Applied Mathematics, 241:103–115.

[Wang, 2017] Wang, L. (2017).
Algorithms and Criteria for Volumetric Centroidal Voronoi Tessellations.
PhD thesis, Université Grenoble Alpes.

[Wang et al., 2016] Wang, L., Hétroy-Wheeler, F., and Boyer, E. (2016).
A hierarchical approach for regular centroidal voronoi tessellations.
In *Computer Graphics Forum*, volume 35-1, pages 152–165. Wiley Online Library.

[Warsi and Thompson, 1990] Warsi, Z. and Thompson, J. (1990).
Application of variational methods in the fixed and adaptive grid generation.
Computers & Mathematics with Applications, 19(8-9):31–41.

[Watson, 1981] Watson, D. F. (1981).
Computing the n-dimensional delaunay tessellation with application to voronoi polytopes.
The computer journal, 24(2):167–172.

[Weatherill, 1988a] Weatherill, N. P. (1988a).
A method for generating irregular computational grids in multiply connected planar domains.
International Journal for Numerical Methods in Fluids, 8(2):181–197.

[Weatherill, 1988b] Weatherill, N. P. (1988b).
On the combination of structured-unstructured meshes.
Numerical grid generation in computational fluid mechanics'88, pages 729–739.

[Weatherill and Hassan, 1994] Weatherill, N. P. and Hassan, O. (1994).

Efficient three-dimensional delaunay triangulation with automatic point creation and imposed boundary constraints.
International journal for numerical methods in engineering, 37(12):2005–2039.

[Whitehead, 1949] Whitehead, J. H. C. (1949).
Combinatorial homotopy. ii.
Bulletin of the American Mathematical Society, 55(5):453–496.

[Wriggers et al., 2016] Wriggers, P., Rust, W. T., and Reddy, B. D. (2016).
A virtual element method for contact.
Computational Mechanics, 58(6):1039–1050.

[Xu and Newman, 2006] Xu, H. and Newman, T. S. (2006).
An angle-based optimization approach for 2d finite element mesh smoothing.
Finite Elements in Analysis and Design, 42(13):1150–1164.

[Yan et al., 2013] Yan, D.-M., Wang, W., Lévy, B., and Liu, Y. (2013).
Efficient computation of clipped voronoi diagram for mesh generation.
Computer-Aided Design, 45(4):843–852.

[Yerry and Shephard, 1984] Yerry, M. A. and Shephard, M. S. (1984).
Automatic three-dimensional mesh generation by the modified-octree technique.
International journal for numerical methods in engineering, 20(11):1965–1990.

[Yi and Moon, 2012] Yi, F. and Moon, I. (2012).
Image segmentation: A survey of graph-cut methods.
In *2012 international conference on systems and informatics (ICSAI2012)*, pages 1936–1941. IEEE.

[Zhang et al., 2010] Zhang, M., Huang, J., Liu, X., and Bao, H. (2010).
A wave-based anisotropic quadrangulation method.
In *ACM SIGGRAPH 2010 papers*, pages 1–8. -.

[Zhou and Jacobson, 2016] Zhou, Q. and Jacobson, A. (2016).
Thingi10k: A dataset of 10,000 3D-printing models.

[Zlámal, 1968] Zlámal, M. (1968).
On the finite element method.
Numerische Mathematik, 12(5):394–409.

[Zunic and Rosin, 2004] Zunic, J. and Rosin, P. L. (2004).
A new convexity measure for polygons.
IEEE Transactions on Pattern Analysis and Machine Intelligence, 26(7):923–934.