

Integrable Frame Fields for Quadrilateral and Hexahedral Meshing

Mattéo Couplet

*Thesis submitted in partial fulfilment of the requirements
for the Degree of Doctor in Engineering Sciences*

September 2024

Thesis Committee:

Prof. Jean-François Remacle (Advisor)	UCLouvain
Dr. Alexandre Chemin	UCLouvain
Prof. Christophe Geuzaine	ULiège
Dr. Étienne Corman	CNRS
Prof. David Bommes	University of Bern
Prof. Edward Chien	Boston University
Prof. Renaud Ronsse (Chair)	UCLouvain

This work was supported by the F.R.S.-FNRS.

Remerciements

Avant de commencer cette thèse, on m'avait donné le conseil "*ne choisis pas un sujet, choisis un promoteur*". Et il faut reconnaître que je n'aurais pas pu faire un meilleur choix : Jean-François, au-delà d'être un esprit brillant et d'une culture scientifique immense, démontre une humanité et une bienveillance rare. Il restera pour moi un exemple de chercheur et de mentor, et je lui souhaite de continuer à inspirer ses doctorants actuels et futurs.

Je souhaite tout autant remercier deux autres chercheurs et amis qui ont été de précieux conseillers. Dans un contexte de confinement difficile, Maxence Reberol m'a initié au sujet fascinant du maillage, avec son expertise hors norme en la matière. Alexandre Chemin a repris le flambeau un an plus tard, et c'est peu dire que ses conseils avisés, son aide dans le code et sa confiance incessante ont été déterminants dans l'aboutissement de ce travail.

I would like to express my gratitude to Christophe Geuzaine, Étienne Corman, David Bommès, Edward Chien and Renaud Ronsse for joining in on my thesis committee, and for their insightful comments that enabled this manuscript to reach its final form. Thank you to David Palmer and David Bommès for the numerous Zoom discussions we had and for your invaluable input; I look forward to many more years of fruitful collaboration. Ed, thank you for warmly welcoming me in your lab; I cannot wait to join your team!

Merci à toute l'équipe Hextreme de m'avoir chaleureusement accueilli au tout début : Célestin, Kilian, Jovana, Christos, Ruili, Ruiyang, Amaury, P-A, Jon. Et aux Eulériens passés et présents pour l'atmosphère scientifique et humaine exceptionnelle : Ange, Anthony, Antoine, Chiheb, Guillaume³, Illy, Insaf, Jose, Julien, Michel, Miguel, Momo, Nathan, Rémi, Simon, Thomas, Vincent, et tous les autres. Les innombrables tartes, team buildings, INMA- et MEMArathons... vont terriblement me manquer de l'autre côté de l'Atlantique.

La vie ne s'arrête pas aux portes de l'Euler. Merci aux copains d'ingé pour les soirées, les skis, les journées coworking : Martin, Constant, Simon, MO, Léa, Julien, Nicolas et les autres. Merci à tous les membres passés et présents de l'OSSEL pour cette aventure musicale unique, et bien sûr les tournées inoubliables. Avec une mention spéciale pour les Bhumis (Pablo, Raph et Val) et notre voyage annuel (depuis 1 an), qui j'espère deviendra une tradition.

Merci à ma famille, proche, élargie ou belle-famille, pour leur soutien constant et leurs encouragements. Avec une pensée toute particulière pour mes

parents, qui m'ont toujours donné l'opportunité de m'épanouir dans tout ce que j'entreprends. Même si loin des yeux, soyez assurés que vous serez toujours proches de mon cœur.

Merci enfin à Diane d'avoir été ma plus grande supportrice, dans les bons moments comme dans les mauvais, et pour ton infinie patience durant le sprint final de cette thèse. J'ai une chance immense de partager ma vie avec toi, et ces quelques mots ne suffiront pas à exprimer tout ce que tu représentes pour moi.

Contents

Contents	5
1 Introduction	7
1.1 Meshes, parametrizations and frame fields	10
1.2 The field-driven meshing pipeline	22
1.3 Related work	23
1.4 Thesis organization	30
2 Coarse quadrilateral layouts using quantization	31
2.1 Introduction	32
2.2 Quasi-structured quadrilateral meshing	35
2.3 Quadrilateral layout simplification	37
2.4 Block-structured quad mesh and high-order quadrilaterals . . .	41
2.5 Results and discussion	43
2.6 Conclusion and future work	45
3 The odeco frame representation	47
3.1 Symmetries of a parametrization’s Jacobian	48
3.2 Representing frames using orthogonally decomposable tensors .	49
3.3 Computing with odeco tensors	54
4 Integrable frame fields using odeco tensors	63
4.1 Curl of an odeco tensor field	65
4.2 Curl optimization	68
4.3 Computing 2D seamless parametrizations	74
4.4 2D results and discussion	75
4.5 3D results and discussion	82
5 Conclusion	87
5.1 Summary of contributions	87
5.2 What it means for meshing	88
5.3 Research avenues	88
Bibliography	91

Introduction

Numerical simulation enables engineers and scientists to simulate physical processes on a computer. In the past few decades, it has radically improved engineering processes across industries: aeronautics and automotive design, civil engineering, and many more. Being able to gain insights on a design without a real-life experimental setup adds immense value to the process, as engineers can perform even complex experiments at virtually no cost. It also allows the study of physical phenomena that are otherwise challenging to observe, e.g., ocean dynamics or turbulent flows. Numerical simulation also has a predictive power as it can make a system evolve in time from an initial state; weather prediction and climate modeling are impactful examples.

Many processes that engineers study can be accurately modeled by the hypothesis of *continuum mechanics*: even if matter is made of atoms, to the naked eye, solids and fluids seem to be continuous, i.e., their characteristic properties (density, elasticity, velocity, etc.) are continuous. This hypothesis is used notably in solid mechanics, fluid mechanics, and electromagnetics, and allows to resort to continuous and differentiable functions to model the useful quantities. They are then governed by *partial differential equations* (PDEs) that describe the physical phenomena through space and time.

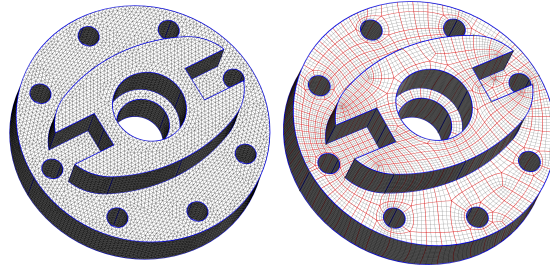


Figure 1.1: An unstructured triangle mesh vs. a block-structured quad mesh.

The vast majority of PDE solvers rely on a spatial discretization of the bulk of the domain. Such a discretization is typically done through a *mesh*, i.e., a partition of the domain into simple elements that are easy to represent on a computer. Since the 1970s and still today, the Finite Element Method (FEM) has dominated the numerical simulation scene, thanks to (i) its ability to solve nonlinear problems, (ii) its ability to deal with complex geometries using unstructured meshes (made of triangles or tetrahedra), and (iii) its mathematical soundness and structure of proof. Yet, a typical finite element algorithm is often memory bound and its performance rarely exceeds 10 % of the peak efficiency of a modern computer.

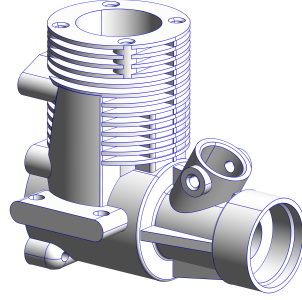
Computational physicists wishing to achieve higher performance typically resort to *block-structured meshes*, made of large quadrilateral or hexahedral *blocks*. These meshes allow to build efficient algorithms that can use over 90 % of the computing power, e.g., high-order finite elements, or finite differences that run on GPU. Currently, these kinds of block decompositions are often created manually by numerical engineers, making the meshing phase allegedly the bottleneck of the numerical simulation process. This motivates the search for automatic and robust algorithms for generating block-structured meshes on the complex geometries that are ubiquitous in engineering.

A task closely related to block-structured meshing is *surface parametrization*, which can be roughly summarized as flattening surfaces in the best possible way. Surface parametrization finds many applications in *computer graphics*: for example, in *texture mapping*, objects in a virtual scene are mapped to the plane, and a flat image texture is then mapped back onto the object through the parametrization. It can also be used as a quad meshing tool, by mapping back the Cartesian grid onto the domain to be meshed. Quad meshes are important for artists in computer graphics as they can be aligned to the curvature of the surface, and can be locally subdivided in a natural way.

Thanks to the efforts of the computer graphics community, quadrilateral meshing is approaching a mature state. Yet, state-of-the-art meshers still suffer from robustness issues when dealing with industrial-grade CAD models. These models possess hundreds of geometrical features to which the quad mesh must strictly conform, at a wide range of scales. The fact that industrial meshers still employ unstructured meshing techniques such as triangle splitting, recombina-

tion, or pattern identification shows that the methods from computer graphics still have room for improvement.

Recently, Reberol et al. (2021) built a quad mesher that combines some of the best techniques from both worlds, where an unstructured mesh is gradually improved under the topological guidance of a cross field. The pipeline produces a *quasi-structured* mesh, i.e., close to block-structured. A **first contribution** of this thesis is a method to make this mesh block-structured by coarsening its quad layout. We borrow ideas from the computer graphics works on quad layout optimization, with the novelty that we do not require any parametrization, but only a quad mesh in input. Everything is put together in an end-to-end robust block-structured mesher, soon to be available in the Gmsh software.



The best approaches from computer graphics for quadrilateral meshing (of which Bommes et al. (2013b) provide a survey) compute a *frame field*, which gives a local orientation and/or sizing of the quad elements. In a second step, a mesh is generated using guidance from the frame field. An issue shared by most existing works is that no quad mesh exactly follows the a priori computed frame field; this is due to it not being *integrable*. This limitation of field-based methods means that the mesh deviates from the frame field and the user must compromise on element quality, control over element size and orientation, and control over the topology of the mesh. A more fundamental issue arises when the frame field has a global topology that is not meshable, making it unusable. The **second contribution** of this thesis addresses this issue and proposes a solver for frame fields that are *integrable* on planar surfaces, providing guarantees on the subsequent meshability.

While quad meshing is a mature topic, all-hexahedral structured meshing is still a very open research problem. It is common knowledge in geometry that algorithms very rarely seamlessly extend from 2D to 3D. In fact, there exists no industrial-grade all-hex mesher, and engineers mostly rely on hex-dominant meshes that admit additional elements: prisms, pyramids, or tetrahedra. In computer graphics, efforts have been made to extend the frame field-based methods to 3D, but it turns out they suffer even more from the topological issues they have on surfaces. The best approach currently available (and arguably the best hex mesher to date, (Liu et al., 2023)) repairs a given 3D frame field in an algorithmic fashion to make it *locally meshable*, but lacks the global meshability property. The third and **last contribution** extends the 2D contribution (it was actually built with 3D in mind) to build the first-ever optimization-based solver that produces *globally meshable* 3D frame fields. By showing the hex meshes these fields generate, we consider this work to be an important first step towards an industrial-grade all-hex mesher.

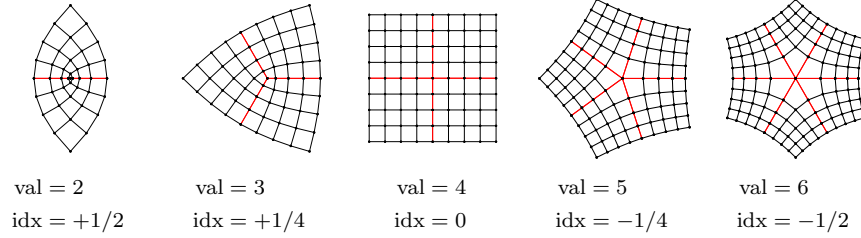


Figure 1.2: Quad mesh singularities of varying indices. Separatrices (in red) partition the domain into quadrilateral patches.

1.1 Meshes, parametrizations and frame fields

We start this chapter by reviewing the foundational concepts in frame field-driven meshing. First, we present the basic topological properties of quadrilateral and hexahedral meshes. We show how such meshes can be equivalently represented by a special parametrization: *integer-grid maps*. Relaxing the integer constraints brings us to *seamless parametrizations*, which can be seen as "infinitesimal meshes". We show how the Jacobian of a seamless parametrization induces an *integrable frame field*, which is continuous everywhere except at singularities. We formally define meshability and integrability of frame fields, and the conditions that guarantee these properties.

Topology of quadrilateral and hexahedral meshes

The topology of a mesh defines its structuredness, and has a large influence on the quality of its elements. It is also closely related to the topology of the parametrizations and frame fields used to generate them. It is thus crucial to have a good understanding of quad and hex mesh topology.

Quad mesh topology. A quad mesh partitions a 2D region of $\mathbb{R}^2$ or, more generally, a 2-manifold $\mathcal{M}$ embedded in $\mathbb{R}^3$, into quadrilateral elements Q , formed by edges E and vertices V . A vertex $v \in V$ is characterized by its *quad valence* $\text{val}(v)$, i.e., its number of incident quads. An interior vertex with valence 4 is *regular*, and otherwise *singular*; a boundary vertex with valence 2 is regular and otherwise singular. Deviation from regularity is measured by the fractional *index*

$$\text{idx}(v) = \begin{cases} \frac{4 - \text{val}(v)}{4} & \text{for interior } v, \\ \frac{2 - \text{val}(v)}{4} & \text{for boundary } v. \end{cases} \quad (1.1)$$

We detail later how mesh singularities relate to the singularities of the underlying parametrization and frame field. The notions of regular and singular vertices broadly classify a mesh as either *structured* or *unstructured*:

1.1. Meshes, parametrizations and frame fields

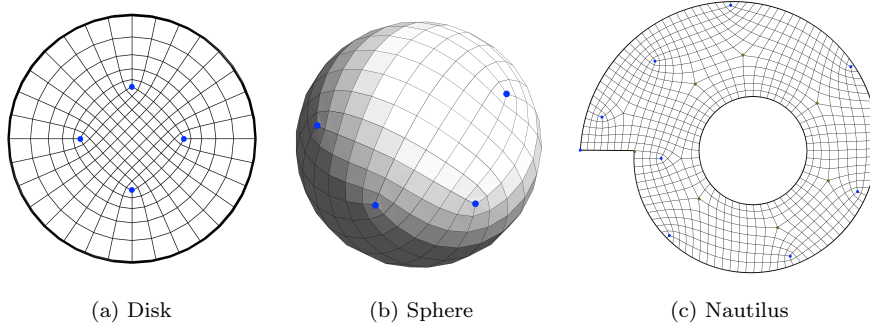


Figure 1.3: Examples of quad meshes on typical geometries. Valence 3 singularities are highlighted in blue, valence 5 singularities in green.

- *Structured* meshes have a grid topology almost everywhere, i.e., nearly all nodes are regular. Structuredness can therefore be roughly measured by the number of singular vertices in the mesh. A structured mesh can also be *block-structured*, meaning that it can be partitioned into coarse quadrilateral patches or hexahedral blocks. This kind of mesh can easily be converted into a high-order mesh, which is ideal for high-order methods. A structured mesh that is not particularly block-structured is sometimes called *quasi-structured*.
- *Unstructured* meshes do not have a grid-like topology, and are characterized by a large number of singular vertices. They are typically computed using indirect methods: for example, starting from a triangle mesh, and either splitting every triangle into three quads, or adequately combining adjacent pairs of triangles, as Remacle et al. (2012). Similar cutting or recombination methods have been developed to indirectly generate a hex mesh from a tetrahedral mesh.

As high quality elements should have angles close to 90° , one should aim at a low number of singular vertices, hence structured meshes are preferred and this thesis is focused on them.

Singularities are not just idiosyncrasies of quad meshes, but are actually strictly necessary on most domains, even in the absence of boundaries; for example, a sphere cannot be quad-meshed without singularities. This is explained by the *Poincaré-Hopf theorem* for quad meshes, stating that the sum of vertex indices is equal to the Euler characteristic of the surface $\mathcal{M}$:

$$\sum_{v \in V} \text{idx}(v) = \chi(\mathcal{M}), \quad (1.2)$$

where the Euler characteristic is linked to the surface's genus g and number of boundaries b by $\chi(\mathcal{M}) = 2 - 2g - b$. For example, a disk (fig. 1.3a) has genus 0 and 1 boundary, hence $\chi = 1$ and it is naturally meshed with 4 singularities of

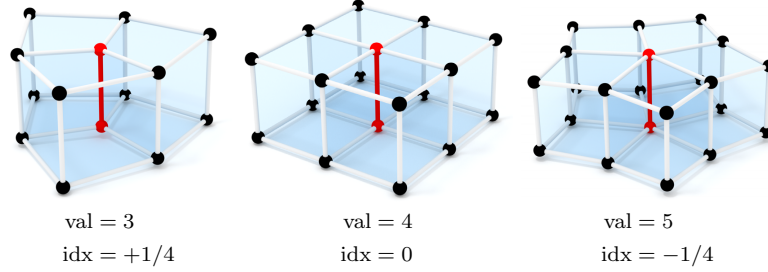


Figure 1.4: Hex mesh singular edges of varying indices, that can be seen as extrusions of quad mesh singularities (fig. 1.2). Figures from (Liu et al., 2018).

index $+1/4$. A sphere (fig. 1.3b) has genus 0 and 0 boundary, hence $\chi = 2$ and it is meshed with 8 singularities of index $+1/4$. There is an infinite number of possible singularity indices, but the ones with a small absolute value (i.e., $\pm 1/4$) are more desirable as they imply higher-quality quad elements. They are also sufficient for generating a mesh on any surface, and singularities of higher index are usually not desired. Note that boundary singularities also count towards the sum in eq. (1.2), hence a square can be meshed without any interior singularity, despite having a disk topology. It is important to note that, when asking if a quad mesh exists for a given singularity configuration, condition eq. (1.2) is necessary *but not sufficient*. For example, the (in)famous *nautilus* geometry (fig. 1.3c) has singularities of index $+1/4$ and $-1/4$ on its boundary and therefore satisfies the condition, but cannot be meshed without at least a pair of interior singularities.

Starting from an edge incident to a singular vertex, one can propagate forward a sequence of mesh edges until hitting another singularity or a boundary; this is called a *separatrix*. Tracing out all separatrices decomposes the mesh into quadrilateral patches, which form the so-called *base complex*. Some engineering applications look for meshes that are *block-structured*, i.e., having a coarse base complex, and some meshing algorithms are specialized in achieving a base complex as coarse as possible.

Hex mesh topology. A hexahedral mesh partitions a volumetric region $\mathcal{M} \subset \mathbb{R}^3$ into hexahedral elements H , formed by quadrilateral faces Q , edges E , and vertices V . Topological concepts introduced for quad meshes roughly extend to hex meshes by gaining an additional dimension. A singular vertex in a quad mesh can be extruded into a singular edge by extruding the incident quads into hexes. Formally, we can define the edge index through the *hexahedral valence* $\text{val}(e)$, i.e., its number of incident hexes:

$$\text{idx}(e) = \begin{cases} \frac{4 - \text{val}(e)}{4} & \text{for interior } e, \\ \frac{2 - \text{val}(e)}{4} & \text{for boundary } e. \end{cases} \quad (1.3)$$

1.1. Meshes, parametrizations and frame fields

A sequence of singular edges with identical index is called a *singular arc*. The set of singular edges, E_S , form a *singularity graph*, where singular arcs are either closed or start and end at *singular nodes* V_S , or the boundary. Singular nodes have a limited number of admissible local topologies, enumerated by Liu et al. (2018); for instance, a valence 3 singular arc cannot directly connect to a valence 5 singular arc. The same authors also find a Poincaré-Hopf-like formula for hex meshes:

$$\sum_{v \in \partial V_S} \frac{1}{2} \left(1 - \frac{\text{val}_h(v)}{4} \right) - \sum_{e \in \partial E_S^-} \text{idx}(e) + \sum_{v \in \overset{\circ}{V}_S} \frac{1}{2} \left(1 - \frac{\text{val}_h(v)}{4} \right) - \sum_{e \in \overset{\circ}{E}_S^-} \text{idx}(e) = 0, \quad (1.4)$$

where ∂V_S and $\overset{\circ}{V}_S$ are the boundary and interior nodes of the singularity graph, and ∂E_S^- and $\overset{\circ}{E}_S^-$ are the *non-closed* boundary and interior singular arcs. Like in the quad case, this condition is necessary but not sufficient for meshability.

Separatrices traced from singular edges are now *singular sheets* that decompose the mesh into a hexahedral base complex.

