
**Indirect generation of quadrilateral and
hexahedral-dominant meshes: a hybrid approach based
on cross-field guidance, topological labelling and patterns.**

Christos Georgiadis

Thesis submitted in partial fulfillment of the requirements for the Ph.D
Degree in Engineering Sciences

THESIS COMMITTEE		
Advisor	Jean-François Remacle	<i>UCLouvain</i>
Jury	Vincent Legat	<i>UCLouvain</i>
	Jonathan Lambrechts	<i>UCLouvain</i>
	Koen Hillewaert	<i>ULiege</i>
	Cecil Armstrong	<i>QUB</i>
	Franck Ledoux	<i>CEA</i>
	Gaëtan Compère	<i>Siemens</i>
Chairman	Thomas Pardoen	<i>UCLouvain</i>

Louvain-la-Neuve, May 2021

Acknowledgements

Throughout the writing of this dissertation I have received a great deal of support and assistance. I would like to express my gratitude to all the people who accompanied me all along this long journey.

First of all, I would like to express my sincere gratitude to Prof. Jean-François Remacle for his continuous support and guidance during these years. I could not imagine a better advisor and mentor. His motivation and enthusiasm have made my Ph.D. experience truly enjoyable and kept me inspired throughout these years.

I would also like to acknowledge the invaluable help of Dr. Jonathan Lambrechts, who was always present to help me with my research. I would further like to acknowledge the other members of my jury, Prof. Vincent Legat, Prof. Koen Hillewaert, Prof. Cecil Armstrong, Dr. Franck Ledoux, Dr. Gaëtan Compère and the chairman, Prof. Thomas Pardoën, for their attention on reading my thesis and the valuable insights and comments during my defense.

I truly enjoyed my time as a researcher, thanks to all the people in the Euler building that made the working environment so pleasant and interesting. I am very grateful for being a member of the Hextreme group and I would like to thank all its members, my friends and colleagues: Jovana, Célestin, Kilian, Pierre-Alexandre, Jeanne, Amaury, Ruili, Arthur, and Ruiyang. I would like to especially thank Alexandre for his valuable contributions to my research and my office mate Maxence for his contributions, sharing his knowledge, and all the interesting discussions we had.

The support of my friends and family was of utmost importance for the completion of this thesis. I am extremely grateful for having friends that made my life happier. Last but not least, I want to express my heartfelt gratitude to my parents. It would not be possible for me to achieve anything without their endless support and encouragement.

Christos Georgiadis

Declaration

The candidate confirms that the work submitted is his own, except where work which has formed part of jointly authored publications has been included. The contribution of the candidate and the other authors to this work has been explicitly indicated below. The candidate confirms that appropriate credit has been given within the thesis where reference has been made to the work of others.

The core work of this thesis is in the form of independent research articles, presented in Chapters 3,4,5, and 6, which include joint work. In Chapter 6, Maxence Reberol and my supervisor Jean-Francois Remacle are the main contributors of 2D scaled cross fields and the cavity remeshing strategy. In Chapter 7, Alexandre Chemin has developed the method for 3D scaled frame fields.

Christos Georgiadis

Contents

Acknowledgements	i
Declaration	iii
Contents	v
1 Introduction	1
2 Related work	7
2.1 Surface meshing	7
2.2 Quadrilateral meshing	9
2.3 Hexahedral meshing	14
3 In a nutshell	19
3.1 Cross-fields and size-fields	21
3.2 Generating points	23
3.3 Obtain a triangular/tetrahedral mesh	24
3.4 Combining triangles/tetrahedra into quadrilaterals/hexahedra .	25
3.5 Post-process	26
4 Cross-field guided frontal point generation.	29
4.1 Introduction	30
4.2 Frontal point insertion	32
4.3 Recombination to quadrilaterals	37
4.4 Results	39
4.5 Conclusions	42
5 Indirect quadrilateral meshing using bipartite labelling.	45
5.1 Introduction	46
5.2 Overview	50
5.3 Point Generation	53
5.4 Surface Meshing	54
5.5 Bipartite quadrilateral labeling	57

5.6	Results	64
5.7	Conclusions	66
6	Quasi-structured quadrilateral meshing.	69
6.1	Introduction	70
6.2	Context and motivation	77
6.3	Scaled cross field computation	80
6.4	Size map and curve quantization	88
6.5	Unstructured frontal quadrilateral mesh generation	92
6.6	Topological quadrilateral meshing	97
6.7	Results and discussions	107
6.8	Perspectives and future work	112
7	Hexahedral-dominant meshing.	117
7.1	Introduction	118
7.2	Frame field integrability equations	120
7.3	Point generation for hex-dominant mesh generation	126
7.4	Results	129
7.5	Conclusion and perspectives	133
8	Conclusion	135
8.1	Perspectives	137
	Bibliography	143

CHAPTER



Introduction

Mesh generation is an essential step of the majority of engineering design methods. Most of the common numerical methods use a mesh to describe the underlying geometry of the model. Furthermore, the success and accuracy of a simulation are usually highly dependant on the mesh quality. The conceptual motif of most numerical methods is to divide a complex problem into a number of simple sub-problems. Three general steps are necessary: i) pre-processing, where the geometry is created and meshed (i.e., the model is subdivided into simple shapes), ii) solver, where the solution is approximated with appropriate formulation on mesh elements iii) post-processing, where the results are analyzed and visualized. The fundamental objective of mesh generation is to provide a discrete version of a continuous domain. Mesh elements are local approximations of a continuous geometric representation.

We can outline the historical development of mesh generation in conjunction

with the development of numerical methods. Finite Difference Methods relied upon structured meshes (also referred to as grids). Advancements of the theoretical and mathematical foundation, along with progress in digital computing, gave birth to more sophisticated methods such as the Finite Element Method and the Finite Volume Method. The underlying principles of these approaches could accommodate more complex representations. At the same time, there were advancements in Computer-Aided Design systems that could accommodate the creation of more complicated geometric models. Even though we can trace the origins of CAD and the necessity of numerical methods for meshes at the same period, there was little communication between the two fields. The growth of numerical methods and CAD systems that could handle more complicated models gave birth to increased demand for more complicated meshes, and led to mesh generation forming an autonomous research topic. It evolved to be a highly interdisciplinary field, lying on the intersection between computational geometry, topology, engineering, and computer science.

Mesh generation is not a trivial task. It is commonly reported as being one of the most significant bottleneck of workflows involving numerical methods. An interesting report on the process of engineering analysis (that can be found in [Boggs et al. \[2021\]](#)) states that the pre-processing step amounts up to 80% of time, while the mesh-related tasks of geometry decomposition (a necessary step when full automation is not possible), meshing, and mesh post-process accounting to around half of the total time. These numbers agree with the commonly reported 80% of time to create the model and generate a mesh. Moreover, according to the same report, the meshing is the step most frequently revisited for correction/improvements.

We focus here on two of the main reasons for the difficulties of mesh generation. On the one hand, computing capabilities have come a long way; it is common nowadays for engineers and scientists to perform simulations on meshes with hundreds of millions of elements. Even though mesh generation technology improved, the overall time consumed remains significant due to bigger and more complicated models. Methods for generating triangular/tetrahedral meshes are considered mature and robust algorithms exist to generate them. At the same time, automatic quadrilateral and especially hexahedral meshing are infantile in comparison, posing several difficulties related to their topologically constrained nature and the geometric "stiffness" of quad/hex elements. Even though we can acknowledge a variety of solutions for quadrilateral meshing, there is not up to date a fully automatic algorithm for the generation of well-aligned and high-quality hexahedral meshes. On the other hand, besides the inherent difficulties of meshing, another part of the problem comes from CAD interoperability. There is a diverse variety of CAD representations based on different geometry kernels, making communication between them difficult. CAD models may consist of thousands of patches (often with gaps and overlaps be-

tween them) and may have excessive detail that is redundant for simulation purposes. Thus, to become suitable inputs for a mesh generator CAD models usually need extra pre-processing for geometry clean-up, removal of unnecessary features, fixing the topology, and so on. Simply put, there is a gap between CAD representations and mesh generation requirements. It is interesting to note that avoiding the peculiarities of meshing is the main impetus for researchers to develop methods that bypass this step. Notable examples include Isogeometric Analysis, where the numerical method is formulated directly on the CAD representation [Hughes et al., 2005], and the Immersed Boundary methods for Computational Fluid Dynamics, which uses Cartesian grids that are non-conformal to geometry [Peskin, 1977].

As stated, the major open problems in mesh generation are related to quadrilateral and hexahedral meshing. It is not within the scope of this work to revisit the long-standing debate of triangles (tetrahedra) versus quadrilaterals (hexahedra) [Benzley et al., 1995; D’Azevedo, 2000]. Triangular/tetrahedral meshes are the common standard. We have concise methods to generate them, they allow geometric flexibility since they are easier to adapt to geometric features, and also it is more straightforward to adapt them for non-uniform size-fields or anisotropic metrics. On the other hand, quadrilateral/hexahedral meshes (colloquially also referred to as quad/hex meshes) are more challenging to generate but prove to be advantageous in certain aspects. They have fewer elements for the same number of vertices, they are capable of providing a block-structure and they can provide better alignment to geometric features, as well as to problem-specific features, providing better numerical behavior for specific physical phenomena (a typical example is the demand for structured and aligned boundary layers in Computational Fluid Dynamics).

Besides engineering analysis, the other field that has been traditionally invested in mesh generation is Computer Graphics, mainly focused on design, animation, graphic art and visualization. Even though the context and objectives of the two communities differ in certain aspects, the exchange of ideas between them has been proven highly beneficial recently. Useful lines of research emanating from Computer Graphics found their way in engineering applications and vice versa, most notably the use of cross-fields to assist optimal element creation.

Motivation and objective of this work The general objective of this work is to develop methods for the generation of quadrilateral and hexahedral-dominant meshes. We build upon classic engineering techniques of mesh generation, and we enhance them with contributions from computer graphics and graph theory. We thus follow a hybrid approach, aiming to incorporate the best elements from each field.

Traditional fully-automatic approaches on quad/hex meshing usually have a

trade-off between robustness, output quality and efficiency. For example, grid-based methods, where the domain is superposed on a structured grid, are able to produce all-quad and all-hex meshes but come at the cost of degraded boundary elements. Advancing-front approaches are more flexible, but they pose difficulty in terminating with success (robustly ‘closing’ opposite fronts with high-quality elements). On the other hand, cross-field based global parameterization methods (a line of research stemming from the computer graphics community) have recently gained attention as a promising approach for automated quad/hex meshing [Bommes et al., 2013b]. These methods produce high-quality meshes with underlying structure (coarse quad layouts), but are computationally intensive due to integer formulations and dependent on cross-fields/frame-fields that must have a topological structure that is quad/hex-meshable, which is not always feasible and especially for hex-meshing is an open problem.

We build our method based on a classic approach by generating points in an advancing front manner, triangulating them, and obtaining a quad/hex mesh by indirect combination of triangles to quadrilaterals and tetrahedra to hexahedra. We enhance the method with cross-field (and frame-field for 3D) guidance for point insertion. We use these direction fields as an auxiliary tool and without relying upon them, since they can have defects or not being able to accommodate desired sizing constraints. Furthermore, we enrich the geometric propagation of points inside the domain with a bipartite topological labeling scheme (Section 5.5) that enables us to obtain quad meshes directly, and possibly reduce the complexity of the problem of hex-meshing.

We aim for high efficiency; thus we avoid ‘heavy’ global strategies, for example, mixed-integer formulations of global parameterization methods, the L_p Central Voronoi Tessellation of Lévy and Liu [2010] for point sampling, or the indirect triangle-pairing of BlossomQuad algorithm [Remacle et al., 2012]). Instead, we develop methods relying on local operations. For example, we locally sample the 2D cross-fields and 3D frame-fields with advancing-front propagation. Our frontal approach still benefits from global information contained in the cross-field, i.e., that it is smooth and contains singularities, but only in an indirect way, so our approach works even if the guiding fields are imperfect. In the same spirit, we present a straightforward algorithm for the indirect conversion of a triangular to an all-quad mesh by using bipartite topological labeling which enforces a global topological property for quad meshes (even-sided boundary) and fixes inconsistencies with local operations.

One of our main interests is to be able to produce meshes on any input, regardless of its complexity. Reviewing the relevant literature, we observe that the majority of the works present results on relatively simple models. We work on large model databases (that are recently becoming the common standard for validation purposes on bulk numbers of models [Koch et al., 2019; Ledoux; Zhou and Jacobson, 2016]), and we aim to have a close to 100% success rate

on them. To be generic, we develop methods that can have as input only a triangular mesh of the model, avoiding the dependency on the underlying CAD representation. In Figure 1.1 we present results of meshes obtained with our methods on the same model. We remesh the input triangulated surface to a triangular mesh with right-angled triangles aligned with the input metric (cross-field with size-field), convert it to a quadrilateral mesh, optimize it to a quasi-structured one, and finally obtain a hex-dominant mesh of the volume.

The present manuscript is organized in the following way. In Chapter 2 we provide an overview of the literature relevant to our goals: i) surface meshing, ii) quadrilateral meshing, and iii) hexahedral meshing. In Chapter 3 we present an outline of the design goals and the methodology followed, and we bind together the various components of the pipeline in a concise manner. The core work of this thesis is developed on four independent research works. The corresponding chapters are presented in sequential order, chronologically and logically:

- Chapter 4: *Cross-field guided frontal point generation for quadrilateral meshing*. We implement an advancing-front method for the optimal placement of points across the domain.
- Chapter 5: *Indirect quadrilateral meshing using bipartite labeling*. First, we develop a robust surface meshing algorithm. Second, we present a novel approach to convert it efficiently to a high-quality all-quad mesh.
- Chapter 6: *Quasi-structured quadrilateral meshing*. A strategy for the topological and geometric optimization of quad meshes based on local remeshing, that is used as a component of an end-to-end quad meshing pipeline.
- Chapter 7: *Hexahedral-dominant meshing using 3D integrable frame-fields*. We develop a pipeline for the generation of hexahedral-dominant meshes guided by 3D frame-fields. We investigate the integrability condition for 3D frame-fields and we obtain promising results using scaled frame-fields.

A lot of the implementation details are strictly technical, and there would be no point in detailing every process. The work has been implemented in the HXT library of the Hextreme project [Hextreme, 2021] and has been incorporated as an external library to GMSH [Geuzaine and Remacle, 2009], with all the codes being open source.

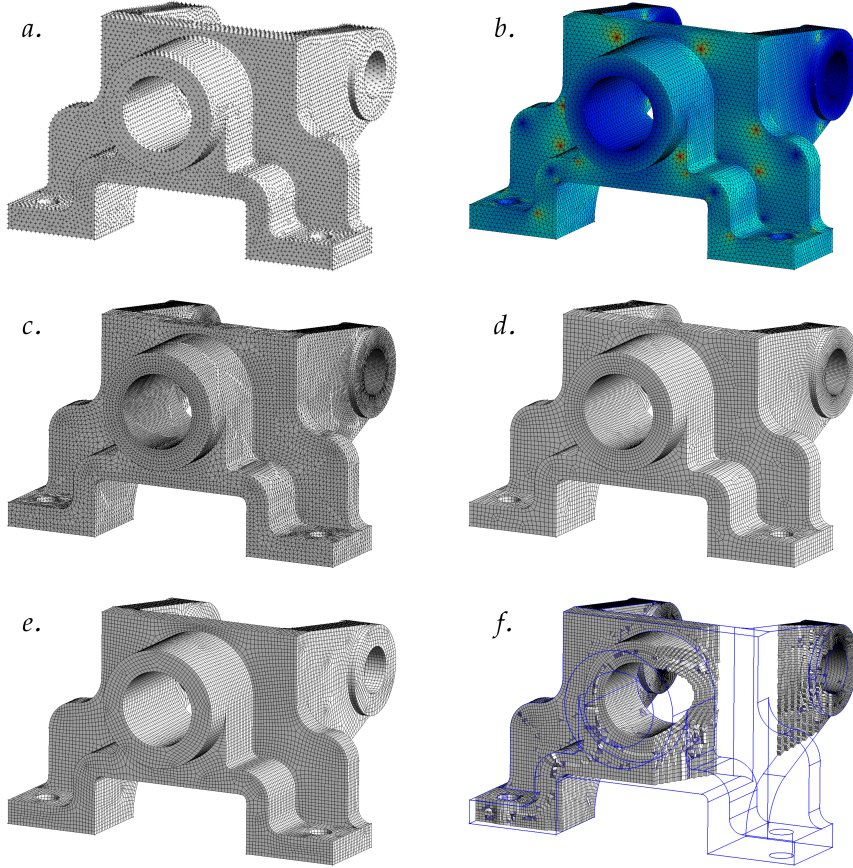


Figure 1.1: Overview of the results of this work: a) Cross-field and b) size-field based on conformal scaling, computed on the input mesh. c) Triangular mesh with cross-field aligned right-angled triangles (created with the surface meshing algorithm of Chapter 5). d) Quadrilateral mesh by converting the triangular mesh using bipartite topological labelling (Chapter 5). e) Optimized quasi-structured quadrilateral mesh based on a local remeshing strategy (Chapter 6). f) Hexahedral-dominant mesh using 3D integrable frame fields and indirect combination of tetrahedra to hexahedra (Chapter 7) .

CHAPTER



Related work

A vast amount of relevant bibliography exists on the field of Mesh Generation. Here, we will present an outline of relevant research works, where we mostly focus on unstructured mesh generation. Concise and more detailed presentations on mesh generation can be found on relevant books [[Thompson et al., 1999](#); [Frey and George, 2008](#); [Lo, 2015](#)] and survey papers [[Owen, 1998](#); [Sheffer et al., 2006](#); [George et al., 2007](#); [Bommes et al., 2013b](#)].

2.1 Surface meshing

Surface meshing methods can be broadly classified into two main categories [[Borouchaki et al., 2000](#); [Lo, 2015](#)]: i) Parametric (or indirect) approaches and ii) Direct approaches (also referred as mesh adaptation or mesh modification strategies, especially in the context of remeshing surfaces)

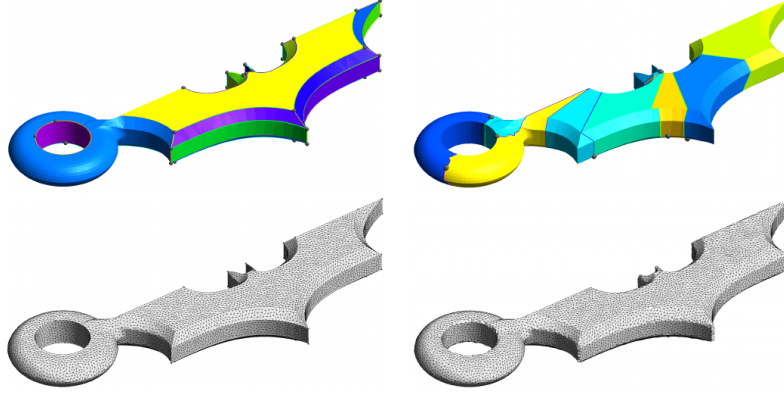


Figure 2.1: Surface remeshing based on reparametrization (figure from [Beaufort et al. \[2020\]](#)).

In *parametric approaches*, the 3D surface is mapped to a 2D parametric space [[Lo, 1988](#); [Sheng and Hirsch, 1992](#); [Cuillière, 1998](#); [Lee and Hobbs, 1998](#); [Shimada and Gossard, 1998](#); [Borouchaki et al., 2000](#); [Floater and Hormann, 2005](#)]. Since the CAD surfaces (typically NURBS patches) have underlying u, v representation, it can be efficient to generate a mesh in 2D with standard meshing techniques and after map it back to the 3D space. While the meshes produced with this method are of high quality for smooth surfaces, quality degrades with increasing model complexity, due to distortions in the underlying mapping. A strategy to handle complex models is to subdivide into a union of more simple patches [[Lo, 1988](#); [Beaufort et al., 2020](#)]. In that case, conformity between patches has to be ensured. On the other hand, manipulation of sub-surfaces this way forbids global strategies, for example, aligning with a cross-field to indirectly generate quadrilateral meshes. Parametrization techniques can also be used to remesh triangulations [[Remacle et al., 2010](#); [Beaufort et al., 2020](#)].

Direct approaches generate elements directly on the surface. They can be based on quad-trees [[Shephard and Georges, 1991](#); [Frey and Marechal, 1998](#)], advancing front [[Löhner and Parikh, 1988](#); [Lau and Lo, 1996](#)] or Delaunay strategies [[Baker, 1989](#); [Borouchaki et al., 1997](#)]. One of the main advantages of the direct approach is the usefulness for models where an underlying parametrization is not available or when it is degraded. Local mesh modifications strategies to remesh models described by STL triangulations are proposed in [Wang et al. \[2006\]](#); [Béchet et al. \[2002\]](#). One of the main difficulties of this class of methods is that the manipulation of geometry directly on the 3D space is a challenging task that may lead to geometric or topologic ambiguities.

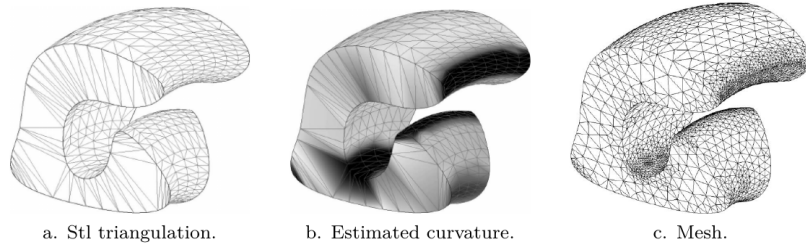


Figure 2.2: Local mesh modification remeshing of STL input (figure adapted from [Béchet et al. \[2002\]](#)).

2.2 Quadrilateral meshing

Automated quadrilateral mesh generation is considered one of the most difficult problems in mesh generation. Even though a diversity of methods exist [[Owen, 1998](#); [Schneiders, 2000a](#); [Bommes et al., 2013b](#)], that are based either on strong mathematical foundations or to more heuristic or semi-automatic approaches, there is not up to date a definite algorithmic method that can encompass the variety of design goals. A concise presentation of algorithmic characteristics and design goals of quad meshing (as well as hexahedral/hex-dominant meshing) can be found in [Sabin \[1991\]](#) and [Johnen \[2016\]](#).

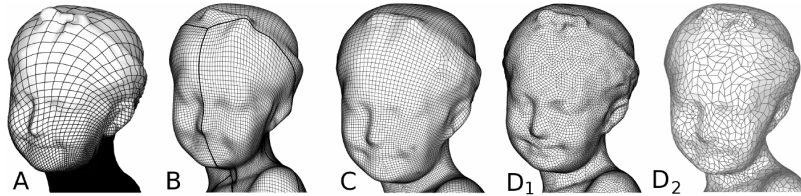


Figure 2.3: Classification of quadrilateral meshes. A: regular. B: semi-regular. C: valence semi-regular. D1-D2: unstructured (D2 being more irregular than D1). Figure from [Bommes et al. \[2013b\]](#).

A useful classification of quadrilateral meshes can be based on the underlying structure and regularity of them (Figure 2.3). As regular vertices, we consider those having a *valence* of 4, i.e., they are adjacent to four quadrilaterals. Quadrilateral meshes can therefore be:

- regular (where all vertices are regular)
- semi-regular (a small number of irregular vertices and a block-structure)

- valence semi-regular (small number of irregular vertices)
- unstructured ('relatively' large number of irregular vertices)

where the borders between the last three categories is not clearly defined. For most models, irregular vertices are necessary to accommodate geometric variations, sizing constraints, and the topological characteristics of the model. In the context of this work, we are mainly concerned with the last two categories.

Regarding the methods to generate a quad mesh, the following classification is useful for our purposes:

- *Direct quadrilateral meshing*, where quadrilateral elements are formed directly.
- *Indirect quadrilateral meshing*, where an intermediate triangular mesh is transformed to a quad one.

2.2.1 Direct quadrilateral meshing

Direct quadrilateral meshing methods are commonly based upon *grid based*, *advancing-front* and *domain decomposition* approaches.

Grid-based methods can be based on a Cartesian grid or more commonly on an underlying quad-tree to control element sizing and to fit elements on the domain boundary [Baehmann et al., 1987; Frey and Marechal, 2000; Rushdi et al., 2017; Pochet et al., 2017]. Another method sharing similarities with those is the one of Sarrate and Huerta [2000], where the domain is recursively decomposed in quadrilaterals. Even though they are considered a robust way to generate quad meshes (the same holds for hexahedral ones using oct-trees) grid-based methods cannot naturally offer alignment to geometric characteristics and have low-quality boundary elements.

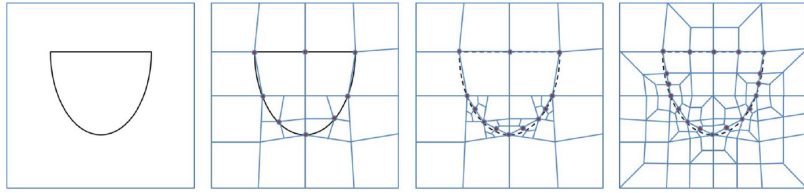


Figure 2.4: Quadtree workflow for quad meshing. The quadtree is refined to accommodate the geometry (figure from Pochet et al. [2017]).

Advancing front methods start from a pre meshed boundary and propagate a *front* of elements towards the interior, starting from a pre meshed boundary. The paving algorithm proposed in [Blacker and Stephenson \[1991\]](#) directly forms rows of quadrilaterals, while in [White and Kinney \[1997\]](#) the method is improved. Advancing front methods require special treatment to handle opposite fronts collisions and are usually complemented with heuristic clean-up operations.

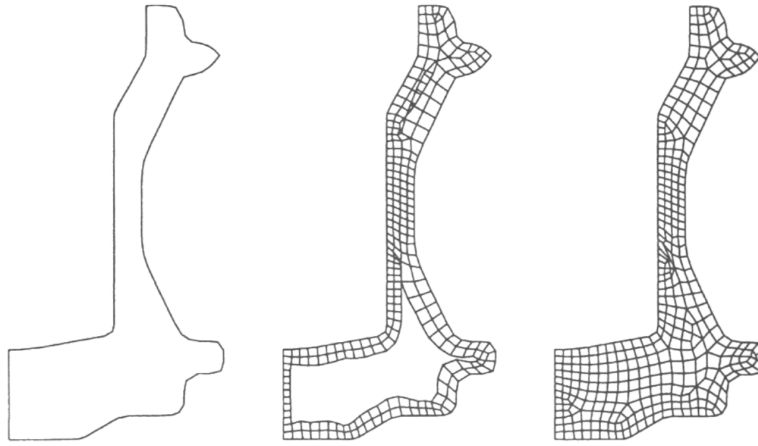


Figure 2.5: Paving method. In the middle we see the intersection between opposing fronts (figure from [Thompson et al. \[1999\]](#)).

Domain decomposition methods aim to decompose the domain into simpler regions that can be quad-meshed in a straightforward fashion. In [Tam and Armstrong \[1991\]](#) the medial axis is used to this end. More modern approaches utilize cross-fields to obtain a quad layout [[Kowalski et al., 2013](#); [Jezdimirovic et al., 2019](#); [Campen, 2017](#)].

Cross-fields have obtained a considerable amount of attention the recent years, offering a powerful tool to enhance the quality of quadrilateral meshes [[Ray et al., 2006](#); [Palacios and Zhang, 2007](#); [Ray et al., 2008](#); [Beaufort et al., 2017b](#)]. A survey on directional fields (which include cross-fields) can be found in [Vaxman et al. \[2016\]](#). An advantage is that they encapsulate topology information, with the singularities of the field corresponding to irregular nodes of quad meshes [[Bunin, 2008](#); [Ray et al., 2008](#)]. They are typically designed to be as smooth as possible, aligned with the principal curvature directions and domain boundaries/feature curves [[Ray et al., 2009](#); [Nieser et al., 2012](#)]. In practice, they define preferred directions which align the elements. To optimally accommodate for the directional variations of the cross-field we can make it curl-free

[Ray et al., 2006] or compute a conformal scaling of it [Ben-Chen et al., 2008]. We can thus obtain a scaled size field matching the prescribed directions. The opposite, i.e., computing a cross-field from a given size-field, is an interesting open problem.

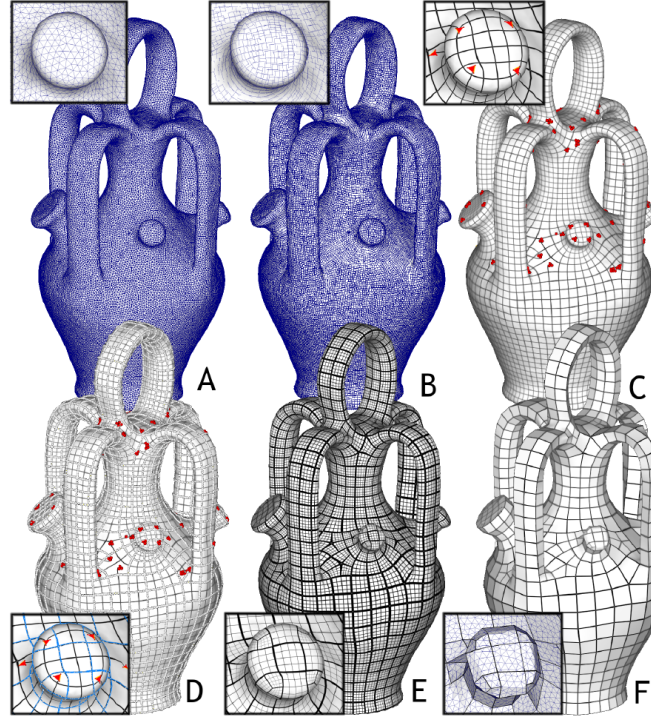


Figure 2.6: Global parameterization based on cross-fields (figure from Ray et al. [2006]).

Global parametrization pipelines are typically based on cross-fields [Bommes et al., 2013b]. It must be noted that since we can decompose cross-field computation from the subsequent quad mesh generation, they can also be utilized by advancing-front approaches [Remacle et al., 2013]. Global parametrization methods try to find a u, v seamless parametrization, whose gradients match the given cross-field, and map from a 2D parameter domain to the model surface [Kälberer et al., 2007; Bommes et al., 2009; Campen et al., 2015]. These methods provide impressive results, and furthermore are capable of producing optimal block-structured (semi-regular) decompositions of the domain. Yet, we are not aware of their industrial-grade applicability. These methods rely upon the topological integrity of the cross-field and utilize computationally demanding integer formulations, which could be a bottleneck for increasing model

complexity.

2.2.2 Indirect quadrilateral meshing

In *indirect quadrilateral meshing*, an intermediate triangular mesh can be transformed to a quadrilateral one, using mid-point subdivision or, more commonly, by merging pairs of triangles (also referred to as quad conversion or triangle-pairing methods).

The former is a straightforward way to obtain a quadrilateral mesh from a triangular mesh by splitting each triangle to three quadrilaterals, commonly known as Catmull-Clark subdivision [Catmull and Clark, 1978]. The same procedure can be used to transform a quad-dominant mesh to an all-quad one. The cost for simplicity is quadrupling the number of elements and diminishing accuracy on the boundary.

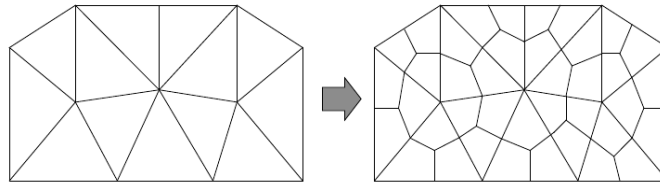


Figure 2.7: Midpoint subdivision to obtain quadrilateral mesh from triangles (figure from Owen [1998]).

A more optimal strategy is to identify suitable pairs of triangles and merge them onto quads. In one of the first relevant approaches [Zhu et al., 1991] triangles are formed in an advancing manner and then combined into quads. In Lo [1989], heuristic measures are proposed to optimally pair triangles, leading to quad-dominant meshes. In Lee and Lo [1994] this method is enhanced by utilizing an advancing front pairing. Building upon Zhu et al. [1991], an advancing front strategy is also used in Q-Morph algorithm [Owen et al., 1999], that, by using suitable edge swaps and insertion of vertices, is able to reduce the number of irregular nodes and increase boundary alignment and orthogonality. The methods of Zhu et al. [1991] and Owen et al. [1999] are difficult to implement since they are advancing-front and they have to ensure the closing of opposing front in a conformal way without creating odd bounded subdomains. Q-Tran [Ebeida et al., 2010] is another method that uses triangle transformations leading to quad meshes with provably-good quality.

Even though indirect approaches are considered robust and efficient, they usually lead to isolated remaining triangles due to the local nature of operations. A sophisticated global solution is proposed in Remacle et al. [2012] that uses the

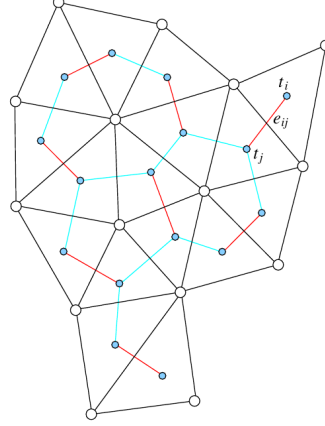


Figure 2.8: Perfect matching (red edges) on the graph of a triangular mesh (figure from [Remacle et al. \[2012\]](#)).

Blossom algorithm from graph theory to compute a perfect matching on the dual of the triangular mesh to optimally pair triangles. An important aspect of these methods is the position of vertices on the base triangular mesh. In [Lévy and Liu \[2010\]](#) the points are optimally placed by minimizing an L_p Centroidal Voronoi Tessellation objective function. [Remacle et al. \[2013\]](#); [Baudouin et al. \[2014\]](#) build upon a similar concept, but aiming for efficiency by placing the points in a frontal manner instead of an iterative one.

The number of irregular vertices and the overall quality of the resulting meshes can be optimized in post-processing. Geometric optimization is commonly based in Winslow smoothing schemes [[Knupp, 1999](#); [Karman, 2010](#)]. On the other hand, topological improvement may be based on the dual representation with chord-collapsing [[Daniels et al., 2008](#)], on quad patch replacement [[Bunin, 2006](#); [Verma and Tautges, 2011](#); [Takayama et al., 2014](#)] and on fundamental local operators [[Kinney, 1997](#); [Tarini et al., 2010](#)].

2.3 Hexahedral meshing

The problem of hexahedral meshing is considered to be the 'holy grail' of the mesh generation field. We refer here to the problem of finding an algorithmic solution to generate in a robust and efficient fashion 'good' hexahedral meshes for general domains, in the sense that they are aligned with the boundary and can provide geometric quality guarantees (in order to be practical for numerical simulations). There exist a large variety of methods that are extensions from 2D

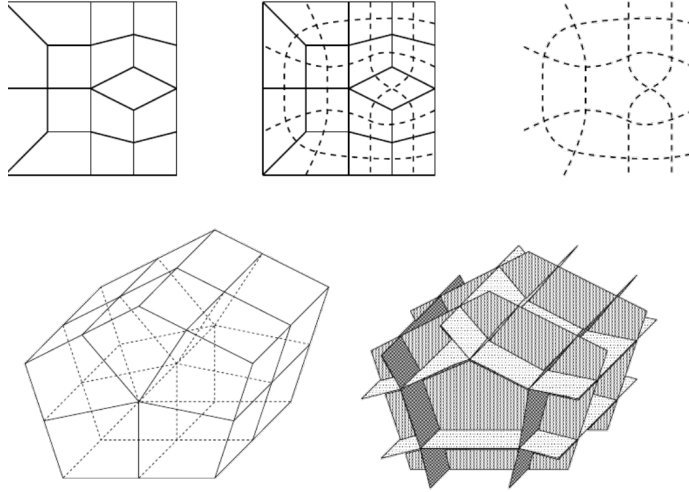


Figure 2.9: Dual representations of a quadrilateral mesh (top) and a hexahedral mesh (bottom) (figures adapted from [Schneiders \[2000a\]](#)).

quad meshing, use practical heuristics or rely upon topological constructions. Yet, a fully automated algorithmic method has not been developed. Open questions still exist about the feasibility and the method attributes for hex meshing. For example, generating a hexahedral mesh for a volume enclosed by a given quad mesh or constructing a 3D cross-field that corresponds to a valid hexahedral mesh topology.

[Mitchell \[1996\]](#); [Eppstein \[1999\]](#); [Erickson \[2014\]](#) describe the topological conditions of hex meshing, as well as theoretical constructions of topological hex meshes. Given a topological quadrilateral mesh bounding the volume, two conditions are given: i) it has an even number of quads and ii) no odd cycle in the quad mesh bounds an interior surface. A useful tool to understand the global nature of constraints involved in hexahedral meshing is the dual representation of the Spatial Twist Continuum [\[Murdoch et al., 1997\]](#). Concise presentations of topological and geometrical properties and constraints of hexahedral meshing can be found in [Shepherd and Johnson \[2008\]](#) and [Ledoux and Shepherd \[2010\]](#)

As in quadrilateral meshing, a common way to classify methods for hexahedral meshing is in: *direct* and *indirect* approaches [\[Lo, 2015\]](#).

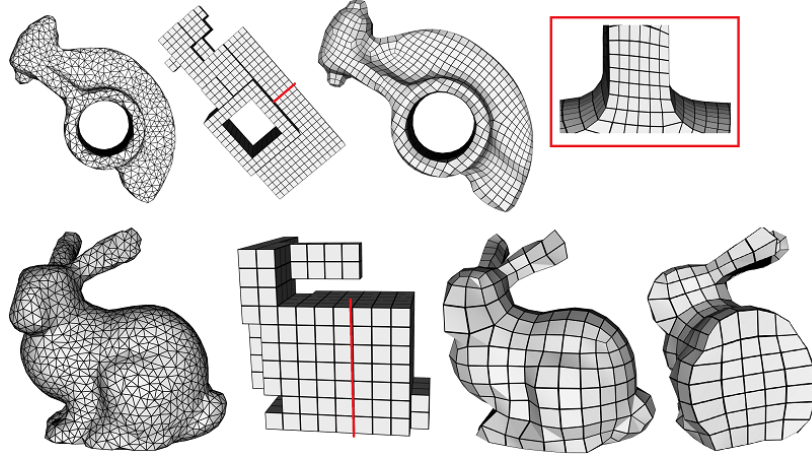


Figure 2.10: Polycube hexahedral meshing (figure from Yu et al. [2014]).

2.3.1 Direct hexahedral meshing

Classic methods in the first category include mapping [Li and Cheng, 2000], sweeping [Lai et al., 2000; Roca and Sarrate, 2006; Ruiz-Gironés et al., 2009; Rypl, 2010] overlay grid [Schneiders, 1996; Zhao et al., 2008], decomposition methods [Price et al., 1995; Liu and Gadh, 1997; Lu et al., 2001] and plastering [Stephenson et al., 1992; Staten et al., 2005]. Sweeping and plastering can be seen as the 3D analogous of advancing front methods such as paving.

In Tautges et al. [1996] a topological approach is followed based on the Spatial Twist Continuum [Murdoch et al., 1997], by trying to progressively build the dual representation from a quadrilateral boundary mesh. A set of useful criteria to evaluate hex-meshing techniques is provided in Tautges [2001]. Most of the methods mentioned can generate well-shaped structured hexahedra on domains with relatively simple boundary and topology.

More modern approaches include polycube methods, a form of sub-mapping where a polycube structure resembling the domain geometry is used [Hormann et al., 2004; Gregson et al., 2011; Liu et al., 2015; Guo et al., 2020] and 3D cross-field methods (commonly referred to as 3D frame-fields) [Huang et al., 2011; Li et al., 2012; Kowalski et al., 2014; Gao et al., 2017]. Cross-field methods, similarly to 2D, involve three steps: i) generating a 3D frame-field, ii) computing a volumetric parameterization based on it and iii) extracting a hexahedral mesh. One of the most important open problems is to develop 3D frame-field with topology that results to a singularity graph with a hex-meshable structure [Viertel et al., 2016; Reberol et al., 2019]. Without this, the applicability of a 3D global parametrization approach on large and complex models is pro-

hibiting. Another interesting approach producing impressive results has been recently proposed in LoopyCuts [Livesu et al., 2020], where the domain is decomposed by cutting loops based solely on a surface direction field, resulting in a 'metamesh' with predominantly hex-meshable blocks.

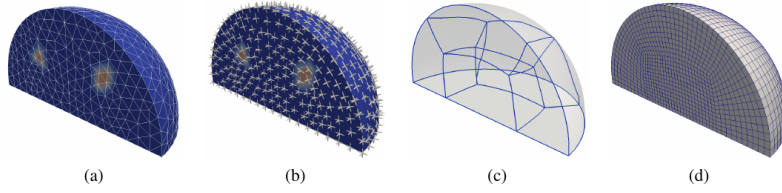


Figure 2.11: Block-structured hexahedral mesh. (a) tetrahedral input, (b) 3D frame field (c) singularity graph (d) hex mesh (figure from Kowalski et al. [2014]).

2.3.2 Indirect hexahedral meshing

Regarding *indirect* methods, it is not straightforward to extend triangle merging methods from 2D to 3D "tet-merging". While for a quad we usually need just a pair of triangles, there are *174 subdivisions of the hexahedron to tetrahedra* [Pellerin et al., 2018b]. It is thus very difficult to obtain a full hexahedral mesh by using a tet to hex combination approach, even with well placed vertices. In principle, it is always possible to subdivide a tetrahedron into four hexahedra with midpoint subdivision, but that leads to elements with poor quality. Usually, indirect methods strive for high-quality hex-dominant meshes, with pyramid or prism elements accommodating the transition between hexahedral and tetrahedral elements [Owen et al., 1997].

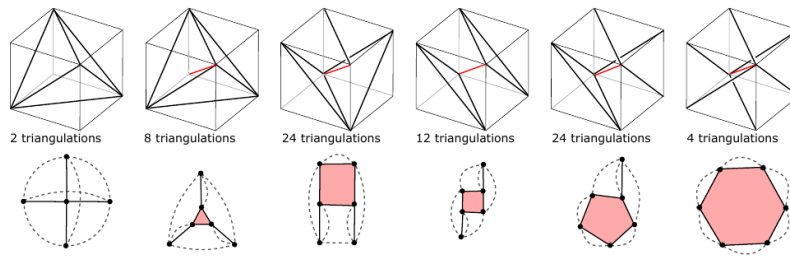


Figure 2.12: Combinations of hexahedron to tetrahedra along with their dual representation (figure from Pellerin et al. [2018a]).

Predefined patterns of possible decompositions are used in Meshkat and Talmor [2000]; Botella et al. [2016]. In Pellerin et al. [2018a] the process is improved

by identifying all possible combinations of tetrahedra into hexahedra, along with a greedy algorithm to choose them. A 3D cross-field can be used in an advancing front manner to generate aligned tetrahedral elements that are more suited for combination as in [Baudouin et al. \[2014\]](#); [Bernard et al. \[2016\]](#). In [Sokolov et al. \[2016\]](#) a parametrization technique is used instead to sample the points that leads to improved hex-dominant meshes in terms of quality and hex proportion.

CHAPTER



In a nutshell

Our starting point is a triangulation that describes the model. This input triangulation may be arbitrarily complex and of low quality, but it must be topologically valid. It should not have intersecting or duplicated elements, gaps between surfaces or non-conformal connections between them. The surfaces can be classified (denoted by distinct colors), with the interface edges between them considered as feature edges (Figure 3.1). Feature edges can also be edges detected by the dihedral angle, user-prescribed or problem-specific edges, and non-manifold edges. The general objective of our work is to transform this triangulation to a high quality quadrilateral mesh that may also serve as a boundary mesh of a hex-dominant mesh. We develop two pipelines: one for quadrilateral meshing and one for hexahedral-dominant meshing. We build upon theoretically sound considerations to produce high-quality results, but at the same time we follow a pragmatic approach; we want to produce results for

general inputs in an efficient and robust way. We thus define the design goals of interest for our methods:

- **Robustness.** In general, one of the most critical aspects of mesh generation is robustness. Our point of view is to develop methods that are robust, i.e., that will always provide a solution. We thus favor simplicity in our approach and we try to develop algorithms that are parameter-free.
- **Features preservation.** We consider complex models, possibly with non-manifold features, that may contain feature/floating edges and faces that must be preserved through the meshing process.
- **Mesh quality.** The output meshes must be of high-quality. We extensively use cross-fields and size-fields to guide the mesh generation process. The use of cross-fields allows generating meshes that are well suited for indirect conversion to quad/hex-dominant meshes.
- **Efficiency.** This is an obvious requirement in mesh generation: the method should be able to generate large meshes in terms of minutes. Our approach is based on local operations and the algorithms developed are of linear complexity.
- **Versatility.** A sequential approach is followed, based on modular building blocks. We can thus adapt our algorithms for different objectives.
- **Automation.** Our methods work with minimal user input. We want to avoid fine-tuning depending on the application.

The various parts of the method fall under a well-structured framework utilized both for 2D and 3D meshing. We use here the term 2D meshing for surface meshing (even if it also refers to general 3D surfaces) and the term 3D meshing for the volume case. Conceptually, we can divide our method to generate meshes on the following five general steps:

- (i) Computing a cross-field and a size-field to guide point generation.
- (ii) Generating the points.
- (iii) Triangulating the points.
- (iv) Combining triangles/tetrahedra into quadrilaterals/hexahedra respectively.
- (v) Post-processing - Optimization.