Integer-grid maps

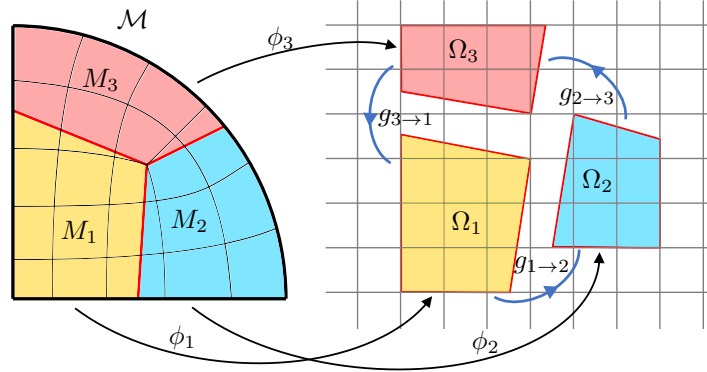


Figure 1.5: Domain $\mathcal{M}$ is mapped onto a parametric plane by an integer-grid map ϕ , allowing to map a valid quad mesh back onto $\mathcal{M}$. In this example the matchings are $r_{1 \rightarrow 2} = 0$, $r_{2 \rightarrow 3} = 1$ and $r_{3 \rightarrow 1} = 0$, such that $r_p = 1$ and the singularity index is $+1/4$.

A quad or hex mesh can be equivalently represented by an *integer-grid map* (IGM), a special kind of parametrization mapping from region $\mathcal{M}$ to the Cartesian space $\mathbb{R}^2$ or $\mathbb{R}^3$, such that each quad or hex corresponds to an "integer cell" of $\mathbb{R}^2$ or $\mathbb{R}^3$. We start by formally defining integer-grid maps for two-dimensional $\mathcal{M}$, then we extend the definition to 3D regions.

2D integer-grid maps. A 2-manifold is defined by an *atlas*, i.e., a collection of *charts* ϕ_i homeomorphically mapping a subset M_i of $\mathcal{M}$ to a subset Ω_i of

$\mathbb{R}^2$. This atlas already provides a parametrization of the surface, though an arbitrary choice of charts is not very useful and a set of conditions has to be satisfied for this atlas to be an IGM. First, the map must be *locally injective* to avoid any inverted element when pulling back the Cartesian grid onto $\mathcal{M}$. The 2×2 Jacobian matrix $\mathbf{J}_{\phi_i} = [\frac{\partial \phi}{\partial x}, \frac{\partial \phi}{\partial y}]$ must have positive determinant:

$$\det \mathbf{J}_{\phi}(p) > 0 \quad \forall p \in \mathcal{M}. \quad (\text{local injectivity})$$

Charts are connected by *transition functions*: if ϕ_i and ϕ_j are two charts, then on the intersection of their domains $M_i \cap M_j$ there exists another homeomorphism $g_{i \rightarrow j} : \phi_i(M_i \cap M_j) \rightarrow \phi_j(M_i \cap M_j)$ with $g_{i \rightarrow j} = \phi_j \circ \phi_i^{-1}$. In order for the mesh to seamlessly stitch between pieces M_i , the transition functions need to preserve the Cartesian grid, which yields the *conformity* condition

$$g_{i \rightarrow j}(z) = \mathbf{R}_{i \rightarrow j} z + \boldsymbol{\tau}_{i \rightarrow j}, \quad \boldsymbol{\tau}_{i \rightarrow j} \in \mathbb{Z}^2, \quad (\text{conformity})$$

where $\mathbf{R}_{i \rightarrow j}$ is any rotation multiple of $\pi/2$; this multiple is denoted $r_{i \rightarrow j}$ and usually called a *matching*. The use of charts allows the presence of interior singularities in the parametrization: consider a closed cycle in a neighborhood of point p , traversing charts $M_{i_1}, \dots, M_{i_n}$. One can measure the sum α_p of the *interior angles* in parameter space (i.e., the interior angles of $\Omega_{i_1}, \dots, \Omega_{i_n}$); point p is *singular* if $\alpha_p \neq 2\pi$, and the angle defect provides the singularity index $\frac{2\pi - \alpha_p}{2\pi}$. An alternative way to characterize singularities is to accumulate the matchings along the closed cycle: let $r_p = r_{1 \rightarrow 2} + r_{2 \rightarrow 3} + \dots + r_{n \rightarrow 1}$. Point p is singular if $r_p \neq 0$, and the singularity index is given by $r_p/4$. In order for a singularity to correspond to a vertex in the mesh, it must be mapped to an integer point in parameter space, giving a third condition

$$\phi(p) \in \mathbb{Z}^2 \quad \forall \text{ singular } p. \quad (\text{singularity placement})$$

Lastly, the mesh induced by the IGM must correctly stitch to the boundary of $\mathcal{M}$, or any other user-imposed feature, which we consider part of the boundary $\partial\mathcal{M}$ for simplicity. Boundary points must have at least one integer coordinate, so that the corresponding integer isoline is mapped to the mesh boundary:

$$\text{either } \phi^u(p) \in \mathbb{Z} \text{ or } \phi^v(p) \in \mathbb{Z} \quad \forall p \in \partial\mathcal{M}. \quad (\text{boundary alignment})$$

An atlas of charts $\{(M_i, \phi_i)\}$ where all maps ϕ_i and transitions $g_{i \rightarrow j}$ satisfy (local injectivity), (conformity), (singularity placement) and (boundary alignment) is called an *integer-grid map*, and pulls back the Cartesian grid through ϕ^{-1} to a boundary-aligned quad mesh with preserved topology. Integer-grid maps are sometimes also called *integer seamless* or *grid-preserving* parametrizations.

3D integer-grid maps. Integer-grid maps naturally extend from 2D to 3D, mapping subsets M_i of $\mathcal{M}$ to subsets Ω_i of $\mathbb{R}^3$ in a way that pulls back the integer isosurfaces of $\mathbb{R}^3$ onto a hex mesh. The local injectivity requirement eq. (local injectivity) still holds, and conformity extends to the group Oct of the 24

1.1. Meshes, parametrizations and frame fields

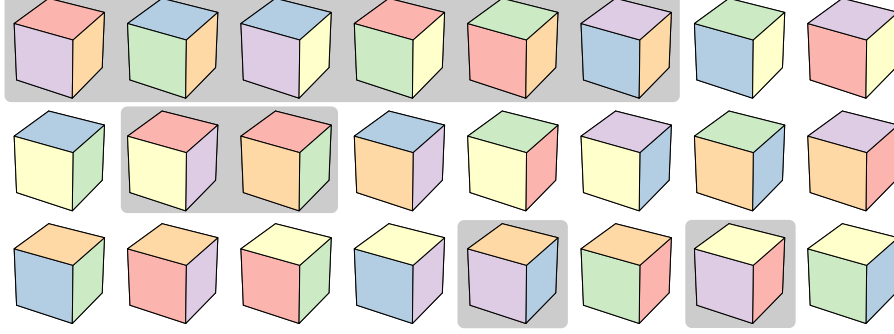


Figure 1.6: Group Oct of the 24 orientation-preserving octahedral symmetries, here represented with a cube which is dual to the octahedron. The 10 elements highlighted in gray are the ones that, starting from the top-left cube, rotate only around one axis. These rotations are the only ones allowed in a *meshable* seamless parametrization.

orientation-preserving octahedral symmetries, illustrated in fig. 1.6:

$$g_{i \rightarrow j}(z) = \mathbf{R}_{i \rightarrow j} z + \boldsymbol{\tau}_{i \rightarrow j}, \quad \mathbf{R}_{i \rightarrow j} \in \text{Oct}, \quad \boldsymbol{\tau}_{i \rightarrow j} \in \mathbb{Z}^3. \quad (\text{conformity})$$

The holonomy of a cycle around a point p is defined in the same way as before. Now singularities can have one of their 3 parameters non-integer, such that they lie on an edge of the integer grid; let us call this space of edges G_e :

$$\phi(p) \in G_e \quad \forall \text{ singular } p. \quad (\text{singularity placement})$$

Similarly, let G_f denote the space of grid faces, having at least one of their coordinates integer. Boundary alignment is guaranteed by assuring that the boundary and features are mapped to G_f :

$$\phi(p) \in G_f \quad \forall p \in \partial \mathcal{M}. \quad (\text{boundary alignment})$$

An atlas of charts satisfying the four new conditions is a 3D integer-grid map.

Seamless parametrizations

We can further relax the mesh representation by removing all integer constraints. The resulting mapping, called a *seamless parametrization* (fig. 1.7), can be seen as an *infinitesimal* quad or hex mesh, since locally one can always find a quantization of the map that yields a valid quad/hex mesh. We formally present the seamless parametrization constraints only in 3D; the corresponding 2D equations naturally follow.

Across charts, transition functions are still related by an octahedral permutation, so that the parametrization isosurfaces connect seamlessly. The translation term on the other hand is completely free:

$$g_{i \rightarrow j}(z) = \mathbf{R}_{i \rightarrow j} z + \boldsymbol{\tau}_{i \rightarrow j}, \quad \mathbf{R}_{i \rightarrow j} \in \text{Oct}, \quad \boldsymbol{\tau}_{i \rightarrow j} \in \mathbb{R}^3. \quad (1.5)$$

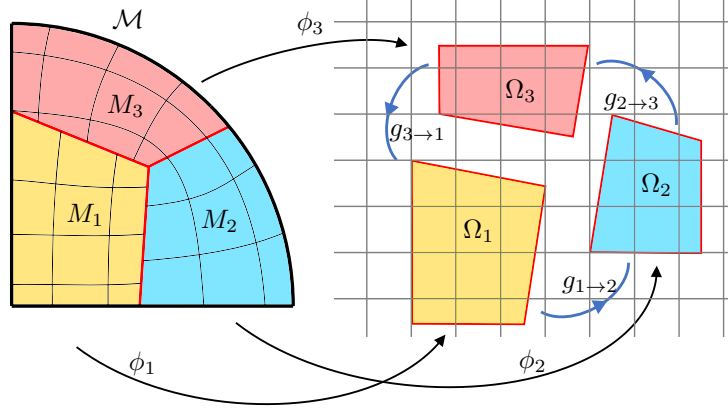


Figure 1.7: Domain $\mathcal{M}$ is mapped onto a parametric plane by a seamless parametrization ϕ , allowing to map a grid back onto $\mathcal{M}$. Note that the singularity is not mapped to an integer location, and the boundaries are not mapped to the grid.

This relaxed conformity condition can be equivalently expressed through the parametrization Jacobian $\mathbf{J}_\phi$. On the chart boundary we have that $\phi_j = g_{i \rightarrow j} \circ \phi_i$, and differentiating on both sides yields

$$\mathbf{J}_{\phi_j}(p) = \mathbf{R}_{i \rightarrow j} \mathbf{J}_{\phi_i}(p), \quad p \in M_i \cap M_j, \quad \mathbf{R}_{i \rightarrow j} \in \text{Oct} \quad (1.6)$$

(note that $\mathbf{R}_{i \rightarrow j}$ acts on the rows of $\mathbf{J}_\phi$ and not on its columns, meaning that the gradient vectors are not actually rotated, but rather the different gradients are permuted). Singularities can now lie at any position in the parameter space. As for boundaries, one of the parameter isolines should coincide with the boundary, meaning that one of the parameters must be constant along the boundary. This is easily imposed by ensuring that at least one of the parametrization gradients is orthogonal to the boundary; if $\mathbf{e}_t$ is the tangent vector at $\partial\mathcal{M}$ then

$$\nabla\phi^u \cdot \mathbf{e}_t = 0 \text{ or } \nabla\phi^v \cdot \mathbf{e}_t = 0 \text{ or } \nabla\phi^w \cdot \mathbf{e}_t = 0. \quad (1.7)$$

Parametrization-induced frame fields

The conditions defining a seamless parametrization (local injectivity, conformity and boundary alignment) can all be expressed as conditions on the parametrization Jacobian $\mathbf{J}_\phi$. The latter is discontinuous across charts where it undergoes a rotation of the octahedral group Oct. To remove this discontinuity, we can define the equivalence class of $\mathbf{J}_\phi$ under this group:

$$[\mathbf{J}_\phi] := \{\mathbf{R}\mathbf{J}_\phi \mid \mathbf{R} \in \text{Oct}\}, \quad (1.8)$$

which we call a *frame*. The corresponding *frame field* $[\mathbf{J}_\phi](p)$ is then continuous on the whole parametrization, except at singularities where it is undefined

1.1. Meshes, parametrizations and frame fields

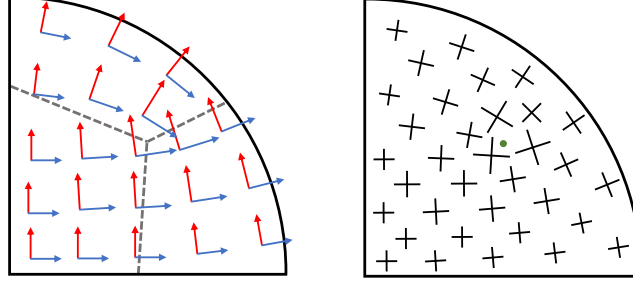


Figure 1.8: Left: Jacobian of the parametrization of fig. 1.7 ($\nabla\phi^u$ in blue, $\nabla\phi^v$ in red); notice how the field is discontinuous across the $2 \rightarrow 3$ chart transition, and the amplitude grows close to the singularity as the integer isolines get closer. Right: induced frame field, which is continuous everywhere except at the singularity. Note that the fields need not be orthogonal nor isotropic.

(fig. 1.8). This observation opens the way for methods computing parametrizations through continuous frame fields, which do not require integer constraints or cutting the domain into charts. However, for a frame field to be truly equivalent to a seamless parametrization, a parametrization must exist of which the frame field is the Jacobian: it must be *integrable*. Before formally stating the condition for integrable frame fields, we review some basic concepts of vector field topology and calculus, which will allow us to formally define the different ways in which a frame field is *meshable*.

Frame field meshability and integrability

The notion of meshability of 2D and 3D frame fields is discussed in depth by Liu et al. (2023); we recall some of the main results and refer the reader to the paper for more details. First of all, we make the general assumption that at each non-singular point, the local injectivity condition is satisfied, meaning that all frame vectors are non-zero and linearly independent. This prevents any degenerate mesh elements with zero volume.

We define three frame field properties: *meshability*, *local meshability*, and *integrability*.

- A frame field is **(globally) meshable** if a quad or hex mesh exists with identical topology (i.e., singularity configuration).
- A frame field is **locally meshable** if at each point of the domain there is an arbitrarily small neighborhood that is quad- or hex-meshable.
- A frame field is **integrable** if a seamless parametrization exists that induces the frame field.

By definition, global meshability implies local meshability. An integrable frame field admits a seamless map of identical topology, however, *seamless maps are*

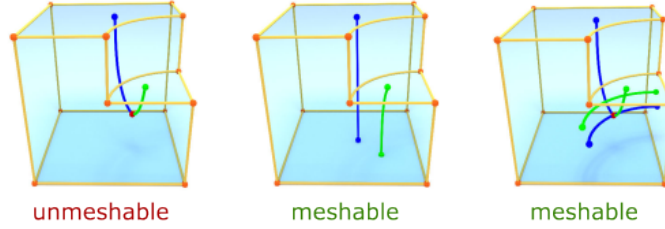


Figure 1.9: Locally unmeshable 3D frame field due to the 3-5 singular node, and meshable alternatives. Figures from (Liu et al., 2023).

not always meshable. An additional requirement for the seamless map (and the frame field) is that around singular arcs, it only rotates around the axis parallel to the arc; other rotations, such as around a diagonal of the cube, are forbidden. This is the *meshable holonomies* condition. The admissible rotations are highlighted in fig. 1.6. If this condition is satisfied, along with the local injectivity and integrability, then the corresponding seamless map can always be quantized to a (possibly very fine) mesh of identical topology. To summarize,

$$\begin{array}{c} \text{integrability} + \\ \text{meshable holonomies} \end{array} \Rightarrow \text{global meshability} \Rightarrow \text{local meshability} \quad (1.9)$$

Besides guaranteeing meshability, integrable frame fields have the property that the corresponding seamless parametrization, and any sufficiently fine mesh quantized from it, *exactly follows the frame field* in terms of element orientation, sizing, anisotropy or non-orthogonality. This is a crucial advantage for designing meshes supporting numerical simulations, as users often wish to specify element sizing fields or large anisotropy.

We now review the set of conditions that guarantee each of the defined frame field properties.

Meshability. A 2D frame field is locally meshable under a mild set of conditions (isolated singularities and quad sectors only; see (Liu et al., 2023)). The *nautilus* model without interior singularities is locally meshable but not globally, because of the presence of a *limit cycle*, i.e., a streamline converging to a closed orbit, in this case the interior boundary. A locally meshable frame field *in the plane* is globally meshable if and only if it has no limit cycles. This condition is not sufficient however for surfaces embedded in $\mathbb{R}^3$, where additional degeneracies exist, e.g., a pair of 3-5 singularities on a torus.

For 3D frame fields, singular arcs can admit several defects that make it non-locally meshable: the frame field can be misaligned with its singular arc, or it can rotate around an axis other than the one tangential to the singular arc. Singular arcs can also connect in a way that is not locally meshable, as in

1.1. Meshes, parametrizations and frame fields

the well-known 3-5 singular node (fig. 1.9, left). The fact that 3D frame fields are not systematically locally meshable is what makes the hex meshing task so difficult. Global meshability is even less understood, as today there is no known set of sufficient conditions for a 3D frame field to be meshable.

Vector calculus. Before moving on to integrability, we recall some basic properties of vector calculus. The fundamental theorem of vector calculus states that a well-behaved vector field $\mathbf{v}$ (2D or 3D) can be decomposed into the sum of a *diverging* vector field $\mathbf{v}_d$ and a *solenoidal* vector field $\mathbf{v}_s$:

$$\mathbf{v} = \mathbf{v}_d + \mathbf{v}_s. \quad (1.10)$$

This is also called the *Helmholtz decomposition*. The diverging vector field $\mathbf{v}_d$ is curl-free, or irrotational, and the solenoidal vector field $\mathbf{v}_s$ is divergence-free:

$$\nabla \times \mathbf{v}_d = \mathbf{0}, \quad \nabla \cdot \mathbf{v}_s = 0. \quad (1.11)$$

On a simply-connected domain, $\mathbf{v}_d$ derives from a scalar potential ϕ , and $\mathbf{v}_s$ derives from a vector potential $\mathbf{a}$, through the identities

$$\mathbf{v}_d = \nabla \phi, \quad \mathbf{v}_s = \nabla \times \mathbf{a} \quad \text{on a simply-connected domain.} \quad (1.12)$$

Notice how plugging these identities into eq. (1.11) reveals the orthogonal nature of the curl and div operators: $\nabla \times \nabla \phi = \mathbf{0}$ and $\nabla \cdot (\nabla \times \mathbf{a}) = 0$. For this reason the diverging and solenoidal vector fields are sometimes called *longitudinal* and *transverse*, respectively.

Moving to general (non-simply-connected) domains, a vector field can be curl-free and divergence-free, and still be nontrivial; it is then called a *harmonic vector field*. Let $\mathbf{v}_h$ denote such a vector field; the term *harmonic* can be explained by its behavior on a simply-connected neighborhood, where its Helmholtz decomposition is given by

$$\mathbf{v}_h = \nabla \phi + \nabla \times \mathbf{a}. \quad (1.13)$$

Stating that $\mathbf{v}_h$ is both divergence- and curl-free we find that

$$\begin{aligned} 0 &= \nabla \cdot \mathbf{v}_h = \nabla^2 \phi, \\ 0 &= \nabla \times \mathbf{v}_h = \nabla \times (\nabla \times \mathbf{a}) = \nabla^2 \mathbf{a}. \end{aligned} \quad (1.14)$$

In other words, locally, $\mathbf{v}_h$ derives from a scalar potential ϕ and a vector potential $\mathbf{a}$ that are solutions to Laplace's equation. This is what makes $\mathbf{v}_h$ *harmonic*. Having defined harmonic vector fields allow us to define the *Hodge decomposition*, which is valid for any vector field $\mathbf{v}$ on general domains:

$$\mathbf{v} = \nabla \phi + \mathbf{v}_h + \nabla \times \mathbf{a}. \quad (1.15)$$

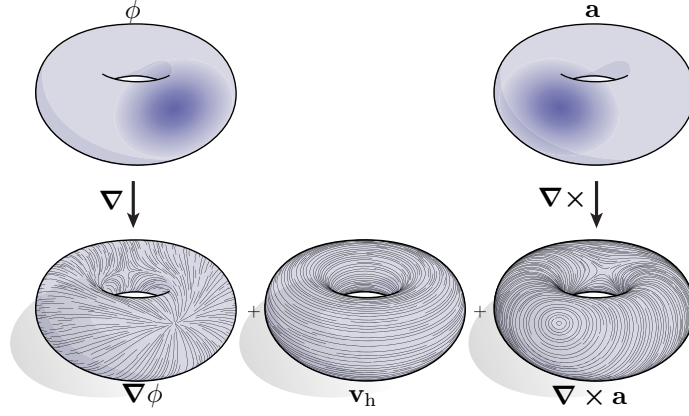


Figure 1.10: Hodge decomposition of a vector field on a torus. Figures from (Crane, 2023).