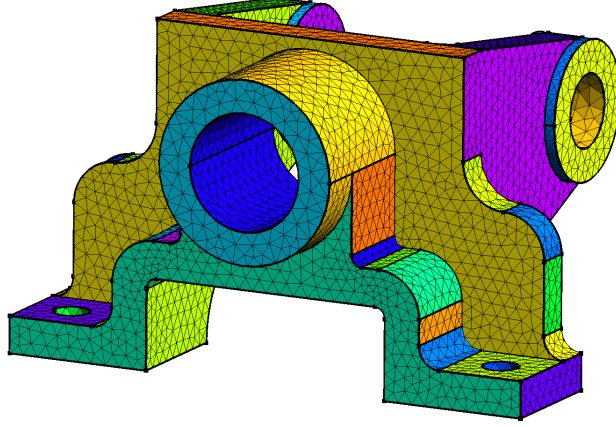


Figure 3.1: Input model with classified surfaces and feature points/edges.

The two pipelines, for the 2D and the 3D case respectively, are based upon these five steps. The sequential order remains the same for both, and this modular approach enables the transfer of ideas and solutions from the 2D case to the 3D and vice versa. Of course, each case retains its unique characteristics due to the different nature of the problems in question. For example, point sampling on the surface is more complicated compared to point sampling on the volume, while topological properties of hexahedral meshes result in much more constraint configurations compared to quadrilateral meshing. In the following, we will outline the steps for each pipeline and reference to the relevant chapter/section. Furthermore, we will explain the reasoning behind certain decisions and give extended remarks where we find it useful. The pipelines for quadrilateral and hexahedral-dominant meshing are schematically outlined in Table 3.1.

3.1 Cross-fields and size-fields

Cross-fields are a potent tool for meshing purposes. They define directions to which the quadrilaterals can be aligned, based on domain features and surface curvature. In this work, a cross-field $\mathbf{f}$ along with a size field $\mathbf{h}$ guides the point generation. Together, they define a metric field which can be viewed as a continuous representation of the domain's geometric characteristics.

In our initial work (Chapter 4), we take as input cross-fields that are computed based on Ginzburg-Landau theory [Beaufort et al., 2017a]. While this method is highly accurate and provides well-placed singularities, it is computationally

	Quadrilateral meshing	Hex-Dominant meshing
Cross-field Size-field	Section 6.3	Section 7.2
Point Generation	Advancing-front: Section 4.2 Section 5.3	Advancing-front: Section 7.3
Triangulation of points	Surface meshing with local modifications: Section 5.4	Constrained Delaunay Triangulation of Marot et al. [2019]
Combining into quads/hexes	Bipartite labelling: Section 5.5	Identifying tets-to-hex combinations of Pellerin et al. [2018a]
Post-Process Optimization	Section 6.6	-

Table 3.1: Outline of the two pipelines developed and the corresponding sections for quick reference.

expensive. In Section 6.3 a more efficient method is developed based on a multilevel diffusion approach.

The size-field can be computed in various ways. It can be a uniform global size, scaled by a factor in order to make the cross-field curl-free or computed from mesh features and curvatures. In Section 6.4 a method combining both is presented, that can smoothly adapt size-field to CAD features.

In practice, the input of our point generation algorithm is the set of direction vectors at each point and a scalar field for the sizing. It must be noted that our point generation module can have any kind of metric input. For example, we can use an anisotropic metric or a prescribed user-defined size-field that is 'incompatible' with the computed cross-field (in the sense of local integrability).

In Chapter 7, we investigate an approach to extend the cross-field integrability condition for 3D frame fields. The 3D scaled frame fields exhibit inherent anisotropy, since the scaling is not uniform in all directions. For generating a volume-bounding triangular mesh we can extract a surface cross-field by filtering out the normal directions. Using the same 3D frame field through the 1D-2D-3D mesh generation process allows ensuring the consistency of the orientation of elements of all dimensions.

3.2 Generating points

Points can be generated on a surface or inside a volumetric domain, guided by the metric field (cross-field and size-field). Computing a point set $\mathbf{P}$ as the optimal discretized form of the input metric field constitutes a very challenging problem.

We follow here an approach based on [Remacle et al. \[2013\]](#). The aim is to optimally place the points in alignment with the underlying directions. An advancing front scheme is used to this end, where we propagate points throughout the domain starting from the discretized boundaries. The algorithm is presented in [Section 4.2](#) and reiterated in [Section 5.3](#) for completeness. To generate each new point from a seed one, we implement a circle intersection operator as in [Baudouin et al. \[2014\]](#), since we want to respect the prescribed edge size and not the geodesic based distance in curved surfaces ([Figure 3.2](#)).

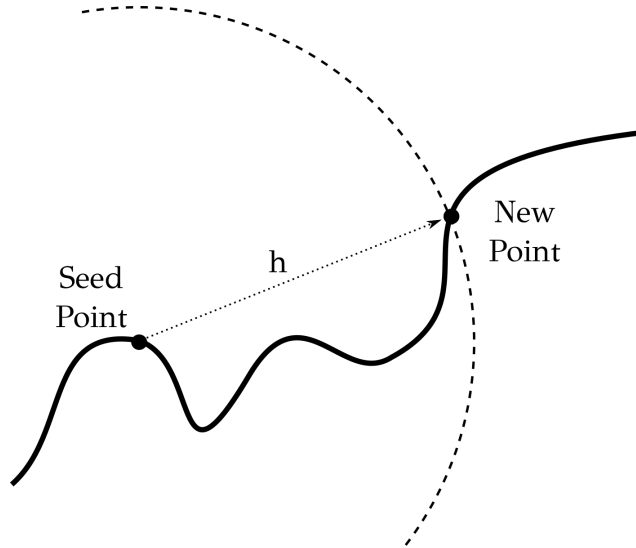


Figure 3.2: Intersection of a circle (with center the seed point and radius the prescribed size h) with the surface. This figure is a conceptual side-view of the [Figure 4.2](#)

For the filtering of points we utilize an R-tree structure ([Section 4.2.4](#)). We also implement an alternative, where the input triangulation serves as an underlying data structure in which generated points are stored. While R-trees offer a more efficient serial implementation, the latter approach can provide some significant advantages: i) a possibility for more straightforward parallelization and ii) better resolving of geometrical ambiguities between surfaces that are too 'close'

(for example two surfaces connected at a very steep angle).

In our initial work, a somewhat naive parallel implementation is presented, that nevertheless can speed up of the algorithm (Section 4.2.5). A better strategy would be to follow an approach as in [Marot et al. \[2019\]](#), where the domain is partitioned, and each part is treated independently.

The extension of metric-guided point generation inside the volume domain follows the same advancing front strategy as in 2D and is presented in Section 7.3. It is worth noting that it is more straightforward and trivial to implement compared to its surface counterpart (one of the rare cases where an extension from 2D to 3D is less challenging) since the circle-intersection operator is redundant for volumes.

3.3 Obtain a triangular/tetrahedral mesh

3.3.1 Surface meshing

The objective here is to generate a triangular mesh given the point set $\mathbf{P}$ that is generated during the previous step. In line with the modular nature of our methodologies, we keep the point generation from the element creation separate. This way enables us to: first, embed points on the domain in an unambiguous way, and second, provide robust surface meshing that is concerned only with element creation. Besides that, we have the possibility of using a different point sampling strategy or another triangulation method.

In Chapter 4, we are using the mesh generator of [Remacle and Lambrechts \[2018\]](#) that is specific to the sphere. The results obtained are of high quality, and we would be very interested in investigating the advantages that quadrilateral meshes would have in ocean modeling, knowing that validation and comparison of meshes for this kind of problems is a very challenging task in itself.

The situation is more perplexed for general 3D surfaces, for which we would like to have a generic, problem-independent approach. Thus, the process we developed in Chapter 5 (Section 5.4) receives as input just the initial triangulated surface and performs local mesh modifications directly on the 3D space. First, generated points are inserted with split edge/triangle operations, and secondly, the points of the initial mesh are removed with half-edge collapsing. Robust implementation of the second part was challenging and constituted one of the most time-consuming part of the current work.

3.3.2 Tetrahedral meshing

For the 3D tetrahedralization (Chapter 7) we are using the very efficient Constrained Delaunay Triangulation algorithm presented in [Marot et al. \[2019\]](#).

The input of this algorithm is a triangular mesh that bounds the volume and the set of points generated inside. Besides its primary objective, 3D triangulation is also a useful way to check the validity of the input surface mesh by identifying common defects as duplicated vertices/elements and intersected entities.

3.4 Combining triangles/tetrahedra into quadrilaterals/hexahedra

3.4.1 Bipartite all-quadrilateral meshing

In Section 5.5, we present a novel approach to obtain directly a quadrilateral mesh from a right-angled triangulation (or a quad-dominant mesh). Triangle-pairing methods commonly follow a greedy approach and produce quad-dominant meshes with isolated triangles. On the other hand, global approaches like BlossomQuad are computationally expensive. A Catmull-Clark subdivision can always be performed on quad-dominant meshes, but it leads to a quadrupled number of elements, a higher number of irregular vertices and inaccurate boundary representation.

We build our approach based on a topological labeling scheme that leads to bipartite meshes. During point generation, we are aiming for the optimal geometric placement of new points, but now we also propagate topological information in the form of binary labels. With each seed point generating one of the opposite label, we know that in the final quad formation we want to keep the edges that have opposite labeled vertices. Thus, we observe that in our right-angled triangulation the triangles with three same-indexed vertices are inconsistent and we add Steiner points to split them. Essentially, it works like a topological error-correcting code that identifies topological inconsistencies. We consider this approach an elegant solution, since it is very efficient, robust, and, in conjunction with the cross-field guidance, leads to high quality elements.

3.4.2 Hex-dominant meshing

The extension of the indirect approach in 3D is a much more constrained problem. Unfortunately, it is not possible to define something similar to a perfect matching (as the one BlossomQuad computes for triangular meshes) that can be used on tet-to-hex merging. The hex-dominant pipeline, along with results using scaled 3D frame-field, is presented on Chapter 7.

Given as input a tetrahedral mesh, we utilize the algorithm of Pellerin et al. [2018a] to find combinations of tetrahedra to hexahedra. The output hex-dominant meshes exhibit a satisfying proportion of hexahedra, both in terms

of element numbers and volumetric percentage. Boundary and smooth interior regions are filled with structured layers of hexahedra, but an inner ‘skeleton’ of tetrahedral regions (mostly around front collisions and non-trivial boundaries) remains.

If we use 3D scaled frame fields we observe improvement in hex proportion and more importantly on the structure of hex-dominant meshes (Section 7.4). We validated our method by meshing models constrained by quadrilateral bounding meshes that are extracted from full-hex meshes and examined the practical advantages on unconstrained medium-complexity models.

3.5 Post-process

The quadrilateral meshes produced with our method are of satisfying quality. We perform further geometric and topological optimization to improve the topological structure and maximize the quality.

A geometric optimization before triangle merging is performed in Section 4.3, by using a novel quality criterion for right-angled triangles. This quality criterion takes into account directionality and sizing. A more straightforward way to evaluate right-angled triangles (without the aforementioned measures) is to compute the normalized distance from the center of the hypotenuse (in practice the longest edge) to the center of the circumscribed circle. The quality of output quads is improved, but it is only a half-measure since the irregularity induced by colliding fronts cannot be eliminated.

Regarding the bipartite quadrilateral mesher of Section 5.5, the new vertices (Steiner points) are inserted using the split-edge operation, and therefore geometric robustness is ensured. In some instances, we may encounter minor geometric defects such as doublets or flat boundary quads (since we cannot reposition boundary vertices). Both of these cases can be locally fixed. The latter by collapsing a vertex to merge the two quads onto one and the former by inserting new vertices and ‘pushing’ the singularity inside the domain. The appearance of these defects arises predominantly for complex models with large difference in size scales, where the size-field may not adequately capture small features.

One of the main interests of this work is to exploit the global information provided by the scaled cross-field on a local basis without ‘heavy’ global strategies. The majority of quadrilaterals in our results are aligned with the cross-field. Still, it is an unstructured method based on an advancing-front scheme, and therefore a minimal number of singularities, let alone a semi-regular structure, is not feasible.

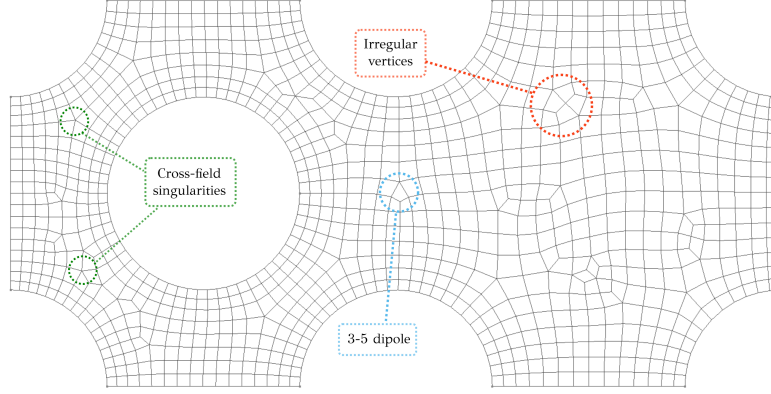


Figure 3.3: Types of irregular nodes (singularities) on a quadrilateral mesh produced with the method of Chapter 5.

We can identify three kinds of singularities (irregular vertices) on our output quadrilateral meshes:

$$S_{\text{total}} = S_{\text{crossfield}} + S_{\text{dipoles}} + S_{\text{redundant}}$$

$S_{\text{crossfield}}$ are singular vertices that naturally arise close to the position of cross-field singularities, without prescribing them. S_{dipoles} are pairs of singularities of index 3 and 5 respectively and imply size transitions. One of the reasons they exist in our meshes is that we do not follow a global quantization strategy for boundary edges (such as in Campen et al. [2015] for example), leading to cases where 'opposite' boundary edges with the same prescribed size-field have a mismatching number of vertices. Eliminating these singularities would require chord collapses that go through domain boundaries (thus changing boundary line discretization). Even though this is feasible for simple models, for complex CAD models as the ones we are interested in is a complicated task due to chord loops around the domain. Finally, $S_{\text{redundant}}$ are the ones due to colliding fronts or propagation of numerical error during the initial sampling of the points. We observe that we can locally remove the majority of redundant irregular vertices. It should be noted that also the 'redundant' singularities come in pairs of valence 3 and 5 singularities that 'cancel' each other. The distinction between those and the necessary 3-5 dipoles is essentially done a posteriori, with the latter being the ones that cannot be further removed without degrading quality and changing boundary discretization.

In Chapter 6 we present a topological optimization strategy to remove as much as possible of the redundant singularities, while keeping the ones accommodating structure and sizing constraints. It is based on continuously improving cavities of quads around irregular vertices by re-meshing or by replacing with predefined patterns. The quadrilateral meshes produced are of very high quality, and the method is able to handle highly complex CAD inputs. The meshes are quasi-structured, in the sense that they have a close-to-minimal number of irregular vertices. The next step of our developments is to convert these quasi-structured meshes to semi-regular ones with topological modifications. The optimization strategy presented in this chapter is a component of a more general end-to-end pipeline for CAD models, but it can also be utilized with the procedure of Chapter 5, i.e. having as input a triangular mesh and handling the model as a whole (instead of acting on each CAD face).

CHAPTER

4

Cross-field guided frontal point generation for quadrilateral meshing.

This chapter is an extended version of the following research note, presented in *International Meshing Roundtable 26*:

Georgiadis, C., Beaufort, P.-A., Lambrechts, J., Remacle, J.-F., *High quality mesh generation using cross and asterisk fields: Application on coastal domains*. In 26th International Meshing Roundtable, Research Notes, September 2017.

Abstract

This paper presents a method to generate high quality triangular or quadrilateral meshes that uses direction fields and a frontal point insertion strategy. Two types of direction fields are considered: asterisk fields and cross fields. With asterisk fields we generate high quality triangulations, while with cross fields we generate right-angled triangulations that are optimal for transformation to quadrilateral meshes. The input of our algorithm is an initial triangular mesh and a direction field calculated on it. The goal is to compute the vertices of the final mesh by an advancing front strategy along the direction field. We present an algorithm that enables to efficiently generate the points using solely information from the base mesh. A multi-threaded implementation of our algorithm is presented, allowing us to achieve significant speedup of the point generation. Regarding the quadrangulation process, we develop a quality criterion for right-angled triangles with respect to the local cross field and an optimization process based on it. Thus we are able to further improve the quality of the output quadrilaterals. The algorithm is demonstrated on the sphere and examples of high quality triangular and quadrilateral meshes of coastal domains are presented.

Keywords: Surface meshing, Quadrilateral meshing, Direction fields, Geophysical flows, Parallel meshing

4.1 Introduction

This work is motivated by the increasing use of unstructured meshes in ocean modelling. Unstructured meshes offer certain advantages in the context of geophysical simulations, such as conforming to the complex geometries of coastal domains and supporting variable mesh sizes [Candy, 2017; Engwirda, 2015]. One of the main difficulty in dealing with geographical data is the over-sampled nature of coastline representations. In Remacle and Lambrechts [2016] we propose an algorithm that automatically unrefines coastline data and we provide a fast and robust mesh generation procedure that is able to generate meshes of the earth system (ocean and continent). However, this method only generates triangular meshes and is specific to the sphere. For geophysical flow simulations quadrilateral meshes are desirable, since they contain twice less elements and can be aligned with the actual flow characteristics.

In this paper we introduce a general method to generate high quality quadrilateral and triangular meshes of surfaces. The input of our algorithm is an

initial mesh $\mathcal{T}_0$, in particular a mesh on the sphere generated with the method proposed in [Remacle and Lambrechts \[2016\]](#). The output is either a high quality triangular mesh or a right-angled triangular mesh that is transformed into an quadrilateral mesh. Both the description of the geometry and the Delaunay kernel are borrowed from [Remacle and Lambrechts \[2016\]](#). There, the point insertion strategy was based on a basic edge saturation Delaunay refinement procedure. The main contribution of this work is a new strategy to generate the vertices of the final mesh in an optimal way. By optimal we mean that the generated Delaunay triangles can be recombined in high quality quadrilaterals or that they are as equilateral as possible.

The method has similarities with [Baker et al. \[1988\]](#) or [Bern et al. \[1994\]](#) where authors use uniform grids or balanced quadrees to locate the internal vertices of the mesh. The drawback of such methods is that quadrees are axis oriented and meshes may be of low quality close to boundaries. The method that is proposed here overcomes these challenges.

All vertices are inserted using a frontal approach before computing their Delaunay triangulation. To steer the insertion of points we utilize direction fields. For quadrilateral meshing, the direction field is a cross field [[Bommers et al., 2009](#); [Kowalski et al., 2013](#); [Remacle et al., 2013](#)], while for triangular meshing the direction field is an asterisk field [[Beaufort et al., 2017a](#)] (Section 4.2.1). This direction field is computed on the base mesh $\mathcal{T}_0$ using a Ginzburg-Landau formulation [[Beaufort et al., 2017a](#)]. To compute the coordinates of the vertices we are using an efficient intersection strategy that does not rely on any global geometrical structures, e.g. octrees (Section 4.2.3). Similarly, filtering of close points is done locally on the fly (Section 4.2.4).

The whole procedure can be parallelized with a multi-threaded strategy proposed in Section 4.2.5. Finally, a novel way to measure the quality of right-angled triangles is proposed and utilized to optimize output meshes (Section 4.3) for combination into quads with the blossom-quad algorithm [[Remacle et al., 2012](#)].

The method is demonstrated here on spherical geometries. We present results for triangular and quadrilateral meshes on the world ocean (Section 4.4). Nevertheless, the algorithm is designed and implemented as a tool that can handle triangulations of any orientable surface $\mathcal{S}$ embedded in $\mathbb{R}^3$.

The method presented has been released as a self consistent open source code that can be used as a stand-alone program or that can be plugged in other software's such as Gmsh [[Geuzaine and Remacle, 2009](#)] or QGIS [[QGis, 2011](#)].

4.2 Frontal point insertion

To begin, we consider a base mesh $\mathcal{T}_0$ of an orientable manifold surface. The objective is to spawn points inside the domain in preferred directions. We essentially want to obtain a set of points $P = \{\mathbf{p}_1, \dots, \mathbf{p}_n\}$ that will either (i) lead to a triangulation $\mathcal{T}_q$ that is well suited for combining triangles into quadrilaterals or (ii) lead to a triangulation $\mathcal{T}_t$ that contains triangles that are close to equilateral. The idea is to use direction fields of two different kinds, depending on whether one wants to generate $\mathcal{T}_q$ or $\mathcal{T}_t$.

4.2.1 Direction fields

A *cross field* $\mathbf{c}$ is a field defined on a surface $\mathcal{S}$ with values in the quotient space S^1/Q , where S^1 is the circle group and Q is the group of quadrilateral symmetry. It associates to each point of a surface $\mathcal{S}$ to be meshed a cross made of four unit vectors orthogonal to one another in the tangent plane of the surface. In the context of quadrilateral mesh generation, a cross field represents at each point of the domain the preferred orientations of a quadrilateral mesh.

It is possible to build direction fields for building $\mathcal{T}_t$ by defining asterisk fields. An *asterisk field* $\mathbf{a}$ is also a field defined on a surface $\mathcal{S}$ but with values in the quotient space S^1/H , where H is the symmetry group of a regular hexagon. Pictorially, it associates to each point of the surface $\mathcal{S}$ an asterisk made of six unit vectors separated by 60 degrees. Such a field will be used to align triangular meshes to pre-defined directions in order to build triangles that are close to equilateral.

Direction fields should be as smooth as possible and should be aligned with the boundary of the domain. One can show that smooth fields only exist for surfaces with a Poincaré characteristic equal to 0. For any other topology, critical points will necessarily occur. More details on direction fields and their computation can be found in Beaufort et al. [2017a].

A cross field and an asterisk field are presented in Figure 4.1.

4.2.2 Generating points across direction fields

Assume a direction field $\mathbf{f}$ defined everywhere on $\mathcal{S}$. Field $\mathbf{f}$ is either a cross field ($N_d = 4$ directions) or an asterisk field ($N_d = 6$ directions). A priority queue is initially filled with all the boundary points of the base mesh $\mathcal{T}_0$. The point $\mathbf{p}_i$ at the top of the queue then tries to insert N_d points $\mathbf{p}_{ij}$ in the $j = 1, \dots, N_d$ directions defined by $\mathbf{f}(\mathbf{p}_i)$ and at a distance $h(\mathbf{p}_i)$ where h is the mesh size field. In order to have points inserted “by layers”, the priority queue that is chosen is a first-in, first-out queue. Our experience shows that ordering the 1D points allows smooth propagation of the fronts inside the domain. We have

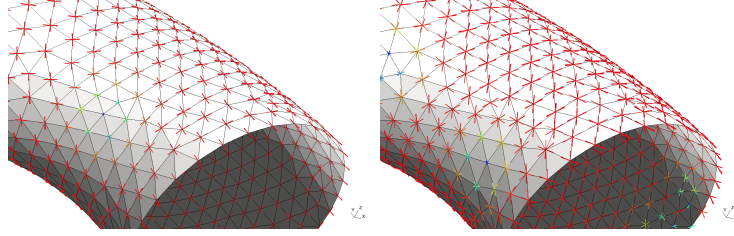


Figure 4.1: Cross field (left) and Asterisk field (right)

tested two kind of ordering: (i) order the points geometrically by sorting them along a space filling curve (Hilbert curve) and (ii) order the points topologically by walking along the 1D boundary. Each point $\mathbf{p}_i$ thus tries to spawn $\mathbf{p}_{ij}$, $j = 1, \dots, N_d$ neighbour points on $\mathcal{T}_0$. Yet, there is no guarantee that point $\mathbf{p}_{ij}$ is not too close to another point of the queue. Points $\mathbf{p}_{ij}$ are hence filtered. An exclusion zone of size $\alpha h(\mathbf{p}_i)$, $\alpha < 1$ is created around every vertex $\mathbf{p}_i$ of the queue in such a way that no new point can be inserted in the queue if it lies inside this zone. Finally, accepted points are added to the end of the queue as well as in P . The procedure terminates when the queue is empty. Algorithm 1 describes the procedure. In the two following subsections we will focus with detail on the two main operations, i.e. the point insertion and the point filtering.

4.2.3 Intersection with triangulation

Assume a point $\mathbf{p}_i$ that lies on one of the triangles of $\mathcal{T}_0$, a direction $\mathbf{d}_{ij}$ i.e. a unit vector tangent to the surface and the mesh size $h(\mathbf{p}_i)$ at that point. The aim is obviously to create an edge of size h . So, the new point $\mathbf{p}_{ij}$ will be located at the intersection of the triangulated surface $\mathcal{T}_0$ and a circle $\mathcal{C}_i$ of center $\mathbf{p}_i$ and radius $h(\mathbf{p}_i)$. $\mathcal{C}_i$ lies on the plane $\mathcal{P}_i$ that is formed by the direction vector $\mathbf{d}_{ij}$ and the normal to the triangulation at our origin point, $\mathbf{n}_i$ (Figure 4.2). To compute $\mathbf{p}_{ij}$ our goal is to find the intersection point of circle $\mathcal{C}_i$ with the triangulation $\mathcal{T}_0$ (Figure 4.2). We start from the triangle of the base mesh $\mathcal{T}_{0i}$ on which $\mathbf{p}_i$ lies. First, we compute the intersection line of the plane $\mathcal{P}_i$ and the plane of the triangle $\mathcal{P}_{\mathcal{T}_{0i}}$. Then, we find the intersection points of this line with the circle $\mathcal{C}_i$ and choose the one that lies in direction $\mathbf{d}_{ij}$. Finally, the barycentric coordinates $\lambda_0, \lambda_1, \lambda_2$ of this point with respect to the current triangle are calculated. In this way we determine whether the intersection point lies on the triangle, and thus a successful intersection with this triangle.

In the case where the current triangle is not intersected, we move forward to

Algorithm 1: Frontal point insertion algorithm.

input : Initial triangulation $\mathcal{T}_0$, mesh size field function $h(\mathbf{x})$ and a direction field calculated on $\mathcal{T}_0$

output: Array of points in preferred directions P to be triangulated.

Place boundary points in a queue;

Create data structure to store in which triangle of $\mathcal{T}_0$ the generated points lie;

while *queue is not empty* **do**

 take the first point $\mathbf{p}_i$ at the top of the queue;

 pop this point out of the queue;

 interpolate the direction vectors for this point;

for N_d *directions* **do**

 Insert point $\mathbf{p}_{ij}$ by intersecting the base triangulation (Section 4.2.3);

 create set of triangles $\mathcal{C}$ in exclusion zone around $\mathbf{p}_{ij}$ to filter (Section 4.2.4);

for $\mathbf{p}_k \in \mathcal{C}$ **do**

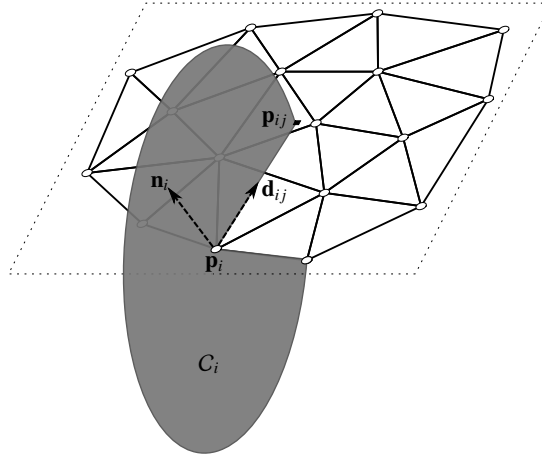
if $\|\mathbf{p}_{ij} - \mathbf{p}_k\| > \alpha h(\mathbf{p}_{ij})$ **then**

 add $\mathbf{p}_{ij}$ in P ;

 push $\mathbf{p}_{ij}$ in the back of the queue;

else

 delete $\mathbf{p}_{ij}$;

Figure 4.2: Computation of point $\mathbf{p}_{ij}$.

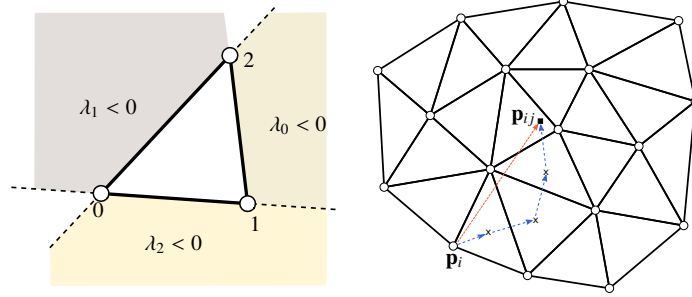


Figure 4.3: Indication of the position of intersection point according to barycentric coordinates (left) and computation of point $\mathbf{p}_{ij}$ by walking in the triangulation in specific direction (right)

another triangle. Since we have already computed the barycentric coordinates with respect to the current triangle $\mathcal{T}_{0i}$, we know where on the plane $\mathcal{P}_{\mathcal{T}_{0i}}$ the intersection point lies (Figure 4.3, left). Therefore, we have an indication on which neighbour triangle we should search after. This procedure continues until a valid intersection point is retrieved.

Essentially, we perform a walk in the triangulation [Devillers et al., 2001] in the desired direction, until we obtain the intersection point. Our experience shows that this method is the most efficient one amongst all possible solutions for this problem. For same mesh size field chosen for the initial and the output mesh, intersection points are found after a little less than 2 triangle visits on average.

4.2.4 Filtering

As it was previously explained, each point generates $\mathbf{p}_{ij}$ points for $j = 1, \dots, N_d$ directions. We have to ensure that new points are not too close to already generated points. Therefore, after each point $\mathbf{p}_{ij}$ is generated, a filtering procedure should follow. We prefer not to utilize a space search structure such as an RTree [Beckmann et al., 1990], since that would make parallelization more challenging.

Instead, we proceed with a more straightforward strategy. During the point insertion process, we store on which triangle of the base mesh $\mathcal{T}_0$ each inserted point lies. For every candidate point $\mathbf{p}_{ij}$, we take the set of triangles $\mathcal{C}$ around our new point that intersect an exclusion zone (Figure 4.4). The exclusion zone is defined as a circle of radius $h(\mathbf{p}_{ij})$ around the candidate point. Therefore we are able to directly obtain the set of points $P_f = \{\mathbf{p}_k, k = 1, \dots, n_f\}$ in the vicinity of $\mathbf{p}_{ij}$. Since our objective is to create right-angled triangles, i.e. equilateral triangles in the $\mathcal{L}_\infty$ norm, we compute the distance between the

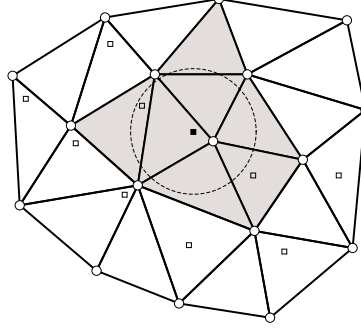


Figure 4.4: Filtering procedure for new points. — —: exclusion zone, $\mathcal{C}$: triangles intersecting the exclusion zone in light grey, ■: candidate point, □: previously generated points stored on $\mathcal{T}_0$

candidate point and its surrounding ones as $\|\mathbf{p}_{ij} - \mathbf{p}_k\|_\infty = \max\{|x_{ij} - x_k|, |y_{ij} - y_k|, |z_{ij} - z_k|\}$. The point is accepted for insertion if condition $\|\mathbf{p}_{ij} - \mathbf{p}_k\|_\infty > \alpha \cdot h(\mathbf{p}_{ij})$ holds for all $\mathbf{p}_k \in P_f$.

4.2.5 Parallel implementation

A multi-threaded Delaunay kernel, as well as a multi-threaded Delaunay refinement process based on it, have already been presented [Remacle and Lambrechts, 2016; Remacle, 2017] and can significantly reduce the time cost of the mesh generation process. The frontal point generation algorithm that was presented could add up to being the most significant term in the total timings of the mesh generation workflow. Therefore, and in order to combine with the parallel Delaunay triangulation process, the generation of points in a multi-threaded fashion would be highly desirable.

Due to the locality of point insertion and filtering, the whole process can be parallelized with the following straightforward strategy. As before, the boundary points are sorted along the boundary elements of $\mathcal{T}_0$. This set of initial points is then split in M queues, where M is the number of threads. An OMP parallel region is initiated, where each thread $m = 1, \dots, M$ handles its own subset of points independently. The threads have shared memory access to the final set of points P and the structure that stores the points for each triangle of $\mathcal{T}_0$. Each point $\mathbf{p}_i^m$ generates $\mathbf{p}_{ij}^m$, $j = 1, \dots, N_d$ ($N_d = 4$ or $N_d = 6$) new points around it. These points are filtered out with respect to the already inserted points. Accepted points in the local filtering process are flagged, but not yet inserted in the respective data structures.

In order to ensure that different threads do not insert points close to each other

and that do not access the global data structures at the same time, an OMP critical region follows. There, the candidate point $\mathbf{p}_{ij}^m$ of the current thread is checked against the already accepted points from the other threads at this point. If no other point is near, the point gets accepted and the procedure goes on. The process terminates when all M queues are empty.

Initial numerical tests show that we can have a significant speedup with the aforementioned strategy. Currently, the algorithm can generate 1 million points in around 10 seconds in 1 thread, while for 4 threads the time decreases to around 3.4 seconds.

4.3 Recombination to quadrilaterals

4.3.1 Right-angled quality criterion

Points along a cross field will generate right-angled triangles that are suitable for merging into quadrilaterals. Our concern now is to further improve the quality of these triangles before applying a recombination procedure. We consider that the optimal triangles would be the right-angled ones in respect to the local cross frame (Fig. 4.5, left), since they can be combined in quadrilaterals of optimal quality. To this end, we are investigating the development of a quality criterion for right-angled triangles, based on how 'close' the triangle is to the optimal one.

For each vertex of the triangle ($i = 1, 2, 3$) we calculate three corresponding local qualities, represented from the following dimensionless quantities:

- (i) a quantity to evaluate how close each angle of the triangle is to 90° :

$$q_a^i = 1 - \frac{|\frac{\pi}{2} - \theta_i|}{\frac{\pi}{2}}$$

- (ii) a quantity to evaluate the nearest edge of each vertex to one direction of our the local cross field. For this, we calculate for the two edges of each vertex their angle with the two main directions $\mathbf{d}_1$ and $\mathbf{d}_2$, thus $j = 4$ angles (Fig. 4.5, right), and we choose the smallest one:

$$q_b^i = \max \left\{ |\cos(2\theta_{ij})| : j = \alpha, \beta, \gamma, \delta \right\}$$

- (iii) finally, the ratio of the two edges of each vertex:

$$q_c^i = 1 - \frac{|e_{ik} - e_{ik}|}{\max(e_{ik}, e_{ik})}, \quad k = \{1, 2, 3 : k \neq i\}$$

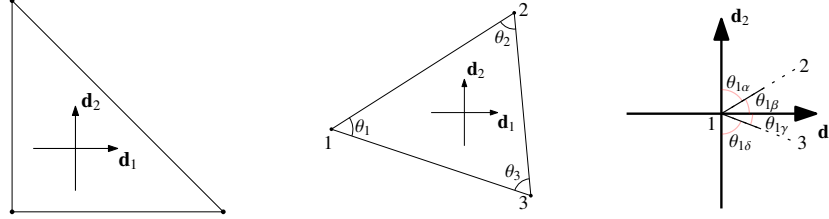


Figure 4.5: Optimal right-angled triangle aligned with the cross frame (left), triangle to be evaluated (middle), angles between edges of vertex 1 with the two main directions (right)

the optimal would be that the edges have equal length, i.e. it is a right-angled triangle, equilateral in the $\mathcal{L}_\infty$ norm. If the objective is to obtain anisotropic quadrangles through the merging of anisotropic right-angled triangles, this quantity can be easily modified to take into account the specified size ratio.

The final right-angled quality of the triangle is the maximum amongst the product of the three local qualities of its vertices: $q_t = \max \left\{ (q_a \cdot q_b \cdot q_c)^i : i = 1, 2, 3 \right\}$.

4.3.2 Optimize for recombination to quadrilaterals

We have now a measure to evaluate the quality of right-angled triangles. We expect that in certain regions the generated points will not create triangles optimal for recombination. Such regions are located for example in parts of the domain where there is transition to different mesh sizes or where fronts of insertion collide (Fig. 4.6, left). Herein we follow a simple procedure in order to optimize these regions, based on the quality measure presented in section 4.3.1.

For each interior point of the domain we take its corresponding cavity of surrounding triangles. The minimum quality amongst the cavity's triangles is set to be the objective function to be maximized. A local maximum is then searched in the line connecting the original position of the point and the centroid of the cavity in the infinity norm, and the point is relocated accordingly. The aforementioned optimization process is done for all the interior cavities of the mesh until points are not relocated further. The improvement of the right-angled quality of triangles in an insertion front collision is depicted on Figure 4.6.

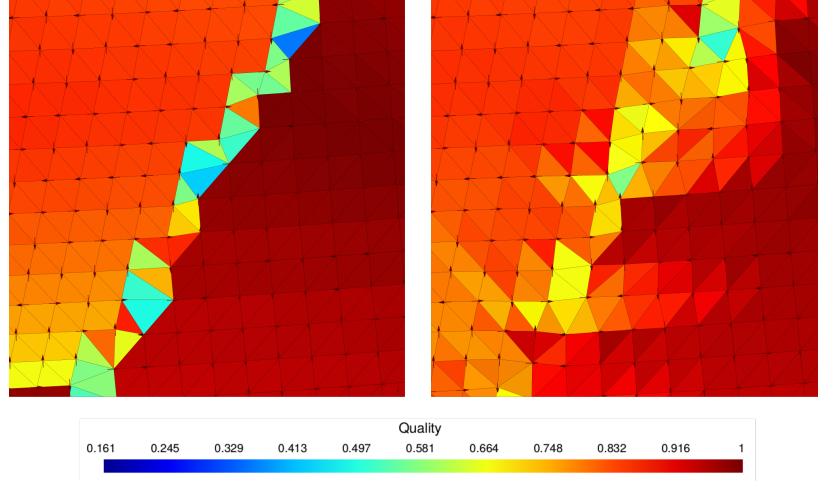


Figure 4.6: Quality distribution on an interface where fronts of insertion collide (one originating from the west and one from the east): before (left) and after (right) optimization. Low quality triangles are transformed and a smooth transition with pairs of right-angled triangles is obtained.

4.4 Results

4.4.1 Triangulations

We present here results for mesh generation by using asterisk fields. Figure 4.7 illustrates the process for generating a triangular mesh in such a way. In Figure 4.8 we present a comparison between the base and the output mesh on Baltic sea, choosing a high resolution mesh size field: from $h_{min} = 150m$ on the coast to $h = 3km$ away from it. The final triangular mesh exhibits a smoother distribution of equilateral triangles through the domain.

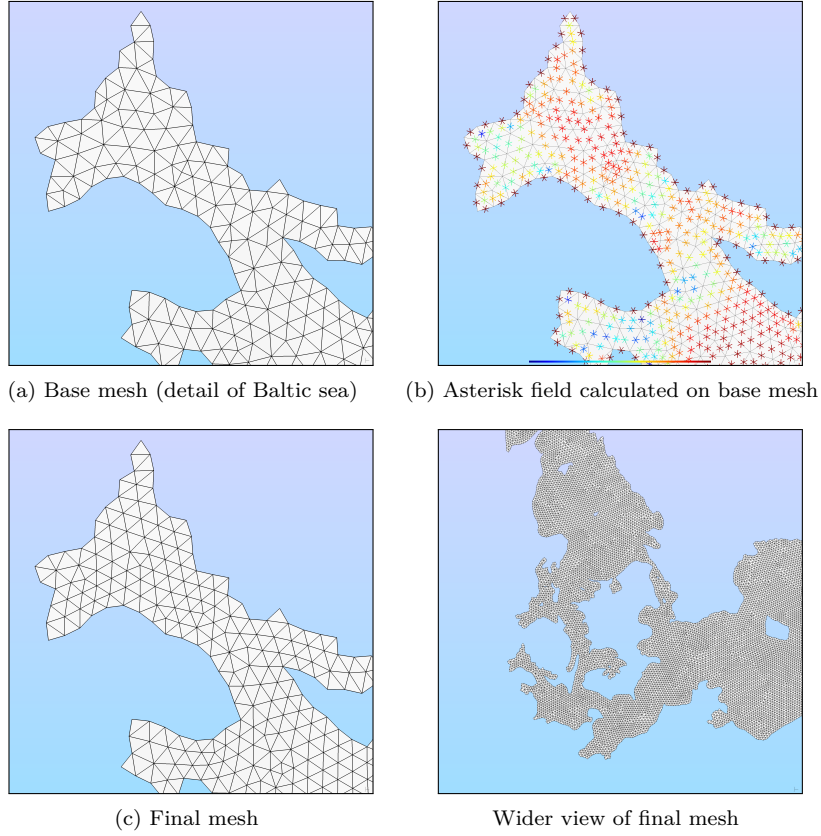


Figure 4.7: Generating a high quality triangular mesh by using an asterisk field.

4.4.2 Quadrangulations

Figure 4.9 shows the discrete steps of quadrilateral mesh generation workflow. Generation of points guided with a cross field leads to a right-angled triangulation. The optimization process described in section 4.3.2 is applied to it and further improves the orientation and pairing of right-angled triangles. The final merging to quadrilaterals is done with the blossom-quad method [Remacle et al., 2012] along with quadrilateral smoothing procedures incorporated in Gmsh [Geuzaine and Remacle, 2009]. It must be noted that a small number of remaining triangles along the coastal boundary may remain. This is due to the highly irregular nature of the boundary geometry and can be easily addressed by choosing a mesh size field smaller than the characteristic length of

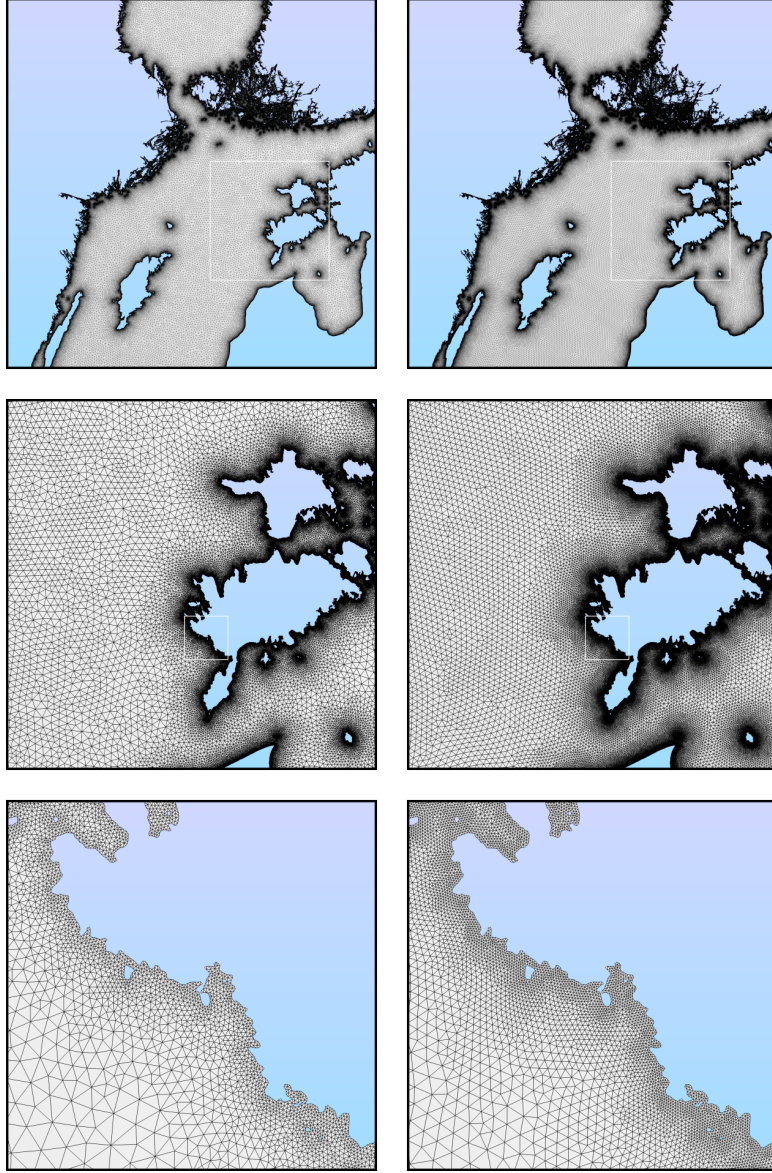


Figure 4.8: Images of base (left) and output (right) triangulation on the Baltic sea. Average radius-ratio quality (defined as $\gamma = \frac{2r_i}{r_c}$, where r_i and r_c are the radii of the inscribed and circumscribed circles of the triangle respectively) is $\bar{\gamma} = 0.947$ ($\gamma_{min} = 0.104$) for the initial mesh. The output mesh improves it to $\bar{\gamma} = 0.981$ ($\gamma_{min} = 0.3231$).

the boundary.

Finally, we utilize our algorithm to generate a quadrilateral mesh of the whole world ocean (Figure 4.10), with a resolution from $h_{min} = 3\text{km}$ to $h_{max} = 60\text{km}$. Generation of the final 2,267,738 points takes around 30 seconds and the triangulation of them around 5 seconds. For comparison, the base mesh generated with the procedure of Remacle and Lambrechts [2016] takes around 50 seconds. The average isotropy measure quality [Johnen et al., 2016] of the output quadrilateral elements is 0.944.