Integrability. We have seen that a curl-free vector field derives from a scalar potential on any simply-connected domain. Let p be a nonsingular point; a *lifting* of frame field $\mathbf{F}$ in a small neighborhood of p is a set of 3 continuous vector fields $\mathbf{u}, \mathbf{v}, \mathbf{w}$, each corresponding to one of the branches of $\mathbf{F}$. In other words, a lifting is a local choice of branches in the frame field. Frame field $\mathbf{F}$ is curl-free at p if any lifting $\mathbf{u}, \mathbf{v}, \mathbf{w}$ in a neighborhood $\mathcal{N}(p)$ is curl-free:

$$\nabla \times \mathbf{u}(p') = \nabla \times \mathbf{v}(p') = \nabla \times \mathbf{w}(p') = \mathbf{0}, \quad \forall p' \in \mathcal{N}(p). \quad (1.16)$$

If $\mathbf{F}$ is curl-free, it is thus the Jacobian of a parametrization ϕ_i on any simply-connected chart M_i , and, stitching the charts together through the octahedral permutations of the frame representation, $\mathbf{F}$ is the Jacobian of a global seamless parametrization on the whole domain. In conclusion, a locally injective, boundary-aligned frame field is *integrable* if and only if it is *curl-free* at each non-singular point. It is then equivalent to a seamless parametrization ϕ up to arbitrary translations in each chart.

Parametrization gradient vs. coordinate vectors

Frame fields are traditionally used to encode the gradients of a seamless parametrization ϕ . *Coordinate vectors* are an alternative representation that is equivalent to the parametrization gradient, used in some works, e.g., (Sageman-Furnas et al., 2019). Consider the inverse of the parametrization $\mathbf{r}$ mapping a pair of coordinates (u, v) in parameter space to a point $\mathbf{p}$ on the domain: $\mathbf{p} = \mathbf{r}(u, v)$. This map can only be defined locally since the parametrization can map different physical points to the same coordinates. The Jacobian matrix of $\mathbf{r}$ defines the *coordinate vectors* $\mathbf{e}^u$ and $\mathbf{e}^v$:

$$\mathbf{J}_{\mathbf{r}} = \begin{pmatrix} \frac{\partial \mathbf{r}}{\partial u} & \frac{\partial \mathbf{r}}{\partial v} \end{pmatrix} = (\mathbf{e}^u \quad \mathbf{e}^v), \quad (1.17)$$

1.1. Meshes, parametrizations and frame fields

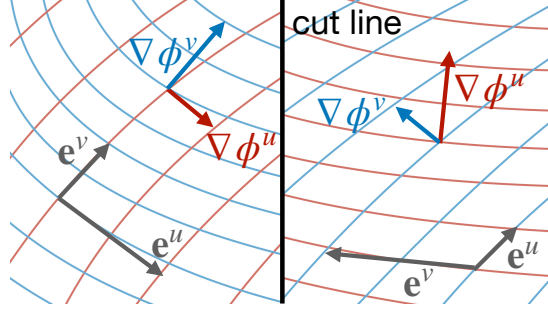


Figure 1.11: A seamless parametrization across a cut. The isolines of the coordinate maps ϕ^u and ϕ^v are depicted in red and blue, respectively. Notice how the gradients $\nabla\phi^u, \nabla\phi^v$ and the frame vectors $\mathbf{e}^u, \mathbf{e}^v$ undergo a 90° rotation as they cross the cut.

forming a *coordinate frame* $(\mathbf{e}^u \ \mathbf{e}^v)$. Since $\mathbf{r}$ is the inverse of the parametrization, their Jacobians are inverse of one another, meaning that the coordinate frame is the inverse of the parametrization's Jacobian:

$$\mathbf{J}_\phi = \mathbf{J}_{\mathbf{r}^{-1}} = (\mathbf{J}_{\mathbf{r}})^{-1} \quad \Rightarrow \quad \begin{pmatrix} \nabla\phi^u \\ \nabla\phi^v \end{pmatrix} = (\mathbf{e}^u \ \mathbf{e}^v)^{-1}. \quad (1.18)$$

Note that, inspecting the off-diagonal terms in $\mathbf{J}_\phi \mathbf{J}_{\mathbf{r}} = \mathbf{I}$, we have $\nabla\phi^u \cdot \mathbf{e}^v = \nabla\phi^v \cdot \mathbf{e}^u = 0$, meaning that the coordinate vectors $\mathbf{e}^u, \mathbf{e}^v$ are orthogonal to the gradients of ϕ^v, ϕ^u , respectively, and therefore parallel to the isolines of ϕ^v, ϕ^u respectively. This shows how the frame field corresponds to a quad mesh through discrete isolines of the coordinate map ϕ^u, ϕ^v . Looking at the diagonal terms, we have $\nabla\phi^u \cdot \mathbf{e}^u = \nabla\phi^v \cdot \mathbf{e}^v = 1$, which indicates that the integer isolines are spaced out according to $\|\mathbf{e}^u\|$ and $\|\mathbf{e}^v\|$. We illustrate the introduced concepts and notations on fig. 1.5.

Orthogonal case. In this work we are interested in frames that are orthogonal: $\mathbf{e}^u \perp \mathbf{e}^v$. Let $\hat{\mathbf{e}}^u = \mathbf{e}^u / \|\mathbf{e}^u\|$ and $\hat{\mathbf{e}}^v = \mathbf{e}^v / \|\mathbf{e}^v\|$; the frame matrix is then diagonal in the local orthonormal basis $(\hat{\mathbf{e}}^u, \hat{\mathbf{e}}^v)$:

$$\begin{pmatrix} \nabla\phi^u \\ \nabla\phi^v \end{pmatrix} = \begin{pmatrix} \|\mathbf{e}^u\| & 0 \\ 0 & \|\mathbf{e}^v\| \end{pmatrix}^{-1} = \begin{pmatrix} 1/\|\mathbf{e}^u\| & 0 \\ 0 & 1/\|\mathbf{e}^v\| \end{pmatrix}. \quad (1.19)$$

The Jacobian of the coordinate map can thus be computed by simply scaling the frame's coordinate vectors: $\nabla\phi^u = \mathbf{e}^u / \|\mathbf{e}^u\|^2$ and $\nabla\phi^v = \mathbf{e}^v / \|\mathbf{e}^v\|^2$.

Integrability for coordinate vectors. A criterion equivalent to the curl-free condition, eq. (1.16), can be obtained for a frame field $(\mathbf{u}, \mathbf{v})$ to be the coordinate vectors of a seamless parametrization: its *Lie bracket* must be zero,

$$[\mathbf{u}, \mathbf{v}] = \nabla_{\mathbf{u}} \mathbf{v} - \nabla_{\mathbf{v}} \mathbf{u} = \mathbf{0}, \quad (1.20)$$

where $\nabla_{\mathbf{u}}$ and $\nabla_{\mathbf{v}}$ are directional derivatives. We refer to (Sageman-Furnas et al., 2019) for the proof of this relationship between the curl-free property and the Lie bracket. Note that all the preceding notions on coordinate vectors extend to 3D frames $(\mathbf{u}, \mathbf{v}, \mathbf{w})$; in particular these are the coordinate vectors of a seamless parametrization if all three pairs of Lie brackets are zero,

$$[\mathbf{u}, \mathbf{v}] = [\mathbf{v}, \mathbf{w}] = [\mathbf{w}, \mathbf{u}] = \mathbf{0}. \quad (1.21)$$

1.2 The field-driven meshing pipeline

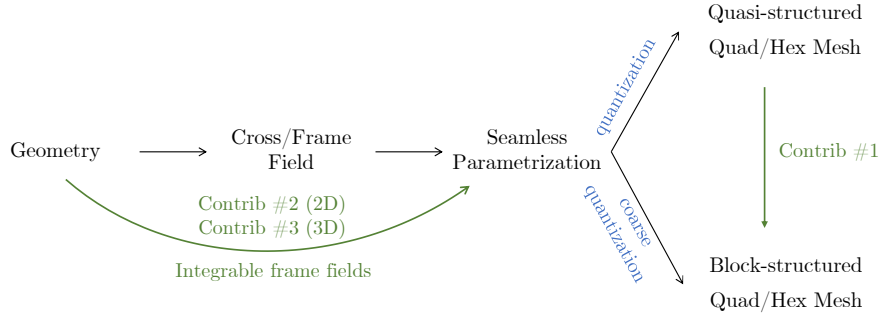


Figure 1.12: A typical structured quadrilateral or hexahedral meshing pipeline.

In the previous section, we have described a sequence of relaxations of quadrilateral and hexahedral meshes: integer-grid maps, seamless parametrizations, frame fields and cross fields. This suggests a *field-driven* meshing pipeline that decomposes the problem into a sequence of steps, each computing a representation that gradually gets closer to a mesh with the desired properties. The majority of current state-of-the-art meshers fit in this pipeline, illustrated in fig. 1.12:

1. A *geometry* is given in input representing the domain to be meshed. This is usually some representation of a surface, which can be manifold or nonmanifold, and closed if a hex mesh is to be computed. Various geometrical representations exist; in this work we assume a watertight CAD representation. Usually the user provides a set of *feature curves*, which are forced to match a sequence of mesh edges. On top of that the user can provide *sizing requirements* for the mesh elements, which can be represented by a size field, or a prescribed number of elements on certain feature curves.
2. A *direction field* is computed indicating a local orientation for the quad or hex elements. This direction field can either be a cross field which has no size, or a scaled frame field (sometimes a two-step approach is used: a

1.3. Related work

cross field is computed and then appropriately scaled). The direction field is required to satisfy the prescribed directional constraints (boundaries and feature curves) and should be as smooth as possible. In computer graphics applications, alignment to the principal curvature directions is often promoted, and the direction field is smoothly interpolated in flat regions.

3. The direction field is used as a guide to build a *seamless parametrization*, mapping the geometric domain to coordinates (u, v) or (u, v, w) . Since the direction field is generally not integrable, a parametrization is computed that aligns best to the prescribed directions in a least-squares sense.
4. A *quad or hex mesh* is extracted from the parametrization by taking iso-lines of u, v, w . The complexity of this step depends on how close the parametrization is to an integer-grid map, and usually requires quantization methods. Some formulations additionally promote the coarseness of the quadrilateral or hexahedral layout to make the mesh block-structured. Alternatively, the parametrization step can be skipped by using the direction field to guide a frontal mesher, which inserts nodes in an advancing front. Then, typically, nodes are connected to form a triangle mesh, which can be recombined into a quad mesh.

1.3 Related work

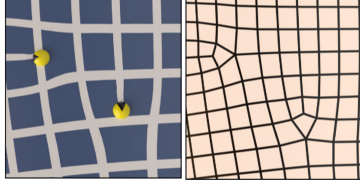
In this section we review the most important research works that are related to this thesis, focusing on step $(1 \rightarrow 2)$ and $(2 \rightarrow 3)$ of the field-driven meshing pipeline. For broader surveys on mesh generation we refer to (Bommes et al., 2013b) for quad meshing, (Pietroni et al., 2022) for hex meshing, and (Campen, 2017) for block-structured quad meshing.

Computing parametrizations from direction fields

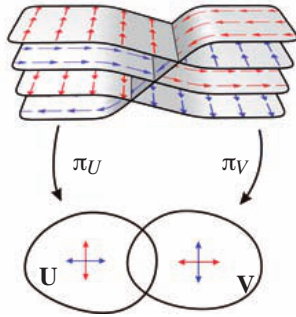
We start this literature review by looking at step

direction field $\rightarrow$ seamless parametrization

of the pipeline. Historically, meshing was not the primary motivation for computing parametrizations on surfaces. The computer graphics community has been interested in parametrizations as a versatile surface representation, with many applications including texture mapping and digital geometry processing. Such parametrizations are typically required to be smooth and aligned with the principal curvature directions. We start this literature review by looking at the most prominent methods for computing a seamless parametrization from a direction field.



Ray et al. (2006) are some of the pioneers in computing parametrizations guided by a pair of input direction fields. They propose a *periodic* representation where the parameter lines correspond to 2π multiples of angle fields (θ, ϕ) . Computations are then done on the complex exponentials $e^{i\theta}$ and $e^{i\phi}$ which are kept close to unit module through a penalty term. The strength of this representation is that the resulting parametrization is implicitly integer. Since fields θ and ϕ are not explicitly coupled, special singularities appear that can be meshed by inserting 3-5 pairs; those singularities were later called *position singularities* by Jakob et al. (2015) (see inset). The subsequent quad mesh has a low distortion, but contains a large number of position singularities.



In *QuadCover*, Kälberer et al. (2007) take as input a frame field and view it as a vector field on a *covering* of the manifold, i.e., a stack of 4 layers, each corresponding to one of the frame vectors (see inset from the same paper). Frame field singularities are *branch points* in the covering, meaning that one changes layer when going around the branch point. This branched covering allows to map a frame field to a continuous vector field, without having to define a frame equivalence class as in eq. (1.8). Finding a seamless parametrization then amounts to finding the gradient field $\nabla\phi$ that is closest to the input vector field. Note that the branch points are fixed to the input frame field singularities, which fixes the topology of the resulting parametrization. The same authors have extended this framework to the 3D case in *CubeCover* (Nieser et al., 2011), where this time a frame field can be mapped to a vector field on a 6-sheeted covering, each sheet corresponding to a frame vector. As in 2D, this covering is not explicitly computed; instead, the topology is fixed by the frame field and a 3D parametrization is computed on a tet mesh, with the linear constraint that it must be seamless across tets.

Bommes et al. (2009) do not work with a covering but instead cut open the triangle mesh to a disk topology, with frame field singularities lying on the disk boundary. This cutting allows to define a global continuous parametrization $\phi = (u, v)$, which is formulated as a least-squares problem with linear conformity constraints across cuts. Local injectivity of the parametrization however is not guaranteed, but a local stiffening scheme strives to prevent inversions. This framework was later extended in (Bommes et al., 2013a) to guarantee local injectivity.

1.3. Related work

Computing 2D guiding cross/frame fields

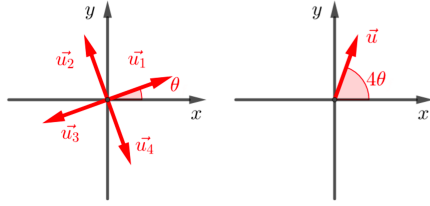
We now turn our attention to the step (geometry) $\rightarrow$ (direction field), which is the most related to our work.

Curvature alignment. When computing parametrizations or quad meshes on surfaces, it is natural to want them to align to the principal curvature directions of the surface. It is less clear what the alignment should be when the curvature is ill-defined, e.g., in flat or umbilical (sphere-like) regions. Many works approach this by striving for a smooth direction field in these regions. Kälberer et al. (2007) blend a smoothing energy E_{smooth} with an alignment energy E_{align} that promotes curvature alignment when it is well-defined, i.e., when the principal curvature values κ_1, κ_2 are distinct. Bommers et al. (2009) approach this differently by first defining a sparse set of salient curvature directions (again based on the anisotropy of the principal curvatures) and then smoothly interpolating these constraints.

The $\cos(4\theta)$ energy. The best-known way to encode a cross field (i.e., a frame field without any sizes) is to define, on each triangle of the input mesh, a local reference frame, and an angle θ_i representing the rotation of a vector $\mathbf{v}_i$ with respect to the reference frame. The cross then is defined by the $\pi/2$ rotations of $\mathbf{v}_i$ on the triangle: $(\mathbf{R}(k\pi/2)\mathbf{v}_i)_{k=0,1,2,3}$. To compare neighboring crosses, Hertzmann et al. (2000) were the first to propose the $\cos(4\theta)$ trick, by defining the smoothness energy between two triangles i and j as $E_{ij} = \cos(4(\theta_i - \theta_j + \alpha_{ij}))$, with α_{ij} the angle formed by neighboring reference frames. This energy is invariant to the cross symmetries since adding an integer multiple of $\pi/2$ to θ_i or θ_j does not alter E_{ij} . Energy minimization is then a non-linear problem, and typically solved using a quasi-Newton method; this is also how Kälberer et al. (2007) proceed. However, this kind of approach is known to suffer from local minima and strong dependence on the initial solution.

Period jumps. Ray et al. (2008) propose a cross field representation that encodes angles θ_i on every triangle, as well as integers p_{ij} , called *period jumps*, that match crosses in neighboring triangles, telling how each cross branch "jumps" across the edge. Determining the singularity index of a given vertex then amounts to going around the vertex and accumulating period jumps. They propose an algorithm to compute a global cross field in this representation, given an imposed singularity configuration. Bommers et al. (2009) encode the period jumps as integer variables, allowing to compute a cross field with automatic placement of singularities, through a mixed-integer optimization problem; note that this time the continuous part (on the angles θ_i) is linear as the angles can be directly compared: $E_{ij} = (\theta_i - \theta_j + \kappa_{ij} + \frac{\pi}{2}p_{ij})^2$.

Implicit cross representations. To get rid of integer variables, some works try to represent crosses in an *implicit way*, i.e., through a mathematical object that encodes the symmetries of the cross. A popular option to encode a cross rotated by angle θ (see inset from (Jezdimirović et al., 2020)) is the pair $\mathbf{u} = (\cos(4\theta), \sin(4\theta))$, which was pioneered by Palacios et al. (2007) and Ray et al. (2009); alternatively, it can be interpreted as a complex number $z = e^{4i\theta}$.



This representation introduces a non-convex constraint: $u_1^2 + u_2^2 = 1$; this poses a challenge as the Dirichlet energy of cross field singularities is then unbounded. Knöppel et al. (2013) relax this constraint by imposing only a global norm on the cross field. Beaufort et al. (2017) and Viertel et al. (2019) instead formulate the pointwise constraint as a penalty term

$\frac{1}{4\epsilon^2} \int_{\Omega} (1 - u_1^2 - u_2^2)^2 d\Omega$. This enables a connection to the *Ginzburg-Landau functional*, of which the asymptotic behavior when $\epsilon \rightarrow 0$ is well characterized: there are few singularities of low index, and they are well separated from each other and from the boundaries.

Frame fields. Later works have looked into computing frame fields that are better suited to extract a mesh, respecting additional constraints of the user such as sizing or non-orthogonality. Panozzo et al. (2014) offer an interesting approach that leverages the mature cross field-guided meshers to generate quad meshes that, following user-prescribed constraints, are non-uniform, anisotropic, or non-orthogonal. Their pipeline goes as follows:

1. A smooth cross field is computed interpolating prescribed orientation constraints.
2. The cross field is smoothly stretched and sheared into a *frame field* to match sizing, anisotropy and non-orthogonal constraints. This transformation is encoded as an SPD matrix field.
3. The surface is deformed according to the metric induced by the frame field, so that the frame field is as close as possible to a cross field on the deformed surface.
4. A smooth cross field is computed on the deformed surface and a uniform quad mesh is extracted.
5. The quad mesh is transformed back onto the initial surface, giving a non-uniform quad mesh matching the given constraints.

The main advantage of this approach is that the cross field computed on the deformed surface will possess appropriate singularities to do the necessary size transitions in the quad mesh, thanks to the curvature introduced by the surface deformation. This makes the method significantly better at managing size constraints than the traditional cross field-based approaches. An intrinsic alternative is proposed by Jiang et al. (2015), who customize a Riemannian

1.3. Related work

metric from the constraints, and compute a smooth cross field in the metric; the advantage is that no surface deformation is necessary. This approach has been extended to 3D frame fields by Fang et al. (2021). However, for each of these methods, the price to pay is that the metric is decoupled from the field orientation, meaning that a corresponding mesh does not necessarily exist.

Diamanti et al. (2014) introduce *PolyVectors* which encode a set of vectors as the complex roots of a complex polynomial. In particular, they use it to represent a (non-orthogonal and anisotropic) frame, by encoding their vectors $\pm u, \pm v$ as the polynomial $p(z) = (z^2 - u^2)(z^2 - v^2)$. A Dirichlet energy is then formulated using the coefficients of polynomial p , and a projection ensures that the frames respect a minimum angle constraint to avoid degeneracies. In (Diamanti et al., 2015) they extend their work to compute *integrable* PolyVector fields. To make the (per-face defined) field curl-free, they match vectors in adjacent faces f, g and impose that the vector flow across the common edge e is the same for both vectors: $\langle \alpha_f, e \rangle = \langle \alpha_g, e \rangle$. The variables of the optimization are two successive vectors (α_f, β_f) of each face f ; hence, they need to guarantee that the ordering of the vectors is preserved across neighboring faces. This requires a special energy term called *PolyQuotient*.