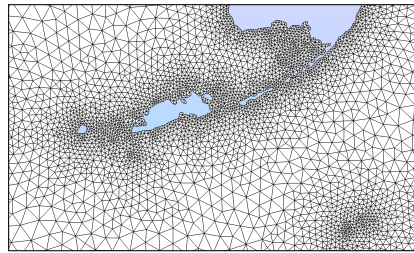
4.5 Conclusions

This paper presents a method to generate high quality meshes by utilizing direction fields. By using solely information from an initial triangulation, the algorithm does not require specific knowledge of the geometrical characteristics of the surface. This holds accurately for the case where the mesh size field is the same for the base and the final mesh.

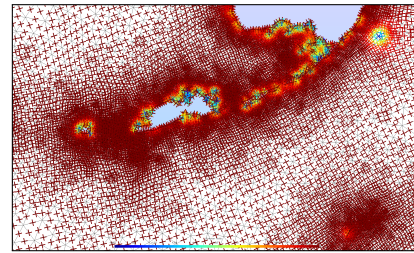
By utilizing an asterisk field (direction field with 6 turns) we can generate triangulations of high quality in comparison with the initial mesh. Combining this method with an appropriate knowledge of the underlying geometry, it could be used as a tool for re-meshing triangulations to finer resolution and improved quality.

Respectively, by utilizing a cross field (direction field with 4 turns), we can generate right-angled triangulations suitable for recombination to quadrilateral meshes. We have developed an optimization procedure that is applied on the output right-angled meshes, in order to further maximize the quality of the resultant quadrilaterals.

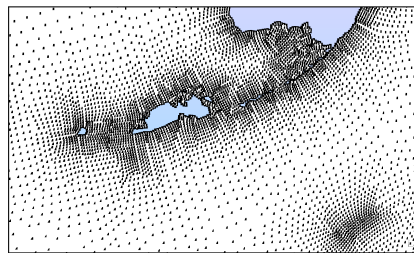
A multi-threaded strategy for our frontal point generation algorithm is presented. A more detailed analysis of the multi-threaded implementation will follow in the future. The following step of our work is the generation of points in an anisotropic fashion and thus the generation of boundary layer meshes.



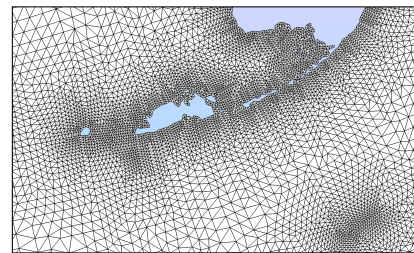
(a) base mesh (white is water)



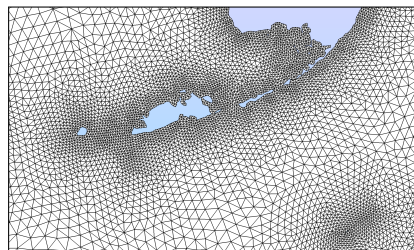
(b) cross field on base mesh



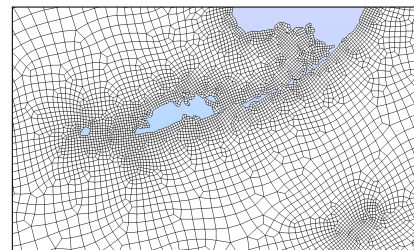
(c) generation of points in the domain



(d) right-angled triangulation



(e) optimization of right-angled triangulation



(f) final quadrilateral mesh

Figure 4.9: Step by step generation of a non uniform quadrilateral mesh around the southern tip of Florida peninsula. Generation of points directed by the cross field and optimization of right-angled triangulation maximizes the quality of the output quadrilateral mesh.

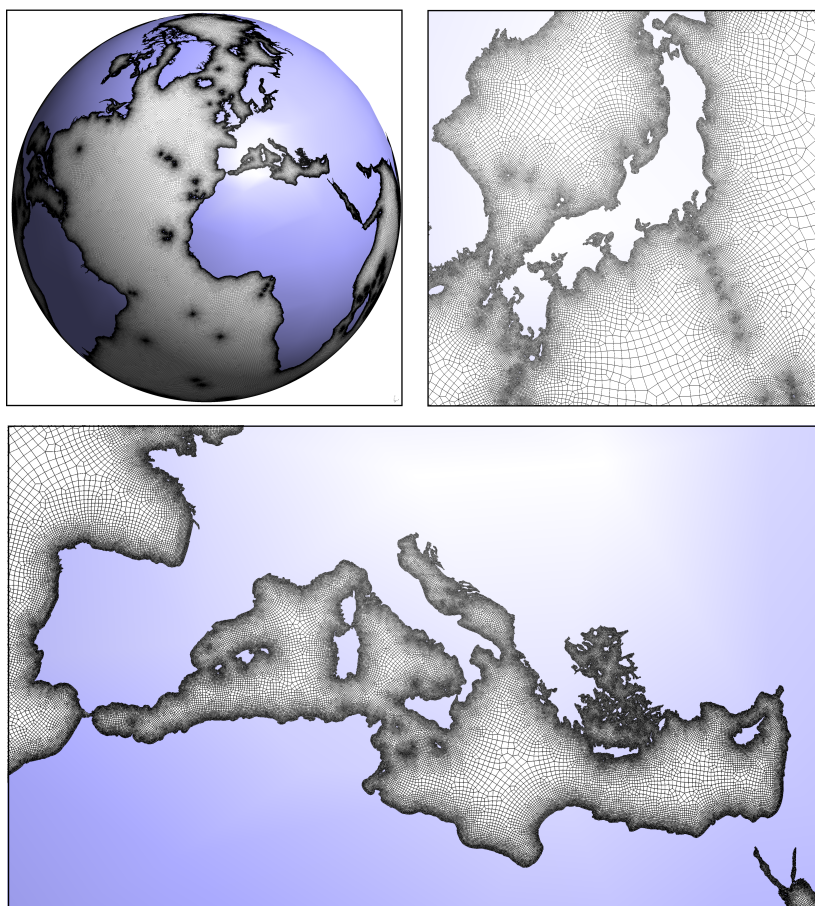


Figure 4.10: Quadrilateral mesh of world ocean (top left) with a zoom on the Mediterranean sea (bottom) and on the sea of Japan (top right).

CHAPTER



Indirect quadrilateral meshing using bipartite labelling.

This chapter is a reproduction of the following paper:

Georgiadis, C., Reberol, M., Remacle, J.-F., to be submitted, *Indirect quadrilateral meshing based on robust surface meshing and bipartite labelling.*

Abstract

Quadrilateral meshes offer certain advantages compared to triangular ones, such as reduced number of elements and alignment with problem-specific directions. We present a pipeline for the generation of quadrilateral meshes on complex geometries. It is based on two key components: robust surface meshing and efficient indirect conversion of a triangular mesh to an all-quad one. The input is a valid geometric surface mesh, i.e., a triangulation that accurately represents the geometry of the model. A right-angled triangular mesh is initially created in the 3D space by continuously modifying the input mesh while always preserving its topological validity. The main advantages of our local mesh modification based approach are to (i) allow the generation of meshes that are globally aligned with a given direction field and (ii) to handle robustly non-manifold feature edges (in multi-volume models) and small features. The final quadrilateral mesh is obtained by merging pairs of triangles into quadrilaterals. A bipartite labeling scheme is utilized to identify and correct inconsistent pairings. The algorithm is based on local operations and is much more efficient than previous global strategies for tri-to-quad conversion. The whole pipeline is tested on a large number of models with diverse characteristics.

Keywords: surface meshing, quadrilateral meshing, bipartite labeling, cross fields

5.1 Introduction

The automatic generation of quadrilateral meshes is still regarded as a challenging problem in mesh generation. Even though a lot of different approaches exist, there is not to date a conclusive method, analogous to triangular meshing which is considered highly mature and for which there are robust algorithms based on strong mathematical foundations.

The purpose of this work is to address the problem of generating quadrilateral meshes for complex 3D geometries. We strive for generality in our approach; our input is simply a triangulation of the model. The triangulation can be an STL representation of the geometry, triangulation of scanned data or a mesh generated from a CAD model with standard Delaunay meshing techniques. The input meshes may be of bad quality and contain non-manifold feature edges. Our goal is to design a pipeline satisfying the following design goals:

(i) robustness, i.e., guarantee of termination regardless of the complexity/bad quality of input data, (ii) feature preservation, i.e., the persistence of some user-defined internal and boundary curves, (iii) high element quality, and (iv) efficiency, i.e., providing a meshing algorithm with a running time comparable to or faster than conventional indirect technologies.

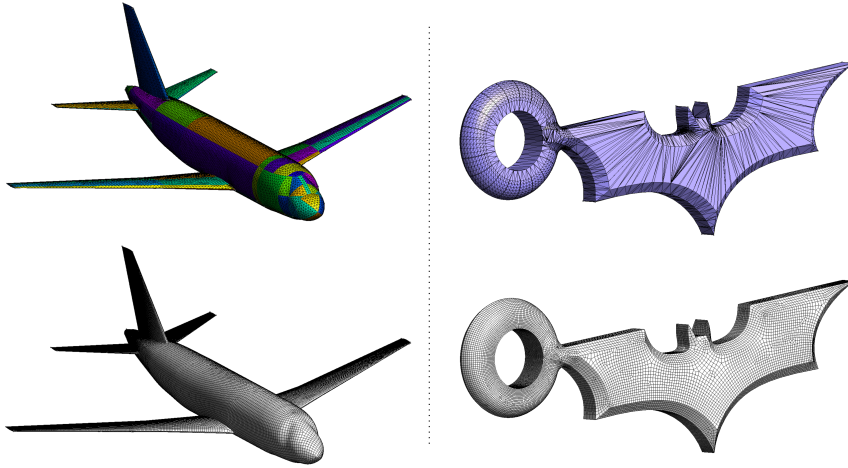


Figure 5.1: Quadrilateral meshes produced by our algorithm with input a CAD model (left) and an STL triangulation (right).

To generate a high-quality unstructured quadrilateral mesh that preserves user-defined features, we develop three independent steps: (i) we sample points on the model edges and surfaces, guided from a metric field (cross-field and an associated size-map), (ii) we use local-mesh modifications to continuously remove points of the initial triangulation and add the new ones, leading to a right-angled triangular mesh and (iii) we convert the triangular mesh into an all-quad mesh using a novel approach based on a topological labeling scheme.

It is quite amusing to see that, when it comes to mesh generation, the expected reliability rate of industrial-grade algorithms is essentially 100%, whereas few engineers would ask for the same reliability rate for a CFD calculation, for example. The input data supplied to our meshing algorithms are often noisy. Inputs may be huge, with a wide range of scales. It is not surprising that the most complex/tricky part of our work is related to robustness. We have worked very hard to ensure that this mesh generator provides results regardless of the complexity of the input data, as long as it is correct (i.e., a watertight non-manifold triangulation). All the details of implementation cannot be fully

described in a paper and, in order to ensure reproducibility of this work, the whole implementation is available in Gmsh, the open-source mesh generator [Geuzaine and Remacle, 2009]. To demonstrate the robustness of our algorithm, we applied it to a large number of CAD models found in large CAD part databases.

5.1.1 Related Work

Surface Meshing Surface mesh generation poses various difficulties related to robustness, efficiency, and optimality. Of the several methods proposed in the bibliography, we can identify two main categories [Borouchaki et al., 2000; Lo, 2015]: i) *Parametric* approaches, where the surface mesh is generated in the parametric space, and ii) *Non-parametric (direct)* approaches, where the surface mesh is generated directly in the 3D space

In *parametric approaches*, the 3D surface is mapped to a 2D parametric space [Lo, 1988; Sheng and Hirsch, 1992; Shimada and Gossard, 1998; Borouchaki et al., 2000; Floater and Hormann, 2005]. Since the CAD surfaces (typically NURBS patches) have underlying u, v representation, it can be efficient to generate a mesh in the plane with standard meshing techniques and after map it back to the 3D space. Generating planar meshes in the parameter space is a robust approach that is usually able to provide high quality meshes. Yet, approaches that use the parameter plane are able to consider surfaces that are isomorphic to a punctured disk. Meshing complex models with a parameter space approach does not allow to globally align a mesh with a cross-field, since each discrete patch of a CAD model may be equipped with an independent parametrization and the feature edges that separate those patches are not necessary aligned with the cross-field. Parametrization techniques can also be used to remesh triangulations [Remacle et al., 2010; Beaufort et al., 2020]

Non-parametric (also referred to as *direct surface meshing* in the literature, creating some ambiguity with the terms of *direct/indirect* approaches commonly used in quad meshing) approaches to surface meshing can be based on quad-trees [Shephard and Georges, 1991; Frey and Marechal, 1998], advancing front [Löhner and Parikh, 1988; Lau and Lo, 1996] or Delaunay strategies [Baker, 1989; Borouchaki et al., 1997]. One of the main advantages of the direct approach is its usefulness for models where an underlying parametrization is not available or when it is degraded. Local mesh modifications strategies to remesh models described by STL triangulations are proposed in Wang et al. [2006]; Béchet et al. [2002]. One of the main difficulties of this class of methods is that the manipulation of geometry directly on the 3D space is a challenging task that may lead to geometric or topologic ambiguities.

Quadrilateral Meshing Initial efforts to automatically generate quadrilateral meshes include *grid based* and *paving* algorithms. Grid-based methods start with the generation of a background Cartesian or a quad-tree grid with the subsequent snapping of elements to the domain boundaries [Baehmann et al., 1987; Schneiders, 1996; Frey and Marechal, 2000]. The paving algorithm, first introduced by Blacker and Stephenson [1991] generates quadrilateral elements in an advancing-front fashion, propagating from the boundary to the interior. Both classes of methods suffer from a degraded quadrilateral quality and high node irregularity on specific regions of the domain: the latter on the domain boundaries and the former on the front collisions on the interior. In Mitchell et al. [2014], a bichromatic Delaunay quadrangulation method is presented, with our current work building upon a similar concept.

On the other hand, quad conversion or indirect methods are based on the merging of pairs of adjacent triangles of an input mesh to quadrilaterals [Lee and Lo, 1994; Borouchaki and Frey, 1998]. In Q-Morph [Owen et al., 1999], triangles are transformed into quadrilaterals with an advancing-front algorithm. Blossom-Quad algorithm [Remacle et al., 2012] computes a perfect matching to optimally pair triangles, while Remacle et al. [2013]; Baudouin et al. [2014] produce meshes better suited for triangle-pairing by generating aligned right-angled triangles.

Cross Fields Cross fields are nowadays commonly used in the context of mesh generation, a line of research stemming from the computer graphics community and global parametrization methods [Bommes et al., 2013b]. For quad/hex meshing, cross-fields define preferred orthogonal directions on the domain to guide creation of optimal elements [Ray et al., 2006; Palacios and Zhang, 2007; Beaufort et al., 2017b]. Cross fields should be as smooth as possible (except at singular points) and aligned with the boundary of the domain.

5.1.2 Contributions

The quad meshing pipeline that is proposed follows a modular approach, with each of the steps being an independent algorithm that can be re-used in various situations (Figure 5.2). As stated before, our main concern is to provide a reliable solution, i.e., we want the quad meshing pipeline to be *resilient to complex/ill-conditioned inputs*.

The two main contributions of this work are:

1. **Robust surface meshing.** In our procedure, we follow the idea of Baudouin et al. [2014] of separating the generation of the points and the creation of the elements. The main novelty of this work is our direct approach. In this work, the initial triangular mesh is *continuously modified*

through robust local mesh modifications. The word continuous is chosen in purpose: each local mesh modification that is performed guarantees the topological integrity of the current triangular mesh. At the end of the remeshing process, most of the points of the initial triangular mesh, and in most cases all those points are removed from the triangulation and replaced by the ones created to accommodate the cross field and size field characteristics.

2. **Straightforward and efficient all-quad meshing.** We propose a bipartite labeling scheme that propagates topology information on the vertices during point generation. By using this information on the right-angled triangular mesh, we are able to optimally place new Steiner points to fix topological inconsistencies (odd-bounded regions on the interior) and recover a bipartite, all-quad mesh. The approach is very efficient since it converts a global problem to a local one.

5.2 Overview

Our algorithm takes as input a watertight, possibly non-manifold triangulation $\mathcal{T}_0$. This initial mesh can be an STL triangulation, another triangular mesh created by commonly used surface meshers, or the extracted boundary surface of a volumetric mesh. The input triangulation $\mathcal{T}_0$ is *classified*: the word classified indicates the fact that the triangles are grouped into *colors* and interfaces between colors are considered as *feature edges* that must be conserved in the remeshing process. Internal feature edges may also exist that lie inside a group of triangles with the same color. Note that feature edges can also be detected based on the dihedral angle, given a user-defined threshold. We take into account two categories of feature edges in $\mathcal{T}_0$: (i) non-manifold feature edges with two or more adjacent surfaces and (ii) internal (embedded) feature edges inside surfaces (e.g., a crack for solid modeling)

All the necessary topological information is thus available in the initial mesh. Surfaces are bounded by closed feature edges and those feature edges by feature points. Consecutive feature edges form *feature curves*. Feature curves must be preserved during the continuous mesh modification process. A key element of our method is the ability to handle the model as a whole. We do not follow a patch-wise approach where we handle each surface independently, followed by a connection of curves to ensure conformity. A half-edge data structure [Kettner, 1998] is used to get connectivity information, and an array of boundary edges that define feature curves is stored. We extend this data structure by storing the type of boundary edge (manifold or non-manifold), along with the triangles connected to each (one triangle for open boundary edges, two triangles for

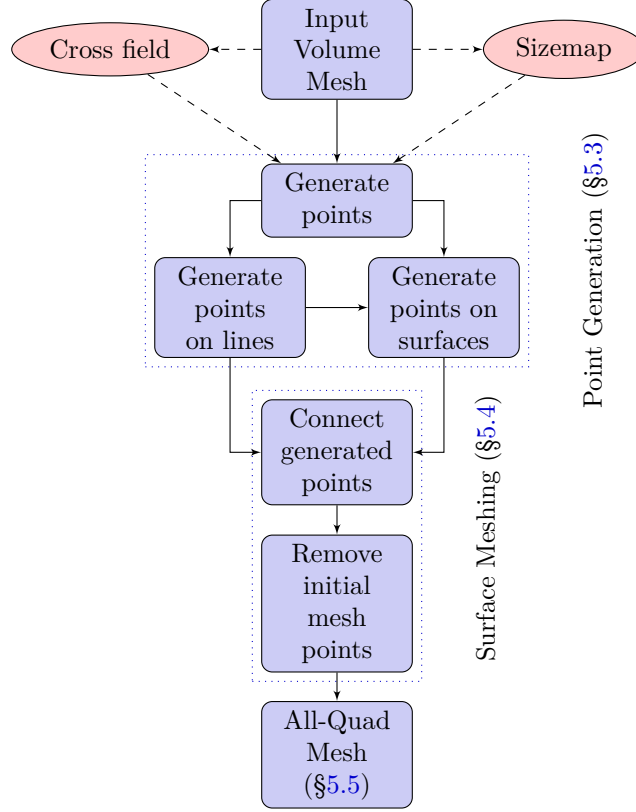


Figure 5.2: Schematic outline of the pipeline proposed for quadrilateral meshing

manifold edges, and a larger than two number for non-manifold ones). This feature enables us to efficiently treat boundaries during surface meshing.

The scheme provides the flexibility to preserve the topological characteristics of the input mesh, such as 'hard' edges or user-defined feature curves, without relying on extensive a priori knowledge of domain characteristics or a complicated feature recognition preprocess [Shamir, 2008; Vieira and Shimada, 2005; Sunil and Pande, 2008; Thakur et al., 2009]. Furthermore, by utilizing an appropriate data structure on top of the half-edge one, we can handle non-manifold configurations that may occur in industrial CAD models.

The input triangulation $\mathcal{T}_0$ is considered as the geometric model. Two other inputs are required for running the algorithm: (i) a unit cross-field $\mathbf{f}$ and (ii) a size field $h(x)$ that are both used for guiding the point insertion process. Although $\mathcal{T}_0$, $\mathbf{f}$ and h can be independent, it is beneficial to have (i) a cross-

field $\mathbf{f}$ that is aligned with the feature edges of $\mathcal{T}_0$ and (ii) a size field h that takes into account both the local change of direction of the cross-field and small features of the geometry. By default, $\mathbf{f}$ and h are computed using $\mathcal{T}_0$ as support.

The transformation of the triangulation $\mathcal{T}_0$ into the final quad mesh $\mathcal{T}_q$ is done in three steps:

1. *Point generation* (§5.3). Points of the final mesh $\mathcal{T}_q$ are generated on the triangles of this base/initial mesh $\mathcal{T}_0$, guided by both the cross and the size field.
2. *Point replacement* (§5.4) The initial mesh $\mathcal{T}_0$ is continuously transformed into another triangular mesh $\mathcal{T}_1$ by connecting the newly generated points on $\mathcal{T}_0$ and subsequently removing the initial mesh points, utilizing local mesh modifications.
3. *From triangles to quads* (§5.5) Mesh $\mathcal{T}_1$ is transformed into a quad mesh by combining pairs of triangles. Here, a brand new approach is proposed that is general, fully robust, and surprisingly simple.

The final quad mesh can be further optimized, both topologically and geometrically, with the procedure outlined in Section 5.5.3. The simple model of Figure 5.3 is used as an example to describe those three steps.

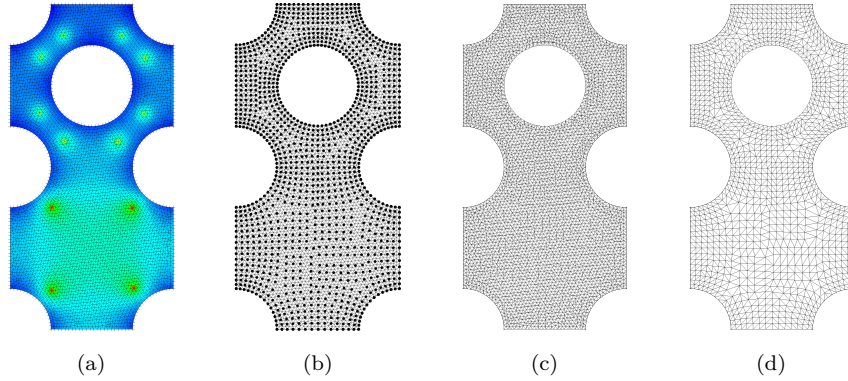


Figure 5.3: Surface meshing steps. (a) Cross-field and size-field computed on the input mesh. (b) Generated points embedded on the input mesh. (c) Inserting points to the triangulation with local mesh modifications. (d) Right-angled triangular mesh of the generated points.

5.3 Point Generation

In this work, we take a standard *surface-to-volume* point of view of mesh generation on Gmsh [Geuzaine and Remacle, 2009] which essentially consists of a bottom-up procedure. Model edges are discretized at first. Mesh lines on the model edges are used as boundaries of model faces and mesh triangles on model faces are used as the boundary of model volumes.

A set P of points on curves and surfaces are generated at first using an approach that is similar to Baudouin et al. [2014]. The main difference with Baudouin et al. [2014] is that all the operations are performed here directly on $\mathcal{T}_0$ without using a parametric space. Our frontal approach requires the use of a cross-field $\mathbf{f}$ that allows to structure the quad mesh and a size field h that allows taking into account the various feature sizes of a model as well as changes of directions of $\mathbf{f}$ (Figure 5.3a). Both cross and size fields that are used in this work have been borrowed (Chapter 6 - Section 6.3). We refer to this paper for more information on those two auxiliary fields. The method we follow has been presented in detail on Georgiadis et al. [2017b]. In the following we will outline the basic steps of this method for point sampling.

5.3.1 Line Point Generation

Each model line is uniquely defined from a list of non-intersecting connected boundary edges. Given a size field h defined by a value at each boundary point, we mesh the discrete representation of each line by following the general guidelines of Frey and George [2008]. This leads to the set of points $P_l = \{\mathbf{p}_i \mid i = 1, \dots, N_{gl}\}$. It is important to note that at this step, we can control the generation to have an even number of points for each boundary curve. This provides us with a topologically necessary condition for all-quadrilateral meshing (see §5.5).

5.3.2 Surface Point Generation

Starting now from the N_{gl} generated points on mesh lines, we want to spawn a set of points $P_s = \{\mathbf{p}_i \mid i = 1, \dots, N_{gs}\}$ on the surfaces in the directions provided by the cross-field $\mathbf{f}(\mathbf{x})$ and with respect to the underlying size field $h(\mathbf{x})$. The point set P_s , along P_l , will lead to a right-angled triangulation $\mathcal{T}_1$ that is well suited for combining triangles into quadrilaterals and form $\mathcal{T}_q$.

The cross field $\mathbf{f}$ gives $N_d = 4$ orthogonal directions. A priority queue is initially filled with the N_{gl} boundary points. In order to have points inserted “by layers”, the priority queue that is chosen is a first-in, first-out queue. The point $\mathbf{p}_i$ at the top of the queue then tries to insert N_d points $\mathbf{p}_{ij}$ in the $j = 1, \dots, N_d$ directions defined by $\mathbf{f}(\mathbf{p}_i)$ and at a distance $h(\mathbf{p}_i)$. We want to

to create the new point exactly at distance $h(\mathbf{p}_i)$ from the seed one and embed it unambiguously in the underlying surface. For this, we use a circle-intersection operation. We define the circle $\mathcal{C}_i$ of center $\mathbf{p}_i$ and radius $h(\mathbf{p}_i)$ that lies on the plane formed by the direction vector $\mathbf{d}_{ij}$ and the normal to the triangulation at our origin point $\mathbf{n}_i$. We compute the new point $\mathbf{p}_{ij}$ as the intersection point of the circle $\mathcal{C}_i$ with the underlying triangulation by successively in the triangulation.

For the filtering of nearby points during generation we have used two options. For the first one, we can use the input mesh $\mathcal{T}_0$ as a data structure [Georgiadis et al., 2017b]. The second choice is to simply use an R-tree for spatial queries. Both choices gives us a list $P_f = \{\mathbf{p}_k, k = 1, \dots, n_f\}$ of nearby points to the point in question. For each of these points we compute if they are inside an exclusion box defined by the input metric field (with the orientation of the box given by the directions). A different metric input can be trivially used if necessary.

The whole procedure finalizes when no points are left in the priority queue and our domain is filled with the cross-field aligned point set P (Figure 5.3b).

5.4 Surface Meshing

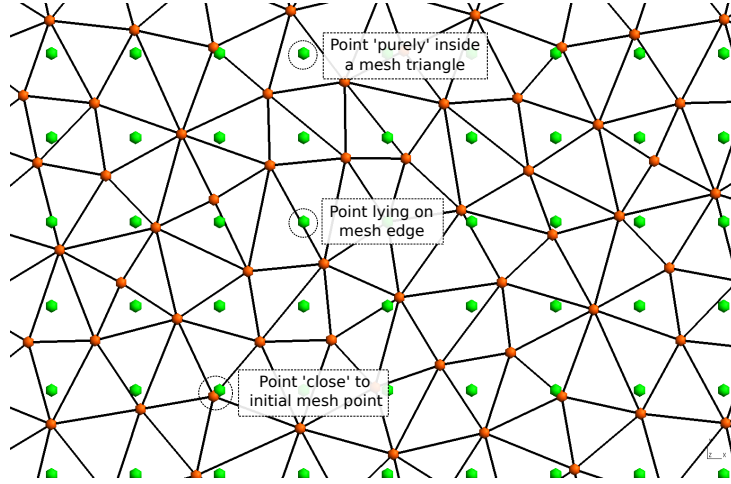


Figure 5.4: Generated points (green) - Initial mesh (red points to be removed)

The objective now is to create a surface mesh with the set of optimal points $P = \{\mathbf{p}_i \mid i = 1, \dots, N_g\}$ (where $N_g = N_{gl} + N_{gs}$) that have been generated on curves and surfaces. The idea is straightforward: connect the generated points P on the initial mesh $\mathcal{T}_0$, and subsequently remove the initial mesh points

(Figure 5.4). A similar idea but in a different context has been used in Ray et al. [2018], leading to a quad-dominant mesh.

Robustness is of crucial interest in our method, since modifying geometrical aspects of general surfaces in 3D space is a delicate task. With our approach, the main goal is to preserve the topological integrity of the mesh through each step of the process, while not compromising the accuracy of the geometrical representation of the surface.

We can identify two complementary sets of generated points $\mathbf{p}_i$:

- i. N_{gl} points lying on lines (boundary edges)
- ii. N_{gs} points lying purely on surface triangles

with a corresponding unique parent element (boundary edge or mesh triangle respectively) already stored for each of these points. Correspondingly, there are N_r points from the initial mesh: N_{rl} points of the boundary edges and N_{rs} points of the 'interior' surface. This division enables us to perform a bottom-up procedure where topological entities are handled independently (first mesh lines and then surfaces). We can therefore ensure that each step will have a well defined 'predecessor' mesh to build upon.

5.4.1 Local Mesh Modifications

The basic operations utilized to locally modify the mesh follow:

Split Triangles Given a surface point and its parent triangle, split it by replacing it by three triangles. This operation is trivial to implement since it cannot change the geometry or the topology of the mesh.

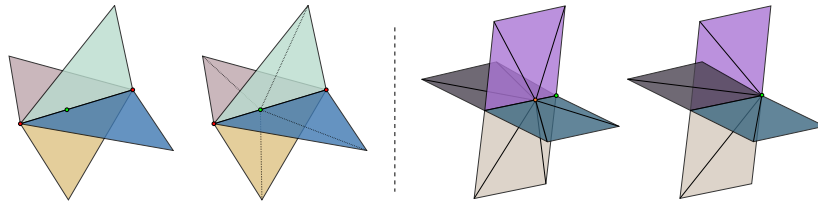


Figure 5.5: Splitting (left) and collapsing (right) a non-manifold feature edge

Split Edges Given a point that lies on a mesh edge, split this edge. For points on boundary edges we already know the parent edge, while for points on triangles it is easy to compute if the point lies on a triangle edge (given a user-defined threshold value ε). For non-manifold boundary edges, we split all

the corresponding triangles connected to this edge (Figure 5.5, left). This set of triangles is readily accessible from the extended data structure for boundary edges.

Collapse Edges Remove points from the triangulation by collapsing an edge. Collapsing an edge is not always a valid operation since it can create flipped or degenerate elements. We check the fan of n triangles connected to the vertex in question and choose an edge that can be collapsed. The resulting $n-2$ triangles should not intersect with neighbouring ones. Again, non-manifold edges can be collapsed if all corresponding 'half-fans' pass the validity test (Figure 5.5, right).

Swap Edges Given an appropriate quality criterion, swap the edge if the topology is not violated. Obviously, a boundary edge cannot be swapped. Edge swaps have a significant role during collapsing of edges, since it is not always feasible to collapse all unnecessary vertices at the first pass. Edge swaps serve at this point to create improved conditions for the next collapsing iteration. Collapsing numerous mesh vertices leads to steep angles; therefore we swap edges if it does not result to bigger dihedral angles. At the final step, swaps can be performed based only on quality criteria.

5.4.2 Outline

Algorithm 2: Direct Surface Meshing with Local Modifications

```

input : Initial triangulation  $\mathcal{T}_0$ 
         generated set of points  $P$  to be triangulated.
output: A new mesh  $\mathcal{T}_1$ 
for  $N_{gl}$  points on lines do
    | Insert points on mesh lines by splitting boundary edges;
for  $N_{gs}$  points on surfaces do
    | Insert point on mesh surfaces by splitting edges or triangles;
Swap edges of intermediate mesh  $\mathcal{T}_1$ ;
while  $N_r \neq 0$  do
    | for  $N_{rl}$  points on lines do
    | | Collapse initial points on mesh lines;
    | for  $N_{rs}$  points on lines do
    | | Collapse initial points on mesh surfaces;
    | Swap edges;

```

The procedure consists of the following (Algorithm 2). All generated points are flagged to be inserted while all initial points are flagged to be removed. Starting from the model lines, each generated point splits its corresponding parent boundary edge, which can be manifold or non-manifold. Each point has a unique parent boundary edge and the splitting is done based on its parameter $t \in [0, 1]$, thus defined in a unambiguous way. If a point to be inserted is 'close' (w.r.t to ε) to an initial mesh point, we flag the latter to be preserved instead of the former in order to avoid small-scale geometric ambiguities that may occur. The procedure follows until all points on model lines are inserted into the mesh. Following, points on surfaces are inserted by splitting triangles. These points have a unique parent triangle, and the splitting is based on the barycentric coordinates. If a point is 'close' w.r.t to ε on one of the triangle's edges, we split the edge, and if a point lies 'close' to an initial point, it gets disregarded (as explained for points on lines).

At this stage, we have an 'enhanced' intermediate mesh containing the initial (red) and generated (green) points (Figure 5.3c). This mesh is of low quality, but our interest here is that it represents as accurately as possible the underlying surface and respects the topology of the initial mesh. A quality improvement with edge swaps can be performed here, though it is not a necessary condition to continue to the next step.

We want now to remove the initial mesh points. Starting again from boundary edges, we iterate for all boundary points and examine if one of the two connected boundary edges can be collapsed (obviously, we do not want to collapse a boundary point inside the surface). Similarly with the procedure of splitting boundary edges, we can collapse non-manifold boundary edges without compromising the conformity of the mesh. Subsequently, points on surfaces are removed by collapsing interior edges. Since not all points are guaranteed to be removed in the first pass, the two discrete loops for points are encapsulated in a while loop that terminates when no point to be removed remains or breaks if the remaining 'red' points cannot be removed at all. This is a crucial part since it prevents the algorithm from producing invalid topology. When an initial mesh vertex cannot be collapsed, it can simply remain in the final mesh. In practice, we observe that no more than two iterations are necessary to remove unwanted vertices for most of the models tested. While inserting points in the triangulation by split edge/triangle operators is trivial, robustly implementing the collapsing step proved to be a fairly strenuous task for the general case.

5.5 Bipartite quadrilateral labeling

At this point, we have in hand a mesh $\mathcal{T}_1$ that is right-angled and ready to be transformed to an all-quad mesh by triangle pairing. A necessary condition for an all-quad mesh to exist is to have an even number of edges on the boundary

of each surface of the model. In this work, we trivially impose this condition on the feature curves that bound the surfaces during the first step of the pipeline (Section 5.3.1). Disappointingly, dividing each curve into an even number of segments is not sufficient to ensure the existence of a perfect matching, i.e., a triangle-pairing that involves all triangles of $\mathcal{T}_1$. This is a typical obstacle of triangle-pairing methods, leading to isolated triangles that cannot be locally removed. A naive way to eliminate those triangles is to perform a Catmull-Clark subdivision [Catmull and Clark, 1978], converting a quad-dominant mesh to an all-quad at the expense of vastly increasing the number of elements as well as the number of irregular vertices. In Blossom-Quad [Remacle et al., 2012] a more sophisticated solution is proposed by computing a minimum-cost perfect matching of the dual graph to offer a global solution. Still, ensuring a graph to contain at least one perfect-matching remains quite complicated.

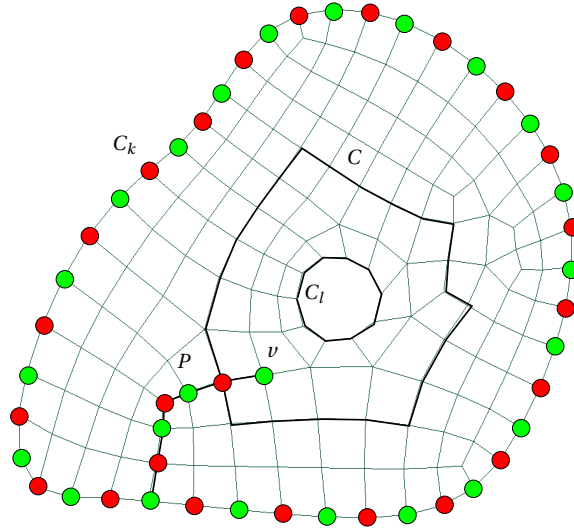


Figure 5.6: Every cycle C is even so the graph is bipartite and is 2-colorable. Starting from one 2-colored boundary C_k , it is possible to find the color of any vertex $v \notin C_k$ by 2-coloring any path P between C_k and v .

We propose here a new idea for constructing one perfect matching in a triangular mesh: all edges of the matching will be used to combine their two neighboring triangles and form a quad mesh.

Let us first recall some elements of graph theory. A graph labeling is the assignment of labels to the nodes of the graph. A graph coloring is a special case of graph labeling; it is an assignment of labels traditionally called *col-*

ors to the nodes of the graph with the constraint that the graph contains no monochromatic edge, i.e., no edge connecting two nodes with the same color.

A *bipartite graph* $\mathcal{G}$ is a graph that can be 2-colored. An important property of a bipartite graph is that *it does not contain any odd-cycles*¹.

Let us now look at the quadrilateral mesh of a domain Ω as a graph $\mathcal{G}$. We assume here that the boundary $\partial\Omega$ of Ω is divided into n separated sub-boundaries $\partial\Omega_i$, $i = 1 \dots n$, with each of the sub-boundaries $\partial\Omega_i$ forming a boundary cycle C_i in $\mathcal{G}$. We assume that each C_i is an even-cycle. Under these conditions, it is easy to see that $\mathcal{G}$ is bipartite. We consider a cycle C of $\mathcal{G}$: C bounds a quadrilateral mesh and it is known [Beaufort et al., 2017b] that every quadrilateral mesh has an even number of boundary vertices. If the domain $\mathcal{D}$ that is enclosed by C is simply connected, then C is its only boundary and C is an even-cycle. If $\mathcal{D}$ is not simply connected, its boundary contains other boundary cycles C_l that are by hypothesis even-cycles. Thus, ensuring that every boundary cycle C_k is even is sufficient to ensure that any other cycle C is even as well. Figure 5.6 illustrates the aforementioned reasoning. It should be noted that not all quad meshes are bipartite: it only holds when every boundary cycle is even.

It should be noted as well that, starting from the 2-coloring of one of the boundary cycle C_k , the 2-coloring of the rest of the graph is unique: the color of any vertex $v \notin C_k$ is found by coloring any path P between any vertex of C_k and v (see Figure 5.6).

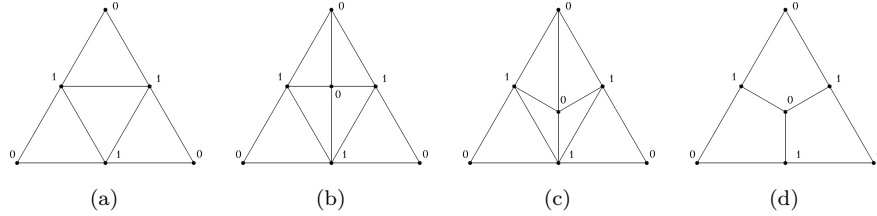


Figure 5.7: Bipartite topological correction. (a) Initial labeling on the even-sided boundary. (b) Splitting topologically inconsistent triangle. (c) reposition of vertex according to opposite labelled stencil. (d) recovery of quadrilaterals (defined by same-labelled edges).

Now let's come to our point. Assume a triangulation of Ω and a bipartite labeling of its vertices. The graph of a triangulation is obviously not a bipartite graph – every triangle forms an odd-cycle and therefore it cannot be 2-colored.

Assume now that the labeling is done in such a way that there exist no monochromatic triangle (a triangle is monochromatic if its three vertices have

¹An odd-cycle is a cycle of odd length.

the same label). Then, the set of monochromatic edges of the triangulation forms a perfect matching and removing monochromatic edges thus leads to the desired quadrangulation. The proof is simple: every triangle of the mesh has exactly one monochromatic edge, which means that each triangle will be chosen exactly once for creating a quad: this is by definition a perfect matching. The only possible issue would be the existence of monochromatic edges on $\partial\Omega$. This issue is avoidable if every boundary cycle is even: coloring should be done at first on $\partial\Omega$ and then be propagated inside.

Now the last question: is it possible to avoid monochromatic triangles? The answer is no, at least not without modifying the triangulation. We start by analyzing the simplest possible case of an even triangulation without perfect-matching (see Figure 5.7a). In this example, all labelings leave the internal triangle monochromatic so there exists no perfect matching in this graph. In other words, merging any of the three possible pairs of triangles would leave two non-connected triangles 'hanging'. In general, picking edges in the graph while constructing a matching may split the graph into disconnected sub-graphs with an odd number of edges on their boundaries, thus not allowing a perfect matching. This condition is a special form of a well-known graph theory theorem from Tutte [Bondy and Murty, U. S. R, 1976]. By inserting an additional Steiner point of the opposite label (by splitting one edge of the triangle in question) our condition is met (Figure 5.7b). A perfect matching actually exists and is composed of all monochromatic edges as demonstrated above (see Figure 5.7d).

5.5.1 Random labeling

Let us start by 2-coloring the boundary cycles C_k of Ω and assign a random label to all internal vertices. Figure 5.8(a) shows this initial random labeling. Random labeling is not the worst-case scenario: it typically produces 25% of monochromatic triangles. Yet, this number is too big because the number of Steiner points that should be added to remove all those monochromatic triangles is sufficiently large to damage the mesh size field.

It is possible to dramatically decrease the number of monochromatic triangles by applying the following *smoothing algorithm*. Each internal vertex is re-labeled if changing its label reduces the number of (its adjacent) monochromatic triangles. The smoothing usually ends with monochromatic triangles that are either isolated or form adjacent pairs (see Figure 5.8(b)). In this specific example, no pairs are observed but they cannot be avoided in the general case.

Then, bipartite topological correction is performed. When monochromatic triangles appear in pairs, their common edge is split. When they are isolated, we choose to split their longest edge, to not degrade the accordance with the size-

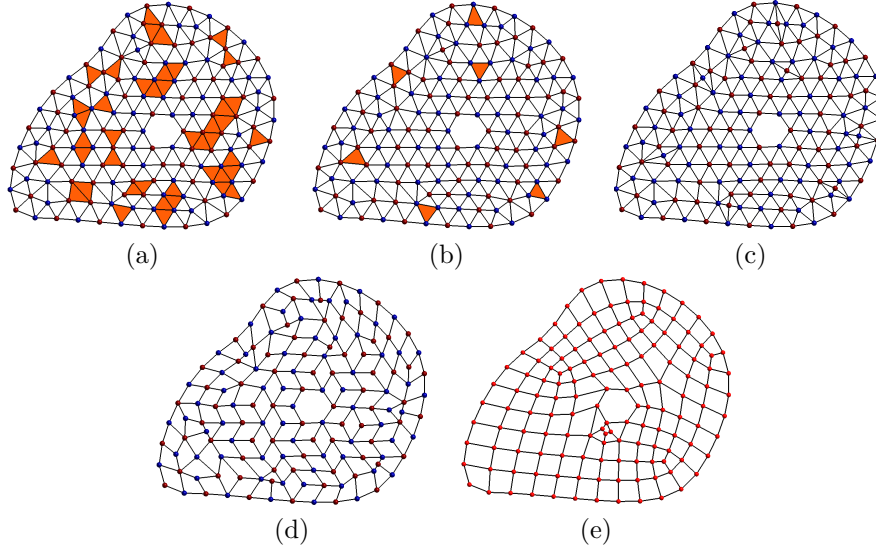


Figure 5.8: Different stages of the quadrangulation process. (a) Random labeling of the internal vertices, monochromatic triangles are colored. (b) Local label smoothing. (c) Splitting of remaining isolated monochromatic triangles. (d) Merging triangles using monochromatic edges and obtention of a 2-coloring. (e) Final quad mesh after topological optimization.

field. This process continues until all monochromatic triangles are eliminated. Figure 5.8(c) shows the final labeling without any monochromatic triangle.

The triangular mesh of Figure 5.8(c) can then be transformed into a quadrilateral mesh by removing its monochromatic edges as depicted in Figure 5.8(d). Mesh of Figure 5.8(d) is not so good (and it is a euphemism). We had to use all Gmsh's heavy optimization weaponry to obtain the final mesh of Figure 5.8(e). We must confess here that we were not expecting a random labeling to provide such a good mesh: this tells a lot about the excellent performances of our recent optimization strategy for quad meshes (Chapter 6).

5.5.2 Cross field guided labeling

The triangular mesh of Figure 5.8(a) is not suited to be transformed into a quadrilateral mesh; obtaining a good mesh at the end heavily relies on advanced optimization. In Section 5.3.2, we proposed an advancing front scheme where each point tries to add other points in 4 orthogonal directions provided by a cross-field. The cross-field guided point insertion process essentially produces

an excellent labeling scheme: new vertices added by a vertex v have the opposite label of the one of v (since the edges formed between these vertices will be the cross-field aligned ones that we want to have in the final quad mesh). Figure 5.9a shows the initial labeling after frontal point insertion. As few as 14 monochromatic triangles are present in the triangular mesh, leading to a very small amount of Steiner points. More importantly, no Steiner point has been inserted at the vicinity of the boundary, leading to several perfect layers of quads (see Figure 5.9d).

At this stage, we have constructed the underlying topological connectivity of an all-quad mesh (Figure 5.9b). Mesh vertices can be repositioned at the center of the stencil of opposite labeled neighbors (essentially a single iteration Laplacian smoothing, e.g. Figure 5.9c). In the case of non-planar surfaces, the repositioned vertices are projected onto the original triangulation. Further and more suited smoothing (e.g., Winslow smoothing [Knupp, 1999]) can be performed on the final quadrilateral mesh. We can then trivially extract an all-quad mesh, since each pair of triangles whose common edge has two same labels defines a quadrilateral (Figure 5.9d). At this point, the advancing labeling scheme is converted to the true 2-coloring of the final bipartite quadrilateral mesh. Another way to see this is that the set of same-colored edges is perpendicular to the set of the edges of the perfect-matching of the dual. It is interesting to note that forming the quads this way before inserting the additional vertices would lead to a quad-dominant mesh with non-quad polygonal regions that are even-sided due to the bipartite condition.