Relation to our work. Integrable frame fields can be considered equivalent to seamless parametrizations, up to a rigid rotation of the coordinate maps. Being able to compute them reliably then allows to perform the two first steps of the meshing pipeline in one. Our contribution on 2D integrable frame fields was first intended as an intermediate step before treating the 3D case, but we believe it is a valuable alternative to the integrable PolyVector fields. The frame representation is very different, though, as we use 2D odeco tensors which can be seen as an extension of the $(\cos(4\theta), \sin(4\theta))$ cross representation. It is implicitly orthogonal, which is both a limitation (good parametrizations are not always orthogonal) and a strength, as it guarantees local injectivity and we do not have to ensure order preservation of the frame matchings. The principal strength compared to PolyVectors is that our odeco frames extend to 3D, as does our integrability formulation.

Computing 3D guiding cross/frame fields

Smooth 3D cross fields. Motivated by the effectiveness of cross fields to generate quadrilateral meshes, researchers have extended the smooth cross field approaches to 3D. Huang et al. (2011) can be considered to be a pioneering work in 3D cross field computations. They are the first to propose encoding a 3D rotation in a polynomial defined on the sphere, which takes into account all required symmetries. This polynomial is then decomposed in a basis of spherical harmonics, which allows to define a Dirichlet energy. The optimization variables are the ZYZ Euler angles (α, β, γ) on each node, and the Dirichlet energy is optimized using an L-BFGS solver, coupled with boundary alignment constraints. The singularity graphs produced by such a frame field smoothing

method typically possess many defects that lead to a degenerate parametrization. Jiang et al. (2014) give a procedure to clean the singularity graph and make it amenable for computing a valid parametrization. Ray et al. (2016) extend this work by proposing a better initialization obtained by smoothing the spherical harmonics coefficients, and projecting them afterwards with a gradient descent procedure. They also formulate the boundary alignment constraints directly in terms of the spherical harmonics. Solomon et al. (2017) use the same representation, coining them as *octahedral fields* in reference to their symmetries. They propose to use the boundary element method (BEM) to sample a cross field at any point in the volume, given boundary alignment specifications. Chemin et al. (2019) connect the usual spherical harmonics representation with a special subspace of *fourth-order tensors*, by noticing that homogeneous polynomials are equivalent to symmetric tensors. This allows to implicitly represent a cross as a tensor respecting a set of quadratic equations. Thanks to this, explicit Euler angles are not needed and the problem can be formulated as the minimization of a Ginzburg-Landau functional.

Smooth 3D frame fields. Palmer et al., 2020 propose a representation for *scaled* 3D crosses, i.e., frames. Relying on the connection to fourth-order tensors, they identify 3D frames with *orthogonally decomposable (odeco)* tensors, which have been thoroughly studied by Robeva (2016). Like octahedral frames, these odeco frames are defined by a set of known algebraic equations, and the authors also characterize the varieties that these frames live on (geodesics, tangent and normal spaces) and propose a precise projector based on a semidefinite relaxation. Two smooth frame field solvers are presented: one based on a diffusion-projection scheme, and the other leverages the knowledge of the variety to perform manifold optimization. The authors argue that the implicit scaling enabled by the odeco representation allows vanishing frames at singularities, preventing the Dirichlet energy from blowing up on fine supporting meshes.

Meshable cross fields from prescribed singularity graph. For most geometric models, smooth octahedral fields have a singularity graph that is not appropriate for hexahedral meshing, giving a hard time to integer-grid map solvers. Liu et al. (2018) have characterized the necessary conditions for a singularity graph to be globally hex-meshable, by enumerating the admissible singular node configurations. They also propose an algorithm that takes as input a meshable singularity graph, and produces an octahedral field that is valid for hexahedral meshing. In a similar vein, Corman et al. (2019) also take a singularity graph in input, and the *derivative* of the octahedral frame field as representation. Using the derivatives removes the need to take the frame symmetries into account, enabling the formulation of an optimization problem without integer variables; the price to pay is an additional quadratic

1.3. Related work

integrability constraint (read: integrability of the frame field derivative, and not the frame field itself).

Singularity graph repair. Motivated by the failure of smooth cross/frame fields to be hex-meshable, later works have focused on "repairing" unmeshable singularity graphs. Reberol et al. (2019) have tried different frame field correction techniques (fig. 1.13) that are based on how a human would tackle the problem. The cube with a notch (a) produces a well-known example of unmeshable frame field. In (b), the sharp feature curve is replaced by a fillet, but altering the geometry is generally not desirable. In (c), the feature curves are propagated in the volume through the frame field, producing "feature sheets" to which the frame must align. In (d), the boundary singularities are propagated inside to form the singular curves. In (e), the invalid 3-5 singular curve is snapped to the boundary. Techniques (c) and (d) fail when interacting with additional singular curves. Technique (e) is the most sensible but it produces hexes of bad quality, and it is not robust when singular defects are away from feature curves.

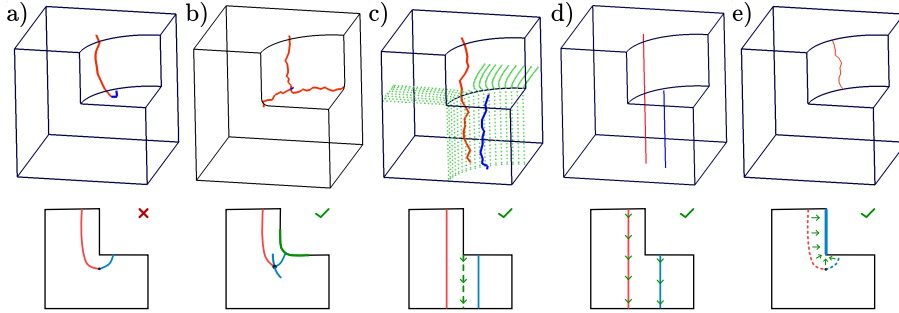


Figure 1.13: Frame field correction techniques from (Reberol et al., 2019).

Liu et al. (2023) push this approach much further by proposing an algorithm to robustly repair a given frame field. The fundamental tool of their work is the concept of arc *zipping* and *unzipping*, which are used to remove the invalid singular nodes (see inset). Upon termination they produce a *locally meshable* frame field, but do not guarantee global meshability, which is the principal source of failure. Combining the repair algorithm with state-of-the-art tools for computing frame fields, parametrizations, and extracting a mesh, their mesher went from a success rate of 2% to 58%, making it the best frame field-driven mesher to date.

Relation to our work. All existing works proposing a method for computing 3D cross or frame fields aim for *smooth* fields, which are known to produce unmeshable singularity graphs. Repair methods modify the singularity graph a posteriori, but can only guarantee local meshability. We are the first to propose a method for computing 3D *integrable* frame fields, motivated by the fact

that integrability guarantees global meshability. This is achieved by leveraging odeco tensors and formulating an integrability criterion on odeco frame fields.

1.4 Thesis organization

This thesis is organized as follows:

- Chapter 2 is a reproduction of our paper (Couplet et al., 2021) presenting a pipeline to turn a quasi-structured quad mesh into a block-structured (or coarse high-order) quad mesh. Coupled with the mesher of Reberol et al. (2021), this forms an end-to-end robust block-structured meshing pipeline.
- Chapter 3 presents orthogonally decomposable (*odeco*) tensors which serve as the algebraic representation of choice for our frame fields. We introduce novel tools on odeco frames that support our integrability formulation.
- Chapter 4 is the bulk of this work, presenting an energy functional that measures the integrability of an odeco frame field, as well as an optimization formulation to compute integrable frame fields. Results in 2D and 3D are presented and discussed.
- Chapter 5 concludes the thesis, providing perspectives for future research, hoping to one day achieve the holy grail of block-structured quadrilateral and hexahedral meshing.

CHAPTER 2

Coarse quadrilateral layouts using quantization

This chapter is a reproduction of the following paper:

Mattéo Couplet, Maxence Reberol and Jean-François Remacle (2021). Generation of high-order coarse quad meshes on CAD models via integer linear programming. In *AIAA Aviation 2021 Forum* (p. 2991).

Abstract. We propose an end-to-end pipeline to robustly generate high-quality, high-order and coarse quadrilateral meshes on CAD models. This kind of mesh enables the use of high-order analysis techniques such as high-order finite element methods or isogeometric analysis. An initial unstructured mesh is generated; this mesh contains a low number of irregular vertices but these are not necessarily aligned, causing a very dense quad layout. A T-mesh is built on the mesh which allows to modify its topology by assigning new integer lengths to the T-mesh arcs. The task of simplifying the quad layout can be formulated as an Integer Linear Program which is solved efficiently using an adequate solver. Finally, a high-order quad mesh is extracted from the optimized topology. Validation on several CAD models shows that our approach is fast, robust, strictly respects the CAD features, and achieves interesting results in terms of coarseness and quality.

2.1 Introduction

Computer Aided Design (CAD) systems are used extensively for industrial design in many domains, including automotive, shipbuilding, aerospace industries, industrial and architectural design, prosthetics, and many more. Engineering designs are encapsulated in CAD models, which, up to manufacturing precision, *exactly represent their geometry*. While the engineering analysis process begins with such CAD models, the predominant method of analysis (the finite element method) requires an alternative, discrete, representation of the geometry: a finite element mesh. In a mesh, the CAD model is usually subdivided into a large collection of simple geometrical shapes. Mesh generation can thus be seen as the transmission belt between CAD and analysis. Notice that, even though the mesh is not geometrically as precise as the CAD, mesh data are usually way larger than CAD data.

The surface of a CAD model is typically defined by piecewise-smooth CAD patches. CAD models are filled with various idiosyncrasies: periodic surfaces, trimmed patches, exotic or singular parametrizations. The idea of this paper is to provide a representation of the CAD that is light, universal and precise. For that, we build a high-order quad layout of the whole CAD model. A quad layout is the partitioning of an object's surface into simple networks of conforming quadrilateral patches. A high-order quad layout is a quad layout where individual patches are endowed with a geometry. Here, individual patches are both (i) truly smooth and (ii) share the same topology (four sides, no internal boundaries). The geometry of individual high order quad patches can be described either using high order finite elements or can inherit from the CAD representation (e.g. surface B-splines). The quad patches can be aligned with some of the feature edges of the CAD model but can also ignore some.

Such a coarse high-order quad layout can be used as a CAD model but also as a mesh in a high-order finite element context (p -FEM), or in numerical techniques with NURBS-based representation of the geometry, such as isogeometric analysis (Hughes et al., 2005) and NEFEM (Sevilla et al., 2008). It can also be used as a lightweight datastructure for forming an inline block-structured quad mesh. Block-structured meshes are of great interest for geometric multigrid or fast to design computation kernels.

In this paper we propose a practical approach to the generation of high-quality high-order quad layouts, where the emphasis is on robustness with respect to complex CAD models commonly occurring in engineering analysis. Our approach consists in first generating an unstructured linear and fine quadrilateral mesh, then modifying its topology until it is possible to extract a coarse quad layout.

Three-dimensional CAD models are represented on a computer using a *Boundary Representation* (BRep): a volume is bounded by a set of faces, a face is bounded by a series of curves and a curve is bounded by two end points. The BRep is a discrete object: it is a graph that contains model entities together with all their topological adjacencies. Then a geometry is associated to each

2.1. Introduction

model entity. Here, we aim at handling *real life* CAD models occurring in engineering analysis, which serve as a support for finite element or finite volume methods. Such models may be arbitrary complex, because of their size (large number of model entities) and/or because of the existence of sharp angles and features at very different scales. This complexity makes the robust treatment of CAD models particularly challenging.

In this work, we make the following assumption: the model is assumed to be a 2-manifold. Each model curve must be adjacent to at most two model faces. This limitation is due to implementation difficulties and could be overridden in a future work. At first let us define specifications for the high-order quad layout:

1. The layout should be *conforming*, i.e., free of T-junctions. This is a necessary condition for most applications (multiblock meshing, isogeometric analysis).
2. The quadrilaterals of the layout should strictly *conform to the CAD*, i.e., the quad corners should exactly lie on the underlying model surfaces and quad edges should strictly be aligned with a set of user-flagged model curves.
3. The mesh should be *valid* and, to a certain extent, *geometry-consistent*; the exact meaning of these terms will be detailed later.
4. The layout should be *coarse*; hence our primary concern is to minimize the number of quads in the layout.

The idea here is to start from a quad layout that does not fulfill all specifications and then to transform it to obtain compliance with the specifications. In our approach, the starting point is a valid quad mesh: a quad layout can be extracted from a conforming quad mesh and this layout satisfies all expected specifications except the last one: a quad mesh does not usually produce a coarse layout (see section 2.2 and fig. 2.2). This is mainly caused by non-alignment of the irregular vertices.

Related work

The generation of coarse quadrilateral layouts, also called base-complex in the literature, is a well-studied problem. For planar models and open surfaces, starting from a fine quad mesh, a simple approach is to iteratively collapse the chords (dual quad loops) which separate the irregular vertices that should be aligned. Unfortunately, this does not generalize to closed surfaces because the chords tend to wind up around the whole model, and collapsing them would simply destroy the mesh.

Bommes et al. (2011) automatically detect helical configurations, i.e., quad loops forming helices winding around the model, and correct them by modifying the mesh directly using elementary operations (such as collapsing and shifting edges) that preserve the quad-nature of the mesh. But this approach is not sufficient to deal with many other configurations, such as self-intersecting quad loops.

Tarini et al. (2011) extract the quad layout from the input mesh and simplify it by iteratively changing connections between irregular vertices in a greedy fashion, in order to minimize concurrently the separatrices' length and drift (how much they deviate from the underlying cross field). Once the new separatrices are optimized, the mesh is re-parametrized to align it with the new layout.

Razafindrazaka et al. (2015) optimize the network of separatrices connecting irregular vertices by formulating it as a graph matching. The resulting network is optimal with respect to a user-specified balance between coarseness and geometric feature alignment.

An alternative approach is to directly produce the coarse quadrilateral layout from a seamless parametrization (consistent global integration of a cross-field). Lyon et al. (2021) first build a T-mesh (non-conforming quad layout with T-junctions) from a seamless global uv -parametrization. The integer lengths of the T-edges are then optimized to achieve a conforming coarse layout. The benefit of this approach is that it encodes an infinite number of layout connections in a finite structure, and allows to flexibly impose constraints.

These papers have in common that they focus on computer graphics applications, where CAD consistency is generally not a concern and where the number of feature curves is limited. Our work aims at achieving similar results but with the additional constraint that the quad layout must comply with a complex CAD model.

Overview

For the construction of the coarse layout, we build on the recent quantization simplification of Lyon et al. (2021) as it provides the most flexibility to impose the constraints related to CAD compliance. Besides additional CAD constraints, the main difference with their work is that we use only quadrilateral meshes: neither a global seamless parametrization nor a cross field is needed here. In a CAD context, we believe that starting from an unstructured quad mesh is more robust as it is much easier to generate one compared to a valid global seamless parametrization.

Our complete method for building a high-order quadrilateral layout from a CAD model works as follows, and is illustrated in fig. 2.1:

1. First, an *unstructured* quad mesh is built on the model. This step relies on our previous work (Reberol et al., 2021); see section 2.2.

2.2. Quasi-structured quadrilateral meshing

2. Then, the topology of the mesh is modified to obtain a coarse layout, using our adaptation of the T-mesh approach of (Lyon et al., 2021); see section 2.3.
3. The geometry of the coarse layout is optimized by quantizing the patches and smoothing the resulting fine mesh; see section 2.4.
4. High-order quadrilaterals are eventually extracted from the block-structured fine mesh and the CAD geometry; see 2.4.

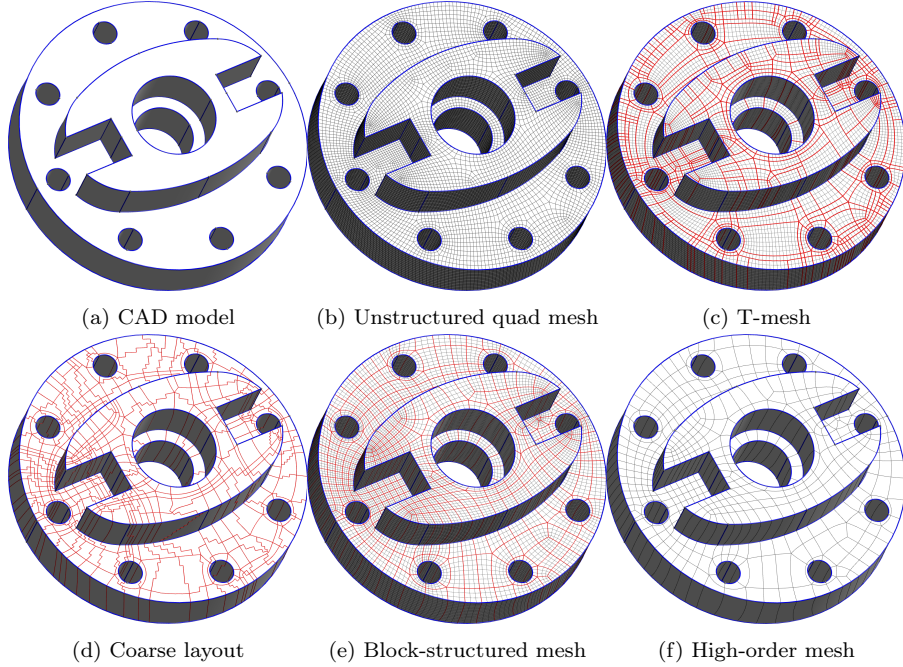


Figure 2.1: Overview of our high-order quadrilateral meshing pipeline. The illustrated CAD model is M1 from the MAMBO dataset (Ledoux, 2019).

2.2 Quasi-structured quadrilateral meshing

The first step of our pipeline is to obtain a high-quality unstructured quadrilateral mesh from the given CAD model. A mesh vertex is *regular* if its valence (number of incident faces) is equal to 4; otherwise it is *irregular*. A quad mesh is said to be *quasi-structured* if it has few irregular vertices compared to meshes produced with usual unstructured techniques. Our group has recently proposed a robust end-to-end pipeline to achieve this task (Reberol et al., 2021), which is freely available in the open-source software Gmsh (Geuzaine et al.,

2009). The idea is to use a boundary-aligned smooth cross field to guide the point insertion in a frontal mesher, then to improve the quad mesh topology by remeshing large convex cavities while respecting the cross field singularities. This method forms the first step of our pipeline and is used to generate the initial unstructured quad mesh.

Even though we are able to produce good linear quadrilateral meshes (in terms of number of irregular vertices and quad shapes), they cannot be coarsened directly due to the density of their *quad layout*. The quad layout is a minimal partition of the mesh into structured quadrilateral *patches*, obtained by propagating separatrices from the irregular vertices until they reach other irregular vertices. A dense layout is illustrated in fig. 2.2: very few patches of the layout have more than one single quad of the initial mesh so the number of patches in the layout is about the same as the number of quads in the initial mesh. There are two main reasons for the quad layout to be very dense for our quasi-structured meshes:

- Irregular vertices are not correctly *aligned* on the mesh, causing most separatrices to wind around the model until meeting an irregular vertex potentially far away.
- Many irregular vertices do not match singularities from the cross field, but are present to accomodate size transitions and small CAD features. They are typically grouped in valence 3-5 pairs. These irregular vertices are not all necessary to produce good high-order quadrilateral patches.

In the next section, we will detail a methodology that allows to correct these defects and produce a mesh with a coarse quadrilateral layout.

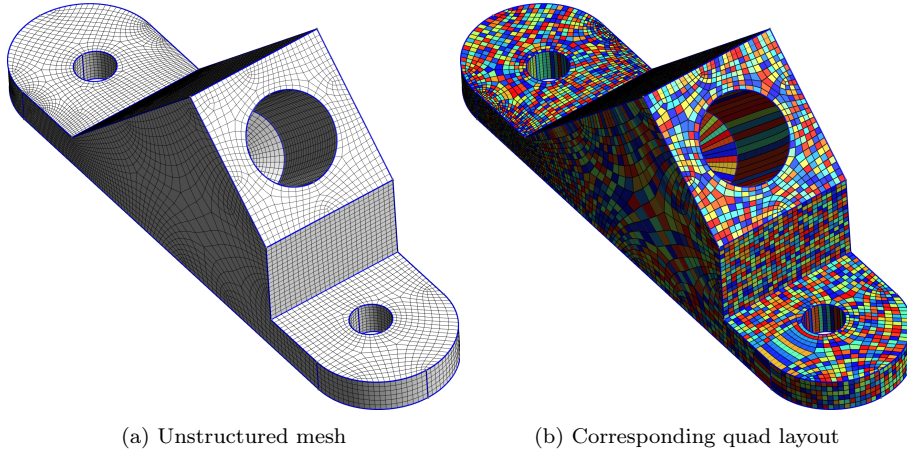


Figure 2.2: Quadrilateral layout of an unstructured mesh generated by our quasi-structured mesher. The quad layout is very dense. CAD model is M2 from MAMBO dataset (Ledoux, 2019).