Finally, it must be noted that our algorithm is straightforward, and linear in time compared to the computation of an optimal perfect matching, which is quadratic in time and very complex. For the sake of completeness, the algorithm is presented on Algorithm 3.

5.5.3 Topological optimization

Quad mesh $\mathcal{T}_q$ is usually of high quality since it follows the cross-field directions. However, it can further be optimized, both geometrically and topologically (Figure 5.12). We are not going into the details of the optimization step. The next step in our developments will be to complement our approach with the procedure proposed in Chapter 6 that takes as input the quad mesh $\mathcal{T}_q$, the cross-field $\mathbf{f}$ as well as the size field h . The main features of this optimization process are:

- All vertices of high valence are eliminated.
- All boundary vertices have their optimal valence in such a way that we can create a boundary layer mesh without effort.

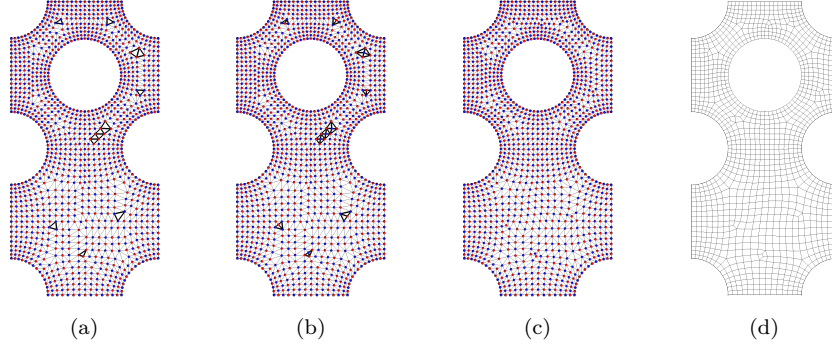


Figure 5.9: Bipartite all-quad meshing. (a) Topologically inconsistent triangles. (b) Insert vertices (Steiner points). (c) Reposition of vertices. (d) Bipartite all-quad mesh.

Algorithm 3: All-Quad Surface Meshing

input : (Right-Angled) Triangulated Surface $\mathcal{T}_r$
 binary label for each vertex $v_i \in \mathcal{T}_r$
output: All-quad mesh $\mathcal{T}_q$

for N_t *triangles* **do**
 | Flag triangles with 3 vertices of same label;
for N_e *edges* **do**
 | **if** adjacent triangles t_i and t_j are flagged **then**
 | Split edge;
for the rest (isolated) flagged triangles **do**
 | Split longest edge of the triangle;
for N_e *edges* **do**
 | **if** edge vertices have the same label **then**
 | Form a quadrilateral from the two adjacent triangles of the
 | edge;

- Isolated irregular vertices corresponding to the singularities of the cross field are preserved.
- Advanced vertex relocation schemes are performed to obtain a geometry regular quad mesh.

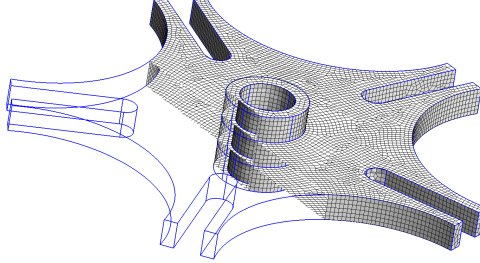


Figure 5.10: CAD model with multiple volumes, leading to non-manifold feature curves.

5.6 Results

To demonstrate the capabilities of our methodology, we have tested the whole pipeline on a variety of cases with diverse characteristics (Figure 5.11). One of the most challenging aspects of our methodology is the removal of the initial mesh points by collapsing and testing multiple 'extreme' cases on that is an ongoing process. As desired, non-manifold features are well-handled and conformity is preserved. The final conversion to an all-quad mesh proves to be straightforward and efficient, given that the input respects the fundamental topological condition (i.e., even number of edges on each curve). In certain cases, especially when the requested size field is much coarser than the geometric characteristics of the model, we may encounter minor defects such as doublet quads or flat quads on the boundary (since boundary vertices cannot be repositioned). Both cases can be locally handled, the former by merging the doublet into one quad and the latter by inserting additional vertices and thus 'pushing' the boundary irregular vertex inside the domain.

A desired property of our mesh generation process is to keep user intervention at a minimum. Input parameters are a threshold value ε to suppress potential discrepancies during local mesh modifications and the angle threshold to detect hard edges. This angle threshold may be disregarded, and the algorithm can initiate from a random edge of the initial mesh instead of the model curves. We keep in mind that this may result in low-quality meshing on hard edges.

Cross-fields are precomputed and considered input to the algorithm, or they can be taken as the unit axis directions for practical purposes. Size-fields can be a uniform input size or computed from the curvature and geometric characteristics of the input mesh. Both cross and size fields impact considerably

the output mesh quality, and we can observe that the directionality of the elements matches the input cross-field directions. The output meshes shown here consist of the 'raw' quadrilateral meshes, without any smoothing or further optimization (an example of further optimizing a quad mesh is presented on Figure 5.12).

One of our main interests is the computational efficiency of the method. The majority of operations are local and the algorithms of linear complexity. The most time-consuming parts of our pipeline is the (inevitable) use of spatial search structures (involved in filtering and also in projection of points onto the triangulation in the all-quad conversion). Indicatively, the generation of quad mesh of $\sim 100k$ vertices takes on average ~ 1 second for the generation of points, ~ 2.5 seconds for the surface meshing and less than 1 second for the all-quad conversion, with a single thread.

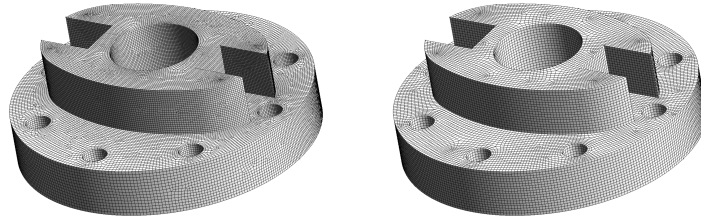


Figure 5.12: Left: unstructured quad mesh produced with our pipeline, where we can observe the excess of irregular vertices. Right: quasi-structured quad mesh obtained with the topological optimization presented on Chapter 6.

5.7 Conclusions

We have presented a method for the creation of quadrilateral surface meshes on general complex geometries. One of the important aspects of our pipeline is its modular nature. Each step can be independently used, for example, a more efficient point sampling method could be used, or a typical CAD meshing algorithm to generate the mesh with these points.

The surface meshing algorithm proposed is used to produce right-angled triangles, but it can be repurposed as a re-meshing tool for triangulations, given appropriate input (i.e. an 'asterisk' field [Beaufort et al., 2017b; Georgiadis et al., 2017a]). The proposed two-step method for surface meshing aims at balancing the trade-off between output quality and robustness. One of its strengths is the ability to handle the model as a whole and thus create cross-aligned

elements globally. While producing high-quality elements in this aspect, we always respect the topology and succeed at meshing small-scale and non-manifold features as shown in the results demonstrated (Section 5.6).

A novel method to obtain an all-quad mesh from any quad-dominant mesh with minimal, if any, post-processing clean-up is introduced. By using a bipartite labeling scheme, we simplify the global treatment required for quad meshing to a set of localized operations that can be performed on a quad-dominant mesh. We are thus enabled to produce always a high-quality quad mesh, regardless of the complexity of the model. Our meshes can be further optimized with a strategy to remove the majority of irregular vertices, presented on Chapter 6.

We are currently in the process of incorporating this quad surface mesher as a component of a general high-quality hex-dominant meshing pipeline, where it will be used as the volume bounding surface. An interesting open problem to this end is if hexahedral element generation should be constrained by a quadrilateral mesh, and if yes, what would be its optimal characteristics. Besides that, we are investigating ways to take advantage of the topological labels used here for quadrilateral elements to the generation of hexahedra, possibly with a labeling scheme that can codify more topological information.

CHAPTER



Quasi-structured quadrilateral meshing.

This chapter is a reproduction of the following paper:

Reberol, M., Georgiadis, C., Remacle, J.-F., submitted, *Quasi-structured quadrilateral meshing in Gmsh – a robust pipeline for complex CAD models.*

Abstract

We propose an end-to-end pipeline to robustly generate high-quality quadrilateral meshes for complex CAD models. A initial quad-dominant mesh is generated with frontal point insertion guided by a locally integrable cross field and a scalar size map adapted to the small CAD features. After triangle combination and midpoint-subdivision into an all-quadrilateral mesh, the topology of the mesh is modified to reduce the number of irregular vertices. The idea is to preserve the irregular vertices matching cross-field singularities and to eliminate the others. The topological modifications are either local and based on disk quadrangulations, or more global with the remeshing of patches of quads according to predefined patterns. Validity of the quad mesh is guaranteed by monitoring element quality during all operations and reverting the changes when necessary. Advantages of our approach include robustness, strict respect of the CAD features and support for user-prescribed size constraints. The quad mesher, which is available in Gmsh, is validated and illustrated on two datasets of CAD models.

Keywords: quadrilateral meshing, unstructured meshing, computer-aided design

6.1 Introduction

There are essentially two main scientific communities that have been historically interested in generating quadrilateral meshes: the engineering analysis community, as these meshes are good geometrical support for the finite element and finite volume methods, and the computer graphics community which, besides application in numerical simulations, has good use for them in surface modeling (*e.g.* subdivision surface) and texturing.

The engineering analysis community has been working on algorithms for generating quadrilateral meshes for several decades, leading to two main categories of industrial-grade techniques. The first one is to manually decompose the CAD model in quadrilateral patches, with potentially some semi-automatic assistance. The patches are then filled with quads while considering user element sizing specifications, *e.g.* anisotropic quads with geometric progression from the boundary layers. This approach is very time-consuming and is mainly used for demanding numerical simulations such as CFD. On the other end, various fully

automatic approaches have been developed for generating unstructured quadrilateral meshes, such as paving [Blacker and Stephenson, 1991] or Blossom-Quad [Remacle et al., 2012] with frontal Delaunay point insertion [Remacle et al., 2013]. Their disadvantages are the potentially high number of irregular vertices, the non-regular alignment of edges in the mesh (affecting smoothness of gradients in simulations) and the lack of user control. Both approaches are now available in most mesh generation packages, including Gmsh [Geuzaine and Remacle, 2009].

In the last decade, the computer graphics research community has thoroughly developed and explored cross-field based techniques for quadrilateral meshing, with a focus on the automatic generation of high-quality block-structured quadrilateral meshes, or equivalently coarse *quadrilateral layouts* which partition the object surfaces into conforming quadrilateral patches. Finite element practitioners quickly became interested in these ideas because of their potential to replace semi-automatic block-structured mesh generators by fully automatic ones. A fruitful line of research revolves around the construction of a global parametrization from which we can extract an all-quadrilateral mesh. The global parametrization is a discontinuous mapping that maps a 2D grid from the parameter domain to the model surface, with discontinuities to allow for the existence of irregular vertices (Sec. 6.1.2 for more details). The work presented in the current paper started from an attempt to apply such a global parametrization technique in the context of CAD meshing in Gmsh, as explained later (Sec. 6.2).

The quasi-structured quadrilateral meshing pipeline that we propose exploits the strength of both worlds. We rely on the robustness of automatic unstructured quad meshing techniques, and we use the topological and geometrical information from cross fields to transform unstructured meshes into high-quality ones, with few but well located irregular vertices. Our approach has many advantages which are important for CAD meshing: it always produces valid quadrilateral meshes, it strictly respects all CAD features and it supports non-uniform element size maps. The new quad mesher is integrated in Gmsh (version 5 and higher) [Geuzaine and Remacle, 2009] and its implementation is open-source.

6.1.1 Overview

This paper proposes a robust end-to-end pipeline to build quasi-structured quadrilateral meshes for arbitrary CAD models. Our only requirement of the input CAD model is that it is valid in the sense that it is possible to generate a triangulated mesh with standard techniques. This includes non-manifold surfaces, which typically appear when the CAD models contains multiple volumes. By quasi-structured quadrilateral mesh, we mean that the number of

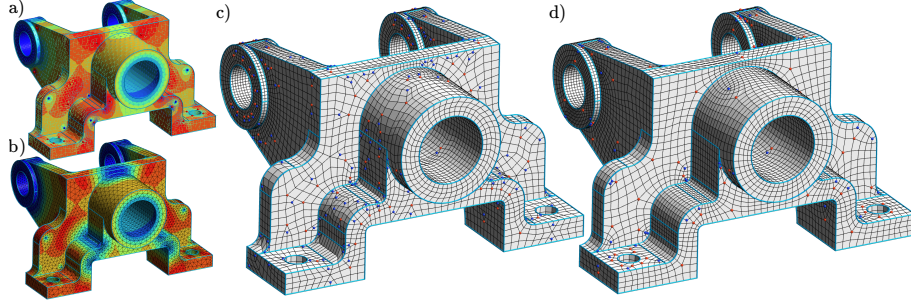


Figure 6.1: The quasi-structured quad meshing pipeline applied on the model $M3$ [Ledoux] (111 CAD corners, 169 curves, 60 faces). The cross field and its associated conformal scaling is computed on a initial triangulation (a). The size map (b) combines the cross field scaling and is adapted to the small CAD features. The initial unstructured all-quadrilateral mesh (c) is computed with a frontal approach, and contains 3429/54982 irregular vertices. After cavity remeshing, the final quasi-structured mesh (d) contains 606/56743 irregular vertices. 78 of them match the cross field singularities (a) and the others allow the mesh size transitions (b).

irregular vertices is small compared to the usual unstructured techniques, and close or equal to the number of irregular vertices obtained with cross-field based parametrization approaches when the target mesh size is smaller than the CAD features (*e.g.* Fig. 6.2). Our meshing pipeline, illustrated on Fig. 6.1, is:

- Step 1: Compute a scaled cross field on each CAD face, using an initial triangulation. The cross directions are computed with a fast multilevel heat diffusion approach and a conformal scaling is obtained by solving a linear system, see Sec. 6.3.
- Step 2: Compute a global metric field (combined direction field and scalar size map) from the prescribed sizing constraints, the small CAD features and the conformal scaling associated to the cross field, see Sec. 6.4.
- Step 3: Generate the meshes of the CAD curves considering the size map and global topological constraints, see Sec. 6.4.3.
- Step 4: Generate the quasi-structured quadrilateral meshes of the simple CAD faces which match predefined patterns, see Sec. 6.6.1.2.
- Step 5: Generate unstructured quadrilateral meshes on the remaining CAD faces, using a frontal approach guided by the cross field scaled by the size map, see Sec. 6.5.

Step 6: Modify the quadrilateral mesh topology to make it quasi-structured, by removing defects with local disk-quadrangulation operations (Sec. 6.6.3) and by remeshing convex cavities to match the cross field singularities (Sec. 6.6.2).

With this pipeline the ultimate focus is on robustness, which has precedence over all other considerations. The unstructured quad mesh is geometrically and topologically valid by construction (quad-dominant mesh subdivided into full-quad). For all the remeshing operations, we always verify that the topology and the geometry remain valid, and we reject the ones which invalidate or degrade the element qualities.

The scaled cross field guides the unstructured point insertion and its singularities are used to constrain the cavity remeshing. All the steps produce valid outputs even if the cross field is not accurate or is topologically invalid. The key idea to robustly deal with complex CAD models is to use the cross field as an auxiliary tool without entirely relying upon it, since it may and usually contains many defects (does not take small CAD features or quantization into account, may have wrong singularities, may have insufficient accuracy).

Figure 6.2.2. illustrates our pipeline for a single surface. In practice, this surface should be seen as a particular CAD face of a larger model, so its 1D boundary mesh cannot be modified without affecting its neighbors. This compatibility constraint between the meshes of adjacent CAD faces is one of the major sources of difficulty when dealing with complex CAD models. We can look at a more realistic CAD model on Fig. 6.1, where the CAD faces are meshed independently. After topological improvement (Fig. 6.1.d.), there are still irregular vertices that do not match singularities from the cross field. These are due to the curve quantization which follows the size map (Fig. 6.1.b.), and which is necessary to take small CAD features into account. For more detailed discussions and applications to complex models, see Sec. 6.7.

Because we present an end-to-end mesher, a lot of ground is covered and it is not practical to describe everything in details. We try to provide extensive references to the literature so the reader can refer to them for further details on specific parts of the pipeline.

6.1.2 Related work

Unstructured quadrilateral meshing A classic approach to generate unstructured quadrilateral meshes is to start from the boundary and to progressively insert points in the surface. The *paving* algorithm [Blacker and Stephenson, 1991] directly generates quadrilateral in an advancing-front fashion. Indirect methods first generate a triangulation with well-placed vertices

then merge pairs of triangles into quads [Lee and Lo, 1994; Borouchaki and Frey, 1998]. There are many ways on how to choose the merged triangles. In Q-Morph [Owen et al., 1999], triangles are transformed in quadrilaterals with an advancing-front algorithm. BlossomQuad [Remacle et al., 2012] computes a global perfect matching to optimally pair triangles, which works especially well with right-angled triangles generated by Remacle et al. [2013].

These methods generally suffer from degraded mesh quality and high vertex irregularity, especially at front collisions. But on the other hand, they are highly robust. Even when a pure quad mesh cannot be obtained directly, it's always possible to perform a midpoint subdivision, also called Catmull-Clark subdivision, to obtain an all-quadrilateral mesh by splitting the triangles into three quads and the initial quads into four sub-quads.

Cross fields In the last decade, cross fields [Hertzmann and Zorin, 2000; Ray et al., 2006; Palacios and Zhang, 2007] have become very popular for quadrilateral meshing. Crosses can be seen as infinitesimal quads and they have topological structures which match quad mesh topology, such as cross singularities which have one-to-one correspondence with quad mesh irregular vertices [Bunin, 2006; Ray et al., 2008].

Cross field can be computed by minimizing a smoothness energy, either by directly optimizing the angles [Bommes et al., 2009] or by smoothing a vector representation with a way to ensure that the variables stay crosses, such as Ginzburg-Landau penalization [Beaufort et al., 2017b] or with iterative projection [Viertel and Osting, 2019]. In this paper, we use the MBO method [Viertel and Osting, 2019] where we alternate diffusion and projection, with varying time stepping [Palmer et al., 2020] and additional optimizations (Sec. 6.3.1).

It is important to note that cross fields are not *integrable* by default. On a cut domain, a cross field can be seen as two orthogonal unit vector fields. To have integrable vector fields, it is necessary to break the orthogonality [Sageman-Furnas et al., 2019] or to scale the amplitudes. Ray N. et al. Ray et al. [2006] introduce a correction step to make the two vector fields curl-free. Another approach is to solve a linear problem to compute a global conformal scaling from the cross field singularities [Bunin, 2008; Ben-Chen et al., 2008], which can be applied to both vector fields. We compute a similar conformal scaling (Sec. 6.3.3) by canceling a Lie bracket, without using the cross field singularities.

There are various ways to exploit the cross field information for quad meshing. It can be used as a directional guide for frontal point insertion, as in Remacle et al. [2013] and in our unstructured initialization (Sec. 6.5). A popular alternative is to directly build the quadrilateral layout by tracing separatrices in the vector fields, *e.g.* [Campen et al., 2012; Myles et al., 2014; Pietroni et al.,

2016]. Another common approach is to *integrate* the cross-field by computing a parametrization.

Cross field based parametrization A parametrization associated to a cross field is usually made of two scalar fields u, v such that their gradients are aligned with the cross field directions. Such parametrization is a mapping from the 2D parameter space to the 3D surface. The mapping discontinuities, which are cuts on the model surface, allow for the existence of irregular vertices on the quadrilateral mesh. QuadCover [Kälberer et al., 2007] introduced globally coherent parametrization to *integrate* a cross field, with integer grid-preserving constraints on a cut-graph to ensure global coherency. This idea have been highly refined and improved in the form of integer grid mapping [Bommes et al., 2009, 2013a; Campen et al., 2012; Ebke et al., 2013]. With these approaches, the irregular vertices in the final quad mesh strictly match the cross field singularities. Alternatively, the parametrization formulation can be relaxed, either by computing local parametrizations [Jakob et al., 2015] or a loosely constrained global parametrization which relies on periodic functions [Ray et al., 2006]. In both cases, there are additional irregular vertices in the final quad mesh introduced by the relaxation. Our experience with a global parametrization attempt is detailed in Sec. 6.2.

Topological meshing and modifications Unstructured quadrilateral meshing methods do not naturally offer the regularity and structure obtained with global parametrization methods. Is it possible however to reduce the number of irregular nodes and improve quality by performing topological modifications on an unstructured quad mesh. Due to topological constraints in quadrilateral meshes (Sec. 6.1.3), there is no simple local operation such as the edge-split or edge-collapse, but there are still many possibilities, some global such as chord collapsing [Daniels et al., 2008] and other more local [Docampo-Sánchez and Haimes, 2019] but which are restricted to specific vertex-valence configurations.

In this work, we are particularly interested in operations that modify the content of a patch of quads inside the mesh, called cavity, without changing its boundary. For triangular and pentagonal cavities, Bunin G. [Bunin, 2008] has shown that we can find a topological quad mesh replacement with a single irregular vertex by solving a linear system. This local clean-up is a fundamental piece of the Jaal quad mesher [Verma and Tautges, 2011], with a later extension to more generic replacement patterns [Verma and Suresh, 2015].

This problem of remeshing a cavity is equivalent to the problem of topologically finding a quadrilateral mesh for a given polyline boundary. It has been studied in the computer graphics context [Yasseen et al., 2013], with a generic integer linear formulation for generic N -sided patches [Takayama et al., 2014]. The

local remeshing operator has been used in interactive software to help users improve their quad meshes [Marcias et al., 2015].

6.1.3 Topology of quadrilateral meshes

In a polygonal mesh, the Euler-Poincaré formula $\chi = n - n_e + n_f$ provides a relationship between the n vertices, the n_e edges, the n_f facets and the Euler-Poincaré characteristic χ of the associated surface, which is given by the relation $\chi = 2 - 2g - b$ with b the number of holes and g the number of handles.

In a quadrilateral mesh, we can follow Beaufort et al. [2017b] to derive the relationship

$$4n_f = \sum_k 4(n - n_b - n_k) + 2(n_b - m_k) + (4 - k)n_k + (2 - k)m_k$$

between the number of vertices of various indices. m_k is the number of boundary vertices with valence $2 - k$ and n_k the number of interior vertices with valence $4 - k$. When $k \neq 0$, the vertices are said to be irregular of index k . By combining this relation with the Euler formula, we obtain [Beaufort et al., 2017b] the relation (6.1) between the number of irregular vertices on the boundary and on the interior in a quadrilateral mesh:

$$\chi = \sum_k \frac{k}{4}(n_k + m_k) \quad (6.1)$$

This formula has some interesting topological implications: regular vertices do not count, pairs of irregular vertices of opposite indices cancel themselves (*e.g.* valence three ($k = 1$) and valence five ($k = -1$)), surfaces with $\chi = 0$ (topological ring) are the only ones that admit fully regular quadrilateral meshes (only valence four inside and valence two on the boundary).

We will see later in this paper that it is possible to generate quadrilateral meshes with interior irregular vertices of indices $+1$ and -1 only, corresponding respectively to quadrilateral valences of 3 and 5. This restriction to irregular vertices of low index is usually better to generate high-quality quads, as irregular vertices are associated with geometric distortions. In this paper, we thus restrict ourselves to irregular boundary vertices of valence 1, 3 and 4, *i.e.* of indices 1, -1 and -2 , and to interior irregular vertices of valences 3 and 5, *i.e.* of indices $+1$, -1 . The relation (6.1) between the surface Euler-Poincaré characteristic and the quadrilateral mesh irregular vertices becomes:

$$n_1 - n_{-1} = 4\chi + m_{-1} - m_1 + 2m_{-2} \quad (6.2)$$

We say that a quadrangulation of a surface with Poincaré characteristic χ is *minimally irregular* if it contains exactly $4|\chi|$ isolated irregular vertices of

index $\text{sign}(\chi)$ (counting the boundary irregular vertices). For example, in the surface of Fig. 6.2, we have $\chi = -6$ and $m_1 = 4$ convex corners, so we should have $n_{-1} = 28$ irregular vertices of valence five to be *minimally-irregular*.

A dipole is a pair of irregular vertices with opposite indices $+1$ and -1 . Adding *dipoles* to a minimally irregular quad mesh may be necessary, for example to allow mesh size variations. A mesh is called *quasi-structured* if it contains $4|\chi|$ isolated irregular vertices of index $\text{sign}(\chi)$ plus a small amount of dipoles. We use the loosely defined adjective *small* because it is *a priori* impossible to determine the minimum amount of dipoles that are necessary for a given application.

6.2 Context and motivation

The more general context around this article is our effort to build a robust high-quality quadrilateral mesher in *Gmsh* that actually *works*. What we mean is that a user should be able to produce a high quality quad mesh on any valid CAD model by only *loosely* specifying the target element sizes, as it is the case for triangular meshing.

Global parametrization Our initial attempt, inspired by state-of-the-art global parametrization techniques [Bommes et al., 2009; Campen et al., 2015], was to build a seamless uv -parametrization from a cross field and then extract a quad-layout. Figure 6.2.1. illustrates the different steps for the construction of a quadrilateral mesh with this approach. Parametrization based techniques cut and flatten the surface of the domain. The cuts (in green on Figure Fig. 6.2.1.b.) allow to find a continuous branch of the cross field and to build a seamless uv parametrization (see Fig. 6.2.1.b.) by solving a linear system with an alignment energy (gradients of u and v must be parallel to the cross field) and specific grid-compatibility linear constraints on the cut graph. To avoid the technical difficulties associated with the integer variables in the standard mixed-integer formulation [Bommes et al., 2009; Kälberer et al., 2007], we directly extracted the (conformal) quadrilateral layout from the iso-value separatrices in the uv parametrization (Fig. 6.2.1.c.). We replaced the integer variables and associated global quantization [Campen et al., 2015] by explicit topological simplification (chord collapse) of the quad layout, since we initially had lot of very thin quad patches as we were doing no snapping or geometric thresholding. The resulting quadrilateral mesh (Fig. 6.2.1.d.) was of very high quality, when it worked.

Even though we were quite excited with our initial results, we failed to generalize the approach into a robust mesher able to handle large and complex CAD models. It took us considerable amount of time and energy to obtain an implementation that is still in a very beta version state. The amount of non-reusable

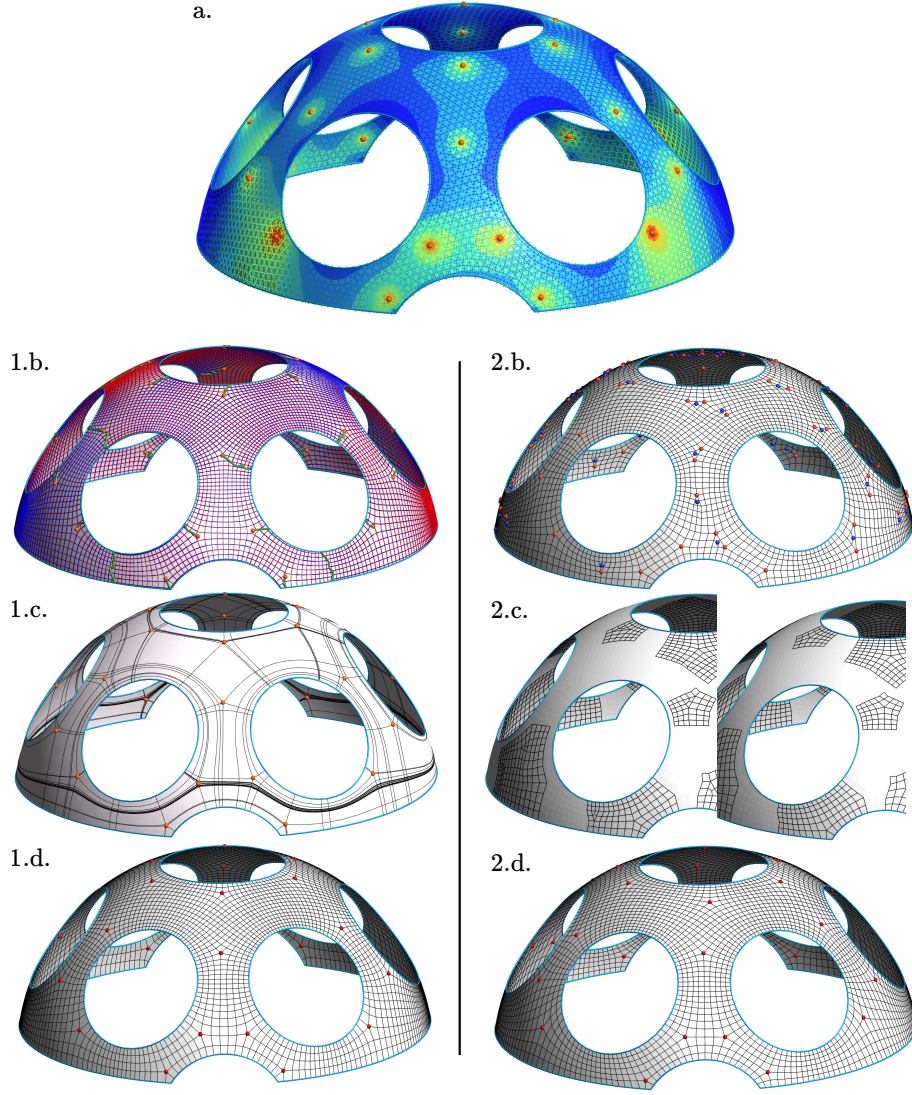


Figure 6.2: Two approaches to build a quasi-structured mesh of a surface. In the global parametrization approach, we build a seamless parametrization (1.b.) by integrating a scaled cross field (a.), then we extract a conformal quadrilateral layout (1.c.) and build the final quad mesh (1.d.) by simplifying the subdivided layout. In the second approach (this paper), we first build an unstructured quadrilateral mesh (2.b.) with a frontal approach guided by the scaled cross field (a.), then we apply cavity remeshing (2.c.) to eliminate unnecessary irregular vertices while preserving the cross field singularities. The final quad mesh (2.d) is very similar to the one obtained with the global parametrization approach and has the same number of irregular vertices.

code that is required for generating the quad mesh of Fig. 6.2.1.d. is quite large and only works when the cross field topology is in perfect accordance with the topology of the surface to mesh, which is often not the case when smoothing continuous representations of crosses. On a given CAD face, the difficulty mainly comes from the linear system that computes the uv -parametrization: it must be strictly right. If there is a small incoherence in the grid-compatibility constraints, *e.g.* due to wrong flagging of singularities, there will be no solution, or a useless one. Even more difficult is the generalization to large CAD models with hundreds of CAD faces, potentially non-manifold at CAD curves if there are multiple volumes. To avoid separatrices winding up around the models for thousands of turns before coming back to their origin, we build independently the quad layouts on each CAD face and then try to reunite them. The union of the quad layouts has many T-junctions on the CAD curves. One potential idea is to use a global integer quantization [Campen et al., 2015] to assign compatible non-zero integers to each edge of the global quad layout (with T-junctions). However, we soon discovered that often there is no non-degenerate solution: the constraints can be circular, *e.g.* $a + b = c, c = d, d = e, e = a$ leading to $b = 0$.

While some issues could be fixed, we do not foresee a robust implementation of that algorithm that would work for industrial grade models within a reasonable timescale. Moreover, such long codes are prone to bugs and are difficult to maintain. Of course, the mesh shown on Fig. 6.2.1.d. is exactly the what most finite element practitioners are willing to use. The main question became:

Can we obtain such high quality mesh using an approach that is both simple and robust?

Unstructured quad meshing Our experience in indirect quad meshing [Remacle et al., 2012, 2013] has been positive in term of industrial use. Indirect quad meshers are essentially based on existing surface meshers (very small amount of extra coding). The transformation of a triangular mesh into a quad mesh by combination of pairs of triangles is an extra step that is simple, robust and efficient [Remacle et al., 2012; Borouchaki and Frey, 1998]. Indirect quad meshers are thus able to treat models of arbitrary complexities with a run time that is not larger than standard triangular meshing. The main drawback of quad meshes generated in an indirect fashion is that they cannot be considered as block-structured or quasi-structured because they contain an arbitrary large number of irregular vertices.

All those reasons have convinced us to move to a new approach that would take *the best of both worlds*. We have decided to use our existing indirect meshing technology and build an *initial unstructured quad mesh* (Fig. 6.2.2.b.) that

is mostly aligned with a *cross field*. Then, information provided by the cross field is used to modify this unstructured mesh and make it *quasi-structured*, *i.e.* with only a few irregular vertices.

Besides working even if the cross field is inaccurate, there is another big advantage: we can use an external non-uniform size maps to locally control the mesh size, independently of the cross field. In practice, we compute a size map adapted to the small CAD features, which has proven to be invaluable when dealing with complex models.

Gmsh surface-to-volume philosophy Gmsh's usual meshing pipeline is essentially "surface to volume", also called "bottom-up". Model curves are discretized (meshed) at first. Then, model surfaces are triangulated/quadrangulated based on a boundary discretization that is imposed by the model curves. Finally, model volumes are tessellated. This approach has many advantages: strict respect of CAD features, flexibility (different meshers can be used for different CAD entities), robustness (we can replace with a different approach on an entity if one failed), easy parallelism, and the automatic support for non-manifold surfaces in models with multiple volumes. Sometimes there is no solution for the meshing of an entity given its boundary mesh (*e.g.* too coarse). In these cases, the boundary mesh is modified (usually a split of some elements) and the meshing of the entities that share this boundary is restarted, until a solution is found or the maximal number of tries is exhausted.

We also follow this philosophy with our quad mesher to benefit from the same advantages, but it should be noted that it has a significant drawback when meshing a CAD face: the boundary mesh (from the boundary CAD curves) is locked and cannot be changed. Thus, the curve quantization (Sec. 6.4.3) is very important. The size map, which controls the curve sampling, must be global and smooth on the CAD curves to have coherent meshes on the adjacent CAD faces. With our current approach, there are some irregular vertices in the final mesh that are due to non-optimal curve quantization, and which could be eliminated with further work (Sec. 6.8).

6.3 Scaled cross field computation

In our pipeline, cross fields fulfill two roles: (a) they guide the frontal point insertion in the unstructured quadrilateral mesher (Sec. 6.5) and (b) they have singularities (Sec. 6.3.2) which correspond to irregular vertices that we want to keep during the topological modifications that lead to the final quasi-structured mesh (Sec. 6.6.2). The cross field is computed with a fast multilevel diffusion algorithm (Sec. 6.3.1) which alternates diffusion steps and projection on the cross space. Then, to improve the accuracy of the following point insertion, we scale the cross field to make it integrable by computing its conformal scaling

(Sec. 6.3.3), which can be seen as a size map (up to a constant) useful to accommodate rotations in the cross field.

6.3.1 Fast cross field computation with multilevel diffusion

A cross field is a field defined on a surface with values in the quotient space S^1/Q , where S^1 is the circle group and Q is the group of quadrilateral symmetry. It associates to each point of the surface to be meshed a cross made of four unit vectors orthogonal with each other in the tangent plane of the surface. In our case, we compute one cross field per CAD face, with crosses aligned with the CAD curves bounding the face.

Cross representation To represent the crosses, we use the standard unit vector representation $\mathbf{u} = (\cos(4\theta), \sin(4\theta)) = (u_1, u_2)$ [Hertzmann and Zorin, 2000] where θ is the angle of one of the four branches with a local reference frame. Contrary to the naive angle representation θ , the vector representation $\mathbf{u}$ is invariant to the quadrilateral symmetries and suitable for linear finite element interpolation. As the linear combination of unit vectors is usually not a unit vector, the interpolated values inside triangles are not a strictly representations of crosses, but the crosses can be recovered easily by projecting on the unit circle, *i.e.* normalizing the vector representation.

Finite element discretization To discretize the vector field representation of the cross field, we define one cross at each edge of the triangulation and use the Crouzeix-Raviart interpolation, as in Beaufort et al. [2017b]. The unknowns of the cross field problem are the representation vector field components u_1, u_2 at each edge e_{ij} . As we want the cross field to be aligned the surface boundaries, the Dirichlet boundary conditions are $\mathbf{u} = (\cos(0), \sin(0)) = (1, 0)$ at each boundary edge.

Smoothing with the heat equation To have a smooth cross field inside the domain, the natural approach is to minimize the Dirichlet energy:

$$\min_{|\mathbf{u}(\mathbf{x})|=1} \int_{\Omega} \|\nabla \mathbf{u}\|^2$$

This objective function is non-linear as the crosses live in the quotient space S^1/Q , which is equivalent to the cross representation being on the unit circle, *i.e.* $\forall \mathbf{x} \in \Omega, |\mathbf{u}(\mathbf{x})| = 1$. A standard relaxation is to allow the cross representation to leave the unit circle, but they should stay close to it. Minimizing the Dirichlet energy by directly solving the Laplacian of the vector representation, *i.e.* $\nabla^2 \mathbf{u} = \mathbf{0}$, is not appropriate as the values may collapse to $\mathbf{0}$ far from the

boundary. One elegant solution to this problem is to solve a Ginzburg-Landau non-linear equation to penalize the values leaving the cross manifold [Beaufort et al., 2017b], but this approach is too expensive for large problems. A more efficient alternative is the MBO approach [Viertel and Osting, 2019], where one alternates heat diffusion (Eq. 6.3) and projection (Eq. 6.4) steps. A further refinement [Palmer et al., 2020] developed for 3D cross fields is to start with large diffusion time-steps and progressively decrease them. In this work, we also use decreasing diffusion time-steps but we select them according to the mesh size considerations, and we regroup them in levels to allow for re-use of the matrix factorization computed by a direct linear solver.

$$\text{Diffusion:} \quad \frac{\partial \mathbf{u}}{\partial t} = \alpha \nabla^2 \mathbf{u} \quad (6.3)$$

$$\text{Projection:} \quad \mathbf{u} \leftarrow \frac{\mathbf{u}}{\|\mathbf{u}\|} \quad (6.4)$$

Multilevel strategy When dealing with large CAD models, there are usually geometric features at many scales. Using a uniform mesh that is sufficiently fine to capture the smallest features is expensive and wasteful, as the cross field is mainly smooth far from the boundaries. In practice, we use triangulated meshes adapted to the geometric features, which can lead to edge lengths varying by many order of magnitudes in the mesh.

Our strategy is to iteratively resolve the cross field at various scales, which we call levels. Each level is characterized by a different diffusion coefficient, the α in (Eq. 6.3). Intuitively, we start with large diffusion lengths and progressively decrease to eventually capture the smallest cross field features. Beside being suited for non-uniform meshes, it is also a great way to accelerate computations on uniform triangulations. This approach is very fast: the cost of computing an accurate cross field is similar to the cost of solving a few linear systems, as we are re-using matrix factorizations inside levels.

After generating a triangulated mesh of the whole CAD model, with mesh sizes adapted to the CAD features, we compute the cross-field on each surface independently. As the boundary condition (cross tangency) is the same on both sides of a curve separating two CAD surfaces, the solutions will be compatible and we can regroup them at the end.

For a given triangulation T of a surface $\mathcal{S}$, we pre-compute N diffusion coefficients that will be used for the N successive heat diffusion levels. At each level, we successively solve the heat equation and project the solution, until the cross field no longer changes. In a way, we are solving the cross field problem (minimizing the Dirichlet energy) for a given resolution. Typically, the appropriate number of levels is between 5 and 10 for a large model but a lower value

around 3 or 4 is sufficient for a CAD face. We chose the initial (largest) and final (smallest) coefficients as

$$\alpha_{\text{initial}} = (0.1 * d)^2 \text{ and } \alpha_{\text{final}} = (3 * h_{\min})^2$$

with d the length of the surface bounding box diagonal and $h_{\min}$ the smallest edge length. The large initial time-step allows a first coarse diffusion of the boundary conditions and the final small time-step allows a local smoothing of the crosses at the finer mesh sizes. Between these extremities, we use linearly interpolated time-steps. While these choices are quite arbitrary, they are the results of many experimentations and we have found that they work pretty well on a large variety of CAD models with features at many scales.

The speed of our approach comes from the re-use of linear factorizations inside each level. From the heat diffusion equation (Eq. 6.3), we derive the discrete version, using finite element for spatial and backward Euler for time discretization:

$$M(U_{i+1} - U_i) = -\alpha K U_i$$

with M the mass and K the stiffness Crouzeix-Raviart matrices. For more details about the stiffness matrix assembly, one can refer to [Beaufort et al. \[2017b\]](#) as our coefficients are the same. With Crouzeix-Raviart, the mass matrix is diagonal and the coefficients are $M_{jj} = \frac{|t1|+|t2|}{3}$ with $|t1|, |t2|$ the areas of the triangles adjacent to the edge j [[Wardetzky et al., 2007](#)].

The heat diffusion system can be re-written as:

$$(\frac{1}{\alpha}I + M^{-1}K)U_{i+1} = \frac{1}{\alpha}U_i$$

At a given level (fixed α), the solution U will evolve, so is the right-hand side, but the matrix part of the system $(\frac{1}{\alpha}I + M^{-1}K)$ will not change, and so can be factorized one time per level. We can also note that the sparsity pattern of the system will be the same for all levels, and thus the preprocessing of the system (reordering, memory allocation) can be done one time at the beginning. While the factorization we use is an operation only available with direct linear system solvers, it can be replaced by an efficient preconditioning when using iterative solvers. Putting all things together, we describe the complete process to compute the cross field in Algorithm 1. Thanks to the system factorizations, the total computational cost of our cross field solver is approximately the cost of a few linear system solves (one per level, typically three or four).

Concerning the accuracy and the convergence of the scheme, it should be noted that we are not interested in reaching the solution of the *true* Ginzburg-Landau continuous formulation. Obtaining such solution is difficult and expensive, as it requires a fine mesh and a uniform triangulation, because the Dirichlet energy

Algorithm 1 Fast heat-based computation of cross field

Input: A triangulation of the surface and global parameters: N the number of diffusion levels (typically $N = 5$) and a norm convergence threshold c (typically $c = 1.e^{-3}$)

Output: A boundary-aligned smooth cross field f sampled at the edges (Crouzeix-Raviart)

Procedure:

compute the sparse Crouzeix-Raviart stiffness matrix K

compute the sparse Crouzeix-Raviart mass matrix M , which is diagonal

process the sparsity pattern of $A = M^{-1}K$ ▷ Direct solver call

$\alpha_{\text{init}} = (0.1 * d)^2$, with d the bounding box diagonal length

$\alpha_{\text{final}} = (3 * h_{\text{min}})^2$, with h_{min} the minimum edge size

for each level $l \in [1, N]$ **do**

$\alpha_l = \alpha_{\text{init}} + (\alpha_{\text{final}} - \alpha_{\text{init}}) * l / (N - 1)$

update the linear system matrix diagonal so $A = (\frac{1}{\alpha_l} I + M^{-1}K)$

factorize the matrix A ▷ Direct solver call

while $\|U_{i+1} - U_i\|_{\infty} > c$ **do**

update the right-hand side $B = \frac{1}{\alpha_l} U_i$

solve the linear system $AU_{i+1} = B$ using the factorization ▷ Direct solver call

project the cross coefficients on the unit disk, $U_{i+1} \leftarrow \text{proj}(U_{i+1})$

end while

end for

tends to infinity at singularities, so it should be uniformly sampled. We are more interested in cross fields that are useful for our quadrilateral meshing approach, and for this a smooth cross field with singularities *reasonably placed* is sufficient. In this context, we can apply the same algorithm parameters $(N, c, \alpha_{\text{init}}, \alpha_{\text{final}})$ to all models as it is not important if a cross field solution is not totally converged.

Boundary condition extension With uniform meshes, minimizing the Dirichlet energy, while staying close to the cross manifold with MBO or Ginzburg-Landau, has a tendency to position singularities close to the boundary. For instance, on a unit circle, singularities are at the radius 0.85. While this is not an issue when computing or studying cross fields, it is not practical for quadrilateral meshing: the associated irregular vertices may be closer to the boundary than the target edge size, complicating vertex insertion in indirect approaches, or there is not enough triangles between the singularity and the boundary to have an accurate parametrization with global parametrization approaches.

To push away the singularities from the feature curves, we use a simple trick: extend the boundary conditions inside the surfaces. On each triangle touching a feature curve, we fix the crosses on its three edges. The cross values are averaged by solving the Laplace equation $\nabla^2 \mathbf{u} = \mathbf{0}$ on this one-triangle boundary layer.

6.3.2 Cross field singularities

To constrain the cavity remeshing (Sec. 6.6.2), we use a list of singularities extracted from the cross field previously computed with the heat-based approach.

Detection of singularities. Cross field singularities are detected by computing, for each point $\mathbf{x}$ on the surface, the angle difference along a small closed circle γ centered on $\mathbf{x}$. This defines a singularity index [Ray et al., 2008]:

$$\text{index}(\mathbf{x}) = \frac{2}{\pi} \oint_{\gamma} d\theta(t) \quad (6.5)$$

In contrast to most related work, we use the factor $\frac{2}{\pi}$ instead of $\frac{1}{2\pi}$ to match the previously defined (Sec. 6.1.3) indices of the irregular vertices in the quadrilateral mesh. When a cross field singularity has index an of -1 with our definition (instead of the usual $-1/4$), it corresponds to an irregular vertex of index $k = -1$ in the quad mesh, *i.e.* valence five.