2.3 Quadrilateral layout simplification

Given a *quasi-structured* quadrilateral mesh (the *input mesh*), we wish to obtain a *block-structured* mesh, having few irregular vertices but these vertices should be *aligned* to be able to extract coarse quadrilateral patches. To this end, we propose a slightly modified version of a method developed by Lyon et al. (2021), adapted to incorporate strict compliance to the CAD geometry. In this section, we present our adaptation of the method.

The objective of the optimization is to produce a layout that is both coarse and consistent with the geometry, i.e., with patches that are not too distorted. There is a trade-off between these two criteria: the coarsest layout will have too much distortion, and the layout with the best geometry is the layout of the input mesh, which, as previously shown, is very dense. To constrain the distortion of the patches, we prevent two irregular vertices from aligning if the corresponding deviation on the input mesh is larger than some prescribed angle α .

The approach works as follows. First, a *T-mesh*, i.e., a quadrilateral mesh with T-junctions, is constructed on the given quasi-structured mesh. Every edge of this T-mesh, or *arc*, has an integer length, corresponding to the number of mesh edges lying on this arc. This integer length assignment is known as a *quantization* of the T-mesh. The idea of the method is to change this quantization in order to optimize the quadrilateral layout. In practice, this will involve assigning many arcs a length of zero. We formulate this task as an *integer linear program* (ILP), i.e., an optimization problem with integer variables and linear objective function and constraints. On typical models such as the ones treated in this work, the ILP is large, with thousands of variables and constraints. But such problems can be solved efficiently thanks to their linearity.

T-mesh construction

From every irregular vertex we propagate *traces*, as if we were drawing the mesh’s quad layout. However, the trace will not be extended until meeting another irregular vertex, but instead stop at some point when reaching another trace, creating a T-junction. Such a construction is sometimes called a *motorcycle graph* in reference to the movie *Tron*. Contrary to a classical motorcycle graph, traces do not stop immediately when reaching another trace. Instead, a trace should stop when it has encountered enough candidates for alignment; specifically, when it has met one candidate on each side of the trace that does not deviate by an angle larger than α . Given a path between two vertices i and j , $\tan(\alpha_{ij})$ is the ratio $\frac{l_{ji}}{l_{ij}}$ between the integer length in the orthogonal direction (l_{ji}) and in the tangential direction (l_{ij}) (Lyon et al., 2021). Formally, we draw a fictitious cone around the trace of half-angle α . The trace stops when it has intersected with two orthogonal traces on each side whose origin

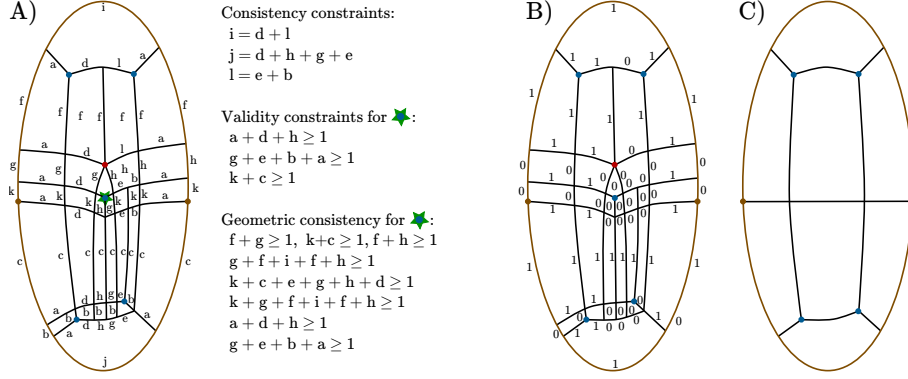


Figure 2.3: Illustration of the simplification of a T-mesh quantization. From the quad mesh of an ellipse, the T-mesh (A) is built, with irregular vertices of valence three in blue, valence five in red and CAD corners in brown. Alphabet letters are the integer lengths in the T-mesh quantization. After minimizing the objective function (weighted sum of lengths) subject to the linear constraints, we obtain the quantization in (B), where many arcs are collapsed (length equals to zero). The equivalent quad layout is coarse (C).

lies inside the cone. The rationale is the following: we want a trace to go far enough to have a couple of potential irregular vertices to align with, but we do not want it too long to prevent aligning irregular vertices far from each other, complexifying the quad layout in vain.

In order to comply with the CAD model, it is forbidden for non-CAD irregular vertices to collapse on a CAD curve. A simple way to address this is to set $\alpha = 0$ for traces lying on CAD curves. This will prevent the trace from diverting towards an irregular vertex away from the CAD curve. Note that, according to the stopping rule, a trace lying on a CAD curve will not stop until meeting another irregular vertex.

The constructed T-mesh $\mathcal{T} = (\mathcal{N}, \mathcal{A}, \mathcal{P})$ consists of nodes $\mathcal{N}$ for every irregular vertex and intersection of traces, arcs $\mathcal{A}$ corresponding to segments of a trace between two nodes, and patches $\mathcal{P}$ which are the quadrilateral regions bounded by the arcs. Every arc $a \in \mathcal{A}$ is assigned an integer length q_a ; the set of integer lengths are the variables of our optimization problem.

Quantization constraints

To achieve our goal of obtaining a coarse quantization that is valid and geometrically consistent, several constraints are enforced.

2.3. Quadrilateral layout simplification

Non-negativity It is clear that every arc should have a non-negative length: for every arc $a \in \mathcal{A}$,

$$q_a \geq 0. \quad (2.1)$$

Consistency The quantization is said to be *consistent* if for every patch, pairs of opposite sides have the same total length. This ensures that the quantization implicitly defines a conforming quad mesh. For every patch with (ordered) sides S_1, S_2, S_3, S_4 , each side being a set of arcs, we write

$$\sum_{a \in S_1} q_a = \sum_{b \in S_3} q_b, \quad \text{and} \quad \sum_{a \in S_2} q_a = \sum_{b \in S_4} q_b. \quad (2.2)$$

Validity For the quantization to be *valid*, irregular vertices must be separated by a strictly positive distance. This may seem complicated to formulate, but luckily Lyon et al. (2021) have proved a simple sufficient condition. Let i be an irregular vertex and t_i one of its traces and let t_* denote the first trace that t_i intersects whose origin $*$ lies in the $\pi/4$ cone around t_i . Let n_{i*} denote the intersection, and S_{i*} the set of arcs from i to n_{i*} . Then, for an irregular vertex i to be separated from any other irregular vertex, it is sufficient that i and n_{i*} are separated. Therefore, for every trace t emanating from an irregular vertex, we set the constraint

$$\sum_{a \in S_{i*}} q_a \geq 1. \quad (2.3)$$

This constraint is forced if n_{i*} lies on a feature curve, to prevent the singularity from collapsing onto it. As explained previously, the input mesh usually possesses many pairs of 3-5 irregular vertices that serve to satisfy size transitions. To further simplify the layout, we make an exception to the validity constraint to allow 3-5 pairs to collapse on each other. Such a collapse is valid as it results in a regular vertex. In practice, the constraint is not imposed if i and n_{i*} form a 3-5 pair.

CAD corners should remain separated in the final layout, and cannot be collapsed onto each other. To address this, we consider them as virtual irregular vertices (regardless of their valence) and apply the same validity constraint. As each feature possesses at least one CAD corner (or CAD point), this also ensures the separation of features.

Geometric consistency In order to ensure consistency with the geometry defined by the input mesh, we must explicitly separate irregular vertices whose traces intersect, but that do not lie in each other's cone. To do so, for every irregular vertex i , every irregular vertex j that is outside the cone of i is separated:

$$\text{for every } n_{ij} \text{ with } \frac{l_{ji}}{l_{ij}} > \tan \alpha_i, \quad \sum_{a \in S_{ji}} q_a \geq 1, \quad (2.4)$$

where n_{ij} is the intersection of t_i and t_j , l_{ji} is the total length from j to n_{ij} , S_{ji} is the set of arcs from j to n_{ij} , and α_i is the cone half-angle for trace t_i (which is either the user-prescribed angle α , or 0 if the trace lies on a CAD curve).

Objective function

To obtain the coarsest possible quadrilateral layout, we need to minimize the number of patches. Let a, b denote the integer dimensions of some patch. If $a = 0$ or $b = 0$, the patch vanishes and does not contribute to the total number of patches. If $a = b = 1$, we add a single patch to our objective function. If $a > 1$ or $b > 1$, it means that there remains T-junctions. To obtain a conforming quad layout, we split the patch into a grid of $a \times b$ patches, each having dimensions 1×1 and contributing as a single patch to the objective function. Hence, the objective that exactly promotes coarseness is the total integer area of the mesh:

$$\min_q \sum_{p \in \mathcal{P}} \text{area}(p) = \sum_{p \in \mathcal{P}} \left(\sum_{a \in S_1} q_a \times \sum_{b \in S_2} q_b \right). \quad (2.5)$$

However, this objective is nonlinear, and using it would not provide us an ILP. Instead, we minimize a *weighted sum of arc lengths*. The weight serves to help the solver prioritize arcs to collapse. To this end, we set the weight of an arc a to be the inverse of its initial integer length l_a . This gives the following objective function:

$$\min_q \sum_{a \in \mathcal{A}} \frac{1}{l_a} q_a. \quad (2.6)$$

Note that this optimization problem is guaranteed to have a solution since the input mesh satisfies every constraint.

The previous T-mesh quantization simplification problem is illustrated on fig. 2.3, where we build the T-mesh of the quad mesh of an ellipse, which is not optimal because the irregular vertices are not aligned and there is an interior 3-5 pair. We list all the consistency constraints, and the validity and geometric consistency constraints associated to the central irregular vertex of valence three. Note that by using a single variable (e.g. length a along the ellipse) for multiple arcs, we already enforce the simple equality consistency constraints. In this particular example, the validity constraints are redundant with some of the geometric consistency constraints, but this is not always the case. The solution of the integer problem (values in B) contains lengths equal to zero and one, where zero corresponds to an arc collapse. By merging the appropriate vertices and smoothing the resulting quad mesh, we obtain the coarse layout in C. The horizontal line is kept because it connects the CAD corners (dots in brown), which are preserved in our formulation.

2.4 Block-structured quad mesh and high-order quadrilaterals

The optimized quantization implicitly defines a topological quad layout. In this section, we explain how we obtain a smooth and fine block-structured mesh. First, a conforming coarse quad mesh is extracted from the quantization, however this mesh has wrongly placed vertices and non-straight edges. Our approach to correct this is to subdivide the coarse patches to obtain a fine mesh, then smooth the fine mesh to optimize the placement of vertices and straighten the edges.

Edge and patch splitting

In the optimized quantization, some arcs will be assigned an integer length $q \geq 2$. These edges are split into q sub-edges. To obtain a conforming quad layout, this split must be "propagated" by splitting the incident patches: if an incident patch has integer dimensions $q \times q'$, and $q' > 0$, it is split into $q \cdot q'$ sub-patches of dimensions 1×1 ; if $q' = 0$ it is split into q sub-patches of dimensions 1×0 . Splitting patches of zero area is necessary to correctly merge vertices in the next step.

Vertex and edge merging

At this point we have a conforming layout but with many zero-length edges and zero-area patches. To get rid of these, we first identify groups of vertices that have zero distance between each other (formally, two vertices are in the same group if they are connected on the T-mesh by a path of zero total length). Every group of vertices is virtually merged into a single "center vertex". This effectively removes all zero-length T-edges. Then, edges bounded by the same pair of center vertices are virtually merged into one edge. This removes all zero-area patches, and leaves us with a clean topological mesh.

Vertex and edge placement

Vertices and edges need to be assigned a geometrical location on the model. This operation requires to pay attention to CAD consistency. Center vertices are chosen in the following way:

- If some vertex in the group corresponds to a CAD vertex, the latter is taken as the center vertex (there is at most one CAD vertex in the group);
- Otherwise, if some vertex in the group lies on a CAD curve, any vertex on the curve is taken as the center vertex (there is at most one such curve in the group);
- Otherwise, any vertex of the group is taken as center.

Afterwards, edges are formed by taking the shortest path on the input mesh between the two centers they connect. If an edge lies on a CAD curve (i.e., some of the T-edges before the merge lied on a CAD curve), its corresponding path is the subset of the curve that connects the two centers.

Patch subdivision

Once we have a conforming quad mesh, we subdivide the coarse patches to refine the mesh. Note that opposite sides of a patch must have the same subdivision. Therefore, an integer width has to be chosen for every so-called "quad loop". A quad loop is a closed sequence of patches obtained by starting from a patch and iteratively go to its neighboring patch, following the same direction.

A target integer length is assigned to every edge, defined as the length of its corresponding path on the input mesh. Since this target length is not constant along the quad loop, the (rounded) mean target length is taken as the quad loop's integer width. Following this process for every quad loop will assign to every edge an integer length. Finally, edges are subdivided into equal parts, and patches are subdivided by transfinite interpolation.

Smoothing

Once we have a fine mesh, we can proceed to smooth the mesh. We perform Winslow smoothing (Winslow, 1966) which is the industry-standard technique for structured grid mesh generation. In an explicit optimization loop, each vertex is moved according to its neighbors. The Winslow coefficients are computed with a finite difference discretization of the Winslow equation (Knupp, 1999). Since this operation moves vertices away from the CAD surface, every vertex is projected back on it by finding the closest point on the surface.

This smoothing tends to create squared quadrilaterals, so it is important that in the previous patch subdivision (2.4), the integer lengths assigned to the quad loops are chosen according to the geometry. After smoothing, the block-structured quad mesh usually contains patches with rectangular shapes (e.g. Fig. 2.1.d.).

The explicit loop, with the finite difference discretization of Winslow equations and CAD projections, is not guaranteed to produce valid quads (strictly positive Jacobian). In practice, the behaviour is generally satisfactory, but negative quads do appear at some concave corners. In the future, we will explore additional untangling techniques for surfaces, or directly optimize the high-order quad mesh produced later in the pipeline.

Extraction of high-order quadrilaterals

From the smooth block-structured quad mesh, we can extract high-order quadrilaterals. Assuming the elements are parametrized with Lagrange finite element

2.5. Results and discussion

functions of order N , the process is straightforward:

- For each vertex v_i of the block structure, we create a high-order vertex.
- For each edge (v_i, v_j) of the block structure, we retrieve the two high-order extremities and we create $N - 1$ interior high-order vertices along the curved edge via polyline equidistant re-sampling. Each new high-order vertex is projected on the CAD.
- For each patch (v_i, v_j, v_k, v_l) , we retrieve the four high-order corners and the four high-order sides, and we create $(N - 1)^2$ interior high-order vertices with transfinite interpolation. Each new high-order vertex is projected on the CAD.

The above process generates high-order meshes with regularly spaced nodes. All the figures shown in this paper were produced with order five polynomials.

For better numerical properties, it is possible to produce Gauss-Lobatto spacing by slightly changing the curve re-sampling step. One could also build B-spline surfaces to better match the initial CAD geometry. In the future, we plan to implement multiple high-order representations and let the user choose the one best suited for his use cases.

Note that the position of the high-order nodes is entirely dependant on the result of the block-structured linear quad mesh smoothing. It may be interesting to employ high-order surface smoothing to further improve the mesh geometry, but such endeavour is challenging and we keep it for further work.

2.5 Results and discussion

Table 2.1: Statistics of results in fig. 2.4.

Model	#Faces ¹	#P _{init} ²	#Irr ³	#Vars ⁴	#P _{final} ⁵	Time ⁶	QuadQS ⁷	ILP ⁸	Smooth ⁹
M1	23707	8625	180	3340	1168	27 s	72 %	4 %	13 %
M2	16059	12253	156	3541	983	17 s	67 %	7 %	13 %
M3	22242	15864	336	5968	2505	28 s	69 %	11 %	9 %
M4	25996	15394	194	3702	1596	30 s	62 %	4 %	23 %
M5	32002	20487	287	5265	2681	45 s	66 %	5 %	18 %
M6	26322	19975	118	2522	1007	31 s	45 %	2 %	43 %
A319	18193	13616	410	7565	4278	173 s	35 %	4 %	53 %
DLR F6	99978	42354	620	11830	6553	282 s	30 %	6 %	57 %

¹ Number of faces in quasi-structured mesh.

² Number of patches in quasi-structured mesh layout.

³ Number of irregular vertices in quasi-structured mesh.

⁴ Number of variables in ILP.

⁵ Number of patches in high-order mesh.

⁶ Total run time from CAD to high-order mesh.

⁷ Proportion of time used for quasi-structured meshing.

⁸ Proportion of time used for solving the ILP.

⁹ Proportion of time used for smoothing and building the high-order mesh.

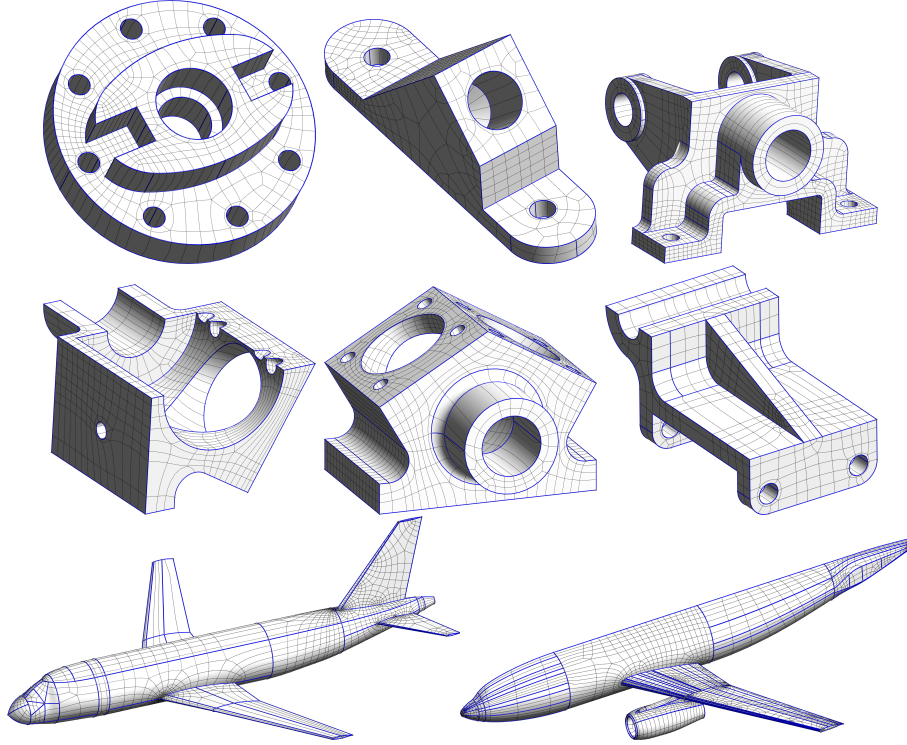


Figure 2.4: Results of our method on a variety of CAD models (5th-order quadrilaterals, $\alpha = \pi/4$). Top six: various models from the MAMBO dataset (Ledoux, 2019) (M1, M2, M3, M4, M5, M6). Bottom two: Airbus A319 and DLR-F6 (Brodersen, 2002).

To validate our approach and its robustness, we apply it on several existing CAD models, and discuss its strengths and weaknesses.

Implementation The complete pipeline is implemented in our meshing software Gmsh. The quasi-structured quadrilateral mesher and Winslow smoother are readily available in Gmsh. For solving the ILP we use `lp_solve` which is a free and open-source Mixed-Integer Linear Programming solver. In our experiments, the complete pipeline is run on a single core on an average laptop computer.

CAD models For illustration, we have chosen six CAD models from the MAMBO dataset (Ledoux, 2019) (M1 to M6) as well as two aircraft models, an Airbus A319 and the DLR-F6 (Brodersen, 2002). As a maximal deviation we chose $\alpha = \pi/4$ as we found that it achieves a good balance between coarseness and mesh quality. The order of the quadrilaterals is $N = 5$.

2.6. Conclusion and future work

On fig. 2.4 we illustrate the high-order meshes produced by our method on the given CAD models. In table 2.1 we provide some statistics about the process.

Discussion First we can notice that the total run time, from a few seconds to a couple of minutes, is reasonable in an engineering analysis context, even on the more complex aircraft models. In all cases, solving the integer optimization takes a very small portion of the total run time. From this observation we are confident that ILP formulations are a viable approach to high-quality, high-order mesh generation. As the model complexity increases, smoothing takes a significant portion of the time.