In the discrete setting, we extract the oriented edge one-ring $(e_{i1}, \dots, e_{in})$ around the vertex i and we compute the sum of the angle differences. In practice there are three cases: the sum is zero and the vertex is regular or the sum is one or minus one and the vertex is a singularity of the cross field.

Because of our Crouzeix-Raviart discretization of the cross field (one angle per edge), cross field singularities may lie on vertices, on edges or on triangles of the mesh. When a singularity lies on a edge, both its two adjacent vertices have an index different from zero, and when it lies on a triangle, we observe three adjacent vertices with an index different from zero. For our application it is not necessary to locate the singularities on mesh vertices, so we explicitly store the 3D position of the singularities, which can be the middle of a singular edge or the barycenter of a singular triangle.

Compatibility with quad mesh topology. In our experience, cross fields computed with a smoothing scheme and a continuous vector representation $(\cos(4\theta), \sin(4\theta))$ fail to be compatible with quadrilateral mesh topology (Eq. 6.2) when there are acute corners (angle $< 45^\circ$) in the CAD face or if the triangulated mesh is not refined enough along curved CAD curves.

To *correct* the first issue, we artificially add one singularity of index 1 to our list of floating singularities at each acute corner of the CAD face. With this simple addition, there is no need to modify the triangulation or to recompute a new cross field. For the second issue, we increase the sampling of the CAD curves based on the curvature during the initial triangulation process.

While both techniques usually work, and largely improve our results, there is no guarantee that the inconsistency between the singularity list and the CAD

face topology will always be fixed. Consequently, for sake of robustness, *we do not assume that the cross field singularity list is topologically correct in the following steps of the meshing pipeline.* This constitutes a major advantage of our unstructured approach: the ability to work with cross fields that are *topologically wrong*.

6.3.3 Scaling the cross field for integrability

The cross field that has been described in Sec. 6.3.1 is made of unit vectors. It is not integrable in the sense that if we extract (locally) two orthogonal vector fields from its branches, they will not commute. In this section, we show how to scale the branches in order to ensure local integrability of the cross field.

Theoretical considerations Assume two vector fields $(\mathbf{T}, \mathbf{S})$ defined in the tangent plane of $\mathcal{S}$. We look for a condition on $(\mathbf{T}, \mathbf{S})$ such that they are *the tangent vectors*, or *directional derivatives*, of a map $\mathbf{X} : (u, v) \in \mathbb{R}^2 \mapsto \mathcal{S}$. Formally, we want $\partial_u \mathbf{X} = \mathbf{T}$ and $\partial_v \mathbf{X} = \mathbf{S}$. The map $\mathbf{X}$ could be constructed “naively” in the following way. Let us pick a point $\mathbf{x} = (x, y) \in \mathcal{S}$ and choose $\mathbf{X}(0, 0) = \mathbf{x}$. We first advance “in time” of a time increment u along the flow $\Psi_{\mathbf{T}}(\mathbf{x}, u)$ of $\mathbf{T}$. The flow can be seen as follows: if $\mathbf{T}(\mathbf{x})$ is seen as the velocity of a fluid at point $\mathbf{x}$, a massless particle at point $\mathbf{x}$ will be advected by $\mathbf{T}$ along integral lines of $\mathbf{T}$. After a time increment of t , the particle starting at $\mathbf{x}$ will be located at $\Psi_{\mathbf{T}}(\mathbf{x}, u) \in M$. Note that the corresponding location in the parameter plane is $(u, 0)$. Then we advance in time of a time increment v along the flow of $\mathbf{S}$ which lead us to point (u, v) in the parameter plane and at point $\Psi_{\mathbf{S}}(\Psi_{\mathbf{T}}(\mathbf{x}, u), v)$. A map is only possible if the two flows commute:

$$\Psi_{\mathbf{S}}(\Psi_{\mathbf{T}}(\mathbf{x}, u), v) = \Psi_{\mathbf{T}}(\Psi_{\mathbf{S}}(\mathbf{x}, v), u) \quad (6.6)$$

From differential geometry, we know that both flows commute if and only if the Lie bracket is zero:

$$[\mathbf{T}, \mathbf{S}] = J_{\mathbf{S}}\mathbf{T} - J_{\mathbf{T}}\mathbf{S} = \mathbf{0}$$

where $J_{\mathbf{T}}$ and $J_{\mathbf{S}}$ are the Jacobian matrices of the vector fields. In our case, we assume that $\mathbf{T}$ and $\mathbf{S}$ have locally the same length:

$$\|\mathbf{T}(\mathbf{x})\| = \|\mathbf{S}(\mathbf{x})\| = h(\mathbf{x}) = e^{H(\mathbf{x})}$$

Choosing $h = e^H$ is adequate because vector lengths should be strictly positive so the map $\mathbf{X}$ is injective. In 2D, $\mathbf{T}$ and $\mathbf{S}$ are two branches of the cross field that are mutually orthogonal and scaled by h . Those vector fields can thus be parametrized using an angle $\theta(\mathbf{x})$:

$$\mathbf{T}(\mathbf{x}) = e^{H(\mathbf{x})} \begin{pmatrix} \cos(\theta(\mathbf{x})) \\ \sin(\theta(\mathbf{x})) \end{pmatrix} \quad \text{and} \quad \mathbf{S}(\mathbf{x}) = e^{H(\mathbf{x})} \begin{pmatrix} -\sin(\theta(\mathbf{x})) \\ \cos(\theta(\mathbf{x})) \end{pmatrix}. \quad (6.7)$$

Assuming $\mathbf{T}$ and $\mathbf{S}$ of the form (6.7), the Jacobian is

$$J_{\mathbf{T}\mathbf{S}} = e^{2H} \begin{pmatrix} -H_x \sin \theta \cos \theta + \theta_x \sin^2 \theta + H_y \cos^2 \theta - \theta_y \sin \theta \cos \theta \\ -H_x \sin^2 \theta - \theta_x \cos \theta \sin \theta + H_y \sin \theta \cos \theta + \theta_y \cos^2 \theta \end{pmatrix}.$$

By enforcing $H_x = -\theta_y$ and $H_y = \theta_x$, the jacobian reduces to $J_{\mathbf{T}\mathbf{S}} = e^{2H} \nabla \theta$ and similarly $J_{\mathbf{S}\mathbf{T}} = e^{2H} \nabla \theta$, fulfilling the commutativity condition $[\mathbf{T}, \mathbf{S}] = J_{\mathbf{S}\mathbf{T}} - J_{\mathbf{T}\mathbf{S}} = \mathbf{0}$. Thus, the condition on the cross field scaling $h = e^H$ for local integrability is

$$\nabla^\perp \theta = \begin{pmatrix} -\theta_y \\ \theta_x \end{pmatrix} = \begin{pmatrix} H_x \\ H_y \end{pmatrix} = \nabla H \quad (6.8)$$

Discretization From the cross field computation, we know at each edge (i, j) the angle θ_{ij} between one of the cross branches and the edge tangential direction. We now describe how to compute the scaling factor $h(\mathbf{x})$ at each vertex of the triangulated mesh.

Consider the triangle (i, j, k) , we first compute the angle differences

$$\Delta\theta_{jk} = \text{diff}(\theta_{jk}, \theta_{ij}) \quad \text{and} \quad \Delta\theta_{ki} = \text{diff}(\theta_{ki}, \theta_{ij}).$$

Operator $\text{diff}(\theta_{jk}, \theta_{ij})$ is not equivalent to a simple difference of angles because (i) angle θ_{jk} is measured on a different system of coordinates as θ_{ij} and because (ii) the cross branches have a 4-symmetry which implies that $\theta_{jk} \equiv \theta_{jk} + k_2 \frac{\pi}{2}$ is the same cross for any $k_2 \in \mathbb{Z}$. We choose k_1 and $k_2 \in \mathbb{Z}$ in such a way that

$$\theta'_{ij} = \theta_{ij} + k_1 \frac{\pi}{2} \in \left[0, \frac{\pi}{2}\right] \quad , \quad \theta'_{jk} = \theta_{jk} - \alpha_j + k_2 \frac{\pi}{2} \in \left[0, \frac{\pi}{2}\right]$$

with α_j the angle between the tangential vectors of the edges (i, j) and (j, k) , and then define

$$\text{diff}(\theta_{jk}, \theta_{ij}) = \begin{cases} \theta'_{jk} - \theta'_{ij} & \text{if } |\theta'_{jk} - \theta'_{ij}| \leq \frac{\pi}{4} \\ \theta'_{jk} + \frac{\pi}{2} - \theta'_{ij} & \text{if } \theta'_{jk} - \theta'_{ij} < -\frac{\pi}{4} \\ \theta'_{jk} - \frac{\pi}{2} - \theta'_{ij} & \text{if } \theta'_{jk} - \theta'_{ij} > \frac{\pi}{4} \end{cases}$$

It is thus possible to compute $\nabla \theta$ as a constant vector of the tangent plane in every triangle. Computing H can be done by solving

$$\min_H \int_S |\nabla H - \nabla^\perp \theta|^2 d\mathbf{x} \quad (6.9)$$

using P^1 finite elements. The scaling factor is simply obtained as $h(\mathbf{x}) = e^{H(\mathbf{x})}$. An example of conformal scaling is illustrated on Fig. 6.2.a.

Equation (6.9) only involves ∇H so H is defined up to a constant C . It is indeed possible to choose C in such a way that the final mesh contains about N quads. We actually know that

$$N \simeq \int_S \frac{1}{h^2(\mathbf{x})} d\mathbf{x} = \int_S e^{-2(H(\mathbf{x})+C)} d\mathbf{x} = e^{-2C} \int_S e^{-2H(\mathbf{x})} d\mathbf{x}.$$

Thus,

$$e^{-2C} = \frac{N}{\int_S e^{-2H(\mathbf{x})} d\mathbf{x}} \quad \rightarrow \quad C = -\frac{1}{2} \ln \left(\frac{N}{\int_S e^{-2H(\mathbf{x})} d\mathbf{x}} \right).$$

Comparison with related work The scaling that we are computing is a conformal scaling associated to the cross field. It has been computed in the past directly from the cross field singularities by solving a linear system [Bunin, 2006; Ben-Chen et al., 2008]. The difference is that we do not *robustly* know the singularities. By solving the least square problem (Eq. 6.9) which only involves cross gradients, we always have a scaling which is usable even if our cross field is not globally consistent (*e.g.* wrong singularities).

6.4 Size map and curve quantization

To generate a mesh, it is necessary to have some kind of size map specifying the target size of the elements on the domain. When an explicit scalar size field is not used, it can be defined implicitly by simply using a uniform target size, or prescribed sizes on some CAD entities (corners, curves) with a maximal uniform size far from them. If there are small CAD curves in the model, or small prescribed sizes, the size transition to the coarser regions is usually constrained by a maximal mesh gradient which is a parameter (often hidden) of most meshers.

Most complex CAD models have features of many scales and using the smallest feature size as a uniform target size is not practical as it would lead to meshes with an excessive amount of elements. In our experience, letting the meshing algorithms, which usually have not been designed with mesh size transitions in mind, to just *do their things* and hope for the best is not optimal as it often leads to poor quality quad meshes. To robustly produce high-quality meshes on generic models, we have to use a smooth size map to control the transition from small features to much larger ones.

The conformal scaling associated to the cross field (Sec. 6.3.3) induces a size map (up to a constant) which takes into account the cross gradients (Eq. 6.9). This size map is critical to be able to generate square-shaped quads in the smooth regions of the model. Yet, it ignores small CAD features. Thus, trying

to build a quad mesh with such discrepancy between the forced mesh edges (from the CAD) and the (implicit or explicit) size map used by the quad surface mesher leads to poor quality elements, even if the process is robust. The ideal solution would be to have a cross field which respects the CAD features and mesh size prescriptions, such that its conformal scaling would be naturally compatible with the CAD geometry. Unfortunately, this problem is still open.

Here, a smooth size map is computed from the CAD feature sizes and is blended with the size map from the conformal scaling by taking the minimum of both size maps. The final size map allows for smooth transitions from the smallest CAD features to the coarse regions where it is dominated by the conformal scaling.

6.4.1 Size map from small CAD features

Small CAD feature is a generic term used for the regions where a small mesh size is required to capture the geometry with well-shaped elements. A small CAD curve is a small feature, as well as two CAD objects that are close to each other.

In this work, a *minimal size* field $s^{\min}(\mathbf{x})$ is built for taking into these small CAD features. $s^{\min}$ is a nodal field whose support is the triangular mesh that has been used for computing the cross field. Its values are initially computed on CAD curves. Mesh size $s^{\min}(\mathbf{x})$ at a vertex $\mathbf{x}$ that belongs to a CAD curve $\mathcal{C}_j$ (i) is never larger than the actual length of $\mathcal{C}_j$ and (ii) is never larger than the distance between $\mathbf{x}$ and the closest CAD curve $\mathcal{C}_k$ that is not topologically adjacent to $\mathcal{C}_j$. The field values in $s^{\min}$ are subsequently propagated to the internal nodes of the CAD surfaces, ensuring a smooth gradation.

Technically, the propagation consist in a Dijkstra-like algorithm: the size $s^{\min}$ is updated on every edge $(\mathbf{x}_1, \mathbf{x}_2)$ of the triangulation (including edges classified on CAD curves) in order to limit the size gradient to a prescribed value $g_{\max}$:

$$s^{\min}(\mathbf{x}_2) = \min(s^{\min}(\mathbf{x}_2), s^{\min}(\mathbf{x}_1) + g_{\max} d(\mathbf{x}_1, \mathbf{x}_2)) \quad (6.10)$$

This simple approach works actually well in practice. For more information on mesh size vs. CAD features, the reader can refer to [Bawin et al. \[2020\]](#).

6.4.2 Global size map

To build a global continuous and smooth size map $s(\mathbf{x})$, we combine the minimal size field $s^{\min}(\mathbf{x})$ with the size maps $s_i^{cf}(\mathbf{x})$ associated to the cross field conformal scaling (Sec. 6.3.3) computed on each CAD face $\mathcal{S}_i$.

The conformal scaling fields (Sec. 6.3.3) are defined up to a constant C_i that can be tuned in order to have N_i quads on the surface $\mathcal{S}_i$. Starting from a

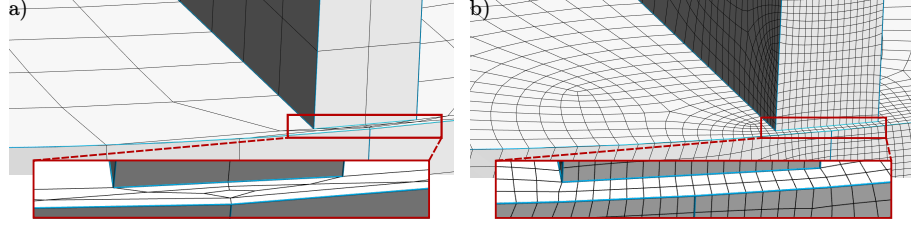


Figure 6.3: Focus on a small CAD feature of the *Block* model. Without adaptation of the size map (a), there are low quality elements around the small feature. With size map adaptation (b), there are small high-quality elements around the feature and irregular vertices to make the transition to coarser regions.

global target number of quads N , we distribute the quads on each CAD face based on their areas, *i.e.* $N_i = \frac{\text{area}_i}{\text{total area}} N$. Another advantage of this simple approach is that we can easily take user preferences into account. If a user want a specific number of quads on a CAD face, we can use this value instead of the one proportional to the area.

The conformal scaling size maps are not continuous at the CAD interfaces. Consider a point $\mathbf{x}$ on a CAD curve common to the faces $\mathcal{S}_i$ and $\mathcal{S}_j$, we have $s_i^{cf}(\mathbf{x}) \neq s_j^{cf}(\mathbf{x})$ because the cross field gradients, and thus the conformal scaling, are not equal on both sides of the curve. To make a continuous and smooth union of the s_i^{cf} fields in a global one s^{cf} , we apply a similar smoothing process as before (Sec. 6.4.1). Initialize the union of the size map, taking the minimum on the shared vertices (CAD corners and curves): $s^{cf}(\mathbf{x}) = \min_i(s_i^{cf}(\mathbf{x}))$. Then apply a one-way smoothing (Eq. 6.10) with the maximal gradient g_{max} .

To blend the small feature size map and the conformal scaling, we simply take the minimum everywhere:

$$s(\mathbf{x}) = \min(s^{cf}(\mathbf{x}), s^{min}(\mathbf{x}))$$

A local example of our size map taking the small CAD features into account is shown on Fig. 6.3, where small quads are used in the region where two feature curves are close to each other. This figure is a zoom on a small part of the larger *Block* model (Fig. 6.9), which have lot of small features and where significant size map transitions are necessary in many regions.

6.4.3 Curve quantization

We call quantization the process of going from continuous fields (cross field, size map) to a quadrilateral mesh which is characterized by discrete quantities, such

as the number of points on a model curve or the number of quads in a model face. As we are following the robust Gmsh’s “surface to volume” philosophy, the curve quantization is important in our pipeline.

The process of first meshing the model curves then the model surfaces, constrained to the curve discretization, works well for the generation of triangulations (or tetrahedrizations) but it is not optimal for quadrilateral meshing (or hexahedral meshing) as the quad topological constraints (chords) are global and go through the model curves. For instance, imagine a simple rectangular face for which the ideal edge numbers, according to the continuous cross field induced size map, are $[9.4, 4.1, 10.6, 3.9]$ on the four sides. A simple quantization by integer rounding would choose respectively $[9, 4, 11, 4]$ edges. While a quadrilateral mesh with this quantization exists, it is not structured as it must include one pair of $+1/-1$ irregular vertices to make the transition from 9 edges on one side to 11 edges on the opposite side.

Finding a *good* curve quantization for quad meshing, for a generic CAD model with CAD faces of arbitrary topology, is a complicated and totally open problem, which has been mostly ignored in the literature. If we restrict ourselves to quadrilateral CAD faces (with T-junctions), there is the global quantization in [Campen et al. \[2015\]](#), but in our case we do not have such quadrilateral layout.

For the current paper, our quantization of CAD curves remains quite simple. We use the integer-rounded value of the ideal number of edges computed by integrating the global size map (Sec. 6.4.2), except for curves which are on quadrilateral CAD faces, where we impose equality on the opposite sides. This non-optimal quantization leads to many necessary dipoles (pair of $+1/-1$ irregular vertices) in the final quad meshes of the different CAD faces. Eliminating them requires to change the CAD curve 1D meshes, which is out of the scope of the current paper.

Size map integration on curves Consider a CAD curve $\mathcal{C}$ parametrized by t varying from a to b . The *floating-point* number of edges on the curve according the size map is: $\bar{n}_e = \int_a^b dl/s(\mathbf{x})$ with $dl = \|x'(t)\| dt$. To get a integer number of edges n_e , we use a simple rounding: $n_e = \text{round}(\bar{n}_e)$. The parameter $t_i, i \in [1, n_e - 1]$ associated to the i -th interior point of the curve is such that:

$$\int_a^{t_i} \frac{dx}{s(\mathbf{x})} = \frac{i}{n_e} \int_a^b \frac{dl}{s(\mathbf{x})} \quad (6.11)$$

This integral equation can be solved via numerical integration, by adding values along the curves until the sum is equal to $i\bar{n}_e/n_e$, with linear interpolation between the samples.

To mesh the curve $\mathcal{C}$, we compute the vertex positions $\mathbf{x}_i, i \in [1, n_e - 1]$ by computing the parameters t_i with (Eq. 6.11) and evaluating the CAD curve

parametrization: $\mathbf{x}_i = f(t_i)$. With this approach, the points are well placed on the curves according to the size map, making smooth transitions from the small CAD feature regions to the coarser ones, where smaller but useful variations come from the cross field conformal scaling.

Topological constraints on quadrilateral patches Integrating the size map does not take into account the particularities of quad mesh topology, such that the quads are organized in topological chords (dual loops of adjacent elements). In this work, we adjust the number of edges n_e on the opposite sides of quadrilateral CAD faces.

Consider a CAD face whose boundary is made of four CAD curves ($\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3, \mathcal{C}_4$), we force the number of points to be equal on the opposite edges, *i.e.* $n_{e1} = n_{e3}$ and $n_{e2} = n_{e4}$, except if the integrated values are very different. When two adjacent quadrilateral CAD faces share a curve $\mathcal{C}_c$, the value n_{ec} must be the same on both faces. This means the equality constraints propagate in the CAD model. To resolve the propagation, we build the topological chords associated to the quadrilateral CAD faces. A chord is made of topologically parallel CAD curves, which all receive the same fixed number of points, that we compute by averaging the ideal values previously computed on each curve.

With this simple propagation, the chords propagate only if two adjacent quadrilateral faces share a single curve which is one of their four sides. We do not propagate across T-junctions. Our quantization applies only on the connected subsets of conformal quadrilateral patches of the CAD model. Dealing with T-junctions is more complicated and corresponds to the quantization phase of [Campen et al. \[2015\]](#), for which there is not always a solution (Sec. 6.2).

6.5 Unstructured frontal quadrilateral mesh generation

Now that the 1D meshes of the CAD curves have been generated, it's time to move to the initial quad-meshing of the CAD surfaces. The all-quad mesh is generated by following a robust three step approach: (1) generate a triangulation with right-angle triangles using the methodology in [Remacle et al. \[2013\]](#), (2) combine some triangles into quads to form a quad-dominant mesh and (3) apply midpoint subdivision to produce an all-quad mesh. We also describe the geometry smoothing and untangling techniques (Sec. 6.5.3) which are used extensively in our pipeline.

6.5.1 Frontal point insertion

A scaled cross field $\mathbf{f}(\mathbf{x})$ has been generated where the size of the crosses is the global size map $s(\mathbf{x})$ (Sec. 6.4.2). Mesh vertices are created on every face $\mathcal{S}$

in a frontal fashion, starting from its boundary $\partial\mathcal{S}$. Vertices of $\partial\mathcal{S}$ are added in a priority queue. Vertex $\mathbf{x}$ at the top of the queue tries to add four new vertices in the domain in the 4 directions of the local cross field $f(\mathbf{x})$ and at a distance $s(\mathbf{x})$. New vertices are only added (i) if they lie inside $\mathcal{S}$ and (ii) if they are not too close to any existing vertex. When a vertex is successfully added, it is inserted at the end of the queue and the procedure continues until no new vertices can be generated. The points are iteratively inserted in the triangulation with a Delaunay kernel, as described in [Remacle et al. \[2013\]](#). This mesher is close to classic frontal Delaunay triangular meshing [\[Rebay, 1993\]](#), except that the goal is to build right-angled triangles in agreement with the cross field.

6.5.2 Quad combination and midpoint subdivision

The triangulation is transformed into a quad-dominant by combining pairs of triangles into quads. As triangles are allowed, there is no need for advanced matching technique such as Blossom-Quad [\[Remacle et al., 2012\]](#) and we simply use a greedy selection. All the merge candidates (two triangles $\rightarrow$ one quad) are computed and sorted by the geometric quality (Sec. 6.5.3) of the resulting quads, weighted by the alignment with the cross-field. Then the quads are iteratively selected, as long as their quality is superior to a minimal threshold ($\text{SICN} > 0.1$ in our case).

The all-quad mesh is obtained by subdividing all quads into four sub-quads and all triangles into three sub-quads. This subdivision is called midpoint subdivision and is topologically equivalent to the Catmull–Clark subdivision surface technique. New vertices are added at the centers of each edge and each element.

To accurately represent the CAD model geometry, it is best to project the new midpoint vertices on the CAD curves and faces. However, this projection is not robust in the sense that it may create invalid quads (negative quality). This difficulty is similar to the one faced when doing CAD snapping for high-order meshing, and is not easy to solve. In our pipeline, we iteratively try to snap all midpoint vertices while monitoring the quality of the adjacent elements, and we revert the projections which lead to negative elements. Even if rejected projections are rare and definitively not ideal, this check is necessary to ensure the geometric validity of the final mesh when dealing with complex CAD models. For better geometric fidelity, one could decrease the size map locally to better capture the geometry and restart the meshing process of the CAD curves and surfaces in the neighborhood.

An example of quad mesh produced by this procedure is shown in Fig. 6.2.2.b. It is actually a fairly good unstructured all-quad mesh, yet containing an excessive number of irregular vertices.

6.5.3 Geometry optimization

Geometrical smoothing of a mesh is the process of finding *good* vertex positions given a fixed mesh topology. Ideally, we would like the positions to maximize some quality functional, but such formulation are usually too expensive to be solved globally and we have to resort to more efficient techniques. Smoothing of unstructured quadrilateral meshes is still a challenging problem in our opinion. Simple and fast techniques, such as Laplacian smoothing and its variants, often produce tangled elements (negative quality) while more advanced non-linear optimization techniques are computationally too expensive to be used on large sets of elements. For CAD surfaces, an additional expensive operation is the projection of a 3D point on the surface, which is typically performed many times during optimization. As an approximation, it is possible to project on a triangulated representation of the geometry, but even this approximation is expensive when applied millions of times.

In this work, we apply a great deal of topological remeshing (Sec. 6.6) and each time we use smoothing techniques to determine the geometry before validating or rejecting the remeshing. Concretely, this means that we are applying smoothing all the time, on many more elements than there are in the initial or final mesh. For instance, if there are $100k$ vertices in the quadrilateral mesh, we will probably smooth millions of vertex positions. While smoothing is always expensive in any mesher, the computational cost is particularly critical in our approach and we have to employ fast techniques and fall-back to expensive ones only locally and in last resort. Before and after any smoothing operation, we compute the element qualities and we revert the vertex positions if the quality decreased.

Quality metric For consistency with other *Gmsh* algorithms, we use the Signed Inverse Condition Number (SICN) metric to evaluate the geometric quality of the quads. It is the inverse of the condition number in Frobenius norm of the mapping from the quad to the regular square. This metric, which is also called *shape quality* [Stimpson et al., 2007], has a behavior similar to the scaled Jacobian: the range is $[-1, 1]$ with negative values for invalid elements and a maximum value of 1 for squares (and rectangles for the scaled Jacobian). As with the Jacobian, the SICN value is not constant inside a quad, so we use the minimum. In practice, most mesher sample the quad quality at the four corners and take the minimum. Assuming the quad corners are $\mathbf{x}_i, i \in [1, 4]$ and $\mathbf{N}_i$ are the CAD surface unit normals, the scaled Jacobian and SICN values at the i -th corner are respectively [Stimpson et al., 2007]:

$$SJ_i = \frac{\mathbf{L}_{i-1} \times \mathbf{L}_i}{\|\mathbf{L}_{i-1}\| \|\mathbf{L}_i\|} \cdot \mathbf{N}_i, \quad SICN_i = 2 \frac{\mathbf{L}_{i-1} \times \mathbf{L}_i}{\|\mathbf{L}_{i-1}\|^2 + \|\mathbf{L}_i\|^2} \cdot \mathbf{N}_i, \quad \text{with } \mathbf{L}_i = \mathbf{x}_{i+1} - \mathbf{x}_i \quad (6.12)$$

When one edge length $\|\mathbf{L}_i\|$ tends to zero (geometrical edge collapse), the scaled Jacobian will not tend to zero but to the sine of the angle at the corner (as $\mathbf{L}_{i-1} \times \mathbf{L}_i = \|\mathbf{L}_{i-1}\| \|\mathbf{L}_i\| \sin(\theta) \mathbf{n}$). This observation have very practical implications: when sampling the quality at the four corners, a quad mesh may have many almost-collapsed edges (*e.g.* $1e^{-12}$) but still a high minimum scaled jacobian (*e.g.* 0.7). On the other hand, the SICN quality metric, also sampled at the four corners, will tend to zero with edge collapse. In our experience, the discretized SICN should be preferred to the scaled Jacobian when evaluating the quality of isotropic quadrilateral meshes.

Laplacian smoothing in CAD parameter domain When we are dealing with a parametrized CAD model, we have a valid uv trimmed parametrization of each CAD face. To smooth the vertices inside a cavity C , which can be a whole CAD face, we extract continuous $\mathbf{uv} \in \mathbb{R}^2$ values for each boundary $\mathbf{x} \in \partial C$. One must be careful with the parametrization jumps (*e.g.* periodic seam on a cylinder). Sometimes it is not possible to have a continuous boundary in the parameter domain (*e.g.* around the pole of a sphere parametrization), and we move to the next smoothing options. Once we have a continuous $\mathbf{uv}$ on the boundary, we simply solve the unweighted Laplacian equations $\Delta u = 0$ and $\Delta v = 0$, where each internal value is the arithmetic average of its edge-connected neighbors. It is important to use the arithmetic average and a direct linear solver to respect the maximum principle and avoid inverted elements in the parameter domain. Depending on the distortions in the CAD parametrization, this technique may both produce high-quality or poor elements. To retrieve the 3D positions, we then apply the CAD face mapping $\mathbf{x} = f(\mathbf{uv})$. Depending on the resulting element qualities, we keep or we reject this smoothing. The huge advantage of this approach is that there is no surface projection and simply two linear solves, making it a very fast smoother. Even if the quality may be poor due to distortion in the CAD mapping, it is still a good initial guess since all elements are usually untangled.

FDM Winslow kernel on regular vertices Winslow smoothing [Knupp, 1999] is the industry-standard technique used for structured grid mesh generation [Thompson et al., 1998]. Given a fixed boundary, the idea is to solve the Winslow non-linear elliptic PDE $\Delta_{\mathbf{x}} \mathbf{u} = \mathbf{0}$, where $\mathbf{u}(\mathbf{x})$ are the coordinates in a certain computational space and $\mathbf{x}$ are the coordinates in the physical space. The advantage of this approach is that the two coordinate components are coupled and the resulting quads are nicely shaped, with some orthogonality enforced even under large distortions. By applying a finite difference (FDM) discretization to the Winslow equation [Knupp, 1999], we can derive a local smoothing kernel for regular vertices in the quadrilateral mesh. Assuming that $(\mathbf{x}_1, \dots, \mathbf{x}_8)$ are the ordered vertices of the stencil around the regular vertex $\mathbf{x}$,

its new position is given by:

$$\begin{aligned} \alpha_0(\mathbf{x}_1 + \mathbf{x}_5 - 2\mathbf{x}) + \alpha_1(\mathbf{x}_3 + \mathbf{x}_7 - 2\mathbf{x}) - \frac{1}{2}\beta(\mathbf{x}_2 + \mathbf{x}_6 - \mathbf{x}_4 - \mathbf{x}_8) &= 0 \\ \text{with } \alpha_0 &= (\mathbf{x}_3 - \mathbf{x}_7) \cdot (\mathbf{x}_3 - \mathbf{x}_7), \\ \alpha_1 &= (\mathbf{x}_1 - \mathbf{x}_5) \cdot (\mathbf{x}_1 - \mathbf{x}_5), \\ \beta &= (\mathbf{x}_1 - \mathbf{x}_5) \cdot (\mathbf{x}_3 - \mathbf{x}_7) \end{aligned} \quad (6.13)$$

This smoothing is moving the vertex outside of the CAD surface. We project back on it by finding the closest point on a triangulated representation of the CAD surface (with a kd-tree for faster spatial queries). If we want a more accurate projection, such as in the last steps of the smoothing, we ask the projection to the CAD geometric kernel, which in our case is OpenCASCADE.

Angle-based kernel on irregular vertices On irregular vertices, we use the angle-based smoothing kernel of Akram et al. [2021], where the idea is to move the vertices on the bisector of the one-ring. As the above Winslow kernel, it works well in regions where the mesh is not too constrained but it may create inverted elements in complex configurations. An alternative for irregular vertices is the unstructured Winslow FDM kernel [Knupp, 1999], but it is more complicated and would suffer from the same issues.

Smoothing loop The smoothing kernels, followed by projection on the CAD, are applied iteratively to all vertices of the current cavity, until the displacement is smaller than a criteria or the maximal number of iterations is reached. This iterative process is slower than the uv-Laplacian smoothing, because of the non-linearity of the kernel and of the surface projections, so we generally use it without waiting for convergence. It should be notated that these kernels may produce tangled elements, especially close to concave CAD features or in very irregular regions. For this reason, we compare the element qualities before and after, and roll back if necessary.

Non-linear optimization Some local configurations are difficult to untangle and smooth, or even impossible. This is generally due to a mix of constraining CAD features and mesh irregularity, such as for the disk quadrangulation remeshing (Sec. 6.6.3). In these tricky situations, we use the non-linear untangler of Mesquite [Brewer et al., 2003] on very small patches of quads. This optimization can fail or may convergence too slowly, so in practice we use a time limit of 1s. Failure to untangle is not a big issue in our pipeline because we start from a geometrically valid unstructured mesh by construction, so untangling is only encountered during remeshing and we can simply rollback if the result is tangled.

6.6 Topological quadrilateral meshing

Unstructured quadrilateral meshes generated frontally (Sec. 6.5) typically have a large excess of irregular vertices (*e.g.* Fig. 6.1.c.). To transform the mesh into a quasi-structured quad mesh (*e.g.* Fig. 6.1.d.), we need to remove most of the irregular vertices while preserving the ones that important to accommodate the model topology and geometry. We achieve this topological improvement by extracting *cavities* of quads in the unstructured mesh and by replacing them by *more regular* quad meshes. We act at three different scales: the cavity may be an entire CAD face when it is topologically simple (Sec. 6.6.1), the cavity may be a convex patch of quads in a CAD face (Sec. 6.6.2), or the cavity may contain only the quads adjacent to a vertex (Sec. 6.6.3). For large cavities (Sec. 6.6.1 and Sec. 6.6.2), we look for replacement meshes in a list of high-quality predefined patterns (Fig. 6.4). For local cavities, we look for replacements in the list of all possible disk quadrangulations (Sec. 6.6.3).

Each time we apply a topological operation (meshing or local remeshing), it is very important to verify that the geometric quality of the mesh is satisfactory. We always apply the geometric smoothing techniques defined in Sec. 6.5.3. If the geometry is invalid, or if the quality decreased too much compared to the initial configuration after remeshing, we cancel the mesh replacement.

Common definitions A *cavity* C is a set of quads forming a simply-connected part of the mesh. The boundary of a cavity ∂C is a closed polyline. Consider a vertex $\mathbf{x} \in \partial C$. Let us call $n_{\mathbf{x}}$ the number of quads adjacent to $\mathbf{x}$, which are also the quadrilateral valences. We distinguish $n_{\mathbf{x}}^{in}$ the number of quads adjacent to $\mathbf{x}$ that belong to C and $n_{\mathbf{x}}^{out}$ the number of quads adjacent to $\mathbf{x}$ that do not belong to C . We have $n_{\mathbf{x}} = n_{\mathbf{x}}^{in} + n_{\mathbf{x}}^{out}$. It should be noted that we usually only consider the quads in the current CAD face $\mathcal{S}$, not the global quadrilateral mesh of the model.

The vertex $\mathbf{x} \in \partial C$ is a convex corner of the cavity if and only if $n_{\mathbf{x}}^{in} = 1$. It is a concave corner of the cavity if the vertex is strictly inside the CAD face, $\mathbf{x} \notin \partial \mathcal{S}$, and if $n_{\mathbf{x}}^{out} = 1$.

By splitting the cavity boundary ∂C at the convex and concave corners, we form m regular *sides*, which respectively contain $N_1, \dots, N_m$ edges.

6.6.1 Pattern-based quadrilateral meshing

Given a cavity with m sides (containing $N_1, \dots, N_m$ edges), it is often (not always) possible to topologically find a quadrilateral mesh to fill the interior from a list of predefined patterns (Sec. 6.6.1.1). Such quadrilateral meshes (Fig. 6.4 bottom) are the anisotropic subdivision of coarse quadrilateral meshes (Fig. 6.4 top). By anisotropic subdivision, we mean that each topological chord

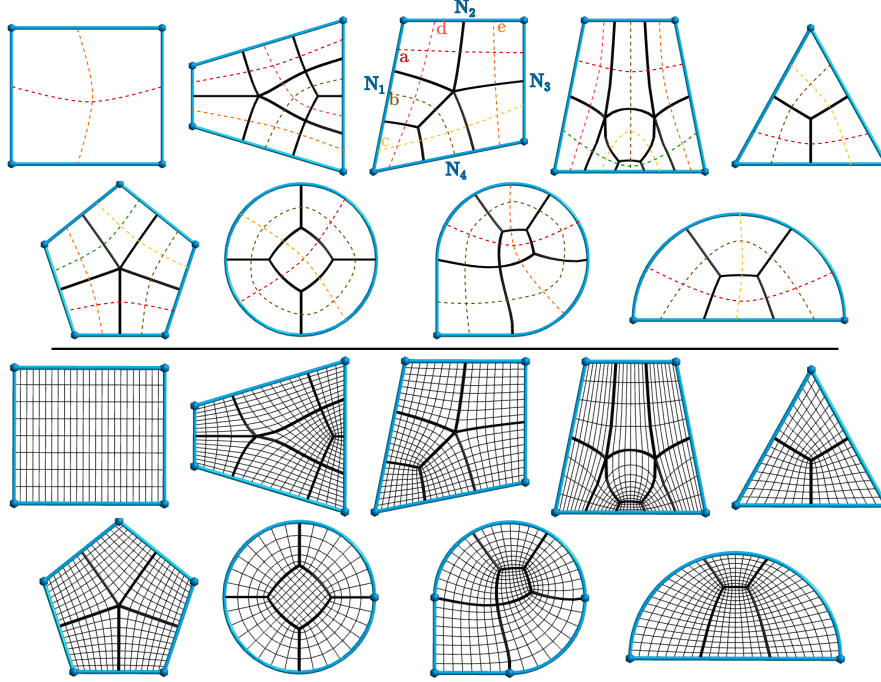


Figure 6.4: Pattern-based topological quad meshing. The patterns are defined by coarse quadrilateral meshes (top). The topological chords (dashed lines) can be subdivided independently to build refined quadrilateral meshes (bottom).

(dashed lines in Fig. 6.4) can be subdivided independently of the others. We use this constrained topological approach to mesh whole CAD faces when their topology is simple (Sec. 6.6.1.2), skipping the frontal unstructured mesher, or to iteratively improve patches of quads in the initially unstructured mesh of a CAD face.

6.6.1.1 Matching patterns

Consider the cavity with side edges $(N_1, \dots, N_m)$, we check if there is a matching with a quad pattern with m sides in our pre-computed pattern list (*e.g.* Fig. 6.4). For each pattern $\mathcal{P}$, we denote $(s_1, \dots, s_c)$ the number of edge subdivision of the c topological chords of the patterns (dashed lines Fig. 6.4). There is a quadrilateral mesh if it is possible to find strictly positive chord subdivisions $s_j > 0, j \in [1, c]$ such that on each side, the sum of the subdivisions of the

chords included in the side is equal to the side number of edges:

$$\forall i \in [1, m], \quad \sum_{j \in [1, c]} w_{ij} s_j = N_i \quad (6.14)$$

where w_{ij} is the number of times the chord j is inside the side i . In practice, w_{ij} is equal to 0 (the chord is not on the side), 1 (the chord goes through the pattern) or 2 (the chord starts and finishes on the side, turning inside). This linear system may be undetermined if there are more chords (columns) than sides (rows). We are not only interested in a solution to the linear system, we want one that is integer and strictly positive because it corresponds to the chord subdivisions of the pattern. We also want the values to be well balanced so the output quadrilateral mesh is not highly distorted. This behavior can be obtained by finding the solution that is the closest to an ideal subdivision of the chords, defined by equi-balancing the values on each side. This corresponds to a non-linear objective function where we minimize a distance, *e.g.* $E = \sum_j (s_j - x_j)^2$ with x_j the ideal subdivisions.

To solve such system, with linear constraints and a quadratic objective function, the ideal approach is to use a suited Mixed-Integer solver. From a practical point of view, such solvers are large software dependencies and would be overkill for our problem. As our system is usually very small, we have adopted a much simpler solution: compute the integer row echelon form of the system matrix, start from an ideal guess (by balancing the values) and use a depth-first search to find a solution satisfying all the constraints by rolling up in the echelon form matrix. However, even if the system is small, the exhaustive search that happens when there is no solution is too slow for practical use. So we abandon the search after a few hundred attempts. While this approach may sound naive, we observe that it works well in practice as the solution, if there is one, is close to the ideal subdivisions (initial guess). It should be noted that when looking for match with a pattern, one must check all the rotations of the sides $[N_1, \dots, N_n]$, clockwise and counter-clockwise.

The advantage of this integer formulation is that it is generic and works on any quadrilateral pattern. The user can add a new pattern by simply specifying the coarse quadrilateral mesh. The construction of the sides, chords and linear systems are all handled automatically. While it is possible to construct thousands of patterns by enumerating the quadrangulations of the disk (Sec. 6.6.3), it should be noted that most of them do not lead to high-quality quadrilateral meshes. For this reason, we restrict ourselves to the patterns of Fig. 6.4 that we selected manually.

For a practical example, consider a rectangular CAD face with respectively $[N_1, N_2, N_3, N_4]$ edges on its sides. It can potentially match with the first four patterns of Fig. 6.4. For the first pattern, the condition is obviously equality of

the subdivisions on the opposite sides ($N_1 = N_3$ and $N_2 = N_4$). Let consider a more interesting case: the third pattern. The sum of the chord subdivisions on each side gives the system:

$$\begin{aligned} a + b + c &= N_1 \\ d + e &= N_2 \\ a + c &= N_3 \\ b + d + e &= N_4 \end{aligned}$$

We can see that b must respect $b = N_3 - N_1$ and $b = N_4 - N_2$, which is a quite natural condition if we look at the dual chords (Fig. 6.4). On the other hand, there is no solution for this pattern if $N_3 - N_1 \neq N_4 - N_2$. Assuming the above condition is respected, the balance between the respective subdivisions a, c and d, e are left to the minimization of the objective function, or to the depth-first search with our technique.

Related work The above integer formulation is not new and has been introduced for patterns with a single interior vertex in Bunin [2008]. A general ILP formulation for more complicated patterns has been developed in Takayama et al. [2014] and is equivalent to ours (Eq. 6.14).

Geometry The integer formulation is purely topological and does not take geometry into account. A very interesting extension would be to find a way to incorporate geometric information from the cavity boundary into the objective function, in order to find optimal chord subdivisions. In our pipeline, the mesh geometry is obtained by smoothing (Sec. 6.5.3) with a fixed boundary, and the new quadrilateral mesh is only kept if its quality is better than the initial one.

6.6.1.2 Application to simple CAD face meshing

In a complicated CAD model (*e.g.* Fig. 6.9), there are usually lot of CAD faces but many of them have the topology of simple polygons (*e.g.* triangles, rectangles, pentagons). Given such polygon and the number of edges on each of its sides, it is often possible to directly find a quadrilateral mesh with our topological matching technique (Sec. 6.6.1.1).

In practice, we loop over all CAD faces and check if they have the same topology (χ and convex corners) as some of our patterns. When the topology is identical, we search for a subdivision of the pattern which matches the number of edges on each side of the CAD face, previously determined by the curve quantization step (Sec. 6.4.3). This matching exists if there is a strictly positive solution to the integer linear problem (Eq. 6.14). Once the quadrilateral mesh topology is determined, we can find the vertex positions by employing our quad mesh

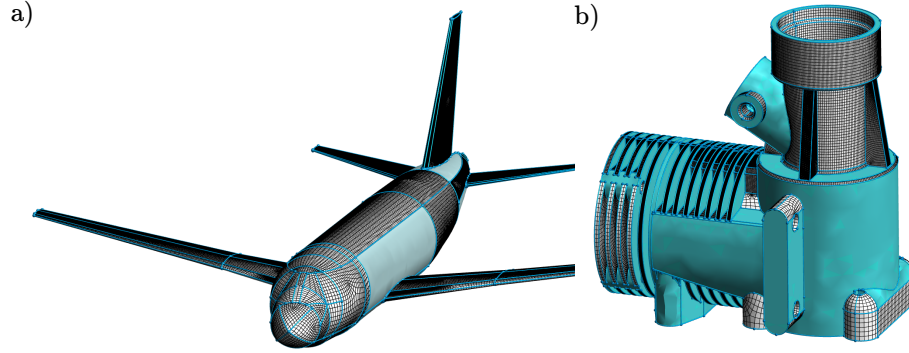


Figure 6.5: Pattern-based quadrilateral meshes of 75/83 CAD faces on the A319 model (a) and 419/533 CAD faces on the Block model (b).

geometry smoother (Sec. 6.5.3). We only keep the quad mesh only if the geometric quality is satisfactory, and if it is not we fall back to the unstructured approach.

Examples To see the usefulness of this pattern-based pre-meshing in real-life situations, we can look at the results on two CAD models. The *A319 model* is made of 83 CAD faces, 275 CAD curves and 287 CAD corners and the *Block model* is made of 533 CAD faces, 1584 CAD curves and 1044 CAD corners. After the curve meshing (Sec. 6.4.3), we apply our pattern-matching on all CAD faces and we are able to directly build the quadrilateral meshes of respectively 75/83 and 419/533 CAD faces, as shown in Fig. 6.5. These results are very interesting because it means that we are able to build high quality meshes on most of the simple CAD faces in a short amount time, for which there is no need to use more computationally expensive techniques.