Our method is able to significantly reduce the complexity of the quad layout, as can be deduced from the $\#P_{\text{init}}/\#P_{\text{final}}$ ratio which ranges from 3 to 11. While the mesh coarseness is satisfying, we believe that it can be further simplified without affecting significantly the geometric consistency. We see two limitations that prevent a better simplification:

- Once a 3-5 pair of irregular vertices has been collapsed into a regular vertex, this vertex cannot be collapsed further due to the validity constraints enforced on the former irregular vertices.
- After quantization, many arcs have an integer length greater than 1, which significantly increases the number of patches after the splitting phase. This is due to our choice of objective function which is a linearization of the total mesh area.

By visual inspection we see that the quadrilaterals have low distortion, and while being coarse, they accurately capture the model's geometry. The meshes we have produced are therefore perfectly suitable for high-order analysis techniques.

2.6 Conclusion and future work

In this work, we have presented a high-order meshing pipeline that is fast, effective and robust with respect to the input CAD model. It combines our robust quasi-structured quad mesher together with a layout simplification based on an ILP formulation that allows to flexibly impose CAD compliance. An additional advantage is that the layout simplification only requires an unstructured quad mesh as input; there is no need for a seamless parametrization or a cross field which are usually hard to build robustly on CAD models. Although we have not achieved industrial-grade performance in terms of computational cost and coarseness, we are confident that further improvements can, paving the way for high-order analysis on arbitrarily complex CAD geometries.

In terms of **computational cost**, we should concentrate our efforts on the smoothing step which takes a large portion of the total time on large

models. Another clue we would like to investigate is parallelism: a first "pre-simplification" could act on every CAD face separately while enforcing a quantization on the boundaries, after which a global layout simplification would finish the work.

In terms of **coarseness**, as previously said, we believe further simplification can be achieved without sacrificing geometric consistency. A first improvement is to allow collapsed 3-5 pairs to further collapse onto model curves or other irregular vertices. This could be achieved, e.g., using *conditional constraints*. Another approach would be to *restart* the algorithm on the new quad mesh, forming an iterative process; however, as it might be highly non-uniform in size, one should calculate the cone angle α_i appropriately to take into account the physical lengths of the T-edges. Furthermore, by trading off computational performance, we could investigate the use of the exact objective function (2.5) with *non-linear solvers*. Finally, we note that the quality of the CAD model severely impacts the coarseness that can be achieved. This is particularly visible on the aircraft models in fig. 2.4 which possess many short CAD curves. Relaxing the CAD constraints by *correcting such defects* would allow to achieve a coarser result.

CHAPTER 3

The odeco frame representation

The contents of this chapter partially overlap with the paper

Mattéo Couplet, Alexandre Chemin and Jean-François Remacle (2024). Integrable Frame Fields using Odeco Tensors. In *Proceedings of the 2024 SIAM International Meshing Roundtable* (p. 53-65).

As has been discussed in chapter 1, a frame field assigns to each point of the n -dimensional geometric domain a set of n directions, called a *frame*, representing the local gradient of a parametrization (or, alternatively, the set of *coordinate vectors*). This chapter focuses on a mathematical representation for frames that is suitable for computations, particularly regarding the desired symmetries. It is based on the works of Robeva (2016) for the general theory of odeco tensors, Palmer et al. (2020) for their application to frame fields, and our own work (Couplet et al., 2024). We propose the following novel contributions to the odeco tensor framework:

- A fast algorithm, based on the Mandel notation, to recover the decomposition of an odeco tensor, which can be used as a projector when the tensor is close to odeco;
- The notion of *second-order part* of a tensor, and the way it can alter the scaling of an odeco tensor without recovering the decomposition;
- The characterization of the *eigenvalue sensitivity*, which generalizes the perturbation problem from matrices to 4th order odeco tensors.

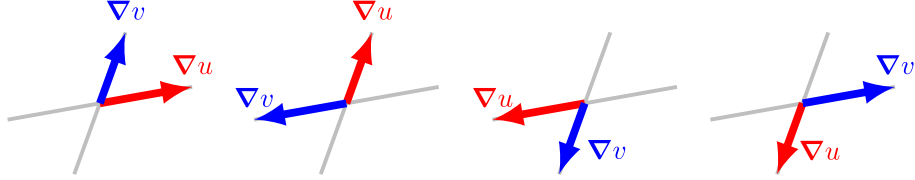


Figure 3.1: Equivalent gradient pairs $(\nabla u, \nabla v)$ across the charts of a global seamless parametrization.

3.1 Symmetries of a parametrization's Jacobian

In chapter 1, we have presented the notion of *frame* as a way to make the Jacobian of a global seamless parametrization continuous across its charts.

Starting with the 2D case, if $\phi = (u, v)$ represents a seamless parametrization, then the parametrization gradients are discontinuous across charts by a permutation

$$\begin{pmatrix} \nabla u' \\ \nabla v' \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}^k \begin{pmatrix} \nabla u \\ \nabla v \end{pmatrix}, \quad k \in \mathbb{Z}. \quad (3.1)$$

Note that we chose the term *permutation* and not *rotation*, since this property does not mean that the individual gradient vectors rotate by a multiple of 90° across the cut, but rather they cycle along the sequence of 4 vectors; this is illustrated in fig. 3.1. Equation (3.1) defines an equivalence relation on pairs of vectors $(\mathbf{u}, \mathbf{v})$. The corresponding equivalence class

$$\left[\begin{pmatrix} \mathbf{u} \\ \mathbf{v} \end{pmatrix} \right] \triangleq \left\{ \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}^k \begin{pmatrix} \mathbf{u} \\ \mathbf{v} \end{pmatrix} \right\} = \left\{ \begin{pmatrix} \mathbf{u} \\ \mathbf{v} \end{pmatrix}, \begin{pmatrix} \mathbf{v} \\ -\mathbf{u} \end{pmatrix}, \begin{pmatrix} -\mathbf{u} \\ -\mathbf{v} \end{pmatrix}, \begin{pmatrix} -\mathbf{v} \\ \mathbf{u} \end{pmatrix} \right\} \quad (3.2)$$

is called a *frame*. Note that swapping $\mathbf{u}$ and $\mathbf{v}$ gives a different frame $\left[\begin{pmatrix} \mathbf{v} \\ \mathbf{u} \end{pmatrix} \right] \neq \left[\begin{pmatrix} \mathbf{u} \\ \mathbf{v} \end{pmatrix} \right]$ even though the four frame vectors are the same. Assuming the frame respects the local injectivity property $\det \begin{pmatrix} \mathbf{u} \\ \mathbf{v} \end{pmatrix} > 0$, it follows that $\mathbf{u} \times \mathbf{v}$ must be positive everywhere, and the swapped frame $\left[\begin{pmatrix} \mathbf{v} \\ \mathbf{u} \end{pmatrix} \right]$ is not locally injective. For this reason we can extend the frame to a larger equivalence class that includes the inverted frame

$$\left[\begin{pmatrix} \mathbf{u} \\ \mathbf{v} \end{pmatrix} \right] \triangleq \left[\begin{pmatrix} \mathbf{u} \\ \mathbf{v} \end{pmatrix} \right] \cup \left[\begin{pmatrix} \mathbf{v} \\ \mathbf{u} \end{pmatrix} \right], \quad (3.3)$$

which we can make correspond to the set of four frame vectors $\{\mathbf{u}, \mathbf{v}, -\mathbf{u}, -\mathbf{v}\}$. This *extended frame* will be useful to make a one-to-one correspondence with the tensor object we define in the next section.

The same machinery can be applied to 3D frames: if $\phi = (u, v, w)$ is 3D seamless parametrization, then its gradients are discontinuous across charts by octahedral permutations

$$\begin{pmatrix} \nabla u' \\ \nabla v' \\ \nabla w' \end{pmatrix} = R \begin{pmatrix} \nabla u \\ \nabla v \\ \nabla w \end{pmatrix}, \quad R \in \text{Oct}. \quad (3.4)$$

3.2. Representing frames using orthogonally decomposable tensors

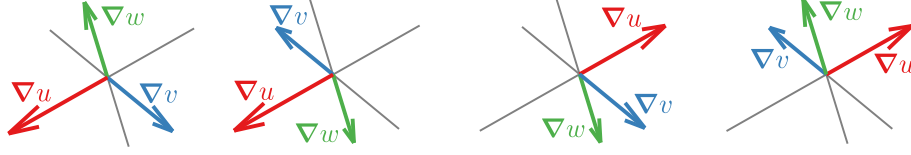


Figure 3.2: Four (out of 24) equivalent gradient triplets $(\nabla u, \nabla v, \nabla w)$ across the charts of a 3D global seamless parametrization.

This defines an equivalence relation on triplets of vectors $(\mathbf{u}, \mathbf{v}, \mathbf{w})$, with equivalence class

$$\begin{bmatrix} \mathbf{u} \\ \mathbf{v} \\ \mathbf{w} \end{bmatrix} \triangleq \left\{ R \begin{pmatrix} \mathbf{u} \\ \mathbf{v} \\ \mathbf{w} \end{pmatrix}, \quad R \in \text{Oct} \right\}. \quad (3.5)$$

Recall that the octahedral group Oct has 24 permutations. The local injectivity property now reads $\det \begin{pmatrix} \mathbf{u} \\ \mathbf{v} \\ \mathbf{w} \end{pmatrix} > 0$, and the inverted frame (there is only one!) can be obtained by swapping two frame vectors, or changing one of the signs. We define the extended frame by including the inversion

$$\left[\begin{bmatrix} \mathbf{u} \\ \mathbf{v} \\ \mathbf{w} \end{bmatrix} \right] \triangleq \left\{ \pm R \begin{pmatrix} \mathbf{u} \\ \mathbf{v} \\ \mathbf{w} \end{pmatrix}, \quad R \in \text{Oct} \right\} = \left\{ R \begin{pmatrix} \mathbf{u} \\ \mathbf{v} \\ \mathbf{w} \end{pmatrix}, \quad R \in 2\text{Oct} \right\}, \quad (3.6)$$

where 2Oct is the *binary* octahedral symmetry group, which contains the 48 rotations *and reflections* of the octahedron. The extended frame then corresponds to the set of six frame vectors $\{\mathbf{u}, -\mathbf{u}, \mathbf{v}, -\mathbf{v}, \mathbf{w}, -\mathbf{w}\}$.

3.2 Representing frames using orthogonally decomposable tensors

In order to compute a field of parametrization gradients that is continuous everywhere (except at singularities), we wish to find a representation of a n -dimensional frame $(\mathbf{v}_1, \dots, \mathbf{v}_n)$ that is invariant under the symmetries in eqs. (3.3) and (3.6). From this chapter on, frames are supposed to be *orthogonal*, i.e., their frame vectors $\mathbf{v}_1, \dots, \mathbf{v}_n$ are pairwise orthogonal. At first, it might be tempting to simply consider the $n \times n$ symmetric positive definite matrix

$$\mathbf{M} \triangleq \sum_{m=1}^n \lambda_m \mathbf{u}_m \mathbf{u}_m^T, \quad (3.7)$$

where $\lambda_m \triangleq \|\mathbf{v}_m\|$ and $\mathbf{u}_m \triangleq \mathbf{v}_m / \lambda_m$ is the normalized frame vector. It indeed respects all required symmetries, as it is invariant under any permutation of the frame vectors or changes in their signs. The issue is that when some frame vectors have identical lengths, they form an eigenspace of $\mathbf{M}$ of dimension greater than 1, and the information on their individual orientations is lost. As

an example, consider a frame that has equal lengths $\lambda_1 = \dots = \lambda_n = \lambda$ (a so-called *isotropic* frame). Its corresponding matrix then becomes

$$\mathbf{M} = \lambda \sum_{m=1}^n \mathbf{u}_m \mathbf{u}_m^\top = \lambda \mathbf{I}, \quad (3.8)$$

from which it is impossible to recover the frame directions $\mathbf{u}_1, \dots, \mathbf{u}_n$. To alleviate this problem, we turn to a higher-order tensor representation

$$\mathbf{T} = \sum_{m=1}^n \lambda_m \mathbf{u}_m^{\otimes d} = \sum_{m=1}^n \lambda_m \underbrace{\mathbf{u}_m \otimes \dots \otimes \mathbf{u}_m}_{d \text{ times}} \quad (3.9)$$

Choosing an odd order d is inappropriate: switching a frame vector's sign would then yield a different tensor, since $(-\mathbf{u}_m)^{\otimes d} = (-1)^d \mathbf{u}_m^{\otimes d}$. Hence, the lowest-order choice is to work with *fourth-order* tensors.

In the remainder of this section we lay out the mathematical framework of symmetric tensors, and we present the tensor decomposition problem. We show how orthogonal frames can be mapped to *orthogonally decomposable tensors*, which form a subspace of symmetric tensors that is fully described by a set of known algebraic equations. In order to emphasize their applicability to both 2D and 3D frames, we develop the theory for an arbitrary dimension n .

Symmetric tensors and homogeneous polynomials. A tensor $\mathbf{T}$ of *dimension* n and *order* d is an array of size $n \times n \times \dots \times n$ (d times) with entries $T_{i_1 \dots i_d}$. This tensor is said to be *symmetric* if its entries are invariant under permutation of the indices: $T_{i_1 \dots i_d} = T_{i_{\sigma(1)} \dots i_{\sigma(d)}}$ for any permutation σ of $\{1, 2, \dots, d\}$. Let $S^d(\mathbb{R}^n)$ denote the space of n -dimensional symmetric tensors of order d ; note that $S^2(\mathbb{R}^n)$ is the space of $n \times n$ real symmetric matrices.

How many independent coefficients does a symmetric tensor $\mathbf{T} \in S^d(\mathbb{R}^n)$ have? To answer this, we count the number of multisets of d elements taken from a set of indices $\{1, \dots, n\}$, which gives

$$\binom{n+d-1}{d} = \frac{(n+d-1)!}{d!(n-1)!}. \quad (3.10)$$

For $d = 4$, we thus have 5 coefficients in 2D and 15 coefficients in 3D.

It is useful to notice right away that there is a one-to-one correspondence between symmetric tensors and homogeneous polynomials. This correspondence provides a nice way to visualize tensors, and allows us to connect tensor representations with the spherical harmonics representations from previous works.

A symmetric tensor $\mathbf{T} \in S^d(\mathbb{R}^n)$ has a corresponding homogeneous degree d polynomial in n variables $p_{\mathbf{T}} \in \mathbb{R}[x_1, \dots, x_n]$, given by

$$p_{\mathbf{T}}(x_1, \dots, x_n) = \mathbf{T} \cdot \mathbf{x}^d \triangleq \sum_{i_1, \dots, i_d=1}^n T_{i_1 \dots i_d} x_{i_1} x_{i_2} \dots x_{i_d}, \quad (3.11)$$

3.2. Representing frames using orthogonally decomposable tensors

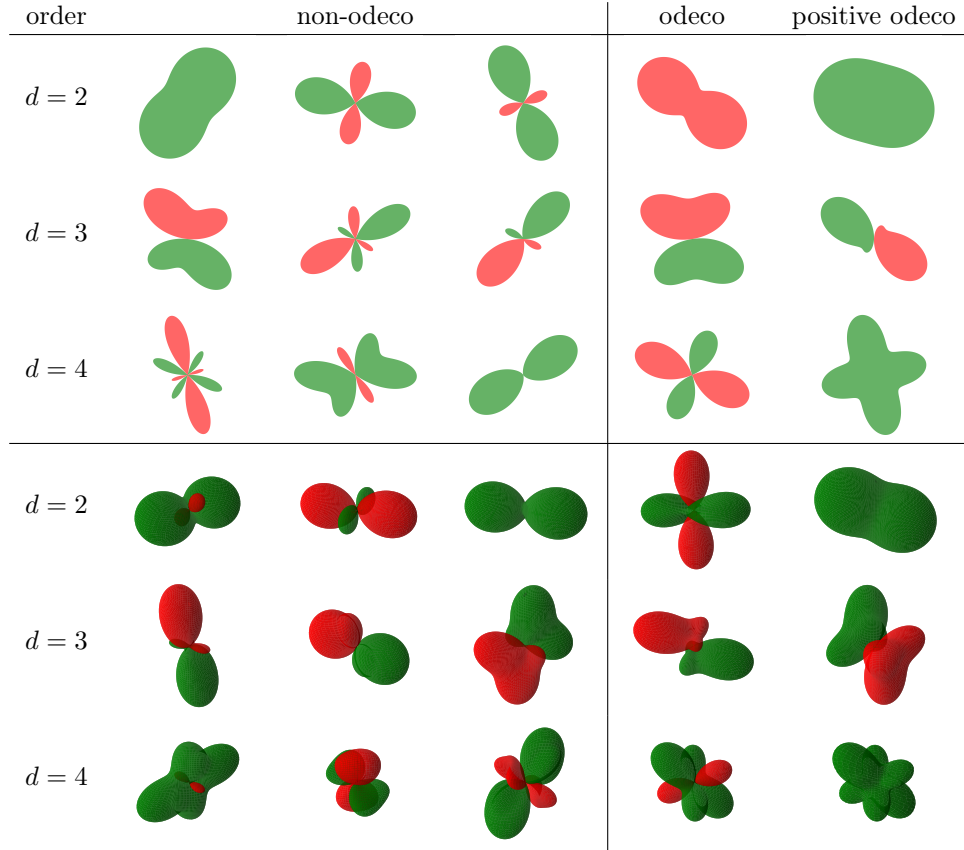


Figure 3.3: Gallery of 2D and 3D symmetric tensors illustrated by their corresponding homogeneous polynomial: $\{|p_{\mathbf{T}}(\hat{\mathbf{x}})| \hat{\mathbf{x}}, \hat{\mathbf{x}} \in \mathbb{S}_{n-1}\}$. Green areas correspond to $p_{\mathbf{T}} > 0$ and red areas to $p_{\mathbf{T}} < 0$. Last two columns illustrate odeco tensors, with positive eigenvalues for the last column. Notice that $p_{\mathbf{T}}$ is an even function when d is even, and an odd function when d is odd. Notice that second-order ($d = 2$) tensors are always odeco.

where we have defined the notation $\mathbf{T} \cdot \mathbf{x}^d$ for the d -uple contraction of a tensor $\mathbf{T}$ with a vector $\mathbf{x}$, which will be used throughout the sequel. $p_{\mathbf{T}}$ is *homogeneous* since for any $\lambda \in \mathbb{R}$,

$$p_{\mathbf{T}}(\lambda x_1, \dots, \lambda x_n) = \lambda^d p_{\mathbf{T}}(x_1, \dots, x_n). \quad (3.12)$$

The homogeneous property allows us to restrict this polynomial to the unit sphere $\mathbb{S}_{n-1}$. This suggests a simple way to visualize $p_{\mathbf{T}}$: take the unit sphere $\mathbb{S}_{n-1}$ embedded in $\mathbb{R}^n$ and "stretch" it according to the value $p_{\mathbf{T}}$ takes on the sphere. Formally, we draw the $(n-1)$ -dimensional manifold

$$\{|p_{\mathbf{T}}(\hat{\mathbf{x}})| \hat{\mathbf{x}}, \quad \hat{\mathbf{x}} \in \mathbb{S}_{n-1}\}. \quad (3.13)$$

The sign of $p_{\mathbf{T}}$ is distinguished by coloring the manifold in two colors: one for positive values and one for negative values. We illustrate the polynomials of some arbitrary tensors in fig. 3.3.

Eigenvectors of tensors. The notions of eigenvalues and eigenvectors are familiar from linear algebra, and play a crucial role in applied mathematics and scientific computing. With the advent of high-dimensional data, matrix algorithms are increasingly being generalized to tensors. A generalization that interests us is the non-standard notion of eigenvectors of tensors, which was pioneered by Lim (2005) and Qi (2007).

The notions of eigenvector and eigenvalue can be generalized to symmetric tensors, and are central to define orthogonally decomposable tensors. To gain an intuitive sense of tensor eigenvectors we start with their interpretation as stationary points of polynomial p_T . Consider the stationary points of the restriction of p_T to the sphere $\mathbb{S}_{n-1}$. For such vectors x , the gradient of p_T is parallel to x . Working out the gradient of p_T gives

$$\frac{\partial p_T(x)}{\partial x_j} = d \sum_{i_1, \dots, i_{d-1}} T_{i_1, \dots, i_{d-1}, j} x_{i_1} \dots x_{i_{d-1}} = d(T \cdot x^{d-1})_j. \quad (3.14)$$

This gradient being parallel to x means that there exists $\lambda \in \mathbb{R}$ such that

$$T \cdot x^{d-1} = \lambda x, \quad (3.15)$$

which gives us the definition of an *eigenvector* x of T with *eigenvalue* λ . When $d = 2$, this definition coincides with the usual definition for matrix eigenvectors. Notice that evaluating p_T at a *unit norm* eigenvector $\hat{x}$ gives

$$p_T(\hat{x}) = T \cdot \hat{x}^d = (T \cdot \hat{x}^{d-1}) \cdot \hat{x} = \lambda \hat{x} \cdot \hat{x} = \lambda, \quad (3.16)$$

which means that the eigenvalue λ visually corresponds to the extent of the plot of p_T at $\hat{x}$, as on fig. 3.3.

How many distinct eigenvectors does a generic tensor have on the unit sphere? First notice that they always come in opposite pairs $\pm x$, of identical

3.2. Representing frames using orthogonally decomposable tensors

eigenvalue if d is even, and opposite eigenvalues if d is odd, since, by eq. (3.12), $p_T(-x) = (-1)^d p_T(x)$. In the 2D case $n = 2$, one can parametrize the variables $x_1 = \cos \theta$, $x_2 = \sin \theta$ for $\theta \in [0, 2\pi)$ and show that p_T has at most $2d$ stationary points. For the 3D case $n = 3$, things are more complicated, and we rely on the work of Kuznetsov et al. (1992) who show that degree d polynomials on the sphere have at most $2(d^2 - d + 1)$ stationary points.

Symmetric tensor decomposition. Just like symmetric matrices, symmetric tensors admit rank one decompositions of the form

$$T = \sum_{i=1}^r \lambda_i v_i^{\otimes d}, \quad (3.17)$$

with vectors $v_1, \dots, v_r \in \mathbb{R}^n$ and scalars $\lambda_1, \dots, \lambda_r \in \mathbb{R}$. The smallest r for which such a decomposition exists is called the *rank* of T . Unlike matrices, the rank is not limited to the dimension n , and is as high as $\binom{n}{d}$ for a generic tensor. Finding tensor decompositions is an NP-hard problem (Hillar et al., 2013), with many applications in engineering.

The odeco variety. Orthogonally decomposable tensors, or *odeco* tensors, are symmetric tensors admitting an *orthogonal decomposition*

$$T = \sum_{i=1}^n \lambda_i v_i^{\otimes d}, \quad \text{with } v_1, \dots, v_n \text{ pairwise orthonormal.} \quad (3.18)$$

The rank of odeco tensors is obviously limited to n since one cannot find more than n pairwise orthogonal vectors in $\mathbb{R}^n$. One can check that the vectors $v_1, \dots, v_n$ are eigenvectors of T with corresponding $\lambda_1, \dots, \lambda_n$, thanks to the pairwise orthonormality. If we restrict fourth-order odeco tensors to have non-negative eigenvalues, $\lambda_i \geq 0$, we can establish a one-to-one correspondence between odeco tensors and the frame equivalence class we defined in eqs. (3.3) and (3.6):

$$\begin{bmatrix} \mathbf{v}_1 \\ \vdots \\ \mathbf{v}_n \end{bmatrix} \leftrightarrow \mathbf{T} = \sum_{i=1}^n \lambda_i \mathbf{u}_i^{\otimes 4}. \quad (3.19)$$

This odeco representation, proposed by Palmer et al. (2020) enables us to represent parametrization gradients without explicitly knowing the frame vectors, which removes the need for matchings between frames, and integer variables altogether.

As odeco tensors only form a small subset of the space of fully symmetric tensors, one needs to characterize what makes a tensor odeco. A major result of Robeva (2016) is the fact that odeco tensors form an algebraic variety defined

by a set of homogeneous quadratic equations in the tensor coefficients. More precisely, $\mathbf{T} \in S^4(\mathbf{R}^n)$ is odeco if and only if the contraction $\mathbf{T} * \mathbf{T}$ defined by

$$(\mathbf{T} * \mathbf{T})_{i_1 i_2 i_3 j_1 j_2 j_3} = \sum_{s=1}^n T_{i_1 i_2 i_3 s} T_{j_1 j_2 j_3 s} \quad (3.20)$$

is a fully symmetric tensor, i.e., $\mathbf{T} * \mathbf{T} \in S^6(\mathbf{R}^n)$. Note that $\mathbf{T} * \mathbf{T}$ already has many symmetries, so the number of equations guaranteeing its full symmetry is limited. In the fourth-order 2D case, for example, the odeco variety is defined by three nontrivial equations only:

$$\begin{aligned} (\mathbf{T} * \mathbf{T})_{111122} &= (\mathbf{T} * \mathbf{T})_{112112} &\Rightarrow T_{1111}T_{1122} + T_{1112}T_{1222} &= T_{1112}^2 + T_{1122}^2, \\ (\mathbf{T} * \mathbf{T})_{111222} &= (\mathbf{T} * \mathbf{T})_{112122} &\Rightarrow T_{1111}T_{1222} + T_{1112}T_{2222} &= T_{1112}T_{1122} + T_{1122}T_{1222}, \\ (\mathbf{T} * \mathbf{T})_{112222} &= (\mathbf{T} * \mathbf{T})_{122122} &\Rightarrow T_{1112}T_{1222} + T_{1122}T_{2222} &= T_{1122}^2 + T_{1222}^2. \end{aligned} \quad (3.21)$$

The number of equations defining the variety grows combinatorially with d and n . In the fourth-order 3D case, the number of linearly independent equations is 27.

3.3 Computing with odeco tensors

Orthogonally decomposable tensors form a subspace of fourth-order symmetric tensors: an algebraic variety defined by a set of homogeneous quadratic equations, or *quadrics*. In this section, we lay out some practical aspects that are needed to do computations with odeco tensors. First we show how symmetric tensors can be represented through a basis of *spherical harmonics*, which defines an adequate metric to compare tensors. Then, we show how a frame can be efficiently recovered from an odeco tensor using eigendecomposition algorithms. We recall the semidefinite optimization approach proposed by Palmer et al. (2020) to project a given tensor onto the odeco variety. Finally, we show how an odeco tensor's *second-order part* can be leveraged to manipulate the lengths of its underlying frame vectors.

Representing symmetric tensors: monomials and spherical harmonics. To store a symmetric tensor on a computer, taking into account the symmetries, one can simply store the coefficients of the corresponding polynomial eq. (3.11), $u_{i_1 \dots i_n}$ being the coefficient in front of monomial $x^{i_1} \dots x^{i_n}$, with $i_1 + \dots + i_n = d$. We relate the monomial coefficients $u_{i_1 \dots i_n}$ to the tensor

3.3. Computing with odeco tensors

coefficients $T_{j_1 \dots j_d}$:

$$\begin{aligned} p_T(x_1, \dots, x_n) &= \sum_{j_1, \dots, j_d=1}^n T_{j_1 \dots j_d} x_{j_1} \dots x_{j_d} \\ &= \sum_{i_1 + \dots + i_n = d} \underbrace{\binom{d}{i_1, \dots, i_n} T_{\underbrace{1 \dots 1}_{i_1 \text{ times}} \dots \underbrace{n \dots n}_{i_n \text{ times}}}}_{u_{i_1 \dots i_n}} x_1^{i_1} \dots x_n^{i_n}. \end{aligned} \quad (3.22)$$

This effectively removes the tensor symmetries, as $u_{i_1 \dots i_n}$ has 5 coefficients in 2D, and 15 in 3D.

Another option is to use the fact that polynomial $p_{\mathbf{T}}$ can be restricted to the unit sphere $\mathbb{S}_{n-1}$, and decompose it into a basis of *spherical harmonics*. To be able to evaluate the smoothness of a tensor field, we need to define a metric for our tensors, i.e., an inner product. A natural choice is the L^2 inner product of functions

$$\langle p^{(1)}, p^{(2)} \rangle = \int_{\mathbb{S}_{n-1}} p^{(1)}(\mathbf{x}) p^{(2)}(\mathbf{x}) d\mathbf{x}. \quad (3.23)$$

This inner product can be easily evaluated if the polynomials are expressed in a basis that is orthonormal with respect to the inner product. Let us concentrate on the $n = 3$ -dimensional case first. *Spherical harmonics* are a special set of functions having this property:

$$Y_l^m(\mathbf{x}), \quad l = 0, 1, \dots, \quad m = -l, \dots, l, \quad (3.24)$$

where l is the *band* of the harmonic and m its *order*. Spherical harmonics of even band are even functions; hence, our degree 4 polynomials can be decomposed into spherical harmonics of bands $l = 0, 2, 4$, which we write compactly as

$$p(\mathbf{x}) = \sum_{l \in \{0, 2, 4\}} \sum_{m=-l}^l q_l^m Y_l^m(\mathbf{x}). \quad (3.25)$$

Note that the number of terms in each band indeed sum to $1 + 5 + 9 = 15$. Spherical harmonics are orthonormal with respect to the defined inner product: $\langle Y_i, Y_j \rangle = \delta_{ij}$, and the inner product of polynomials can therefore be computed directly from the spherical harmonics coefficients:

$$\langle p^{(1)}, p^{(2)} \rangle = \sum_{i=1}^N q_i^{(1)} q_i^{(2)} = \langle \mathbf{q}^{(1)}, \mathbf{q}^{(2)} \rangle. \quad (3.26)$$

Note that the spherical harmonics coefficients q_i , contrary to the monomial coefficients u_i , do not depend on the choice of tensor basis $(\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3)$. Evaluating the distance between tensors can then be done by evaluating the Euclidean distance between their spherical harmonics coefficients:

$$d(\mathbf{T}^{(1)}, \mathbf{T}^{(2)}) = \left\| \mathbf{q}^{(2)} - \mathbf{q}^{(1)} \right\|. \quad (3.27)$$

These properties make the spherical harmonics a very convenient choice for computing with tensors. Note that spherical harmonics are usually defined for a sphere in $\mathbb{R}^3$; however, they have a 2D counterpart, sometimes called *circular harmonics*, which actually correspond to the basis functions of a *Fourier series*:

$$\underbrace{1}_{\text{band 0}}, \underbrace{\cos(2\theta), \sin(2\theta)}_{\text{band 2}}, \underbrace{\cos(4\theta), \sin(4\theta)}_{\text{band 4}}. \quad (3.28)$$

Palmer et al. (2020) have shown that each band has a distinct meaning. Band 0 encodes the tensor *size*, since

$$q_0 = C_0 \sum_m \lambda_m. \quad (3.29)$$

Band 2 encodes the tensor *anisotropy*, since

$$\|\mathbf{q}_2\|^2 = C_2 \sum_{m=1}^n (\lambda_{m+1} - \lambda_m)^2. \quad (3.30)$$

In particular, the tensor is isotropic iff $\|\mathbf{q}_2\| = 0$. Band 4 encodes the tensor *orientation*, since $\mathbf{q}_4$ is (up to a scalar multiple) the spherical harmonics decomposition of the unit frame tensor $\mathbf{T} = \sum_{m=1}^n \mathbf{u}_m^{\otimes 4}$.

Recovering a frame from an odeco tensor. Given an odeco tensor, the corresponding frame can be recovered using a tensor power iteration, as suggested by Robeva (2016):

$$\mathbf{u} \mapsto \frac{\mathbf{T} \cdot \mathbf{u}^{d-1}}{\|\mathbf{T} \cdot \mathbf{u}^{d-1}\|}. \quad (3.31)$$

Under certain conditions, this iteration converges towards an eigenvector of $\mathbf{T}$. However, it has slow convergence and is not even guaranteed to converge when $\mathbf{T}$ is not exactly odeco. Instead, we propose a simple procedure, relying on matrix eigendecompositions.

In the context of linear elasticity, Hooke's law can be generalized to three-dimensional continuous media:

$$\boldsymbol{\sigma} = \mathbf{C} : \boldsymbol{\epsilon}, \quad (3.32)$$

where $\boldsymbol{\sigma}$ and $\boldsymbol{\epsilon}$ are the stress and strain tensors (of rank two), and $\mathbf{C}$ is a fourth-order *elasticity tensor*. This tensor has the same dimensions as ours, albeit less symmetries. In order to perform the required tensor operations, the *Mandel notation* was introduced, mapping a 3×3 tensor to a 6×1 vector and a $3 \times 3 \times 3 \times 3$ tensor to a 6×6 matrix:

$$\tilde{\boldsymbol{\sigma}} = \begin{pmatrix} \sigma_{11} \\ \sigma_{22} \\ \sigma_{33} \\ \sqrt{2}\sigma_{23} \\ \sqrt{2}\sigma_{13} \\ \sqrt{2}\sigma_{12} \end{pmatrix} \quad \tilde{\mathbf{C}} = \begin{pmatrix} C_{1111} & C_{1122} & C_{1133} & \sqrt{2}C_{1123} & \sqrt{2}C_{1113} & \sqrt{2}C_{1112} \\ C_{2211} & C_{2222} & C_{2233} & \sqrt{2}C_{2223} & \sqrt{2}C_{2213} & \sqrt{2}C_{2212} \\ C_{3311} & C_{3322} & C_{3333} & \sqrt{2}C_{3323} & \sqrt{2}C_{3313} & \sqrt{2}C_{3312} \\ \sqrt{2}C_{2311} & \sqrt{2}C_{2322} & \sqrt{2}C_{2333} & 2C_{2323} & 2C_{2313} & 2C_{2312} \\ \sqrt{2}C_{1311} & \sqrt{2}C_{1322} & \sqrt{2}C_{1333} & 2C_{1323} & 2C_{1313} & 2C_{1312} \\ \sqrt{2}C_{1211} & \sqrt{2}C_{1222} & \sqrt{2}C_{1233} & 2C_{1223} & 2C_{1213} & 2C_{1212} \end{pmatrix} \quad (3.33)$$

3.3. Computing with odeco tensors

This notation has useful properties: the dot product and the matrix-vector product are preserved,

$$\boldsymbol{\sigma}_1 : \boldsymbol{\sigma}_2 = \tilde{\boldsymbol{\sigma}}_1 \cdot \tilde{\boldsymbol{\sigma}}_2, \quad \tilde{\boldsymbol{\sigma}} = \tilde{\mathbf{C}} \tilde{\boldsymbol{\epsilon}}, \quad (3.34)$$

which enables the use of the standard matrix storage and algorithms.

Consider an odeco tensor $\mathbf{T} = \sum_k \lambda_k \mathbf{u}_k^{\otimes 4}$, and let the 3×3 matrices $\mathbf{D}_k \triangleq \mathbf{u}_k^{\otimes 2}$. In Mandel notation we have

$$\tilde{\mathbf{T}} = \sum_{k=1}^3 \lambda_k \tilde{\mathbf{D}}_k. \quad (3.35)$$

The 6×1 vectors $\tilde{\mathbf{D}}_k$ are orthonormal since $\tilde{\mathbf{D}}_k \cdot \tilde{\mathbf{D}}_l = \mathbf{u}_k^{\otimes 2} : \mathbf{u}_l^{\otimes 2} = \delta_{kl}$, and therefore eq. (3.35) is an eigendecomposition (note that, if $\mathbf{T}$ is not odeco, $\tilde{\mathbf{T}}$ has up to 6 eigenvectors). If the λ_k are distinct, then the eigendecomposition of $\tilde{\mathbf{T}}$ is unique and the frame vectors are easily computed. Suppose now that the eigenvalues are all equal: $\lambda_k = \lambda$. Then the eigendecomposition of $\tilde{\mathbf{T}}$ correctly recovers λ , but the eigenvectors are mixed in the eigenmatrices $\mathbf{D}_k$:

$$\mathbf{D}_k = \sum_{l=1}^3 \alpha_{kl} \mathbf{u}_l^{\otimes 2}, \quad (3.36)$$

with $\sum_k \alpha_{kl} = 1$. To recover the eigenvectors $\mathbf{u}_m$ in a way that is robust to their algebraic multiplicity, we construct a matrix $\mathbf{M}$ as the weighted sum of matrices $\mathbf{D}_k$ with random weights β_k :

$$\mathbf{M} = \sum_{k=1}^3 \beta_k \mathbf{D}_k = \sum_{l=1}^3 \left(\sum_{k=1}^3 \alpha_{kl} \beta_k \right) \mathbf{u}_l^{\otimes 2}. \quad (3.37)$$

The random weights guarantee that $\mathbf{M}$ has distinct eigenvalues, which allows us to easily recover vectors $\mathbf{u}_l$. In the case where $\mathbf{T}$ is not exactly odeco, we simply take the 3 largest terms (out of 6) in the eigendecomposition of $\tilde{\mathbf{T}}$. If $\mathbf{T}$ is close to odeco, then these 3 largest terms are dominating the decomposition.

The full procedure for recovering the frame vectors from a (nearly) odeco tensor is summarized in algorithm 1. Note that this method is computationally very cheap as it requires only the eigendecomposition of a 6×6 matrix.

Projecting onto the odeco variety. A proper projection onto the odeco variety $\mathcal{F}$ finds the tensor $\mathbf{q}$ that is closest (in the L^2 metric defined above) to a given tensor $\mathbf{y}$:

$$\Pi_{\mathcal{F}}(\mathbf{y}) = \arg \min_{\mathbf{q} \in \mathcal{F}} \|\mathbf{q} - \mathbf{y}\|^2. \quad (3.38)$$

This problem is a quadratically constrained quadratic program (QCQP) and is generally NP-hard. However, Palmer et al. (2020) have shown empirically

Algorithm 1 Recover frame from (nearly) odeco tensor

Input: A 4th-order symmetric tensor $\mathbf{T} \in S^4(\mathbb{R}^3)$.

Output: Frame vectors $\lambda_1 \mathbf{u}_1, \lambda_2 \mathbf{u}_2, \lambda_3 \mathbf{u}_3$. The recovery is exact if $\mathbf{T}$ is odeco.

Using the Mandel notation, eq. (3.33), find an eigendecomposition $\mathbf{T} := \sum_{m=1}^6 \lambda_k \mathbf{D}_k^{\otimes 2}$.
 Keep only the 3 terms with largest eigenvalue: $\lambda_1 \mathbf{D}_1, \lambda_2 \mathbf{D}_2, \lambda_3 \mathbf{D}_3$.
 Generate random weights $\beta_1, \beta_2, \beta_3$ and define $\mathbf{M} = \sum_{k=1}^3 \beta_k \mathbf{D}_k$.
 Compute eigendecomposition of $\mathbf{M}$, providing vectors $\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3$.

that its semidefinite relaxation gives the exact projection, provided that the polynomial $\mathbf{y}$ is positive. The intuition of the semidefinite relaxation is to look for a positive semidefinite matrix $\mathbf{Q} \in \mathbb{R}^{(N+1) \times (N+1)}$ that is a proxy for $\mathbf{q}\mathbf{q}^\top$. This gives a semidefinite problem

$$\begin{aligned} & \min_{\mathbf{Q} \in \mathbb{R}^{N \times N}} \langle \mathbf{Y}, \mathbf{Q} \rangle \\ & \text{subject to } Q_{11} = 1 \\ & \quad \langle \tilde{\mathbf{A}}_i, \mathbf{Q} \rangle = 0, \quad i = 1, \dots, M \\ & \quad \mathbf{Q} \succeq 0, \end{aligned} \tag{3.39}$$

where

$$\mathbf{Y} = \begin{pmatrix} \|\mathbf{y}\|^2 & -\mathbf{y}^\top \\ -\mathbf{y} & \mathbf{I}_{15 \times 15} \end{pmatrix} \quad \text{and} \quad \tilde{\mathbf{A}}_i = \begin{pmatrix} 0 & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_i \end{pmatrix}, \tag{3.40}$$

$\mathbf{A}_i$ being the M symmetric matrices defining the odeco variety. A dimension is added to $\mathbf{Q}$ to have access to the linear terms $\mathbf{q}$ and a constant term (top-left entry). If the optimal solution $\mathbf{Q}^*$ happens to be of rank 1, then it can be decomposed as

$$\mathbf{Q}^* = \begin{pmatrix} 1 \\ \mathbf{q}^* \end{pmatrix} \begin{pmatrix} 1 \\ \mathbf{q}^* \end{pmatrix}^\top, \tag{3.41}$$

and $\mathbf{q}^*$ is then the exact projection of eq. (3.38). This projection is much more costly than the one based on eigendecompositions (algorithm 1), but it is conjectured to be exact whatever the distance to the variety.