6.6.2 Quasi-structured topology with cavity remeshing

On the remaining CAD faces, we start from the unstructured quadrilateral mesh (Sec. 6.5) and we improve its topology by locally remeshing cavities with the pattern-matching technique (Sec. 6.6.1). Our objective is to iteratively reduce the number of irregular vertices in order to reach a quasi-structured topology. Consider a CAD face, there are four sources of irregular vertices: (i) the face topology (see Eq. 6.2), (ii) the face geometry, (iii) the non-ideal quantization of the curves (Sec. 6.4.3) and (iv) the non-ideal behavior of the frontal mesher (frontal collisions, size map not adapted to CAD features or cross field gradients, etc). The irregular vertices from (i) and (ii) are captured as cross field singularities (including the ones we artificially added at acute

corners). In the current paper, we work on removing the irregular vertices (iv) caused by the unstructured quadrilateral mesher, while preserving the irregular vertices (i) and (ii) which are necessary to have a *good* quadrilateral mesh in the end. It should be noted that the irregular vertices (iii) caused by the non-ideal quantization cannot be removed by considering only one CAD face, but the whole model quadrilateral mesh should be taken into account. We leave this last improvement for future work as it is definitively not trivial: the global quadrilateral mesh is potentially non-manifold (*e.g.* multiple volumes) and the CAD features must be strictly preserved while applying global operations across multiple CAD faces.

To remove the unnecessary irregular vertices, we build convex topological cavities in the quadrilateral mesh, which contains three or more irregular vertices, and we replace them by *more regular* quad meshes (the patterns defined in Sec. 6.6.1). This process is illustrated on Fig. 6.6, where a pentagonal cavity with five valence 3 ($k = 1$) and six valence 5 ($k = -1$) irregular vertices ($\sum k = -1$) is replaced by a pentagonal patch with only one valence 5 irregular vertex ($k = -1$). The key to our approach is to not apply this process blindly but in a specific way to preserve the irregular vertices matching cross field singularities.

The idea of remeshing cavities with more regular patterns is not new [Bumin, 2008; Takayama et al., 2014] and has been exploited in quad meshing [Verma and Tautges, 2011; Marcias et al., 2015; Verma and Suresh, 2015]. The difficulty is in choosing which cavities to remesh, because uncontrolled greedy remeshing will remove irregular vertices that were important for mesh quality, leading to highly distorted meshes. For instance, a rectangular cavity may eliminate a valence three singularity close to a convex CAD curve and a valence five singularity close to a concave CAD curve, which were both useful irregular vertices used to accommodate the model geometry.

We call *singularities* the irregular vertices that we preserve, which mainly come from the cross field singularities, and we call *unnecessary* irregular vertices the ones we want to eliminate. The singularities are used to control and constrain the growth of the remeshable cavities that absorb the unnecessary irregular vertices.

6.6.2.1 Growing remeshable cavities

To iteratively improve the mesh of a CAD face $\mathcal{S}$, we grow convex cavities (patch of quads) and replace them. To grow a convex cavity C , we start from a initial simply connected set of quads C_0 (*e.g.* quads adjacent to a vertex) and we iteratively add quads adjacent to the cavity. To ensure the convexity of the cavity, if one of the vertex $\mathbf{x}$ on the boundary ∂C is concave, *e.g.* $\mathbf{x} \notin \partial \mathcal{S}$ and $n_{\mathbf{x}}^{\text{out}} = 1$, we add the adjacent exterior quad in priority. During the growth, we also ensure that (i) a singularity (flagged irregular vertex to preserve) is

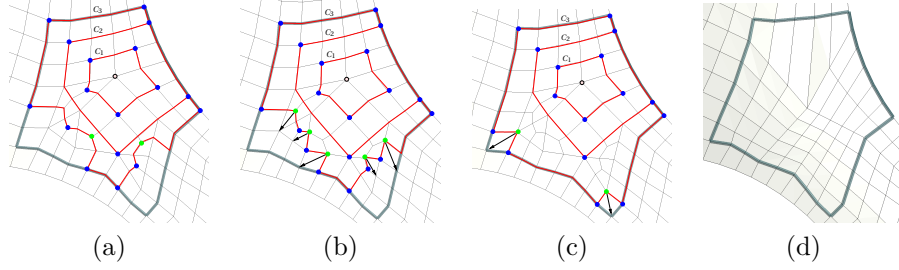


Figure 6.6: Growing a cavity around one vertex of index -1 (in pink). Convex corners are in blue and concave corners are in green. The remeshed cavity (d) has one irregular vertex instead of eleven.

never added to the interior of the cavity and (ii) a CAD concave corner is never fully surrounded by the cavity ($\mathbf{x}$ concave CAD corner, $n_{\mathbf{x}}^{\text{out}} = 0$). This process is illustrated in Fig. 6.6, where a pentagonal cavity is growing starting from a valence-5 singularity. Each time the growing cavity is convex and has absorbed new irregular vertices, we check if it is remeshable by solving the matching problem (Sec. 6.6.1.1) with the target replacement patterns. If it is remeshable, we store it as the *last remeshable* cavity. We either stop if we want a minimal cavity, or we continue the growth if we want a maximal cavity.

6.6.2.2 Cavity remeshing strategy

The choice of cavity seeds and the order in which cavities are grown and remeshed matters. When a triangular or a pentagonal cavity built around a singularity is remeshed, removing pairs of $+1/-1$ irregular vertices, it produces quads with only one irregular vertex (associated to the initial singularity), but the new singularity topological position is *translated* compared to the initial one. On Fig. 6.6.d., we can observe that the new irregular vertex is moved closer to the top side (distance of one edge instead of three initially). Another way to interpret this is that when a pair of $+1/-1$ irregular vertices is eliminated via a singularity, it will attract it or push it away, depending of the pair orientation and the singularity index. This behavior has important consequences: if a mesh singularity is used a lot to absorb all other irregular vertices, it may end up far away from the initial cross field singularity, leading to important distortions in the mesh geometry. So in order to produce a high-quality quasi-structured mesh, we must adopt a cavity remeshing strategy that will minimize the movements of the singular vertices and keep them close to their initial position in the cross field.

On a given CAD face $\mathcal{S}$, we apply the following steps as long as there is im-

provement (*i.e.* in a while loop):

- Step 1: Starting from unnecessary irregular vertices, grow and remesh maximal rectangular cavities that do not contain singularities and that match the regular grid pattern (first one on Fig. 6.4). The goal is to eliminate the opposed irregular $+1/-1$ pairs without distorting the mesh.
- Step 2: Starting from the singularities (index $k = +1$ or $k = -1$), grow and remesh minimal triangular or pentagonal cavities that contain the starting singularity and the non-zero minimum number of unnecessary irregular vertices while keeping the target index (*i.e.* $\sum k = -1$ or $+1$). The goal is to eliminate the irregular vertices while distributing the mesh distortion on all the singularities.
- Step 3: Re-apply Step 1, in case some new regular cavities have been *unlocked* by the previous step.
- Step 4: Starting from unnecessary irregular vertices, grow and remesh minimal rectangular cavities that do not contain singularities and that match one of the four rectangular patterns of Fig. 6.4. The goal is to enable size transitions in the mesh while reducing the number of irregular vertex pairs. To avoid large geometric distortions, we use minimal remeshable cavities and we verify that the mesh quality is not degraded too much after smoothing.

This process is the core of our quasi-structured topological improvement. Fig. 6.2.2. illustrates the absorption of irregular vertices by the cross singularities. For a more complicated example, we can look at the *Block* model illustrated in Fig. 6.9, where the zooms show the kind of mesh size transitions which are produced by the rectangular pattern cavity remeshing.

6.6.3 Local remeshing with disk quadrangulations

Unstructured quad meshes constructed in an indirect fashion may contain a few very irregular vertices, that we call defects. The goal of this local remeshing is to eliminate them before applying the larger cavity remeshing (Sec. 6.6.2). In the interior of a CAD face, we say that a vertex is very irregular if its quadrilateral valence is more than 6, *i.e.* its index is $k \leq -2$. On the CAD curves, we say that a vertex is a defect if its valence is different from two on the adjacent CAD faces. On CAD corners, we want to enforce the ideal valence which is deduced from the associated angle in the CAD faces, *e.g.* valence three for a concave 270° corner. We can also enforce user-prescribed corner valences

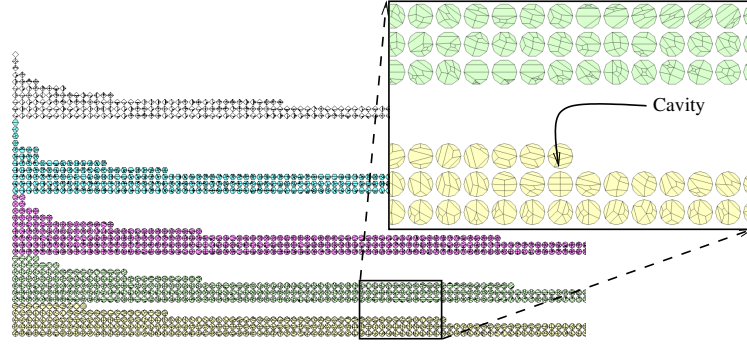


Figure 6.7: All quadrangulations of the disk bounded by 4, 6, 8, 10 and 12 vertices with 0, 1, 2, 3 and 4 internal vertices.

with this technique, which can be useful when hand-crafting boundary layers for specific applications. The goal of the local remeshing step is to eliminate the defects (valence different from the allowed range) by changing the topology locally.

When forming a cavity with the quads adjacent to a vertex, we are building a topological disk. We can replace the interior of the cavity with any quadrilateral mesh as long as we do not change the cavity boundary, which is a closed polyline. Our idea is simple: browse all the possible disk quadrangulations (Sec. 6.6.3.1) and use the best one, according to a *quality* criterion (Sec. 6.6.3.2). An example of elimination of a valence six vertices is shown on Fig. 6.8.

6.6.3.1 Quadrangulations of the disk

The exhaustive list of disk quadrangulations can be built by recursively applying the three edge flips that add a quad to a current quadrangulation, starting initially from a single quad. This is a simple 2D version of the algorithm used to build shellable hexahedral meshes of the sphere in Verhetsel et al. [2019]. Our implementation is open-source¹. The only technical difficulty is in detecting equivalent quadrilateral meshes that have different vertex ordering. For this we use the open-source library *Nauty* [McKay and Piperno, 2014] to compute the canonical labeling of the edge graphs. In our meshing software, we only store the list of the disk quadrangulations up to a certain size in a large table.

¹Software to enumerate disk quadrangulations:
https://git.immc.ucl.ac.be/reberol/disk_quadrangulation

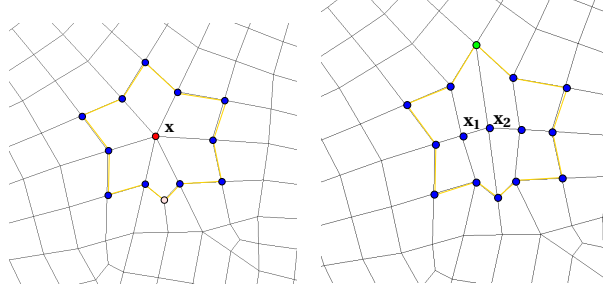


Figure 6.8: Removing a very irregular vertex $\mathbf{x}$ (in red) of index $k = -2$. Blue dots are regular vertices of index $k = 0$. The green dot has an index -1 while rose dot has index $+1$.

For practical meshing, the only disk quadrangulations that are interesting are the ones with valence three, four, five inside (indices $1, 0, -1$) and valences one, two, three on the boundary (indices $1, 0, -1$). On Fig. 6.7, we show the disk quadrangulations with up to 12 vertices on the boundary and up to 4 vertices in the interior. These configurations are more general than the patterns previously used (Sec. 6.6.2) because they include concave vertices on the cavity boundary.

6.6.3.2 Local remeshing

Consider a cavity C that we want to remesh, with its boundary ∂C defined by the polyline $[\mathbf{x}_1, \dots, \mathbf{x}_b]$. We want to select the *best* replacement in the list of disk quadrangulations with b vertices on the boundary, including all rotations clockwise and counter-clockwise of the patterns. As the cavity is immersed in a larger quadrilateral mesh, the quadrilateral valence outside the cavity associated to each vertex of ∂C is fixed: $[n_{\mathbf{x}_1}^{\text{out}}, \dots, n_{\mathbf{x}_b}^{\text{out}}]$. The future quad valences inside the cavity $[n_{\mathbf{x}_1}^{\text{in}}, \dots, n_{\mathbf{x}_b}^{\text{in}}]$ are parameters of each disk quadrangulation. We also know the ideal valences $[\bar{n}_{\mathbf{x}_1}, \dots, \bar{n}_{\mathbf{x}_b}]$ in the global quadrilateral mesh according to their position in the CAD face, *e.g.* $\bar{n}_{\mathbf{x}_j} = 4$ inside if regular, $\bar{n}_{\mathbf{x}_j} = 2$ on a curve, $\bar{n}_{\mathbf{x}_j} = 3$ on a concave corner.

The best disk quadrangulation replacement is the one which minimizes the *irregularity* of the cavity after inclusion in the complete quadrilateral mesh:

$$I_r = \sum_{j=1..b} (\bar{n}_{\mathbf{x}_j} - n_{\mathbf{x}_j}^{\text{out}} - n_{\mathbf{x}_j}^{\text{in}})^2 + \sum_{k \in \{\text{inside}\}} (4 - n_{\mathbf{x}_k}^{\text{in}})^2 \quad (6.15)$$

If there are multiple ideal quadrangulations ($I_r = 0$), we choose the one with the minimal number of vertices. But in practice, it is not always possible to

find ideal replacements, and it is important to not introduce new defects in the mesh. So we restrict the allowed integer ranges of the $n_{\mathbf{x}_j}^{\text{in}}$. For a CAD face interior vertex ($\bar{n}_{\mathbf{x}_j} = 4$), the allowed range is such that $3 \leq n_{\mathbf{x}_j}^{\text{in}} + n_{\mathbf{x}_j}^{\text{out}} \leq 5$. For a CAD curve vertex, we typically force the regularity: $n_{\mathbf{x}_j}^{\text{in}} + n_{\mathbf{x}_j}^{\text{out}} = 2$. For a CAD corner, we also force the ideal valence: $n_{\mathbf{x}_j}^{\text{in}} + n_{\mathbf{x}_j}^{\text{out}} = \bar{n}_{\mathbf{x}_j}$. Because the problem is very constrained when the CAD topology and geometry is locally complex, it is not always possible to satisfy all constraints (even if we are considering thousands of replacement candidates, thus the difficulty of quad meshing). To still improve the quadrilateral mesh, we introduce priorities: it is better to improve the corners, then the curves, then the interior. We allow the creation of curves and interiors defects in order to fix the corners, and then we allow the creation of interior defects when fixing the curves. Such relaxation is necessary to not get stuck early in the improvement process.

Geometry As the disk quadrangulation are purely topological, the actual mesh geometry is computed with our smoothing and untangling technique (Sec. 6.5.3). If we fix the boundary of the cavity, there is usually no geometrically valid solution. This is due to the fact that the quad topology typically changes significantly while the boundary stays a small polygon: there is not enough room to untangle and optimize the elements. To relax the geometric problem, we extend the smoothing cavity to the quads adjacent to the topological cavity, and we apply the optimization on this enlarged cavity, with its boundary fixed. If there is still no satisfactory geometric solution (*e.g.* tangled quad), we cancel the current disk quadrangulation remeshing.

Limitations In practice, most of the defects are eliminated, but sometimes it is not possible to remove some of them when the region is too constrained (*e.g.* almost all vertices of a cavity on CAD curves). With luck, they will be eliminated later by larger convex cavity remeshing (Sec. 6.6.2) but there is no guarantee. That being said, what we call defect are local configurations which are totally acceptable in the broader context of unstructured quad meshing techniques.

6.7 Results and discussions

To validate our approach and confirm its robustness, we applied it on many CAD models from existing databases (Sec. 6.7.1). In the next section (Sec. 6.7.2), we discuss the strengths and weaknesses of the approach and its positioning relative to existing techniques in the literature.

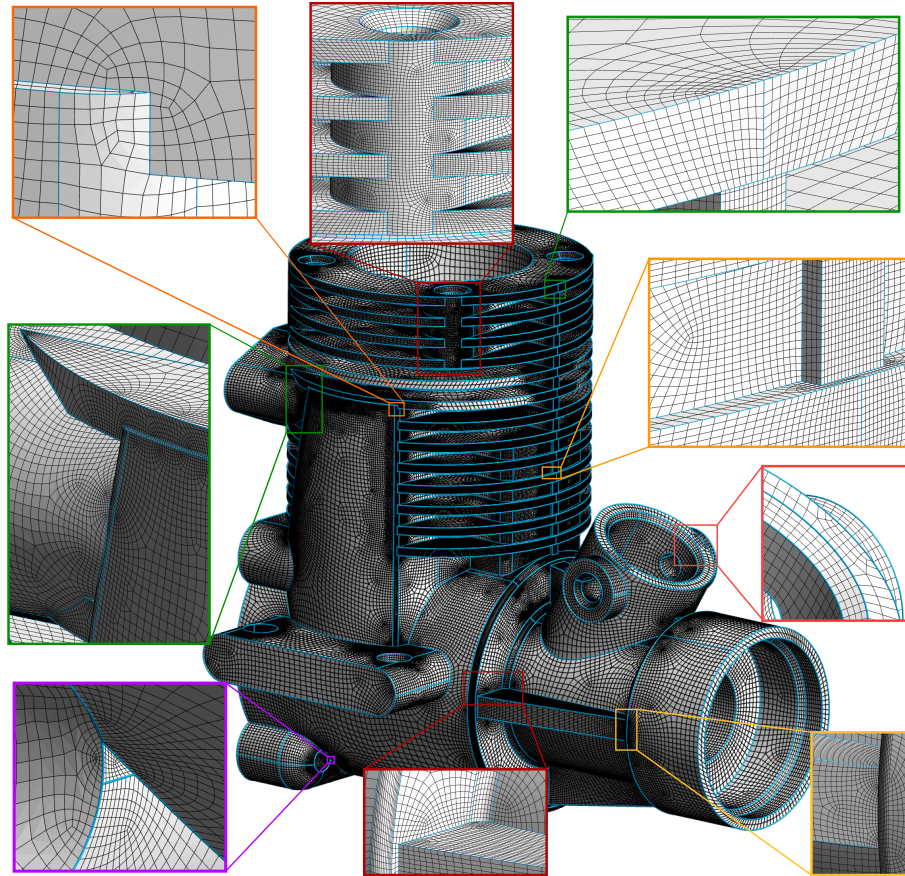


Figure 6.9: Quasi-structured quad mesh (261.5k vertices, 261.6k quads) of the *Block* model (533 faces and 1584 curves). The average SICN quality is 0.87 and the minimum is 0.11. The initial unstructured quad mesh was generated in 58 seconds and the quasi-structured improvement took 33 seconds, both with 4 CPU cores on a laptop. The number of irregular vertices was reduced from 14.4k to 3.6k.

6.7.1 CAD databases

Test cases We applied the end-to-end meshing pipeline to the 114 models of the MAMBO dataset [Ledoux] and to 1000 models the ABC dataset [Koch et al., 2019]. As the ABC dataset is huge (one million models), we randomly selected one thousand not-too-simple models for which it was possible to generate triangulations and extract a single CAD volume. This pre-selection is necessary because many models are very simple (*e.g.* a few CAD primitives), or are not valid, or contain lot of independent components. For all models, we applied a global sizing parameter of 0.1 on mesh edges, *i.e.* edges ten times smaller than the *initial* very coarse CAD sampling in the triangular Delaunay mesher. In our pipeline, a size map is then produced to adapt to CAD feature locally, but the mesh resolution in the final mesh is largely controlled by this single parameter. It should be noted that using the same resolution parameter for all CAD models is very far from ideal, as the resulting meshes are either too coarse or too fine relatively to the CAD geometry, compared to what a user would choose. We still followed this procedure because it allows large scale automation and it is a good way to observe the mesher behavior in non-optimal conditions.

Results We were able to automatically build geometrically valid quad meshes on the 114/114 models of the MAMBO dataset and on 988/1000 models of the ABC dataset. In the 12 failure cases in the ABC dataset, two are topologically valid quad meshes but with tangled elements, the minimum SICN quality being respectively -0.008 and -0.11 . The invalid quads are due to inverted triangles in the quad-dominant mesh, which is too coarse in some regions. The ten other failures are due to either (a) issues in the Delaunay point insertion used for the initial triangulation and frontal quad-dominant mesher or (b) performance issues (exceed the 8GB RAM or timeout after 5 hours) in the quasi-structured topological improvement when the size map is too fine. The cases (b) typically correspond to elongated models with large thin regions. Our automatic size map leads to very small elements and many millions vertices, many of them irregular, in the CAD faces. Our topological improvement strategy is not adapted in these situations because too many cavities (literally millions) are built, smoothed and remeshed, which is computationally too expensive. The above failure cases can be fixed by using a more appropriate global sizing parameter or by manually building a size map better adapted to the thin regions.

Statistics Topological and geometrical statistics, along with figures, are available for each model in the supplementary material. Average values for some quantities are listed in Table 6.1. Statistically, the quads are of very high quality and the topological improvement reduces the number of irregular ver-

Dataset (#)	MAMBO (114)	ABC (990)
#C	46	98
#F	18	38
Q min	0.53	0.44
Q avg	0.98	0.96
#vert	20.2k	49k
% irreg. init.	1.24%	2.2%
% irreg.	0.27 %	0.74 %
time init./improv.	9.6 sec / 4.6 sec	15.3 sec / 140.8 sec *

Table 6.1: Average values over all dataset models. #C and #F are the number of CAD curves and faces. Mesh statistics are computed on the final quasi-structured mesh, except for the initial percentage of irregular vertices, which is computed on the initial unstructured quad mesh. The quality metric Q is the signed inverse condition number (SICN). Time init. is the elapsed time to build the unstructured quad mesh starting from the CAD and time improv. is the elapsed time to make it quasi-structured, both computed with a single core. (*) The average run-time of 140.8 sec is high due to slow outliers (seven models took more than one hour), the median value is 5.9 seconds.

tices by a factor of three (ABC) or five (MAMBO). An important proportion of the remaining irregular vertices are due to large gradients in the size map. This is caused by the global sizing parameter which is often too coarse compared to the smallest features. Global sizing parameter chosen model by model would allow a greater reduction of the number of irregular vertices, typically a factor from 5 to 10. The median and average running times for the whole pipeline are respectively 18 and 155 seconds, using a single-thread on a laptop 2.8 GHz CPU. If we normalize by the number of vertices, our complete pipeline takes around one second per thousand vertices in the output, in average.

6.7.2 Discussion

Statistical results (Sec. 6.7.1) are interesting to confirm the robustness of our approach but they are not very relevant to evaluate the mesher’s behavior. They are strongly affected by the target meshing size, *i.e.* smaller size would improve both regularity and quality. In the same spirit, midpoint subdivision steps would artificially improve the statistics whereas the topological structure is not changed. To better understand the mesher strengths and weaknesses, it is necessary to visually inspect the quadrilateral meshes. A detailed quad mesh of a complex model is illustrated on Fig. 6.9 and some of the database results are shown in the mosaic ² in Fig. 6.10.

²generated with the Mosaic generator (github.com/qnzhou/Mosaic)

Overview Our end-to-end meshing pipeline is able to robustly generate quadrilateral meshes with high-quality elements on almost all CAD models, using a single global sizing parameter. There are a few failure cases, but they are due to either a *very* wrong target size choice or a CAD-adapted size map leading to too many vertices ($> 5M$), for which we have performance issues. When using an appropriate target size, we are able to eliminate many irregular vertices (*e.g.* factor five to ten) in the initial unstructured mesh, while preserving a relevant global structure thanks to the cross-field singularities. In its current state, this new mesher is a viable alternative to triangular meshing in Gmsh for generic CAD models and we expect further improvements in the future. The flexibility offered by the unstructured approach makes it easy to integrate user-prescribed constraints, such as specific element sizes at given points or a custom size map given as a scalar field.

Comparison with unstructured techniques There are two large steps in our approach: the robust generation of the initial unstructured quad mesh and its improvement via cavity remeshing. On the unstructured part, our technique is essentially an incremental improvement of standard indirect meshing [Remacle et al., 2012, 2013]. Thanks to the accurate cross field computation and its inherent size map (conformal scaling), we generate better vertex positions which follow integral lines of the *locally integrable* scaled cross field (*e.g.* Fig. 6.2.2.b). Concerning the topological improvement, we employ two topological techniques. For local and constrained configurations, we pick the best rearrangement by checking in an exhaustive list of disk quadrangulations (Sec. 6.6.3), which is a new technique to our knowledge. For large cavities and simple CAD faces, we use pattern-based remeshing similar to Verma and Suresh [2015] but with better cavities as we are using the cross-field singularities to control their spawning and growth.

Comparison with global parametrization techniques When the size map is dominated by the cross field scaling and not by the adaptation to small features, our approach produces meshes tending to the ones produced by global parametrization techniques (*e.g.* Fig. 6.2), except that there may be a few additional irregular vertices due to the non optimal quantization of the curves. Most global parametrization literature is from the computer graphics community and focus on smooth models without or with few feature curves, without much consideration for CAD entities. Thus, it is difficult to know if they could be adapted to strictly respect them. As they are using a global sizing parameter without adaptation to small features, the basic way to match them is to introduce strong distortions in the parametrization and it is not clear if it is possible to extract geometrically valid quads afterwards. Our approach has the advantage of working with custom size maps containing strong gradients

whereas global parametrization techniques would first require to generate a cross field matching the size map, which is an open problem.

Performance The initial unstructured quad mesh generation, starting from the CAD, is fast and is a few times the cost of triangular meshing. The computational cost of the topological improvement is less predictable as it depends on the number and size of the cavities that are candidate for remeshing. In general, the cost is similar to the initial step but it may be much higher, especially for large CAD faces with lot of vertices ($>100k$). The vast majority of the time is spent doing smoothing, which is essential to guarantee geometric validity of the elements. To ensure limited running times, it is possible for the user to limit the time spent on each CAD face. In the future, we plan to develop better cavity growing strategies to deal more efficiently with large meshes by reducing the number of cavity candidates.

Thanks to the bottom-up approach, meshing and remeshing operations can be applied in parallel on each CAD face, as the boundary meshes are fixed. The scaling with the number of cores is excellent, and we are only limited by the RAM as we are using shared memory in Gmsh parallelism (OpenMP).

6.8 Perspectives and future work

In this work, we have presented a quadrilateral meshing pipeline which is viable for industrial use. The robustness is ensured by starting from a basic unstructured quad mesh (quad-dominant split into quads) and iteratively improving it while always checking the validity of each operation. Additional improvements stemming from recent research on cross fields are incorporated in the pipeline as guiding information, but never at the cost of robustness. Starting from this solid basis, we envision various lines of research to further enhance the topological structure of the final quad meshes.

Remaining non-essential irregular vertices When the target size is fine and the size map is dominated by the cross field and not the CAD features, we would ideally expect to match the cross field singularities. With our approach, it requires an optimal quantization of curves, which is hardly achieved. We see two solutions to this problem, either a better initial quantization or a posteriori remeshing across CAD curves.

For the initial quantization, we should consider more the topological constraints in quadrilateral mesh. In particular, the curves of the polygonal CAD faces whose topology matches the patterns in Fig. 6.4 should have a number of edges such that the quantization is meshable according to the ILP problem. For the CAD faces without cross field singularities, we can assign a direction



Figure 6.10: Quadrilateral meshes of models in the MAMBO [Ledoux] and ABC [Koch et al., 2019] datasets.

$+x, -x, +y, -y$ to each boundary curve and ensure that the sum of the edges over both axis is zero. By combining these linear constraints in a global mixed-integer problem, we should be able to produce a better initial quantization of the whole model.

A more pragmatic alternative is to eliminate the remaining irregular vertices by modifying the mesh topology globally. The remeshing cavities could grow across the CAD curves to surround opposite pairs of $+1/-1$ irregular vertices on distinct faces. The difficulty is in ensuring the CAD curves are still preserved after remeshing and in dealing with non-manifold face junctions. Another possibility is to move the pairs of $+1/-1$ in the global mesh by successively applying local mesh modifications.

Performance and large models The main limitation of our mesher is that it is slow when dealing with large CAD face meshes ($> 100k$ vertices) because there are too many remeshing cavities which are tried, and each one involves smoothing the geometry. We believe there is a lot of room for improvement in the remeshing strategy (Sec. 6.6.2.2), which is quite simple for the moment. A possibility could be to replace the current isotropic growth of cavities by more surgical ones which would target the non-essential irregular vertices and avoid the regions of the mesh which are already regular.

From quasi-structured to block-structured For many applications, the best quadrilateral mesh is block-structured, meaning that its base-complex is coarse and composed of a minimum of large quadrilateral patches. This kind of meshes is interesting because it can be easily converted into a good high-order quad mesh which is ideal for accurate high-order methods. Even at low orders, the block structure allows for fast computational optimizations (*e.g.* implicit indexing, multigrid). Our quasi-structured meshes are not ready for such applications because their base-complexes are composed of too many patches. Even when the number of irregular vertices is low (*e.g.* matching the cross field singularities), the issue is that the irregular vertices are not aligned and consequently the topological chords tend to wind up over the whole model. In the future, we wish to explore ways to align the irregular vertices, such as [Bommes et al. \[2011\]](#), and build coarse layouts on complex CAD models. This is definitively a challenging problem as the CAD features must be fully preserved during the process.

Extension to hexahedral meshes Hexahedral meshing is a very difficult problem, especially when trying to respect the features of CAD models. The current robust techniques are either the subdivision of a tetrahedral mesh into hexahedra or the octree-based approach [[Maréchal, 2009](#)]. Both produce meshes whose topological structure is not very good: there is a large number

of irregular edges and vertices, and the hexahedra are not aligned with the model boundary. We think that exploring ways to extend the cavity remeshing technique to the 3D case could be a fruitful line of research as it would allow iterative and robust topological improvements to reduce the number of irregular edges and vertices in such meshes.

CHAPTER



Hexahedral-dominant meshing using 3D integrable frame-fields.

This chapter is a reproduction of the following paper:

Georgiadis, C., Chemin, A., Remacle, J.-F., draft, *Hexahedral-dominant meshing using 3D integrable frame-fields.*

Abstract

Frame field based approaches on hexahedral mesh generation have gained a lot of attention recently. The success of this class of methods is strongly dependent on the characteristics of the input frame fields, since they should have a topological structure that is *hex-meshable*. In this article we develop mathematical conditions that a frame field has to fulfill in order to be integrable. The resulting problem is highly complex, thus we follow a methodology to scale input frame fields without changing their orientation, in order to be as close as possible to integrability. The resulting scaled frame fields are then used to generate hex-dominant meshes with a frontal point insertion approach, tetrahedralization and indirect combination of tetrahedra to hexahedra. Finally, we investigate the practical benefits of frame field scaling on a variety of models.

Keywords: 3D frame fields, integrable frame fields, hex-dominant meshing

7.1 Introduction

The problem of automatically generating a hexahedral mesh that conforms to the boundary and offers quality and robustness guarantees is widely considered as the most challenging problem of mesh generation. While a variety of related methods have been proposed, their applicability is mostly restricted on simple models. In practice, semi-manual decomposition methods are used to handle more complex models.

In the last years, frame-field based approaches have been widely investigated for the automatic generation of hexahedral meshes, as a natural extension of 2D cross-field based approaches for quadrilateral meshing. Frame fields are auxiliary in 3D mesh generation to define local preferred directions ($\mathcal{D}_{\mathbf{u}}, \mathcal{D}_{\mathbf{v}}, \mathcal{D}_{\mathbf{w}}$) and sizes ($L_{\mathbf{u}}, L_{\mathbf{v}}, L_{\mathbf{w}}$) for hexahedral and hex-dominant meshing. Standard frame field based meshing approaches consist in building a boundary-aligned frame field [Huang et al., 2011; Chemin et al., 2019; Ray et al., 2016] and using it to build a hex-dominant [Gao et al., 2017; Ray et al., 2018; Sokolov et al., 2016] or a fully hexahedral mesh [Zheng et al., 2019; Lyon et al., 2016; Li et al., 2012; Nieser et al., 2011; Liu et al., 2018]. Frame fields can also be utilized by frontal approaches as in Baudouin et al. [2014]; Bernard et al. [2016] for the

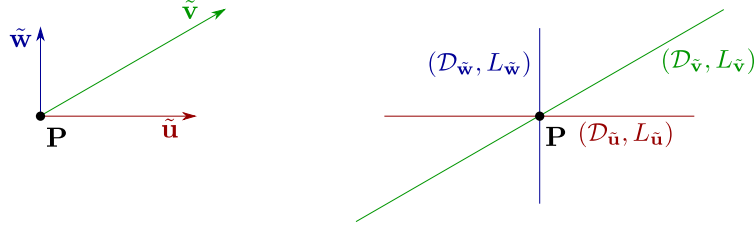


Figure 7.1: Scaled frame (on the left) and preferred directions and sizes (on the right).

generation of hex-dominant meshing. Frontal approaches are more straightforward and flexible compared to parameterization methods, at the cost of resulting hex-dominant meshes with lower percentage of hexahedra.

These methods are all relying on the fact that the considered frame field can be used as a local system of coordinates, and therefore rely on the frame field integrability. When this criterion is not met, global parametrization approaches fail and hex-dominant ones produce a lot of tetrahedra. One manifestation of non integrability is the existence of 3–5 singular curves in the computed frame field, and we refer the reader to [Reberol et al. \[2019\]](#) in which a comprehensive analysis of this problem and existing methods to circumvent it is done. In this case, the problem remains largely unsolved.

We define here a scaled frame $\tilde{f}$ as a set of 3 distinct vectors mutually orthogonal $(\tilde{\mathbf{u}}, \tilde{\mathbf{v}}, \tilde{\mathbf{w}})$ (Fig. 7.1). A scaled frame field $\tilde{F} = \{P \in \Omega \subset E^3 \mapsto \tilde{f}(P)\}$, now, is a rule that associates a frame $\tilde{f}(P)$ to each point P in a subset Ω of E^3 . $\tilde{\mathbf{u}}$, $\tilde{\mathbf{v}}$ and $\tilde{\mathbf{w}}$ are called branches of the frame $\tilde{f}$. A scaled frame f composed of 3 unitary vectors will be called a unitary frame, and a scaled frame field $F = \{P \in \Omega \subset E^3 \mapsto f(P)\}$ associating to each point $P \in \Omega$ a unitary frame $f(P)$ will be called a unitary frame field. The frame field approaches commonly use what we call here unitary frames.

In this article we develop mathematical conditions that a scaled frame field $\tilde{f}$ has to fulfill in order to be integrable, so that preferred directions $(\mathcal{D}_{\tilde{\mathbf{u}}}, \mathcal{D}_{\tilde{\mathbf{v}}}, \mathcal{D}_{\tilde{\mathbf{w}}})$ and preferred sizes $(L_{\tilde{\mathbf{u}}} = \|\tilde{\mathbf{u}}\|, L_{\tilde{\mathbf{v}}} = \|\tilde{\mathbf{v}}\|, L_{\tilde{\mathbf{w}}} = \|\tilde{\mathbf{w}}\|)$ are coherent for hexahedral meshing. As it will be shown later these conditions are coupling branches' sizes with frame field orientation, and are also imposing conditions on the frame field orientation itself. Equations which have to be fulfilled are very difficult to solve if neither orientation nor branches' size are known.

Therefore, exploitation of mathematical results obtained will be limited here to computation of branches' size for an input given frame field in order to be as close as possible to integrability conditions. It means computing $L_{\tilde{\mathbf{u}}} = \|\tilde{\mathbf{u}}\|, L_{\tilde{\mathbf{v}}} = \|\tilde{\mathbf{v}}\|, L_{\tilde{\mathbf{w}}} = \|\tilde{\mathbf{w}}\|$ for a fixed frame field orientation. If given frame

field is not already respecting orientation integrability conditions, which is going to be the case in general, it will be impossible to reach integrability.

We then utilize the scaled frame fields for the generation of hex-dominant meshes with a frontal approach. The idea is to propagate points inside the domain following preferred directions $(\mathcal{D}_{\mathbf{u}}, \mathcal{D}_{\mathbf{v}}, \mathcal{D}_{\mathbf{w}})$ and respecting the spacing induced by preferred sizes $(L_{\mathbf{u}}, L_{\mathbf{v}}, L_{\mathbf{w}})$. Usually, preferred sizes $(L_{\mathbf{u}}, L_{\mathbf{v}}, L_{\mathbf{w}})$ are imposed by the user and are not correlated to the preferred directions $(\mathcal{D}_{\mathbf{u}}, \mathcal{D}_{\mathbf{v}}, \mathcal{D}_{\mathbf{w}})$. A tetrahedral mesh is then created from the generated point set and tetrahedra are combined into hexahedra in order to obtain a hex-dominant mesh. Some tetrahedra are remaining in the mesh as no combinations are possible to eliminate them. However, if the spacing between each generated point is not coherent with regard to the frame field orientation, more tetrahedra will remain after the recombination procedure. Our objective is to examine if imposing scaled sizing on the input frame field can prove beneficial in the context of hex and hex-dominant meshing.

7.2 Frame field integrability equations

We develop here the mathematical conditions that a scaled frame field has to fulfill in order to be integrable. The obtained equations are constraining frame field orientation and coupling the size and orientation of branches. As mentioned in the introduction, these equations are difficult to solve if orientation and size are unknown. We will then assume frame field orientation to be known (coming from a previous boundary aligned frame field computation done by any classical method) and present a method to scale branches in order to be as close as possible to an integrable frame field.

7.2.1 Integrability equations

A unitary frame is represented by 3 orthogonal unitary vectors $\mathbf{u}, \mathbf{v}, \mathbf{w}$ that we can define with the 3 euler angles (ψ, θ, ϕ) :

$$\mathbf{u} = \begin{pmatrix} -s_\phi s_\psi c_\theta + c_\phi c_\psi \\ s_\phi c_\psi c_\theta + s_\psi c_\phi \\ s_\phi s_\theta \end{pmatrix}, \mathbf{v} = \begin{pmatrix} -s_\phi c_\psi - s_\psi c_\phi c_\theta \\ -s_\phi s_\psi + c_\phi c_\psi c_\theta \\ s_\theta c_\phi \end{pmatrix}, \mathbf{w} = \begin{pmatrix} s_\psi s_\theta \\ -s_\theta c_\psi \\ c_\theta \end{pmatrix} \quad (7.1)$$

where $c_\alpha = \cos \alpha$ and $s_\alpha = \sin \alpha$.

We consider the following scaling of the 3 branches $\mathbf{u}, \mathbf{v}$ and $\mathbf{w}$:

$$\tilde{\mathbf{u}} = L\mathbf{u}, \tilde{\mathbf{v}} = M\mathbf{v}, \tilde{\mathbf{w}} = N\mathbf{w}, \quad (7.2)$$

where L, M and N are scalar fields defined on Ω .

The Lie bracket $[\mathbf{u}, \mathbf{v}]$ in the 3D Euclidian space is defined as:

$$[\mathbf{u}, \mathbf{v}] = \nabla_{\mathbf{v}}(\mathbf{u}) - \nabla_{\mathbf{u}}(\mathbf{v}) \quad (7.3)$$

where

$$\nabla_{\mathbf{v}}(\mathbf{u}) = \langle \nabla \mathbf{u}, \mathbf{v} \rangle = u_{i,j} v_j \quad (7.4)$$

Let's define:

$$\begin{cases} \mathbf{R}_0 = [\tilde{\mathbf{u}}, \tilde{\mathbf{v}}] = \nabla_{\tilde{\mathbf{v}}}(\tilde{\mathbf{u}}) - \nabla_{\tilde{\mathbf{u}}}(\tilde{\mathbf{v}}) \\ \mathbf{R}_1 = [\tilde{\mathbf{u}}, \tilde{\mathbf{w}}] = \nabla_{\tilde{\mathbf{w}}}(\tilde{\mathbf{u}}) - \nabla_{\tilde{\mathbf{u}}}(\tilde{\mathbf{w}}) \\ \mathbf{R}_2 = [\tilde{\mathbf{v}}, \tilde{\mathbf{w}}] = \nabla_{\tilde{\mathbf{w}}}(\tilde{\mathbf{v}}) - \nabla_{\tilde{\mathbf{v}}}(\tilde{\mathbf{w}}) \end{cases} \quad (7.5)$$

The scaled 3D frame field $(\tilde{\mathbf{u}}, \tilde{\mathbf{v}}, \tilde{\mathbf{w}})$ is integrable if and only if $\mathbf{R}_i = 0, i \in [0, 2]$, which gives:

$$\begin{cases} \mathbf{R}_0 = M\mathbf{u}\nabla_{\mathbf{v}}L - L\mathbf{v}\nabla_{\mathbf{u}}M + LM[\mathbf{w}(s_{\phi}\nabla_{\mathbf{v}}\theta - c_{\phi}\nabla_{\mathbf{u}}\theta) + (\mathbf{u}\nabla_{\mathbf{u}}\phi + \mathbf{v}\nabla_{\mathbf{v}}\phi) + (\mathbf{u}_{,\psi}\nabla_{\mathbf{v}}\psi - \mathbf{v}_{,\psi}\nabla_{\mathbf{u}}\psi)] & = \mathbf{0} \\ \mathbf{R}_1 = N\mathbf{u}\nabla_{\mathbf{w}}L - L\mathbf{w}\nabla_{\mathbf{u}}N + LN[(s_{\phi}\mathbf{w}\nabla_{\mathbf{w}}\theta - \mathbf{w}_{,\theta}\nabla_{\mathbf{u}}\theta) + \mathbf{v}\nabla_{\mathbf{w}}\phi + (\mathbf{u}_{,\psi}\nabla_{\mathbf{w}}\psi - \mathbf{w}_{,\psi}\nabla_{\mathbf{u}}\psi)] & = \mathbf{0} \\ \mathbf{R}_2 = N\mathbf{v}\nabla_{\mathbf{w}}M - M\mathbf{w}\nabla_{\mathbf{v}}N + MN[(c_{\phi}\mathbf{w}\nabla_{\mathbf{w}}\theta - \mathbf{w}_{,\theta}\nabla_{\mathbf{v}}\theta) - \mathbf{u}\nabla_{\mathbf{w}}\phi + (\mathbf{v}_{,\psi}\nabla_{\mathbf{w}}\psi - \mathbf{w}_{,\psi}\nabla_{\mathbf{v}}\psi)] & = \mathbf{0} \end{cases} \quad (7.6)$$

By defining:

$$\begin{aligned} H_L &= \log L \\ H_M &= \log M \\ H_N &= \log N \end{aligned} \quad (7.7)$$

it becomes:

$$(\nabla_{\mathbf{u}}H_M)\mathbf{v} - (\nabla_{\mathbf{v}}H_L)\mathbf{u} = (s_{\phi}\nabla_{\mathbf{v}}\theta - c_{\phi}\nabla_{\mathbf{u}}\theta)\mathbf{w} + (\nabla_{\mathbf{u}}\phi)\mathbf{u} + (\nabla_{\mathbf{v}}\phi)\mathbf{v} + (\nabla_{\mathbf{v}}\psi)\mathbf{u}_{,\psi} - (\nabla_{\mathbf{u}}\psi)\mathbf{v}_{,\psi} \quad (7.8a)$$

$$(\nabla_{\mathbf{u}}H_N)\mathbf{w} - (\nabla_{\mathbf{w}}H_L)\mathbf{u} = (s_{\phi}\nabla_{\mathbf{w}}\theta)\mathbf{w} - (\nabla_{\mathbf{u}}\theta)\mathbf{w}_{,\theta} + (\nabla_{\mathbf{w}}\phi)\mathbf{v} + (\nabla_{\mathbf{w}}\psi)\mathbf{u}_{,\psi} - (\nabla_{\mathbf{u}}\psi)\mathbf{w}_{,\psi} \quad (7.8b)$$

$$(\nabla_{\mathbf{v}}H_N)\mathbf{w} - (\nabla_{\mathbf{w}}H_M)\mathbf{v} = (c_{\phi}\nabla_{\mathbf{w}}\theta)\mathbf{w} - (\nabla_{\mathbf{v}}\theta)\mathbf{w}_{,\theta} - (\nabla_{\mathbf{w}}\phi)\mathbf{u} + (\nabla_{\mathbf{w}}\psi)\mathbf{v}_{,\psi} - (\nabla_{\mathbf{v}}\psi)\mathbf{w}_{,\psi} \quad (7.8c)$$

It is important to note that these vectorial equations give 9 scalar equations, of which 3 do not depend on branches scaling. Indeed, projecting 7.8a along $\mathbf{w}$, projecting 7.8b along $\mathbf{v}$ and projecting 7.8c along $\mathbf{u}$ gives:

$$\begin{cases} 0 &= s_{\phi}\nabla_{\mathbf{v}}\theta - c_{\phi}\nabla_{\mathbf{u}}\theta - s_{\theta}c_{\phi}\nabla_{\mathbf{v}}\psi - s_{\theta}s_{\phi}\nabla_{\mathbf{u}}\psi \\ 0 &= c_{\phi}\nabla_{\mathbf{u}}\theta + \nabla_{\mathbf{w}}\phi + c_{\theta}\nabla_{\mathbf{w}}\psi + s_{\theta}s_{\phi}\nabla_{\mathbf{u}}\psi \\ 0 &= s_{\phi}\nabla_{\mathbf{v}}\theta - \nabla_{\mathbf{w}}\phi - c_{\theta}\nabla_{\mathbf{w}}\psi - s_{\theta}c_{\phi}\nabla_{\mathbf{v}}\psi \end{cases} \quad (7.9)$$

This set of 3 equations will be further discussed in section 7.5.

We are interested in solving the following 6 scalar equations that arise from equation 7.8:

$$\begin{cases} \nabla_{\mathbf{u}} H_M &= \nabla_{\mathbf{v}} \phi + c_\theta \nabla_{\mathbf{v}} \psi &= R_M^{\mathbf{u}}(\psi, \theta, \phi) \\ \nabla_{\mathbf{v}} H_L &= -\nabla_{\mathbf{u}} \phi - c_\theta \nabla_{\mathbf{u}} \psi &= R_L^{\mathbf{v}}(\psi, \theta, \phi) \\ \nabla_{\mathbf{u}} H_N &= s_\phi \nabla_{\mathbf{w}} \theta - s_\theta c_\phi \nabla_{\mathbf{w}} \psi &= R_N^{\mathbf{u}}(\psi, \theta, \phi) \\ \nabla_{\mathbf{w}} H_L &= -s_\phi \nabla_{\mathbf{u}} \theta + s_\theta c_\phi \nabla_{\mathbf{u}} \psi &= R_L^{\mathbf{w}}(\psi, \theta, \phi) \\ \nabla_{\mathbf{v}} H_N &= c_\phi \nabla_{\mathbf{w}} \theta + s_\theta s_\phi \nabla_{\mathbf{w}} \psi &= R_N^{\mathbf{v}}(\psi, \theta, \phi) \\ \nabla_{\mathbf{w}} H_M &= -c_\phi \nabla_{\mathbf{v}} \theta - s_\theta s_\phi \nabla_{\mathbf{v}} \psi &= R_M^{\mathbf{w}}(\psi, \theta, \phi) \end{cases} \quad (7.10)$$

To compute an integrable scaled frame field, we have to compute (H_L, H_M, H_N) and (ψ, θ, ϕ) fulfilling eq. 7.9 and eq. 7.10. It turns out to be a very difficult problem to solve therefore, as mentioned earlier, a fixed frame field orientation will be assumed. With (ψ, θ, ϕ) imposed, we have no control on eq. 7.9 being fulfilled. The only control we have is on how close we can be to fulfill eq. 7.10. The formulation of the problem we seek to solve is:

$$\begin{aligned} &\text{Find } (H_L, H_M, H_N) \in (\mathcal{C}^1(\Omega))^3 \text{ minimizing} \\ &E(H_L, H_M, H_N) = \frac{1}{2} \int_{\Omega} \left((\nabla_{\mathbf{u}} H_M - R_M^{\mathbf{u}})^2 + (\nabla_{\mathbf{v}} H_L - R_L^{\mathbf{v}})^2 \right. \\ &\quad \left. + (\nabla_{\mathbf{u}} H_N - R_N^{\mathbf{u}})^2 + (\nabla_{\mathbf{w}} H_L - R_L^{\mathbf{w}})^2 \right. \\ &\quad \left. + (\nabla_{\mathbf{v}} H_N - R_N^{\mathbf{v}})^2 + (\nabla_{\mathbf{w}} H_M - R_M^{\mathbf{w}})^2 \right) d\Omega \end{aligned} \quad (7.11)$$

7.2.2 Ill-posed problem

Minimizing eq. 7.11 is in general an ill-posed problem. This can be stressed out with the very simple example presented on Figure 7.2.

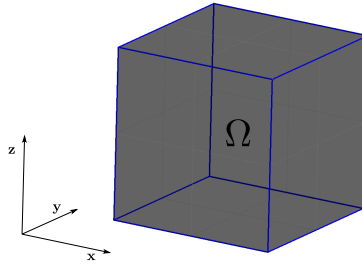


Figure 7.2: Simple domain Ω to demonstrate that the problem is ill-posed.

For this case, the unitary frame field obtained by any method is:

$$\mathcal{F} = \{P \in \Omega \mapsto f(P) = (\mathbf{x}, \mathbf{y}, \mathbf{0})\} \quad (7.12)$$

Euler angles representing this unitary frame field orientation are:

$$\begin{cases} \psi &= 0 \\ \theta &= 0 \\ \phi &= 0 \end{cases} \quad (7.13)$$

Equations 7.9 are fulfilled and equations 7.10 become:

$$\begin{cases} \nabla_{\mathbf{x}} H_M &= \nabla_{\mathbf{z}} H_M &= 0 \\ \nabla_{\mathbf{y}} H_L &= \nabla_{\mathbf{z}} H_L &= 0 \\ \nabla_{\mathbf{x}} H_N &= \nabla_{\mathbf{y}} H_N &= 0 \end{cases} \quad (7.14)$$

for which the solution space $\mathcal{S}$ is :

$$\mathcal{S} = \left\{ (H_L, H_M, H_N) \in (\mathcal{C}^1(\Omega))^3 / \right. \\ \left. (H_L(\mathbf{x}), H_M(\mathbf{x}), H_N(\mathbf{x})) = (F_1(x), F_2(y), F_3(z)), \forall (\mathbf{x}) \in \Omega, \text{ with } (F_1, F_2, F_3) \in (\mathcal{C}^1(\Omega))^3 \right\} \quad (7.15)$$

which gives:

$$\mathcal{S} = ((\mathcal{C}^1(\mathbb{R}))^3 \quad (7.16)$$

Solution space $\mathcal{S}$ is of infinite dimension, thus the problem we are trying to solve is ill-posed. This is due to the fact that conditions on scaling factor derivatives are not sufficient by themselves to lead to a well-posed problem. In order to solve it numerically, we need to add some constraints on the solution we want to obtain. This equation regularization is presented in the next section.

7.2.3 Equations regularization

For the sake of simplicity we outline our method to regularize the problem on a 2D example and then generalize to 3D:

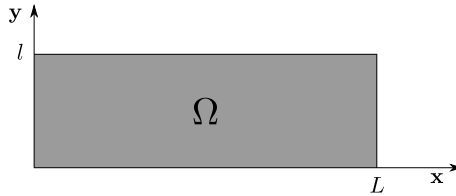


Figure 7.3: Domain Ω considered for the 2D example.

For which we are looking for H such that:

$$\nabla_y H = g, \text{ on } \Omega \quad (7.17)$$

with

$$\begin{cases} g(x, y) = 0 & , \text{ if } 0 \leq x < \frac{L}{3} \\ g(x, y) = (x - \frac{L}{3})^2 * (-\frac{6x}{L+5}) * \frac{9}{L^2} & , \text{ if } \frac{L}{3} \leq x < 2\frac{L}{3} \\ g(x, y) = (\frac{l}{2} - y) * \frac{2}{l} & , \text{ if } 2\frac{L}{3} \leq x < L \end{cases} \quad (7.18)$$

Let us call $\mathcal{S} = \{H \in \mathcal{C}^1(\Omega)/H \text{ verifies eq. 7.17}\}$ the solution space.

This simplified problem is equivalent to the kind of problem (eq. 7.10) we want to solve in $3D$. As we can see, there is no explicit condition on $\nabla_x H$. We can note that, in order for a scalar field H solution of equation 7.17 to exist, we need $\nabla_x H$ to be such that $\nabla \times (\nabla H) = \mathbf{0}$.

It is important to know the physical meaning of H . H is the logarithm of a frame branch size, meaning that it is directly linked to hexahedral element sizes we seek to create. Having that in mind, it seems logical that we would like to have H as smooth as possible in order to generate the best hex-dominant mesh possible. Considering this, the solution we would like to obtain is:

$$\begin{cases} H(x, y) = 0 & , \text{ if } 0 \leq x < \frac{L}{3} \\ H(x, y) = (x - \frac{L}{3})^2 (-\frac{6x}{L+5}) \frac{9}{L^2} (- (y - \frac{l}{2})^2 \frac{1}{l} + \frac{l}{4}) & , \text{ if } \frac{L}{3} \leq x < 2\frac{L}{3} \\ H(x, y) = - (y - \frac{l}{2})^2 \frac{1}{l} + \frac{l}{4} & , \text{ if } 2\frac{L}{3} \leq x < L \end{cases} \quad (7.19)$$

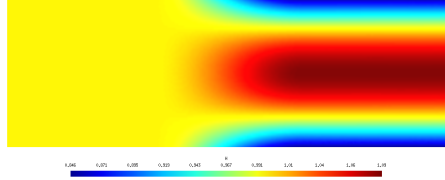


Figure 7.4: Analytical solution plot for $L = 3$ and $l = 1$.

This problem is ill-posed; there is no guarantee that classical numerical methods will converge to a solution. Advantage of this $2D$ example is that with a finite element discretization using order 1 Lagrange elements, preconditioned conjugated gradient and generalized minimal residual method (GMRES) are able to converge to a residual norm of 10^{-8} . All numerical computation presented in the following are done with $L = 3$ and $l = 1$. Figure 7.5 shows the solutions obtained with these 2 methods.

Conjugated gradient is slow to converge and returns a solution satisfying equations 7.17, as well as GMRES. However, solutions obtained are very different



Figure 7.5: Solutions obtained with the two different solvers, for $L = 3$ and $l = 1$. (a) Solution obtained solving equation 7.17 with a preconditioned conjugate gradient. (b) Solution obtained solving equation 7.17 with a generalized minimal residual method.

than the one we would like. GMRES is giving a solution closer to our expectations. This stresses out the need for a problem regularization. First, we cannot know if numerical methods will converge to a solution in the general case, and second, the solutions we obtain depend on the method and are too far from the analytical solution.

In order to regularize the problem, the condition that H should be as smooth as possible is added in the formulation. The problem is then modified as:

$$\text{Find } H \in \mathcal{S} \text{ such as } \|\nabla H\|^2 = \min_{f \in \mathcal{S}} \|\nabla f\|^2 \quad (7.20)$$

which is minimizing the gradient of H on the space of solutions $\mathcal{S}$. This formulation is transforming the initial linear problem into a nonlinear one. By defining :

$$E_{2D}(H) = \int_{\Omega} (\nabla_y H - g)^2 d\Omega \quad (7.21)$$

problem 7.20 can be solved following the method presented in Algorithm 4.

Algorithm 4: Regularized solver

```

 $k = 0;$ 
initial guess  $H^0;$ 
compute  $\epsilon^0 = E_{2D}(H^0);$ 
while  $\epsilon^k < \epsilon^{k-1}$  do
     $k = k + 1;$ 
    find  $H^k$  minimizing  $(E_{2D}(f) + \int_{\Omega} \|\nabla f - \nabla H^{k-1}\|^2 d\Omega), f \in \mathcal{C}^1(\Omega);$ 
    compute  $\epsilon^k = E_{2D}(H^k);$ 
end

```

The solution obtained with the regularized approach (Figure 7.6) is identical to the analytical one (eq. 7.19) we were looking for, and therefore better than the one obtained without regularization (Figure 7.5).

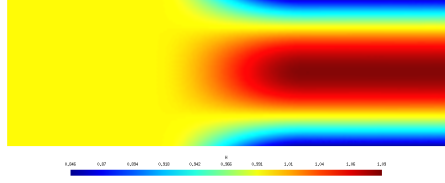


Figure 7.6: Regularized solution.

This method can be extended to solve the general 3D problem (Equations eq. 7.11). The regularized 3D solver becomes:

Algorithm 5: Regularized solver for 3D problem

```

 $k = 0;$ 
initial guess  $H_L^0, H_M^0, H_N^0;$ 
compute  $\epsilon^0 = E(H^0);$ 
while  $\epsilon^k < \epsilon^{k-1}$  do
     $k = k + 1;$ 
    find  $H^k$  minimizing
         $E(f_1, f_2, f_3) + \int_{\Omega} \|\nabla f_1 - \nabla H_L^{k-1}\|^2 + \|\nabla f_2 - \nabla H_M^{k-1}\|^2 +$ 
         $\|\nabla f_3 - \nabla H_N^{k-1}\|^2 d\Omega, (f_1, f_2, f_3) \in (\mathcal{C}^1(\Omega))^3;$ 
    compute  $\epsilon^k = E(H^k);$ 
end

```

7.3 Point generation for hex-dominant mesh generation

To examine the practical benefits of the scaled frame-fields, we use them to generate hexahedral-dominant meshes on volumetric domains. To this end, we develop a pipeline similar to the one proposed in Baudouin et al. [2014] for *indirect* hexahedral-dominant meshing. We follow this approach since it is straightforward and efficient. Besides being suited for the problem in hand, i.e., comparing results with and without scaling of 3D frames, it is also a method that is able to produce results on any kind of model and/or input frame-field regardless the complexity.

The steps of the pipeline can be summarized as follows:

1. Generation of an initial tetrahedral mesh $\mathcal{T}_0$ of the model using the Constrained Delaunay Triangulation algorithm of [Marot et al. \[2019\]](#).
2. Computing a frame-field f and its scaling h on $\mathcal{T}_0$.
3. Generation of a triangular mesh $\mathcal{T}_S$ of the volume-bounding surface.
4. Generation of a point set P_V inside the volume. Points are generated in a frontal fashion, guided by the scaled frame-field.
5. Generation of a frame-aligned tetrahedral mesh $\mathcal{T}_V$ of the point set P_V using [Marot et al. \[2019\]](#).
6. Combination of tetrahedra into hexahedra using [Pellerin et al. \[2018a\]](#).

Steps 1 and 2 Given the input tetrahedral mesh $\mathcal{T}_0$, we compute on it the frame field $f = (u, v, w)$ and its scaling $h = (L, M, N)$. The size-field h can be further scaled if necessary by a factor c relative to the size of the input model. This further scaling does not affect the integrability property.

Step 3 We create a triangular mesh that bounds the volume following the procedure described in Chapter 5. We extract a 2D cross-field f_S from f by eliminating the normal to the surface component at each vertex. Then we follow a bottom-up procedure for the creation of points. Input model lines are uniquely defined from a list of non-intersecting connected boundary edges. Each model line is then discretized with respect to sizes h prescribed at boundary vertices (using the scaling of the frame branch parallel to the boundary line). These points are then used as seed points for the frontal generation of a point-set P_S that is embedded on the volume-bounding surface $\mathcal{S}$. The frontal point propagation is guided by a metric-field consisting of f_S and the corresponding size-field h_S . The usage of the point-set P_S is two-fold. First, the points are used as the seed points for the generation of a point-set P_V inside the volume domain. Secondly, its triangulation $\mathcal{T}_S$ will be a surface mesh (consisting mostly of right-angled triangles) that will be used as the bounding surface for the tetrahedralization. In this work, we do not convert the surface mesh to an all-quad mesh. We are mainly interested on how the scaled frame-field aligns elements inside the volumetric domain. Altering the surface mesh would create incongruities between surface and interior points, for example singular lines inside the domain may not emanate from the equivalent irregular vertices on the surface.

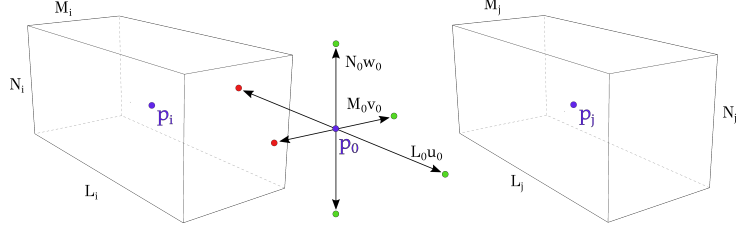


Figure 7.7: The points in blue (p_0 , p_i and p_j) are already generated. The exclusion boxes around the nearby points p_i and p_j are defined taking into account the directions and sizes of the corresponding scaled frames. A point p_0 spawns 6 new points on the corresponding directions. The red ones are discarded since they are included in the exclusion box of p_i and the green ones are accepted and inserted in the queue.

Step 4 Spawning points inside the volume follows. A priority queue is filled with the set of seed points $P_S = \{\mathbf{p}_i \mid i = 1, \dots, N_s\}$ generated on lines and surfaces. Now, each point spawns $N_d = 6$ candidate points, by simply computing $\mathbf{p}_{ij} = \mathbf{p}_i + h(\mathbf{p}_i) \cdot \mathbf{f}(\mathbf{p}_i)$ for $j = 1, \dots, N_d$ frame-field components (Figure 7.7). Similarly to Chapter 5, starting from the parent tetrahedron of the seed point we successively walk into the tetrahedral mesh to find the tetrahedron where the candidate point lies. Points are filtered out if necessary and the process continues until the volumetric domain is filled. For the filtering, a scaled frame-aligned box is used as an exclusion zone around each point, since we want to take into account the anisotropy induced by the scaling of directions (Figure 7.7).

Steps 5 and 6 The volume point-set $P_V = \{\mathbf{p}_i \mid i = 1, \dots, N_v\}$, along with the surface mesh $\mathcal{T}_S$ is the input of the Constrained Delaunay Triangulation algorithm of Marot et al. [2019]. The tetrahedral mesh $\mathcal{T}_V$ produced is not optimized since we want to keep the original frame-aligned position of the vertices. Finally, the algorithm of Pellerin et al. [2018a] is used to find possible combinations of tetrahedra into hexes and choose the final hexahedral elements. We must note that the final hex-dominant mesh produced is not conformal on the hex-tet interfaces. Ensuring conformity by placing transition pyramid elements on the interface will be the next step of our developments. At this stage, the meshes produced are sufficient for our purposes, i.e., examine the applicability of integrable frame-fields. Both algorithms (for tetrahedralization and combination) are highly efficient and are considered state of the art for their respective objective.

		#vertices	#elements	$\%H_{nbr}$	$\%H_{vol}$
bone	unitary	31020	44299	0.522	0.849
	scaled	31665	39617	0.643	0.913
joint	unitary	21791	22099	0.757	0.934
	scaled	22600	21793	0.824	0.954
hanger	unitary	44211	44056	0.765	0.939
	scaled	44127	41346	0.834	0.957
wrench	unitary	19086	17684	0.761	0.953
	scaled	19028	15088	0.936	0.989

Table 7.1: Statistics of hexahedral-dominant meshes on the four validation models with unitary and scaled frame-field input.

7.4 Results

We present here results of hexahedral-dominant meshes aligned with 3D scaled frame-fields. As a first step, we use as input quadrilateral meshes that are extracted from all-hex meshes. The use for these quad meshes is two-fold: i) they provides the seed points on the surface for frontal point generation inside the volume, as well as a triangular mesh (by splitting the quads to triangles) for bounding the 3D Constrained Delaunay Triangulation algorithm and ii) we use them as a boundary condition for the computation of frame-fields and their scaling. We thus start from an 'ideal' configuration, since this quad mesh is already the boundary of an hexahedral mesh and is equipped with the appropriate topological characteristics. Moreover, the input models are relatively simple. The point is to validate the solution and examine the behaviour of the scaling on the interior of the domain. Of course, it is not feasible to reproduce the hexahedral meshes with this kind of advancing-front strategy.

The models that we use are: the hexahedral meshes of the *bone* and *joint* models obtained from Li et al. [2012], and the hexahedral meshes of the *hanger* and *wrench* models obtained from Livesu et al. [2020]. Statistics of the output meshes with unitary and scaled frame-fields are presented on Table 7.1. Hex proportion is significantly improved for scaled frame-fields compared to using unitary ones, both in terms of absolute numbers and more importantly in terms of volumetric percentage. Improvement in hex proportion and element quality is also apparent on the structure of the output meshes, with smoother transitions and ability to better capture singular regions of the frame-field, as shown on Figure 7.8.

We further test our pipeline on a set of CAD models from MAMBO database [Ledoux], in this case without constraining the boundary with a quad mesh.

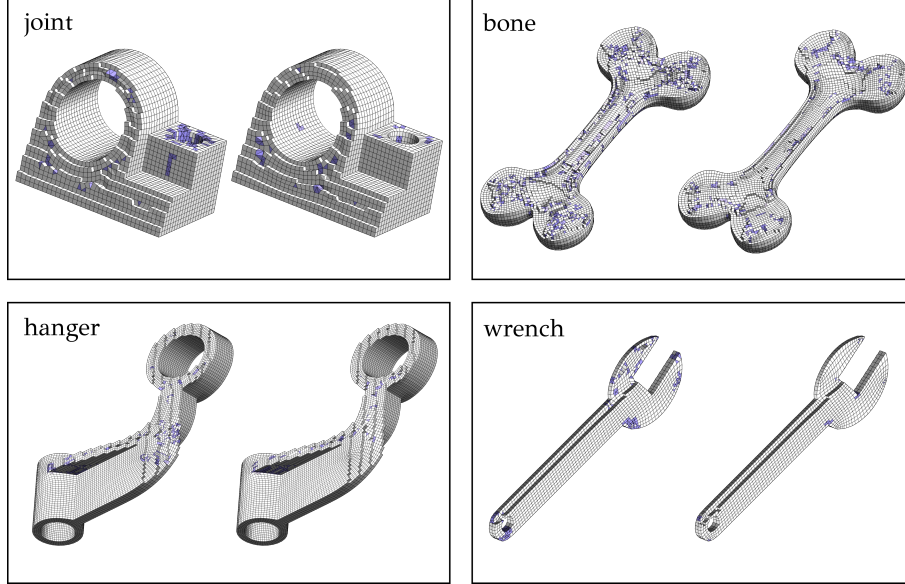


Figure 7.8: Results of hexahedral-dominant meshes on the four validation models (left of each subfigure: with unitary frame-field and right: with scaled frame-field).

The nine models are of medium complexity and presented on Figures 7.9 and 7.10. In Table 7.2 we observe small increase in hex proportions and quality for most of the models, except M8 and M9 where we have diminished numbers. These two models have thin volume regions (on M8 the propeller blades and on M9 the cylindrical volume) and we can identify two issues of our methodology related to this. First, there is not enough space for the scaling to fully develop, probably because of insufficient discretization, and second, the frontal propagation of points creates colliding fronts located close to both boundaries. Nevertheless, the structure of boundary elements on these models is improved.

Our main interest here is the structure (in the sense of frame-field alignment) of the output meshes, where we observe improved accordance with the scaled frame-field structure compared to the unitary one. The majority of remaining tetrahedral regions are due to the frontal approach we follow and the difficulties posed from certain geometric features. Besides the front-collisions, where the method is unable to conformally connect opposite fronts, there are feature-specific issues that are commonly encountered in hex-meshing, for example acute/pyramidal corners and fillet transitions. Furthermore, the anisotropic nature of the input metric may be more challenging to handle by our pipeline, since tetrahedralization and combination steps do not take anisotropy into

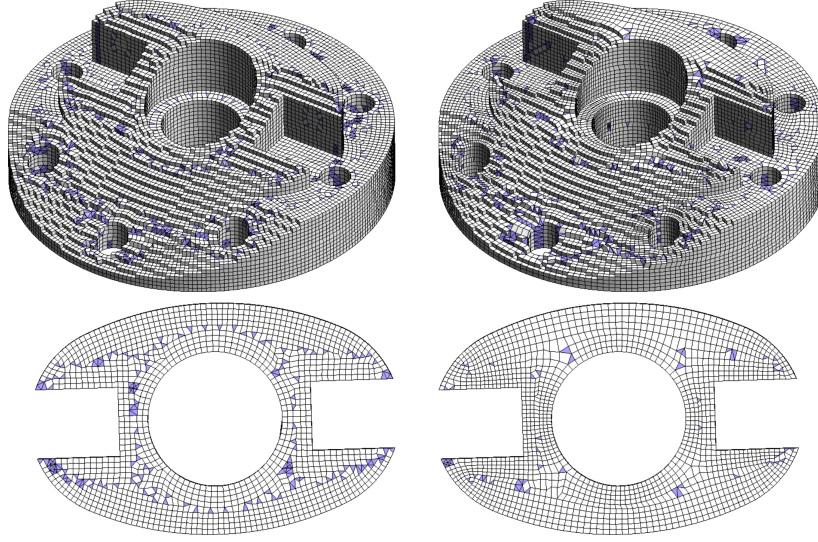


Figure 7.9: Detail of interest on model M1.

		#vertices	#elements	$\%H_{nbr}$	$\%H_{vol}$	Q_{avg}
M1	unitary	99878	128431	0.637	0.916	0.898
	scaled	104566	124741	0.701	0.930	0.912
M2	unitary	79724	110496	0.569	0.890	0.886
	scaled	87315	114827	0.620	0.895	0.889
M3	unitary	158805	187330	0.693	0.932	0.915
	scaled	196762	221795	0.723	0.936	0.913
M4	unitary	150429	190880	0.643	0.919	0.905
	scaled	168530	195044	0.736	0.927	0.923
M5	unitary	298286	380138	0.647	0.920	0.906
	scaled	327983	380563	0.733	0.929	0.920
M6	unitary	188388	214243	0.752	0.948	0.933
	scaled	201765	224776	0.775	0.949	0.934
M7	unitary	331614	449622	0.602	0.903	0.894
	scaled	362488	459558	0.662	0.917	0.903
M8	unitary	328271	395925	0.623	0.911	0.895
	scaled	380194	538848	0.494	0.880	0.826
M9	unitary	136252	169960	0.578	0.895	0.895
	scaled	175486	274080	0.448	0.868	0.804

Table 7.2: Statistics of hexahedral-dominant meshes on the models from [Ledoux](#) with unitary and scaled frame-field input.

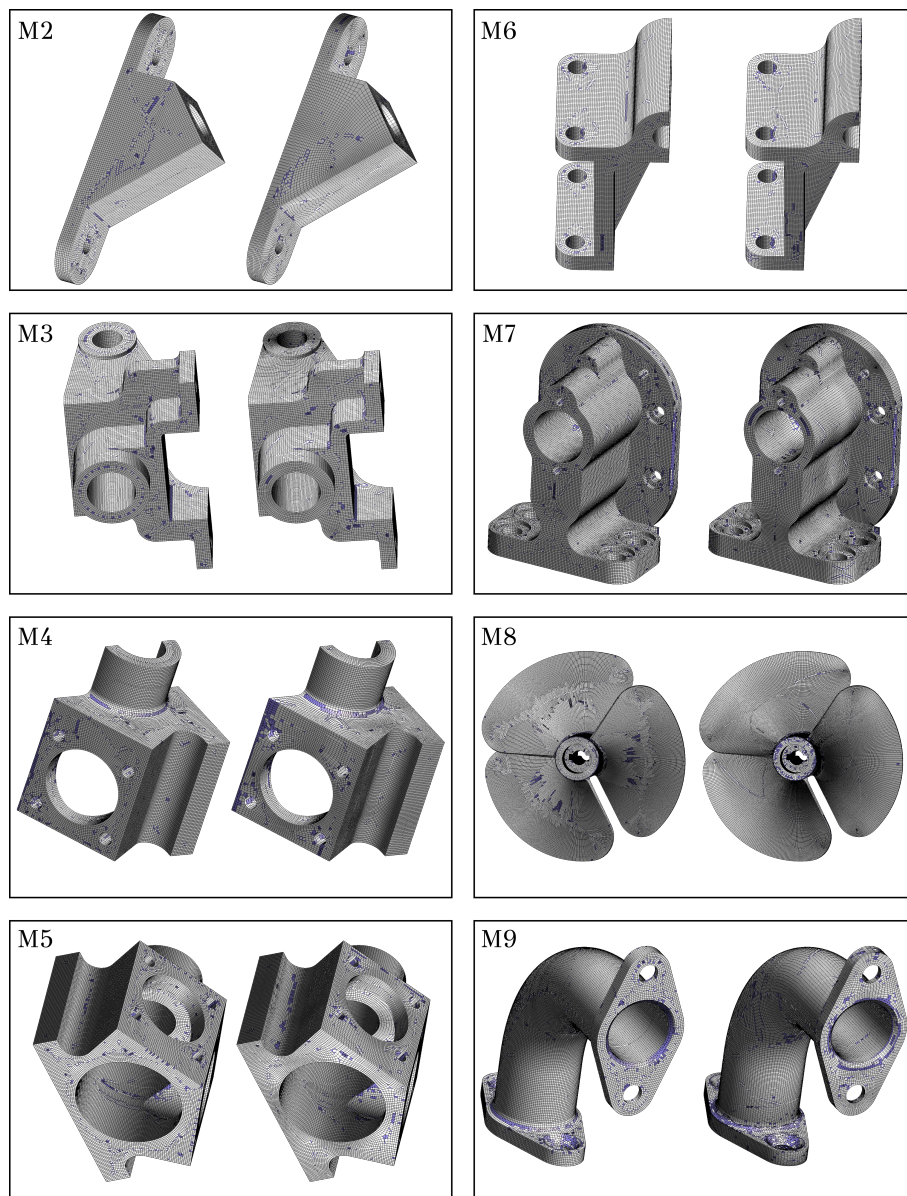


Figure 7.10: Results of hexahedral-dominant meshes on nine medium-complexity models from [Ledoux](#) (left of each subfigure: with unitary frame-field and right: with scaled frame-field).

account.

It is worth noting that we also investigated the possibility to use quad meshes as boundary constraints, as we did for the validation models. In that case though the quad meshes were not extracted from an already formed hexahedral mesh, instead we generated them with the method of Chapter 6. For a lot of the cases tested we obtained lower hex proportion and diminished quality, especially on models with higher complexity. Even if we create a quad mesh respecting the input frame-field, we do not have any guarantee that the topological structure of it will admit a hex-meshable interior. Besides this, in practice these quad meshes may have an excess of irregular vertices compared to the ones the frame-field induces (for example to accommodate size transitions). Therefore we would impose redundant singular lines emanating from these irregular vertices towards the interior. Moreover, if a 'perfect' block structure is not available, the quad mesh may have helical loops encircling the domain that are not hex-meshable. This issue may also appear on the interior, around the remaining cavities of tetrahedra.

7.5 Conclusion and perspectives

In this work we developed the mathematical conditions a frame field has to fulfill in order to be integrable. The underlying formulation leads to a challenging problem coupling frame field orientation and branches' sizes that is difficult to solve. Given that we want to exploit the results on a practical basis, we reduce the problem to the scaling of a given frame field and use these scaled frame fields to generate hexahedral-dominant meshes. The results obtained validate the benefits of integrability conditions, in terms of mesh statistics and topological structure.

While our approach for hex-dominant meshing is pragmatic and offers certain advantages, with the most important being its ability to work on any kind of input, we would like to investigate the use of integrable frame fields in conjunction with a parameterization approach. In our context, the input frame fields provides the guiding directions, while the scaled branches a size field. We keep in mind that in practice it would be preferential to prescribe a user-defined size field and compute a direction field that satisfies it, and we leave it as a challenging open problem for future work.

It is important to note that the 2 set of equations 7.9 and 7.10 have to be verified in order for the 3D frame field to be integrable. Two classical problems encountered in frame field generation are obtaining frame fields with 3-5 singular curves and the presence of limit cycles. Existence of 3-5 singular curves is coming from the fact that equations 7.9 (depending only on frame field orientation) are not fulfilled, and limit cycles from the fact that there exist no scaling

of the branches which can fulfill equations 7.10. Classical ways to compute 3D frame fields are focusing on solving a problem different than this one, and therefore give no guarantee on producing an integrable 3D frame field. This was already known empirically. If the goal is to compute an integrable 3D frame field, we should find a way to compute it verifying the aforementioned equations.

CHAPTER



Conclusion

What makes quadrilateral/hexahedral meshing so complicated in comparison with triangular/tetrahedral one? There is not a definite answer to this general question since one should take into account the design goals and problem-specific characteristics to answer. For example, it is trivial to generate a hexahedral mesh from a tetrahedral one by splitting each tet to four hexes, but the low element quality and high vertex irregularity could render it useless in practice.

In this work, we approached the problem from an engineering point of view, with our main aim being that of providing solutions that are practical for 'real-life' applications, while at the same time not deviating from the underlying theoretical considerations. Four years of working on the topic of mesh generation cultivated a better understanding of the theoretical and practical difficulties involved, which we can outline along with the contributions of the

present work.

Triangular meshing methods build upon the concrete mathematical foundation of Delaunay property and algorithms of complexity $\mathcal{O}(n \log n)$ exist since the 1970s. Constrained Delaunay Triangulation is easy to code, and triangular meshing in 2D is considered a solved problem. Delaunay Triangulations have optimal properties, and there is no need for extended optimization procedures. However, difficulty increases for general 3D surfaces, where the Delaunay property cannot be utilized in the same straightforward way. We have provided in this work a solution for *robust surface meshing*, where we develop a two-step method for triangular meshing on general surfaces, with the geometric validity and fidelity being of outmost importance. A lot of the meshing difficulties arise from the communication issues between CAD representations and meshing algorithms. We tried to address this by having as input solely a triangulated model as the underlying geometric representation.

Tetrahedral meshing is more challenging, even though based on an extension of the same theoretical basis. 3D Constrained Delaunay Triangulation is a hard problem (partly solved in the 1990s) so mesh generation in 3D is a way more recent achievement. While in 2D Delaunay triangulations are flip-connected and unique, no such property holds for 3D. Delaunay tetrahedralizations may contain slivers and thus optimization is required, which is a challenging problem by itself.

Quadrilateral meshing precedes in a sense triangular one, since structured grids with regular quads (on simple domains) were the first used to support Finite Difference Methods. But unstructured (with the definition here covering also block-structured) quad meshing for general geometries is a totally different problem. There is not a nice Delaunay-like property for quad meshes that could permit algorithms like Bowyer-Watson triangulation. Quad elements are less flexible, and the structure of quad meshes is globally constrained. As we have seen in Chapter 2, commonly used methods involve practical heuristics and resulting quad meshes have non-optimal properties. Global parameterization methods offer a sound solution, but they require very complex procedures and, in the author's opinion, it will be difficult for this approach to evolve to a robust engineering tool.

In this work, we have tried to take the best of both worlds. We use cross-fields and take advantage of the global information they provide on a local basis, without building 'heavy' global parameterization strategies. We developed a novel approach for *straightforward quadrilateral meshing* that uses a bipartite labeling scheme which enables us to convert the global problem of finding a perfect matching to a local one and extract a quad mesh whose each edge is defined as the unity of opposite labeled vertices. Efficiency is maximized and the algorithm can always provide a quadrilateral mesh (and in conjunction with

cross-field guidance, a high-quality one). Furthermore, we presented a strategy that produces *quasi-structured quad meshes* by recovering the underlying topological structure of the cross-field. This method improves the topological and geometric properties of quadrilateral meshes and is able to produce meshes of high-quality on complex models.

Hexahedral meshing has been justly dubbed as the 'holy grail' of mesh generation. Its complexity is much higher than quad meshing and it is extremely difficult to generate hex meshes even on very simple models. Most classic methods fail for increased model complexity (with the exception of oct-tree approaches [Schneiders, 2000b; Maréchal, 2009]), and global parameterization lose their properties due to configurations where the frame fields are not hex-meshable. In this work we investigated the extension of scaling for 2D cross-fields to *3D scaled frame fields* and implemented a *hex-dominant meshing* pipeline. We presented hex-dominant meshes based on scaled frame fields where the hexahedral proportion and the structure of the meshes are improved. We are well aware that the advancing-front strategy and indirect combination of tets-to-hexes that we use is not the optimal one for this problem, but obtaining hexahedral meshes for general models with parameterization approaches does not seem feasible in the foreseeable future. We thus favored a pragmatic approach that is robust and efficient, and aimed to maximize its attributes.

8.1 Perspectives

Besides providing solutions and answers to open problems, a research product should raise new questions and pave the way towards paths not yet explored. There are things in our methods that could be improved, as well as ideas not implemented. We provide here an indicative list of possible directions for future work.

Connection with applications. Mesh generation can hardly be considered as a stand-alone domain. We would be very interested in investigating the performance and advantages of the output meshes in numerical simulations.

Surface remeshing as a coarsening tool. The surface meshing algorithm presented proves to be a useful remeshing tool for triangulated surfaces, and it is highly robust in conjunction with an adequate size-field. When the size-field prescribed is much coarser than the geometric characteristics of the model (expressed in the sizing of the input mesh) we essentially have a coarsening (or mesh simplification) tool. Mesh simplification techniques may use quadratic error metrics [Garland and Heckbert, 1997], whose computation can be costly.

Instead, placing a priori a minimal number of points that respect the geometry and performing local mesh modifications could prove more efficient.

Even though our algorithm can be sufficient for such use for most of the cases, we may encounter 'dangerous' configurations. For example, consider two curved surfaces very close to each other, of which the one has a coarser prescribed sizing. It is not within the scope of this work to provide a coarsening tool, but since we consider that the extension is feasible, we can outline some possible solutions for the aforementioned issue:

- Identify a priori thin regions (surfaces 'close' to each other) and flag their corresponding triangles. Our operations are based on local mesh connectivity, but for flagged triangles we could query a spatial search structure (such as R-Trees) for possible intersections in a more extended space.
- Use simplification envelopes, as proposed in [Cohen et al. \[1996\]](#), to limit surface deviation.
- In the case of volume bounding surfaces we could generate a tetrahedral mesh and perform operations on its triangular surface, which always ensure valid tetrahedra.

Cross-fields, sizing and anisotropy. As stated, scaled cross-fields are beneficial for quad/hex generation. We have mainly used size-fields obtained from the scaling of cross-fields/frame-fields. In practice though, a user would commonly prefer prescribing a size-field that is a better fit for the problem in hand. One of the advantages of our approach is that our algorithms can produce meshes regardless the topological integrity of the input metric, for example with a user-defined size-field that is incompatible with the computed cross-field (manifested in the existence of 3-5 dipoles accommodating size transitions). Nevertheless, adapting a direction field to an input size-field would vastly improve the results.

Extending the discussion to general anisotropic metrics, the generation of boundary layer meshes can be investigated. There are two main difficulties related to this. First, successful derivation of an anisotropic, possibly solution-driven, metric for the generation of boundary layers for Computational Fluid Dynamics [[Alauzet, 2010](#); [Loseille and Löhner, 2011](#); [Alauzet and Marcum, 2015](#)]. Second, while it is straightforward to extend the point generation for an anisotropic metric, we consider the triangulation on this metric a much more difficult task. We believe that the best strategy would be to create structured layers of boundary quads/hexes and then subdivide them. While for quadrilaterals we can consider this feasible with the optimization strategy presented

in Chapter 6, we are well aware that for hexahedral elements it is a highly constrained problem with no trivial solution. User input may be necessary to specify desired characteristics, e.g., vertex valence on corners. Finally, it would be possible to exploit the locality of operators in our algorithms to develop an adaptive remeshing technique for transient simulations.

Block-structured quad meshing. We would like to convert the quasi-structured meshes presented in Chapter 6 to semi-regular (block-structured) ones. Topological conversion of this kind may require chord-collapsing. It is not apparent what would be the optimal strategy for chord-collapsing on complex models, since they often have chords with helical structure around the domain, affecting multiple surfaces. A more suited approach could be a global quantization [Campen et al., 2015] strategy acting upon a T-mesh extracted from the quasi-structured one.

Hexahedral-dominant meshing. The indirect approach followed can provide hex-dominant meshes for general models, but hexahedral proportion naturally diminishes for more complex configurations. A few ideas that could enhance the attributes of this approach are the following:

1. While our scheme for point insertion is efficient, robust, and open to parallelization, it still suffers from the drawbacks of advancing-front techniques. It would be possible to avoid the majority of front-collisions by implementing a better strategy, for example a parameterization method for point sampling similar to Sokolov et al. [2016].
2. The last step needed for our hexahedral-dominant meshes to be fully practical is to ensure conformity on the interface between hexahedra and tetrahedra. Unfortunately, it is not trivial to achieve this in a robust and efficient way. The practical solution of inserting small, almost flat pyramids on the quad interfaces should be optimized in a way that ensures untangled elements, conformity and acceptable element quality.
3. We investigated constraining the generation of a hexahedral-dominant mesh with a quadrilateral bounding surface. A very interesting open problem is investigating what kind of properties should this quadrilateral mesh have in order to be compatible with a hex-mesh topology. The information contained on the bounding surface should be sufficient to describe the interior characteristics.

Bipartite hexahedral labeling. We investigated the possibility of using a similar labeling scheme as for quad meshing, since hexahedra are bipartite (two-colored) as well (Figure 8.1, left). The idea is similar; identify and suppress odd-bounded regions during hexahedral combination. The problem arises when we try to recover the bichromatic edges from the tetrahedral mesh, since hex

diagonals also have this property (Figure 8.1, middle). Furthermore, it is not obvious where we could insert Steiner points to avoid odd-bounded regions since we cannot define something similar to a perfect matching like for quads (for example, we could consider in an analogous manner with 2D that tetrahedra with four same-labeled vertices are inconsistent, but it is not clear how to go forward with the rest of the cases). We investigated two ideas that could potentially offer a way to simplify indirect combination.

1. Hypercube labeling.

A possible solution is to develop a more sophisticated labeling scheme that could codify more information. We investigated the possibility of using the labeling of a hypercube or n-cube (with $n=3$ in our case, Figure 8.1, right), commonly used as a communication pattern in network topology [Stricker, 2011]. In that case, the edges of the cube have a difference of one bit, the diagonals of the faces two bits, and the long diagonals three bits. Even though it seems like a promising choice, we should handle various issues before being able to use it in a meaningful way:

- i. It is possible to utilize it as it is only for regular graphs but in our case we have to deal with irregular vertices that lead to non-regular graphs. The first thing that comes to mind is to use it to construct the dual mesh, which in fact is a regular graph. But accordingly, the dual mesh has odd-bounded polyhedra associated with the irregular vertices, where we cannot define a cyclic labeling of this kind.
- ii. We should associate the labeling with geometry information. For example, denoting specific frame field direction with respective changes in the binary digits.
- iii. It is not obvious how to set the boundary conditions, i.e., label the vertices of the bounding triangular or quadrilateral mesh with 2-bit or 3-bit indices.

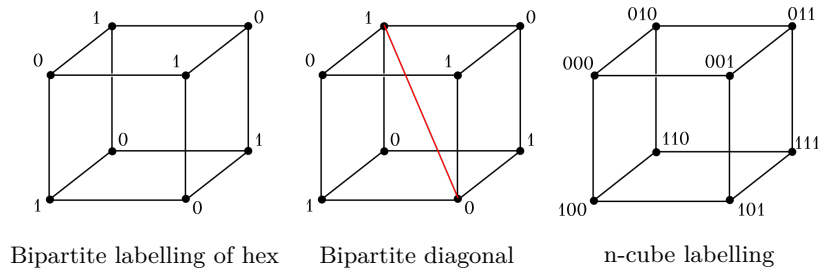


Figure 8.1: Bipartite (left) and hypercube (right) labelling of a cube.

2. Bipartite constraint for tetrahedral combination.

Another possibility to utilize bipartite labeling is to constrain tet-to-hex combination. As stated in Chapter 5, if we extract a quad-dominant mesh (without adding the Steiner points) that respects the bipartite labeling, it will end up having triangular regions that are even-bounded polygons (thus satisfying the topological condition for quad meshing). The main topological conditions for hexahedral meshing is an even number of boundary quadrilaterals and no odd-cycle of quads on its boundary. Accordingly, we could avoid the formation of odd-cycles since bipartite labeling will forbid them.

We have used the work of [Pellerin et al. \[2018a\]](#) for tet-to-hex combination, with which we can identify the set of all potential hexahedra in a tetrahedral mesh. By using the bipartite labeling on the vertices we can constrain the potential hexahedra to the ones that have bichromatic edges. The first immediate advantage of this is that it makes the algorithm much more efficient (since the search space for possible hexahedra is vastly smaller), at the cost of lower hexahedral percentage on the outcome mesh. For example, on a large input mesh with ~ 1.8 million tetrahedra (for the *hanger* model presented in Section 7.4) this algorithm computes the set of potential hexahedra in 23 seconds, while with the bipartite constraint it takes 2 seconds. In the first case the number percentage of hexes in the final mesh is 0.82, while in the second it falls to 0.78. On this model it is not a big difference but we expect that for more complicated cases the hex percentage will be further diminished.

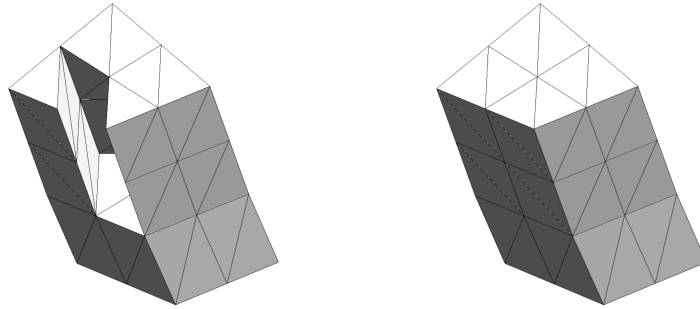


Figure 8.2: Remaining region of tetrahedra after unconstrained (left) and constrained with bipartite labels (right) combination using the algorithm of [Pellerin et al. \[2018a\]](#).

Reducing computation time is not the main aspect of this idea. What we want to achieve is that the non-combined tetrahedral regions that remain on the final hex-dominant mesh will have better topological properties due to the bipartite constraint. As a subsequent step, we could develop a post-processing strategy to manipulate the remaining 'cavities' of tetrahedra, for example, extract re-

gions that are trivially hex-meshable. In Figure 8.2 we can see a simple example of such region, where we visualize the boundary of a remaining tetrahedral region which is surrounded by already formed hexahedra. On the left, two more hexahedra are identified on its interior by the unconstrained algorithm (where the 'gap' is), resulting to a prismatic sub-region. On the right, the bipartite constraint on the vertices forbids their formation and leads to a configuration that is readily hex-meshable.

Bibliography

- M. N. Akram, L. Si., and G. Chen. An embedded polygon strategy for quality improvement of 2D quadrilateral meshes with boundaries. In *Proceedings of the 16th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - GRAPP*,, pages 177–184. INSTICC, SciTePress, 2021. ISBN 978-989-758-488-6. doi: 10.5220/0010209101770184.
- F. Alauzet. Size gradation control of anisotropic meshes. *Finite Elements in Analysis and Design*, 46(1-2):181–202, Jan. 2010. ISSN 0123456789. doi: 10.1016/j.finel.2009.06.028.
- F. Alauzet and D. Marcum. A closed advancing-layer method with connectivity optimization-based mesh movement for viscous mesh generation. *Engineering with Computers*, 31(3):545–560, 2015. ISSN 9783319023359. doi: 10.1007/s00366-014-0385-7.
- P. L. Baehmann, S. L. Wittchen, M. Shephard, K. R. Grice, and M. Yerry. Robust, geometrically based, automatic two-dimensional mesh generation. *International Journal for Numerical Methods in Engineering*, 24:1043–1078, 1987.
- B. S. Baker, E. Grosse, and C. S. Rafferty. Nonobtuse triangulation of polygons. *Discrete & Computational Geometry*, 3(1):147–168, 1988.
- T. J. Baker. Automatic mesh generation for complex three-dimensional regions using a constrained Delaunay triangulation. *Engineering with Computers*, 5(3-4):161–175, June 1989. ISSN 0177-0667, 1435-5663. doi: 10.1007/BF02274210.
- T. C. Baudouin, J.-F. Remacle, É. Marchandise, F. Henrotte, and C. Geuzaine. A frontal approach to hex-dominant mesh generation. *Advanced Modeling and Simulation in Engineering Sciences*, 1(1):8–8, 2014. doi: 10.1186/2213-7467-1-8.

- A. Bawin, F. Henrotte, and J.-F. Remacle. Automatic feature-preserving size field for 3D mesh generation. *arXiv e-prints*, pages arXiv–2009, 2020.
- P. A. Beaufort, J. Lambrechts, F. Henrotte, C. Geuzaine, and J.-F. Remacle. Computing two dimensional cross fields – A PDE approach based on the Ginzburg-Landau theory. In *26th International Meshing Roundtable*, 2017a.
- P.-A. Beaufort, J. Lambrechts, F. Henrotte, C. Geuzaine, and J.-F. Remacle. Computing cross fields A PDE approach based on the Ginzburg-Landau theory. *Procedia Engineering*, 203:219–231, Jan. 2017b. ISSN 1877-7058. doi: 10.1016/j.proeng.2017.09.799.
- P.-A. Beaufort, C. Geuzaine, and J.-F. Remacle. Automatic surface mesh generation for discrete models – A complete and automatic pipeline based on reparametrization. *Journal of Computational Physics*, 417:109575, 2020. ISSN 0021-9991. doi: 10.1016/j.jcp.2020.109575.
- E. Béchet, J. C. Cuilliere, and F. Trochu. Generation of a finite element MESH from stereolithography (STL) files. *Computer-Aided Design*, 34(1):1–17, Jan. 2002. ISSN 0010-4485. doi: 10.1016/S0010-4485(00)00146-9.
- N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. *ACM SIGMOD Record*, 19(2):322–331, May 1990. doi: 10.1145/93605.98741.
- M. Ben-Chen, C. Gotsman, and G. Bunin. Conformal flattening by curvature prescription and metric scaling. *Computer Graphics Forum*, 27(2):449–458, 2008. doi: 10.1111/j.1467-8659.2008.01142.x.
- S. E. Benzley, E. Perry, K. Merkley, B. Clark, and G. Sjaardema. A comparison of all hexagonal and all tetrahedral finite element meshes for elastic and elasto-plastic analysis. In *In Proceedings, 4th International Meshing Roundtable*, 1995.
- M. Bern, D. Eppstein, and J. Gilbert. Provably good mesh generation. *Journal of Computer and System Sciences*, 48(3):384–409, June 1994. doi: 10.1016/S0022-0000(05)80059-5.
- P.-E. Bernard, J.-F. Remacle, N. Kowalski, and C. Geuzaine. Frame field smoothness-based approach for hex-dominant meshing. *Computer-Aided Design*, 72:78–86, Mar. 2016. ISSN 00104485. doi: 10.1016/j.cad.2015.10.003.
- T. D. Blacker and M. B. Stephenson. Paving: A new approach to automated quadrilateral mesh generation. *Int. J. Numer. Meth. Engng.*, 32(4):811–847, Sept. 1991. ISSN 0029-5981, 1097-0207. doi: 10.1002/nme.1620320410.

- P. T. . A. Boggs, A. E. E. . Larzelere, A. R. E. E. . Walsh, E. J. . Clay, R. L., M. F. S. N. L. Hardwick, and C. Livermore. DART system analysis. *August 1*, 31, 2021.
- D. Bommes, H. Zimmer, and L. Kobbelt. Mixed-integer quadrangulation. *ACM Trans. Graph.*, 28(3):1–10, July 2009. ISSN 0730-0301, 1557-7368. doi: 10.1145/1531326.1531383.
- D. Bommes, T. Lempfer, and L. Kobbelt. Global Structure Optimization of Quadrilateral Meshes. *Computer Graphics Forum*, 30(2):375–384, Apr. 2011. ISSN 01677055. doi: 10.1111/j.1467-8659.2011.01868.x.
- D. Bommes, M. Campen, H.-C. Ebke, P. Alliez, and L. Kobbelt. Integer-grid maps for reliable quad meshing. *ACM Trans. Graph.*, 32(4), July 2013a. ISSN 0730-0301. doi: 10.1145/2461912.2462014.
- D. Bommes, B. Lévy, N. Pietroni, E. Puppo, C. Silva, M. Tarini, and D. Zorin. Quad-Mesh Generation and Processing: A Survey: Quad-Mesh Generation and Processing. *Computer Graphics Forum*, 32(6):51–76, Sept. 2013b. ISSN 01677055. doi: 10.1111/cgf.12014.
- J. A. Bondy and Murty, U. S. R. *Graph Theory with Applications*. Macmillan London, 1976. ISBN 0-333-17791-6.
- H. Borouchaki and P. J. Frey. Adaptive triangular-quadrilateral mesh generation. *International Journal for Numerical Methods in Engineering*, 41(5):915–934, Mar. 1998. doi: 10.1002/(SICI)1097-0207(19980315)41:5<915::AID-NME318>3.0.CO;2-Y.
- H. Borouchaki, P. L. George, F. Hecht, P. Laug, and E. Saltel. Delaunay mesh generation governed by metric specifications. Part I. Algorithms. *Finite Elements in Analysis and Design*, 25(1-2):61–83, Mar. 1997. ISSN 0168874X. doi: 10.1016/S0168-874X(96)00057-1.
- H. Borouchaki, P. Laug, and P.-L. George. Parametric surface meshing using a combined advancing-front generalized Delaunay approach. *International Journal for Numerical Methods in Engineering*, 49(1-2):233–259, Sept. 2000. ISSN 0029-5981. doi: 10.1002/1097-0207(20000910/20)49:1/2<233::AID-NME931>3.0.CO;2-G.
- A. Botella, B. Lévy, and G. Caumon. Indirect unstructured hex-dominant mesh generation using tetrahedra recombination. *Computational Geosciences*, 20(3):437–451, June 2016. ISSN 1573-1499. doi: 10.1007/s10596-015-9484-9.
- M. L. Brewer, L. F. Diachin, P. M. Knupp, T. Leurent, and D. J. Melander. The mesquite mesh quality improvement toolkit. In *Proceedings of the International Meshing Roundtable*. Springer, 2003.

- G. Bunin. Non-Local Topological Clean-Up. In P. P. Pébay, editor, *Proceedings of the 15th International Meshing Roundtable*, pages 3–20, Berlin, Heidelberg, 2006. Springer, Springer Berlin Heidelberg. ISBN 978-3-540-34957-0. doi: 10.1007/978-3-540-34958-7_1.
- G. Bunin. A continuum theory for unstructured mesh generation in two dimensions. *Computer Aided Geometric Design*, 25(1):14–40, Jan. 2008. ISSN 0167-8396. doi: 10.1016/j.cagd.2007.05.002.
- M. Campen. Partitioning surfaces into quad patches. In A. Bousseau and D. Gutierrez, editors, *EG 2017 - Tutorials*. The Eurographics Association, 2017. doi: 10.2312/egt.20171033.
- M. Campen, D. Bommes, and L. Kobbelt. Dual loops meshing: Quality quad layouts on manifolds. *ACM Trans. Graph.*, 31(4), July 2012. ISSN 0730-0301. doi: 10.1145/2185520.2185606.
- M. Campen, D. Bommes, and L. Kobbelt. Quantized global parametrization. *ACM Transactions on Graphics*, 34(6):1–12, Nov. 2015. ISSN 0730-0301, 1557-7368. doi: 10.1145/2816795.2818140.
- A. S. Candy. A consistent approach to unstructured mesh generation for geophysical models. *ArXiv*, abs/1703.08491, 2017.
- E. Catmull and J. Clark. Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer-Aided Design*, 10(6):350–355, Nov. 1978. ISSN 0010-4485. doi: 10.1016/0010-4485(78)90110-0.
- A. Chemin, F. Henrotte, J.-F. Remacle, and J. V. Schaftingen. Representing three-dimensional cross fields using fourth order tensors. In X. Roca and A. Loseille, editors, *27th International Meshing Roundtable*, pages 89–108, Cham, 2019. Springer International Publishing. ISBN 978-3-030-13992-6.
- J. Cohen, A. Varshney, D. Manocha, G. Turk, H. Weber, P. Agarwal, F. Brooks, and W. Wright. Simplification envelopes. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques - SIGGRAPH '96*, pages 119–128, Not Known, 1996. ACM Press. ISBN 978-0-89791-746-9. doi: 10.1145/237170.237220.
- J. Cuillière. An adaptive method for the automatic triangulation of 3D parametric surfaces. *Computer-Aided Design*, 30(2):139–149, 1998. ISSN 0010-4485. doi: 10.1016/S0010-4485(97)00085-7.
- J. Daniels, C. T. Silva, J. Shepherd, and E. Cohen. Quadrilateral mesh simplification. In *ACM SIGGRAPH Asia 2008 Papers on - SIGGRAPH Asia '08*. Association for Computing Machinery, ACM Press, 2008. ISBN 978-1-4503-1831-0. doi: 10.1145/1457515.1409101.

- E. F. D'Azevedo. Are Bilinear Quadrilaterals Better Than Linear Triangles? *SIAM Journal on Scientific Computing*, 22(1):198–217, 2000. ISSN 1064-8275. doi: 10.1137/S106482759630406X.
- O. Devillers, S. Pion, and M. Teillaud. Walking in a triangulation. In *Proceedings of the Seventeenth Annual Symposium on Computational Geometry*, SCG '01, pages 106–114, New York, NY, USA, 2001. Association for Computing Machinery. ISBN 1-58113-357-X. doi: 10.1145/378583.378643.
- J. Docampo-Sánchez and R. Haimes. A regularization approach for automatic quad mesh generation. *28th International Meshing Roundtable*, 2019.
- M. S. Ebeida, K. Karamete, E. Mestreau, and S. Dey. Q-tran: A new approach to transform triangular meshes into quadrilateral meshes locally. In S. Shontz, editor, *Proceedings of the 19th International Meshing Roundtable*, pages 23–34, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. ISBN 978-3-642-15414-0.
- H.-C. Ebke, D. Bommers, M. Campen, and L. Kobbelt. QEx: Robust quad mesh extraction. *ACM Transactions on Graphics*, 32(6):1–10, Nov. 2013. ISSN 0730-0301, 1557-7368. doi: 10.1145/2508363.2508372.
- D. Engwirda. Multi-resolution unstructured grid-generation for geophysical applications on the sphere. 00:1–5, 2015.
- D. Eppstein. Linear Complexity Hexahedral Mesh Generation. *Computational Geometry*, 12(1-2):3–16, Feb. 1999. ISSN 09257721. doi: 10.1016/S0925-7721(98)00032-7.
- J. Erickson. Efficiently Hex-Meshing Things with Topology. *Discrete & Computational Geometry*, 52(3):427–449, Oct. 2014. ISSN 1432-0444. doi: 10.1007/s00454-014-9624-3.
- M. S. Floater and K. Hormann. Surface Parameterization: A Tutorial and Survey. In N. A. Dodgson, M. S. Floater, and M. A. Sabin, editors, *Advances in Multiresolution for Geometric Modelling*, pages 157–186. Springer-Verlag, Berlin/Heidelberg, 2005. ISBN 978-3-540-21462-5. doi: 10.1007/3-540-26808-1_9.
- P. Frey and L. Marechal. Fast Adaptive Quadtree Mesh Generation. *Proceedings of the 7th International Meshing Roundtable*, May 2000.
- P. J. Frey and P. L. George. *Mesh Generation: Application to Finite Elements*. Wiley [u.a.], Hoboken, NJ, 2. ed edition, 2008. ISBN 978-1-84821-029-5.
- P. J. Frey and L. Marechal. Fast adaptive quadtree mesh generation. In *In: Proceedings of the Seventh International Meshing Roundtable*, pages 211–224, 1998.

- X. Gao, W. Jakob, M. Tarini, and D. Panozzo. Robust hex-dominant mesh generation using field-guided polyhedral agglomeration. *ACM Transactions on Graphics*, 36(4):1–13, July 2017. ISSN 07300301. doi: 10.1145/3072959.3073676.
- M. Garland and P. S. Heckbert. Surface simplification using quadric error metrics. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques - SIGGRAPH '97*, pages 209–216, Not Known, 1997. ACM Press. ISBN 978-0-89791-896-1. doi: 10.1145/258734.258849.
- P. L. George, H. Borouchaki, P. J. Frey, P. Laug, and E. Saltel. Mesh generation and mesh adaptivity: Theory and techniques. In *Encyclopedia of Computational Mechanics*, chapter 17. American Cancer Society, 2007. ISBN 978-0-470-09135-7. doi: 10.1002/9780470091357.ecm012.pub2.
- C. Georgiadis, P.-A. Beaufort, J. Lambrechts, and J.-F. Remacle. High quality mesh generation using cross and asterisk fields: Application on coastal domains. In *26th International Meshing Roundtable, Research Notes*, Barcelona, Spain, Sept. 2017a.
- C. Georgiadis, P.-A. Beaufort, J. Lambrechts, and J.-F. Remacle. High quality mesh generation using cross and asterisk fields: Application on coastal domains. *arXiv e-prints*, art. arXiv:1706.02236, June 2017b.
- C. Geuzaine and J.-F. Remacle. Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities. *International Journal for Numerical Methods in Engineering*, 79(11):1309–1331, Sept. 2009. ISSN 1097-0207. doi: 10.1002/nme.2579.
- J. Gregson, A. Sheffer, and E. Zhang. All-hex mesh generation via volumetric PolyCube deformation. *Computer Graphics Forum*, 30(5):1407–1416, 2011. doi: 10.1111/j.1467-8659.2011.02015.x.
- H.-X. Guo, X. Liu, D.-M. Yan, and Y. Liu. Cut-enhanced PolyCube-maps for feature-aware all-hex meshing. *ACM Transactions on Graphics*, 39(4), July 2020. ISSN 0730-0301, 1557-7368. doi: 10.1145/3386569.3392378.
- A. Hertzmann and D. Zorin. Illustrating smooth surfaces. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '00*, pages 517–526, USA, 2000. Association for Computing Machinery, ACM Press/Addison-Wesley Publishing Co. ISBN 1-58113-208-5. doi: 10.1145/344779.345074.
- Hextreme. HXT Library - Hexahedral meshing in real time. <https://www.hextreme.eu/>, 2021.

- K. Hormann, P. Cignoni, and C. Montani. PolyCube-maps. *ACM Transactions on Graphics*, 23, July 2004. doi: 10.1145/1186562.1015810.
- J. Huang, Y. Tong, H. Wei, and H. Bao. Boundary aligned smooth 3D cross-frame field. In *Proceedings of the 2011 SIGGRAPH Asia Conference on - SA '11*, page 1, Hong Kong, China, 2011. ACM Press. ISBN 978-1-4503-0807-6. doi: 10.1145/2024156.2024177.
- T. Hughes, J. Cottrell, and Y. Bazilevs. Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement. *Computer Methods in Applied Mechanics and Engineering*, 194(39):4135–4195, 2005. ISSN 0045-7825. doi: 10.1016/j.cma.2004.10.008.
- W. Jakob, M. Tarini, D. Panozzo, and O. Sorkine-Hornung. Instant field-aligned meshes. *ACM Transactions on Graphics*, 34(6):1–15, Oct. 2015. doi: 10.1145/2816795.2818078.
- J. Jezdimirovic, A. Chemin, and J. F. Remacle. Multi-block decomposition and meshing of 2d domain using ginzburg-landau PDE. *28th International Meshing Roundtable, Sandia National Laboratories*, page 17, Oct. 2019.
- A. Johnen. *Indirect Quadrangular Mesh Generation and Validation of Curved Finite Elements*. PhD thesis, Universite de Liege, Liege,Belgique, Feb. 2016.
- A. Johnen, C. Geuzaine, T. Toulorge, and J.-F. Remacle. Efficient Computation of the Minimum of Shape Quality Measures on Curvilinear Finite Elements. *Procedia Engineering*, 163:328–339, 2016. doi: 10.1016/j.proeng.2016.11.067.
- F. Kälberer, M. Nieser, and K. Polthier. QuadCover - Surface parameterization using branched coverings. *Computer Graphics Forum*, 26(3):375–384, 2007. ISSN 1467-8659. doi: 10.1111/j.1467-8659.2007.01060.x.
- S. L. Karman. Virtual Control Volumes for Two-Dimensional Unstructured Elliptic Smoothing. In S. Shontz, editor, *Proceedings of the 19th International Meshing Roundtable*, pages 121–142. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. ISBN 978-3-642-15413-3 978-3-642-15414-0. doi: 10.1007/978-3-642-15414-0_8.
- L. Kettner. Designing a Data Structure for Polyhedral Surfaces. In *Proceedings of the Fourteenth Annual Symposium on Computational Geometry*, SCG '98, pages 146–154, New York, NY, USA, 1998. ACM. ISBN 978-0-89791-973-9. doi: 10.1145/276884.276901.
- P. Kinney. CleanUp: Improving quadrilateral finite element meshes. In *6th International Meshing Roundtable*, pages 449–461, 1997.

- P. M. Knupp. Winslow Smoothing on Two-Dimensional Unstructured Meshes. *Engineering with Computers*, 15(3):263–268, Sept. 1999. ISSN 0177-0667, 1435-5663. doi: 10.1007/s003660050021.
- S. Koch, A. Matveev, Z. Jiang, F. Williams, A. Artemov, E. Burnaev, M. Alexa, D. Zorin, and D. Panozzo. ABC: A big CAD model dataset for geometric deep learning. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9601–9611. IEEE, IEEE, June 2019. ISBN 978-1-72813-293-8. doi: 10.1109/cvpr.2019.00983.
- N. Kowalski, F. Ledoux, and P. Frey. A PDE Based Approach to Multidomain Partitioning and Quadrilateral Meshing. In X. Jiao and J.-C. Weill, editors, *Proceedings of the 21st International Meshing Roundtable*, pages 137–154, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-33573-0.
- N. Kowalski, F. Ledoux, and P. Frey. Block-structured Hexahedral Meshes for CAD Models Using 3D Frame Fields. *Procedia Engineering*, 82:59–71, Jan. 2014. ISSN 1877-7058. doi: 10.1016/j.proeng.2014.10.373.
- M. Lai, S. Benzley, and D. White. Automated hexahedral mesh generation by generalized multiple source to multiple target sweeping. *International Journal for Numerical Methods in Engineering*, 49(1-2):261–275, 2000. doi: 10.1002/1097-0207(20000910/20)49:1/2<261::AID-NME932>3.0.CO;2-G.
- T. S. Lau and S. H. Lo. Finite element mesh generation over analytical curved surfaces. *Computers & Structures*, 59(2):301–309, Apr. 1996. ISSN 0045-7949. doi: 10.1016/0045-7949(95)00261-8.
- F. Ledoux. MAMBO dataset. <https://gitlab.com/franck.ledoux/mambo>.
- F. Ledoux and J. Shepherd. Topological and geometrical properties of hexahedral meshes. *Engineering with Computers*, 26(4):419–432, Aug. 2010. ISSN 0177-0667, 1435-5663. doi: 10.1007/s00366-009-0144-3.
- C. Lee and R. Hobbs. Automatic adaptive finite element mesh generation over rational b-spline surfaces. *Computers & Structures*, 69(5):577–608, 1998. ISSN 0045-7949. doi: 10.1016/S0045-7949(98)80096-5.
- C. Lee and S. Lo. A new scheme for the generation of a graded quadrilateral mesh. *Computers & Structures*, 52(5):847–857, Sept. 1994. ISSN 0045-7949. doi: 10.1016/0045-7949(94)90070-1.
- B. Lévy and Y. Liu. Lp Centroidal Voronoi Tessellation and Its Applications. In *ACM SIGGRAPH 2010 Papers*, SIGGRAPH '10, pages 119:1–119:11, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0210-4. doi: 10.1145/1833349.1778856.

- H. Li and G. Cheng. New method for graded mesh generation of all hexahedral finite elements. *Computers & Structures*, 76(6):729–740, July 2000. ISSN 0045-7949. doi: 10.1016/S0045-7949(99)00193-5.
- Y. Li, Y. Liu, W. Xu, W. Wang, and B. Guo. All-hex meshing using singularity-restricted field. *ACM Trans. Graph.*, 31(6), Nov. 2012. ISSN 0730-0301. doi: 10.1145/2366145.2366196.
- H. Liu, P. Zhang, E. Chien, J. Solomon, and D. Bommes. Singularity-constrained octahedral fields for hexahedral meshing. *ACM Transactions on Graphics*, 37(4):1–17, Aug. 2018. ISSN 0730-0301, 1557-7368. doi: 10.1145/3197517.3201344.
- L. Liu, Y. Zhang, Y. Liu, and W. Wang. Feature-preserving T-mesh construction using skeleton-based polycubes. *Computer-Aided Design*, 58:162–172, Jan. 2015. ISSN 0010-4485. doi: 10.1016/j.cad.2014.08.020.
- S.-s. Liu and R. Gadh. Automatic Hexahedral Mesh Generation by Recursive Convex and Swept Volume Decomposition. In *6th International Meshing Roundtable, Sandia National Laboratories*, pages 217–231, 1997.
- M. Livesu, N. Pietroni, E. Puppo, A. Sheffer, and P. Cignoni. Loopy-Cuts: Practical Feature-Preserving Block Decomposition for Strongly Hex-Dominant Meshing. *ACM Transactions on Graphics*, 39, July 2020. doi: 10.1145/3386569.3392472.
- S. Lo. Finite element mesh generation over curved surfaces. *Computers & Structures*, 29(5):731–742, 1988. ISSN 0045-7949. doi: 10.1016/0045-7949(88)90341-0.
- S. Lo. Generating quadrilateral elements on plane and over curved surfaces. *Computers & Structures*, 31(3):421–426, Jan. 1989. ISSN 00457949. doi: 10.1016/0045-7949(89)90389-1.
- S. Lo. *Finite Element Mesh Generation*. Jan. 2015. ISBN ISBN 9780415690485.
- R. Löhner and P. Parikh. Generation of three-dimensional unstructured grids by the advancing-front method. *International Journal for Numerical Methods in Fluids*, 8(10):1135–1149, Oct. 1988. ISSN 0271-2091, 1097-0363. doi: 10.1002/fld.1650081003.
- A. Loseille and R. Löhner. Boundary Layer Mesh Generation and Adaptivity. *AIAA Conference*, (January):AIAA 2011–894, 2011. ISSN 978-1-60086-950-1. doi: 10.2514/6.2011-894.

- Y. Lu, R. Gadh, and T. J. Tautges. Feature based hex meshing methodology: Feature recognition and volume decomposition. *Computer-Aided Design*, 33(3):221–232, Mar. 2001. ISSN 0010-4485. doi: 10.1016/S0010-4485(00)00122-6.
- M. Lyon, D. Bommès, and L. Kobbelt. HexEx: Robust hexahedral mesh extraction. *ACM Transactions on Graphics*, 35(4):1–11, July 2016. ISSN 07300301. doi: 10.1145/2897824.2925976.
- G. Marcias, K. Takayama, N. Pietroni, D. Panozzo, O. Sorkine-Hornung, E. Puppo, and P. Cignoni. Data-driven interactive quadrangulation. *ACM Transactions on Graphics*, 34(4):1–10, July 2015. ISSN 0730-0301, 1557-7368. doi: 10.1145/2766964.
- L. Maréchal. Advances in octree-based all-hexahedral mesh generation: Handling sharp features. In *Proceedings of the 18th International Meshing Roundtable*, pages 65–84. Springer Berlin Heidelberg, 2009. ISBN 978-3-642-04318-5 978-3-642-04319-2. doi: 10.1007/978-3-642-04319-2_5.
- C. Marot, J. Pellerin, and J.-F. Remacle. One machine, one minute, three billion tetrahedra. *International Journal for Numerical Methods in Engineering*, 117(9):967–990, 2019. ISSN 1097-0207. doi: 10.1002/nme.5987.
- B. D. McKay and A. Piperno. Practical graph isomorphism, II. *Journal of Symbolic Computation*, 60(0):94–112, Jan. 2014. ISSN 0747-7171. doi: 10.1016/j.jsc.2013.09.003.
- S. Meshkat and D. Talmor. Generating a mixed mesh of hexahedra, pentahedra and tetrahedra from an underlying tetrahedral mesh. *International Journal for Numerical Methods in Engineering*, 49(1-2):17–30, Sept. 2000. doi: 10.1002/1097-0207(20000910/20)49:1/2<17::AID-NME920>3.0.CO;2-U.
- S. A. Mitchell. A characterization of the quadrilateral meshes of a surface which admit a compatible hexahedral mesh of the enclosed volume. In G. Goos, J. Hartmanis, J. Leeuwen, C. Puech, and R. Reischuk, editors, *STACS 96*, volume 1046, pages 465–476. Springer Berlin Heidelberg, Berlin, Heidelberg, 1996. ISBN 978-3-540-60922-3 978-3-540-49723-3. doi: 10.1007/3-540-60922-9_38.
- S. A. Mitchell, M. A. Mohammed, A. H. Mahmoud, and M. S. Ebeida. Delaunay Quadrangulation by Two-coloring Vertices. *Procedia Engineering*, 82:364–376, Jan. 2014. ISSN 1877-7058. doi: 10.1016/j.proeng.2014.10.397.
- P. Murdoch, S. Benzley, T. Blacker, and S. A. Mitchell. The spatial twist continuum: A connectivity based method for representing all-hexahedral finite element meshes. *Finite Elements in Analysis and Design*, 28(2):137–149, Dec. 1997. ISSN 0168-874X. doi: 10.1016/S0168-874X(97)81956-7.

- A. Myles, N. Pietroni, and D. Zorin. Robust field-aligned global parametrization. *ACM Trans. Graph.*, 33(4):1–14, July 2014. ISSN 0730-0301, 1557-7368. doi: 10.1145/2601097.2601154.
- M. Nieser, U. Reitebuch, and K. Polthier. CubeCover- Parameterization of 3D Volumes. *Computer Graphics Forum*, 30(5):1397–1406, Aug. 2011. ISSN 01677055. doi: 10.1111/j.1467-8659.2011.02014.x.
- M. Nieser, J. Palacios, K. Polthier, and E. Zhang. Hexagonal global parameterization of arbitrary surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 18(6):865–878, 2012. doi: 10.1109/TVCG.2011.118.
- S. Owen. A survey of unstructured mesh generation technology. *7th International Meshing Roundtable*, pages 25–25, 1998.
- S. Owen, S. Canann, and S. Saigal. Pyramid elements for maintaining tetrahedra to hexahedra conformability. *Trends In Unstructured Mesh Generation*, 220, Jan. 1997.
- S. Owen, M. Staten, S. Canann, and S. Saigal. Q-Morph: An indirect approach to advancing front quad meshing. *International Journal for Numerical Methods in Engineering*, 44(9):1317–1340, 1999. ISSN 0029-5981. doi: 10.1002/(SICI)1097-0207(19990330)44:9<1317::AID-NME532>3.0.CO;2-N.
- J. Palacios and E. Zhang. Rotational symmetry field design on surfaces. *ACM Trans. Graph.*, 26(3):55–es, July 2007. ISSN 0730-0301. doi: 10.1145/1276377.1276446.
- D. Palmer, D. Bommers, and J. Solomon. Algebraic representations for volumetric frame fields. *ACM Trans. Graph.*, 39(2):1–17, Apr. 2020. ISSN 0730-0301, 1557-7368. doi: 10.1145/3366786.
- J. Pellerin, A. Johnen, K. Verhetsel, and J.-F. Remacle. Identifying combinations of tetrahedra into hexahedra: A vertex based strategy. *Computer-Aided Design*, June 2018a. ISSN 00104485. doi: 10.1016/j.cad.2018.05.004.
- J. Pellerin, K. Verhetsel, and J.-F. Remacle. There are 174 subdivisions of the hexahedron into tetrahedra. *ACM Trans. Graph.*, 37(6), Dec. 2018b. ISSN 0730-0301. doi: 10.1145/3272127.3275037.
- C. S. Peskin. Numerical analysis of blood flow in the heart. *Journal of Computational Physics*, 25(3):220–252, 1977. ISSN 0021-9991. doi: 10.1016/0021-9991(77)90100-0.
- N. Pietroni, E. Puppo, G. Marcias, R. Scopigno, and P. Cignoni. Tracing Field-Coherent Quad Layouts. In *Computer Graphics Forum*, volume 35, pages 485–496. Wiley Online Library, Oct. 2016. doi: 10.1111/cgf.13045.

- A. Pochet, W. Celes, H. Lopes, and M. Gattass. A new quadtree-based approach for automatic quadrilateral mesh generation. *Engineering with Computers*, 33(2):275–292, Apr. 2017. ISSN 1435-5663. doi: 10.1007/s00366-016-0471-0.
- M. A. Price, C. G. Armstrong, and M. A. Sabin. Hexahedral mesh generation by medial surface subdivision: Part I. Solids with convex edges. *International Journal for Numerical Methods in Engineering*, 38(19):3335–3359, 1995. doi: 10.1002/nme.1620381910.
- D. T. QGIS. Quantum GIS geographic information system. *Open Source Geospatial Foundation Project*, 2011.
- N. Ray, W. C. Li, B. Lévy, A. Sheffer, and P. Alliez. Periodic global parameterization. *ACM Trans. Graph.*, 25(4):1460–1485, Oct. 2006. ISSN 0730-0301, 1557-7368. doi: 10.1145/1183287.1183297.
- N. Ray, B. Vallet, W. C. Li, and B. Lévy. N-symmetry direction field design. *ACM Transactions on Graphics*, 27(2):1–13, Apr. 2008. ISSN 0730-0301. doi: 10.1145/1356682.1356683.
- N. Ray, B. Vallet, L. Alonso, and B. Levy. Geometry-aware direction field processing. *ACM Trans. Graph.*, 29(1), Dec. 2009. ISSN 0730-0301. doi: 10.1145/1640443.1640444.
- N. Ray, D. Sokolov, and B. Lévy. Practical 3D frame field generation. *ACM Trans. Graph.*, 35(6), Nov. 2016. ISSN 0730-0301. doi: 10.1145/2980179.2982408.
- N. Ray, D. Sokolov, M. Reberol, F. Ledoux, and B. Lévy. Hex-dominant meshing: Mind the gap! *Computer-Aided Design*, 102:94–103, Sept. 2018. ISSN 0010-4485. doi: 10.1016/j.cad.2018.04.012.
- S. Rebay. Efficient Unstructured Mesh Generation by Means of Delaunay Triangulation and Bowyer-Watson Algorithm. *Journal of Computational Physics*, 106(1):125–138, May 1993. ISSN 0021-9991. doi: 10.1006/jcph.1993.1097.
- M. Reberol, A. Chemin, and J.-F. Remacle. Multiple approaches to frame field correction for CAD models. *arXiv preprint arXiv:1912.01248*, 2019.
- J.-F. Remacle. A two-level multithreaded Delaunay kernel. *Computer-Aided Design*, 85:2–9, 2017. ISSN 0010-4485. doi: 10.1016/j.cad.2016.07.018.
- J.-F. Remacle and J. Lambrechts. Fast and Robust Mesh Generation on the Sphere – Application to Coastal Domains. *Procedia Engineering*, 163:20–32, Jan. 2016. ISSN 1877-7058. doi: 10.1016/j.proeng.2016.11.011.

- J.-F. Remacle and J. Lambrechts. Fast and robust mesh generation on the sphere—Application to coastal domains. *Computer-Aided Design*, 103:14–23, 2018. ISSN 0010-4485. doi: 10.1016/j.cad.2018.03.002.
- J.-F. Remacle, C. Geuzaine, G. Compère, and É. Marchandise. High-quality surface remeshing using harmonic maps. *International Journal for Numerical Methods in Engineering*, (March):n/a–n/a, 2010. ISSN 978-1-4577-0079-8. doi: 10.1002/nme.2824.
- J.-F. Remacle, J. Lambrechts, B. Seny, E. Marchandise, A. Johnen, and C. Geuzainet. Blossom-Quad: A non-uniform quadrilateral mesh generator using a minimum-cost perfect-matching algorithm. *International Journal for Numerical Methods in Engineering*, 89(9):1102–1119, Mar. 2012. ISSN 00295981. doi: 10.1002/nme.3279.
- J.-F. Remacle, F. Henrotte, T. Carrier-Baudouin, E. Béchet, E. Marchandise, C. Geuzaine, and T. Mouton. A frontal Delaunay quad mesh generator using the L_∞ norm. *International Journal for Numerical Methods in Engineering*, 94(5):494–512, 2013. doi: 10.1002/nme.4458.
- X. Roca and J. Sarrate. An automatic and general least-squares projection procedure for sweep meshing. volume 26 of *Engineering With Computers - EWC*, pages 487–506, Jan. 2006. ISBN 978-3-540-34957-0. doi: 10.1007/978-3-540-34958-7_28.
- E. Ruiz-Gironés, X. Roca, and J. Sarrate. A New Procedure to Compute Imprints in Multi-sweeping Algorithms. In B. W. Clark, editor, *Proceedings of the 18th International Meshing Roundtable*, pages 281–299, Berlin, Heidelberg, 2009. Springer. ISBN 978-3-642-04319-2. doi: 10.1007/978-3-642-04319-2_17.
- A. A. Rushdi, S. A. Mitchell, A. H. Mahmoud, C. C. Bajaj, and M. S. Ebeida. All-quad meshing without cleanup. *Computer-Aided Design*, 85:83–98, Apr. 2017. ISSN 00104485. doi: 10.1016/j.cad.2016.07.009.
- D. Rypl. Sweeping of unstructured meshes over generalized extruded volumes. *Finite Elements in Analysis and Design*, 46(1):203–215, Jan. 2010. ISSN 0168-874X. doi: 10.1016/j.finel.2009.06.018.
- M. Sabin. Criteria for comparison of automatic mesh generation methods. *Advances in Engineering Software and Workstations*, 13(5):220–225, 1991. ISSN 0961-3552. doi: 10.1016/0961-3552(91)90028-3.
- A. O. Sageman-Furnas, A. Chern, M. Ben-Chen, and A. Vaxman. Chebyshev nets from commuting PolyVector fields. *ACM Trans. Graph.*, 38(6):1–16, Nov. 2019. ISSN 0730-0301, 1557-7368. doi: 10.1145/3355089.3356564.

- J. Sarrate and A. Huerta. Efficient unstructured quadrilateral mesh generation. *International Journal for Numerical Methods in Engineering*, 49(10):1327–1350, 2000. doi: 10.1002/1097-0207(20001210)49:10<1327::AID-NME996>3.0.CO;2-L.
- R. Schneiders. A grid-based algorithm for the generation of hexahedral element meshes. *Engineering with Computers*, 12(3-4):168–177, Sept. 1996. ISSN 0177-0667, 1435-5663. doi: 10.1007/BF01198732.
- R. Schneiders. Algorithms for quadrilateral and hexahedral mesh generation. In: *VKI Lecture Series on Computational Fluid Dynamic. VKI-LS*, 2000, Jan. 2000a.
- R. Schneiders. Octree-Based Hexahedral Mesh Generation. *International Journal of Computational Geometry & Applications*, 10(04):383–398, 2000b. doi: 10.1142/S021819590000022X.
- A. Shamir. A survey on Mesh Segmentation Techniques. *Computer Graphics Forum*, 27(6):1539–1556, Sept. 2008. ISSN 01677055, 14678659. doi: 10.1111/j.1467-8659.2007.01103.x.
- A. Sheffer, E. Praun, and K. Rose. Mesh Parameterization Methods and Their Applications. *Foundations and Trends® in Computer Graphics and Vision*, 2(2):105–171, 2006. ISSN 1572-2740, 1572-2759. doi: 10.1561/06000000011.
- X. Sheng and B. E. Hirsch. Triangulation of trimmed surfaces in parametric space. *Computer-Aided Design*, 24(8):437–444, Aug. 1992. ISSN 0010-4485. doi: 10.1016/0010-4485(92)90011-X.
- M. S. Shephard and M. K. Georges. Automatic three-dimensional mesh generation by the finite octree technique. *International Journal for Numerical Methods in Engineering*, 32(4):709–749, Sept. 1991. ISSN 0029-5981, 1097-0207. doi: 10.1002/nme.1620320406.
- J. F. Shepherd and C. R. Johnson. Hexahedral mesh generation constraints. *Engineering with Computers*, 24(3):195–213, Sept. 2008. ISSN 0177-0667, 1435-5663. doi: 10.1007/s00366-008-0091-4.
- K. Shimada and D. C. Gossard. Automatic triangular mesh generation of trimmed parametric surfaces for finite element analysis. *Computer Aided Geometric Design*, 15(3):199–222, 1998. ISSN 0167-8396. doi: 10.1016/S0167-8396(97)00037-X.
- D. Sokolov, N. Ray, L. Untereiner, and B. Lévy. Hexahedral-Dominant Meshing. *ACM Transactions on Graphics*, 35(5):157:1–157:23, June 2016. ISSN 0730-0301. doi: 10.1145/2930662.

- M. L. Staten, S. J. Owen, and T. D. Blacker. Unconstrained Paving & Plastering: A New Idea for All Hexahedral Mesh Generation. In B. W. Hanks, editor, *Proceedings of the 14th International Meshing Roundtable*, pages 399–416, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. ISBN 978-3-540-29090-2. doi: 10.1007/3-540-29090-7_24.
- M. B. Stephenson, S. A. Canann, and T. D. Blacker. Plastering: A new approach to automated, 3D hexahedral mesh generation. Feb. 1992.
- C. Stimpson, C. Ernst, P. Knupp, P. Pebay, and D. Thompson. The verdict geometric quality library. *Sandia Lab. Report SAND2007-175*, 2007.
- T. M. Stricker. Hypercubes and meshes. In D. Padua, editor, *Encyclopedia of Parallel Computing*, pages 861–871. Springer US, Boston, MA, 2011. ISBN 978-0-387-09766-4. doi: 10.1007/978-0-387-09766-4_408.
- V. Sunil and S. Pande. Automatic recognition of features from freeform surface CAD models. *Computer-Aided Design*, 40(4):502–517, Apr. 2008. ISSN 00104485. doi: 10.1016/j.cad.2008.01.006.
- K. Takayama, D. Panozzo, and O. Sorkine-Hornung. Pattern-based quadrangulation for n-sided patches. In *Computer Graphics Forum*, volume 33, pages 177–184. Wiley Online Library, 2014.
- T. Tam and C. Armstrong. 2D finite element mesh generation by medial axis subdivision. *Advances in Engineering Software and Workstations*, 13(5):313–324, 1991. ISSN 0961-3552. doi: 10.1016/0961-3552(91)90035-3.
- M. Tarini, N. Pietroni, P. Cignoni, D. Panozzo, and E. Puppo. Practical quad mesh simplification. *Computer Graphics Forum*, 29(2):407–418, May 2010. ISSN 01677055. doi: 10.1111/j.1467-8659.2009.01610.x.
- T. J. Tautges. The generation of hexahedral meshes for assembly geometry: Survey and progress. *International Journal for Numerical Methods in Engineering*, 50(12):2617–2642, 2001. ISSN 1097-0207. doi: 10.1002/nme.139.
- T. J. Tautges, T. Blacker, and S. A. Mitchell. The Whisker Weaving Algorithm: A Connectivity-Based Method for Constructing All-Hexahedral Finite Element Meshes. *International Journal for Numerical Methods in Engineering*, 39(19):3327–3349, 1996. ISSN 1097-0207. doi: 10.1002/(SICI)1097-0207(19961015)39:19<3327::AID-NME2>3.0.CO;2-H.
- A. Thakur, A. G. Banerjee, and S. K. Gupta. A survey of CAD model simplification techniques for physics-based simulation applications. *Computer-Aided Design*, 41(2):65–80, Feb. 2009. ISSN 00104485. doi: 10.1016/j.cad.2008.11.009.

- J. F. Thompson, B. K. Soni, and N. P. Weatherill. *Handbook of Grid Generation*. CRC Press, Dec. 1998. ISBN 978-0-367-80246-2. doi: 10.1201/9781420050349.
- J. F. Thompson, B. K. Soni, and N. P. Weatherill, editors. *Handbook of Grid Generation*. CRC Press, Boca Raton, Fla, 1999. ISBN 978-0-8493-2687-5.
- A. Vaxman, M. Campen, O. Diamanti, D. Bommers, K. Hildebrandt, M. Ben-Chen, and D. Panozzo. Directional field synthesis, design, and processing. In *SIGGRAPH ASIA 2016 Courses*, volume 35, pages 545–572. Wiley Online Library, ACM, Nov. 2016. ISBN 978-1-4503-4538-5. doi: 10.1145/2988458.2988478.
- K. Verhetsel, J. Pellerin, and J.-F. Remacle. Finding hexahedrizations for small quadrangulations of the sphere. *ACM Transactions on Graphics (TOG)*, 38(4):53, July 2019. ISSN 0730-0301. doi: 10.1145/3306346.3323017.
- C. S. Verma and K. Suresh. A Robust Combinatorial Approach to Reduce Singularities in Quadrilateral Meshes. *Procedia Engineering*, 124:252–264, 2015. ISSN 18777058. doi: 10.1016/j.proeng.2015.10.137.
- C. S. Verma and T. Tautges. Jaal: Engineering a high quality all-quadrilateral mesh generator. In *Proceedings of the 20th International Meshing Roundtable*, pages 511–530. Springer Berlin Heidelberg, 2011. ISBN 978-3-642-24733-0 978-3-642-24734-7. doi: 10.1007/978-3-642-24734-7_28.
- M. Vieira and K. Shimada. Surface mesh segmentation and smooth surface extraction through region growing. *Computer Aided Geometric Design*, 22(8):771–792, Nov. 2005. ISSN 01678396. doi: 10.1016/j.cagd.2005.03.006.
- R. Viertel and B. Osting. An approach to quad meshing based on harmonic cross-valued maps and the Ginzburg–Landau theory. *SIAM Journal of Scientific Computing*, 41(1):A452–A479, Jan. 2019. ISSN 1064-8275, 1095-7197. doi: 10.1137/17m1142703.
- R. Viertel, M. Staten, and F. Ledoux. Analysis of Non-Meshable Automatically Generated Frame Fields. In *25th International Meshing Roundtable*, page 6, 2016.
- D. Wang, O. Hassan, K. Morgan, and N. Weatherill. EQSM: An efficient high quality surface grid generation method based on remeshing. *Computer Methods in Applied Mechanics and Engineering*, 195(41-43):5621–5633, Aug. 2006. ISSN 00457825. doi: 10.1016/j.cma.2005.10.028.
- M. Wardetzky, M. Bergou, D. Harmon, D. Zorin, and E. Grinspun. Discrete quadratic curvature energies. *Computer Aided Geometric Design*, 24(8-9): 499–518, Nov. 2007. ISSN 0167-8396. doi: 10.1016/j.cagd.2007.07.006.

- D. White and P. Kinney. Redesign of the Paving Algorithm: Robustness Enhancements through Element by Element Meshing. In *Proc. 6 Th Int. Meshing Roundtable*, pages 323–335, 1997.
- Z. Yasseen, A. Nasri, W. Boukaram, P. Volino, and N. Magnenat-Thalmann. Sketch-based garment design with quad meshes. *Computer-Aided Design*, 45(2):562–567, Feb. 2013. ISSN 00104485. doi: 10.1016/j.cad.2012.10.041.
- W. Yu, K. Zhang, S. Wan, and X. Li. Optimizing polycube domain construction for hexahedral remeshing. *Computer-Aided Design*, 46:58–68, Jan. 2014. ISSN 0010-4485. doi: 10.1016/j.cad.2013.08.018.
- G. Zhao, H. Zhang, and L. Cheng. Geometry-adaptive generation algorithm and boundary match method for initial hexahedral element mesh. *Engineering with Computers*, 24(4):321–339, Oct. 2008. ISSN 1435-5663. doi: 10.1007/s00366-007-0085-7.
- Z. Zheng, R. Wang, S. Gao, Y. Liao, and M. Ding. Dual surface based approach to block decomposition of solid models. pages 149–167. July 2019. ISBN 978-3-030-13991-9.
- Q. Zhou and A. Jacobson. Thingi10K: A dataset of 10,000 3D-Printing models. *arXiv preprint arXiv:1605.04797*, 2016.
- J. Z. Zhu, O. C. Zienkiewicz, E. Hinton, and J. Wu. A new approach to the development of automatic quadrilateral mesh generation. *International Journal for Numerical Methods in Engineering*, 32(4):849–866, Sept. 1991. ISSN 0029-5981, 1097-0207. doi: 10.1002/nme.1620320411.