Second-order part of odeco tensors. A very useful tool when manipulating odeco tensors is to look at its *second-order part* $\mathbf{M}$, defined as the second-order tensor (or $n \times n$ matrix)

$$M_{ij} = T_{ijkk}, \tag{3.42}$$

obtained by contracting two of its indices. Note that, thanks to the symmetry of $\mathbf{T}$, it does not matter on which two indices the contraction is done, and $\mathbf{M}$

3.3. Computing with odeco tensors

is a symmetric matrix. Plugging in the odeco definition, eq. (3.18), yields

$$M_{ij} = T_{ijkk} = \sum_{m=1}^n \lambda_m u_i^m u_j^m u_k^m u_k^m = \sum_{m=1}^n \lambda_m u_i^m u_j^m, \quad \text{and} \quad \mathbf{M} = \sum_{m=1}^n \lambda_m (\mathbf{u}_m)^{\otimes 2}, \quad (3.43)$$

where we used the fact that vectors $\mathbf{u}^m$ have unit norm. In other words, the second-order part of a fourth-order odeco tensor is a matrix with the same eigenvalues and eigenvectors. This matrix is very useful since it allows us to scale the eigenvectors of odeco tensor $\mathbf{T}$. First notice that taking matrix $\mathbf{M}$ to the power p changes its eigenvalues accordingly:

$$\mathbf{M}^p = \sum_{m=1}^n \lambda_m^p \mathbf{u}_m^{\otimes 2}. \quad (3.44)$$

And contracting the tensor with this matrix yields the tensor with modified sizes

$$\mathbf{T}^{p+1} \triangleq \mathbf{T} \cdot \mathbf{M}^p = \left(\sum_{m=1}^n \lambda_m^p \mathbf{u}_m^{\otimes 2} \right) \cdot \left(\sum_{m=1}^n \lambda_m \mathbf{u}_m^{\otimes 4} \right) = \sum_{m=1}^n \lambda_m^{p+1} \mathbf{u}_m^{\otimes 4}. \quad (3.45)$$

Integer power p can be chosen arbitrarily and matrix $\mathbf{M}^p$ can be computed using standard matrix algorithms; in particular, $p = -1$ gives a unitary (or *octahedral*) frame tensor $\mathbf{T}^0$ and $p = -2$ inverts the lengths of the frame vectors. We demonstrate in chapter 4 how this tool enables us to properly normalize the integrability energy.

Boundary alignment. It is crucial to be able to impose alignment of a frame field to the boundary or features of the geometry; we explain how it is done using odeco tensors. First notice that a rotation of a spherical polynomial is done by rotating its spherical harmonics coefficients in $\mathbb{R}^{15}$: indeed, rotating the polynomials should preserve the inner product $\langle \cdot, \cdot \rangle$, which, we have shown, corresponds to the inner product in $\mathbb{R}^{15}$; such transformations are called *isometric* and are described by rotation matrices in $\mathbb{R}^{15}$. As Palmer et al. (2020) have shown, these matrices that rotate spherical harmonics are the *Wigner D-matrices*, an essential tool of quantum physics for the study of angular momentum. Consider the space of odeco frames $\mathbf{q}$ that are aligned to axis $\hat{\mathbf{e}}_z$, and have a length of 1 in this direction. This space can be parametrized by the affine transformation

$$\mathbf{q} = \mathbf{q}_z + \mathbf{B}_z \mathbf{s}, \quad (3.46)$$

where $\mathbf{s} \in \mathbb{R}^5$ represents a 2D odeco tensor, and $\mathbf{q}_z \in \mathbb{R}^{15}$ and $\mathbf{B}_z \in \mathbb{R}^{15 \times 5}$ are constants. Then, the space of odeco frames that are aligned to an arbitrary surface normal $\mathbf{n}$ (we will heavily use this constraint for computing 3D frame fields), is obtained by applying any rotation matrix $\mathbf{R}_n \in \mathbb{R}^{15 \times 15}$ that brings $\hat{\mathbf{e}}_z$ to $\mathbf{n}$:

$$\mathbf{q} = \mathbf{R}_n(\mathbf{q}_z + \mathbf{B}_z \mathbf{s}) = \mathbf{q}_n + \mathbf{B}_n \mathbf{s}. \quad (3.47)$$

For our purposes, instead of such an affine parametrization, we are interested in finding an affine equation

$$\mathbf{A}_n \mathbf{q} + \mathbf{b}_n = \mathbf{0} \quad (3.48)$$

that is verified by any $\mathbf{q}$ respecting the alignment constraint. We propose a simple procedure that does not require to figure out $\mathbf{R}_n$ nor $\mathbf{B}_z$ (or even its dimension), and can be adapted to any kind of alignment constraints (e.g., feature edge or corner constraints). First form a matrix $\mathbf{Q}$ that contains as columns a batch of random tensors respecting the *linear part* of the alignment constraint; in this case, frames aligned to $\mathbf{n}$ but having zero length in that direction. Let $\mathbf{N}$ be a basis of the null space of $\mathbf{Q}^\top$, such that $\mathbf{N}\mathbf{Q}^\top = \mathbf{0}$; we then have $\mathbf{A}_n = \mathbf{N}^\top$. Now let $\mathbf{q}_0$ be any tensor respecting the affine constraint; then $\mathbf{b}_n = -\mathbf{A}_n \mathbf{q}_0$. This procedure gives us the affine equation, eq. (3.48).

Eigenvalue sensitivity for tensors. A last tool we add to our "tensor toolbox" is a result on eigenvalue sensitivity. It answers the question "*given an odeco tensor, how do its eigenvectors change when slightly perturbing its coefficients?*". This tool will help us in the next chapter to express integrability in terms of the tensor coefficients, as it can transform spatial derivatives on the frame vectors into spatial derivatives on the tensor.

This *eigenvalue perturbation problem* is well-known for matrices, but is easily generalized to higher-order tensors, provided they are odeco. Consider an odeco tensor $\mathbf{T} = \sum_{m=1}^n \lambda_m \mathbf{u}_m^{\otimes 4}$, or, in index notation,

$$T_{j_1 j_2 j_3 j_4} = \sum_{m=1}^n \lambda_m u_{j_1}^m u_{j_2}^m u_{j_3}^m u_{j_4}^m. \quad (3.49)$$

Consider the contraction of $\mathbf{T}$ with one of its eigenvectors, $\mathbf{u}_k$:

$$\begin{aligned} T_{j_1 j_2 j_3 j_4} u_{j_4}^k &= \sum_{m=1}^n \lambda_m u_{j_1}^m u_{j_2}^m u_{j_3}^m u_{j_4}^m u_{j_4}^k \\ &= \sum_{m=1}^n \lambda_m u_{j_1}^m u_{j_2}^m u_{j_3}^m \delta_{mk} \\ &= \lambda_k u_{j_1}^k u_{j_2}^k u_{j_3}^k, \end{aligned} \quad (3.50)$$

or, written compactly, $\mathbf{T} \cdot \mathbf{u}_k = \lambda_k \mathbf{u}_k^{\otimes 3}$. Contracting again and following the same procedure we find that $\mathbf{T} \cdot \mathbf{u}_k^{\otimes 2} = \lambda_k \mathbf{u}_k^{\otimes 2}$ and $\mathbf{T} \cdot \mathbf{u}_k^{\otimes 3} = \lambda_k \mathbf{u}_k$, the latter being the definition of a tensor eigenvector. Consider now a specific eigenpair $(\lambda, \mathbf{w})$, with $\mathbf{w}$ having unit norm. We wish to express the variation of the eigenpair $(\delta\lambda, \delta\mathbf{w})$ given some perturbation of the tensor $\delta\mathbf{T}$. We write the remainder of the proof in compact notation for brevity, but the same steps can be performed in index notation. Since $\mathbf{w}$ has unit norm, it is orthogonal to its variation:

$$(\mathbf{w} + \delta\mathbf{w}) \cdot (\mathbf{w} + \delta\mathbf{w}) = 1 \quad \Rightarrow \quad \mathbf{w} \cdot \delta\mathbf{w} = 0, \quad (3.51)$$

3.3. Computing with odeco tensors

neglecting the higher-order terms $(\delta \mathbf{w} \cdot \delta \mathbf{w})$. Writing the eigenvector definition $\mathbf{T} \mathbf{w}^{\otimes 3} = \lambda \mathbf{w}$ for the new eigenpair,

$$\begin{aligned}
 (\mathbf{T} + \delta \mathbf{T})(\mathbf{w} + \delta \mathbf{w})^{\otimes 3} &= \lambda \mathbf{w} + \delta(\lambda \mathbf{w}), \\
 \cancel{\mathbf{T} \mathbf{w}^3} + 3\mathbf{T} \mathbf{w}^2 \delta \mathbf{w} + \delta \mathbf{T} \mathbf{w}^3 &= \cancel{\lambda \mathbf{w}} + \delta(\lambda \mathbf{w}) && \text{(eigenvector definition),} \\
 \cancel{3\lambda \mathbf{w}^2 \delta \mathbf{w}} + \delta \mathbf{T} \mathbf{w}^3 &= \delta(\lambda \mathbf{w}) && (\mathbf{T} \mathbf{w}^2 = \lambda \mathbf{w}^2 \text{ and } \mathbf{w} \cdot \delta \mathbf{w} = 0).
 \end{aligned} \tag{3.52}$$

We find the final eigenvalue sensitivity result: $\delta(\lambda \mathbf{w}) = \delta \mathbf{T} \mathbf{w}^3$; the variation in a scaled eigenvector is obtained by contracting the tensor perturbation three times with the unit eigenvector (note that an eigenvector's sensitivity only depends on itself and not explicitly on the other eigenvectors). From this result we find the partial derivatives

$$\frac{\partial(\lambda w_i)}{\partial T_{j_1 j_2 j_3 j_4}} = w_{j_1} w_{j_2} w_{j_3} \delta_{i j_4}. \tag{3.53}$$

Chapter 4 will show how this result is put to use to derive an integrability expression.

CHAPTER 4

Integrable frame fields using odeco tensors

The contents of this chapter partially overlap with the paper

Mattéo Couplet, Alexandre Chemin and Jean-François Remacle (2024). Integrable Frame Fields using Odeco Tensors. In *Proceedings of the 2024 SIAM International Meshing Roundtable* (p. 53-65).

In chapter 1 we have seen that a frame field is integrable, i.e., is the Jacobian of a global seamless parametrization, if it is *curl-free*. In chapter 3 we have presented how 2D and 3D orthogonal frames can be encoded by orthogonally decomposable (odeco) tensors, and we have detailed a set of tools to manipulate them. In this chapter we show that the frame field curl can be expressed as a function of the corresponding odeco tensor field, which constitutes the major result of this thesis. We formulate the search for an integrable frame field as a non-convex optimization problem on a tensor field, and show how our approach can compute seamless parametrizations of planar domains while precisely respecting sizing or anisotropy constraints. Finally, in 3D, we show that our solver can find globally meshable singularity graphs that were previously impossible to obtain otherwise than with explicit repair techniques. Coupling our frame field solver with a state-of-the-art hex mesher, we produce desirable full-hex meshes on usually hard-to-mesh geometries.

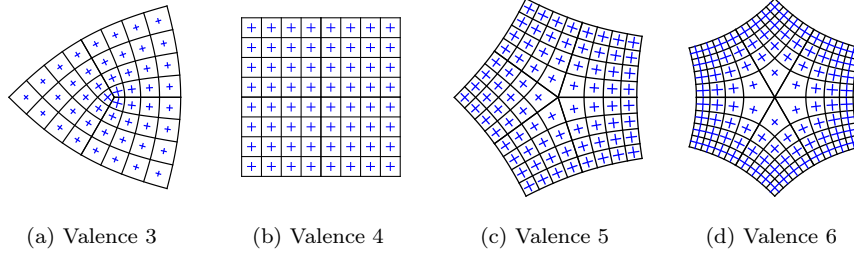


Figure 4.1: Parametrization isolines (in black) and their gradient frames (in blue) defined by eqs. (4.1) and (4.4) for a range of singular indices. Notice how the frame field grows or shrinks according to the singularity index.

Frame field singularities

The behavior of the parametrization gradient deserves special attention in the vicinity of singularities, as the corresponding frame can either vanish or blow up. Consider a polar coordinate system $(\hat{\mathbf{e}}_r, \hat{\mathbf{e}}_\theta)$ with a singularity located at the origin, and a cut along $\theta = 0$. The parametrization jumps by an angle of $k\pi/2$ and thus the singularity has index $k/4$. A natural definition for the coordinate maps is

$$\begin{aligned} u(r, \theta) &= f(r) \cos((1 - k/4)\theta), \\ v(r, \theta) &= f(r) \sin((1 - k/4)\theta), \end{aligned} \quad (4.1)$$

where $f(r)$ is an arbitrary strictly increasing function. The parametrization gradients are given by

$$\begin{aligned} \nabla u &= f'(r) \cos((1 - k/4)\theta) \hat{\mathbf{e}}_r - (1 - k/4) \frac{f(r)}{r} \sin((1 - k/4)\theta) \hat{\mathbf{e}}_\theta, \\ \nabla v &= f'(r) \sin((1 - k/4)\theta) \hat{\mathbf{e}}_r + (1 - k/4) \frac{f(r)}{r} \cos((1 - k/4)\theta) \hat{\mathbf{e}}_\theta. \end{aligned} \quad (4.2)$$

Note that since both gradients have the same amplitude, this parametrization is *isotropic*: at any point, the isolines of u and v are equally spaced. If we now restrict the parametrization to be orthogonal, we want ∇u and ∇v to be orthogonal:

$$\nabla u \cdot \nabla v = (f'(r))^2 \cos(\cdot) \sin(\cdot) - (1 - k/4)^2 \left(\frac{f(r)}{r} \right)^2 \cos(\cdot) \sin(\cdot) = 0, \quad (4.3)$$

which is the case when $f'(r) = (1 - k/4)f(r)/r$. The solution to this differential equation is

$$f(r) = c r^{1-k/4}, \quad (4.4)$$

where c is an arbitrary positive constant. Plugging this back into eq. (4.2) we find that

$$\|\nabla u\| = \|\nabla v\| = c(1 - k/4)r^{-k/4}. \quad (4.5)$$

4.1. Curl of an odeco tensor field

The calculated parametrization and corresponding gradients are illustrated on fig. 4.1 for a range of singular indices. This means that for singularities of index $+1/4$ ($k = 1$) and higher, the frame field blows up when approaching the singular point, and for singularities of index $-1/4$ ($k = -1$) and lower, the frame field vanishes at the singular point. This observation is particularly important for designing our computational methods as they must allow both kinds of frame field singularities to appear. Even though we derived this for a specific parametrization expression, eq. (4.1), we found that the conclusion remains the same for the various frame fields we compute with our methods.

4.1 Curl of an odeco tensor field

Consider an n -dimensional frame field $\mathbf{F}(\mathbf{x})$ on a domain $\Omega \subset \mathbb{R}^n$. At each non-singular point $\mathbf{x} \in \Omega$, a local lifting of frame vectors $\mathbf{v}_1(\mathbf{x}'), \dots, \mathbf{v}_n(\mathbf{x}')$ can be defined on a small neighborhood $\mathcal{N}$ of $\mathbf{x}$. We assume the frame field respects the local injectivity property: $\det(\mathbf{v}_1 \ \cdots \ \mathbf{v}_n) > 0$; this means that three vectors cannot be coplanar, two vectors cannot be colinear, and a vector cannot vanish. As we stated in chapter 1, the frame field $\mathbf{F}(\mathbf{x})$ is *integrable* if its frame vectors are curl-free everywhere:

$$\nabla \times \mathbf{v}_m(\mathbf{x}) = \mathbf{0}, \quad m = 1, \dots, n. \quad (4.6)$$

An equivalent statement is that the squared norms of the frame vector curls sum to zero:

$$\sum_{m=1}^n \|\nabla \times \mathbf{v}_m(\mathbf{x})\|^2 = 0. \quad (4.7)$$

The attractiveness of this statement is that the left-hand side does not depend on the choice of local lifting for the n frame vectors, as it is invariant under octahedral permutations of the vectors. We perform a normalization by the squared norm of the frame vectors to make the expression dimensionless, which defines the *normalized squared curl*

$$h_{\mathbf{F}}(\mathbf{x}) \triangleq \sum_{m=1}^n \frac{\|\nabla \times \mathbf{v}_m\|^2}{\|\mathbf{v}_m\|^2}. \quad (4.8)$$

At the end of this section we elaborate on why this normalization is particularly important for the optimization process. In chapter 3 we have seen that there is a one-to-one correspondence between locally injective orthogonal frames $\mathbf{F}$ and odeco tensors $\mathbf{T}$. This means that the integrability measure $h_{\mathbf{F}}$ can be written as a measure $h_{\mathbf{T}}$ on the odeco tensor field. The main challenge is to find a closed-form expression of $h_{\mathbf{T}}$ in terms of the tensor coefficients. This expression is a major outcome of our work, and the remainder of this section is dedicated to show how it is constructed.

Let $\mathbf{T}$ be an odeco tensor field decomposing as

$$T_{k_1 k_2 k_3 k_4} = \sum_{m=1}^n \lambda_m u_{k_1}^m u_{k_2}^m u_{k_3}^m u_{k_4}^m, \quad (4.9)$$

where $\lambda_m = \|\mathbf{v}_m\|$ is the norm of the m -th frame vector, and $\mathbf{u}_m = \mathbf{v}_m/\lambda_m$ is its normalized version. In the following development, all repeated indices correspond to implicit summations. We start by developing the curl of a given arbitrary eigenvector $\mathbf{v} = \lambda \mathbf{u}$ of $\mathbf{T}$:

$$\begin{aligned} (\nabla \times \mathbf{v})_j &= \epsilon_{jab} \frac{\partial v_b}{\partial x_a} && (\epsilon \text{ is the Levi-Civita symbol}) \\ &= \epsilon_{jab} \frac{\partial v_b}{\partial T_{k_1 k_2 k_3 k_4}} \frac{\partial T_{k_1 k_2 k_3 k_4}}{\partial x_a} && (\text{chain rule}) \\ &= \epsilon_{jab} u_{k_1} u_{k_2} u_{k_3} \delta_{k_4 b} \frac{\partial T_{k_1 k_2 k_3 k_4}}{\partial x_a} && (\text{eigenvector sensitivity; eq. (3.53)}) \\ &= u_{k_1} u_{k_2} u_{k_3} \epsilon_{jab} \frac{\partial T_{k_1 k_2 k_3 b}}{\partial x_a} && (\text{sum on } k_4) \\ &= u_{k_1} u_{k_2} u_{k_3} (\nabla \times \mathbf{T}_{k_1 k_2 k_3})_j && (\text{curl definition}), \end{aligned} \quad (4.10)$$

where $\mathbf{T}_{k_1 k_2 k_3} \in \mathbb{R}^n$ is the vector with components $(T_{k_1 k_2 k_3 1}, \dots, T_{k_1 k_2 k_3 n})$. The objective, now, is to make the frame vectors u_k disappear on the right-hand side, to obtain an expression that only depends on the tensor coefficients instead. To do so, we multiply the expression by the i -th component of $\mathbf{v}$ (notice that i and j are free indices here):

$$v_i (\nabla \times \mathbf{v})_j = \lambda u_{k_1} u_{k_2} u_{k_3} u_i (\nabla \times \mathbf{T}_{k_1 k_2 k_3})_j. \quad (4.11)$$

Summing this expression for each of the eigenvectors $\mathbf{v}_m$ makes the tensor coefficients appear on the right-hand side:

$$\begin{aligned} \sum_{m=1}^n v_i^m (\nabla \times \mathbf{v}_m)_j &= \sum_{m=1}^n \lambda_m u_{k_1}^m u_{k_2}^m u_{k_3}^m u_i^m (\nabla \times \mathbf{T}_{k_1 k_2 k_3})_j \\ &= T_{k_1 k_2 k_3 i} (\nabla \times \mathbf{T}_{k_1 k_2 k_3})_j \quad \text{by eq. (4.9)}. \end{aligned} \quad (4.12)$$

Writing this compactly using the outer product ($\otimes$) and with explicit summations yields the $n \times n$ matrix

$$\sum_{m=1}^n \mathbf{v}_m \otimes (\nabla \times \mathbf{v}_m) = \sum_{k_1, k_2, k_3=1}^n \mathbf{T}_{k_1 k_2 k_3} \otimes (\nabla \times \mathbf{T}_{k_1 k_2 k_3}). \quad (4.13)$$

This is a powerful result: with the frame vectors $\mathbf{v}_m$ being linearly independent, the frame field represented by tensor field $\mathbf{T}$ is curl-free if the right-hand side matrix is zero. We are one step away from crafting the normalized squared curl

4.1. Curl of an odeco tensor field

defined by eq. (4.8). Remember that in chapter 3 matrix $\mathbf{M}$ was defined through the contraction $M_{ij} = \sum_k T_{ijkk}$, and has the frame vectors as eigenvectors: $\mathbf{M} = \sum_m \lambda_m \mathbf{u}_m^{\otimes 2}$. We use this matrix as a tool to normalize our curl expression: applying $\mathbf{M}^{-2} = \sum_m \lambda_m^{-2} \mathbf{u}_m^{\otimes 2}$ to both sides of eq. (4.13) yields

$$\sum_{m=1}^n \underbrace{\mathbf{M}^{-2} \mathbf{v}_m}_{\mathbf{u}_m / \lambda_m} \otimes (\nabla \times \mathbf{v}_m) = \sum_{k_1, k_2, k_3=1}^n \underbrace{\mathbf{M}^{-2} \mathbf{T}_{k_1 k_2 k_3}}_{\mathbf{T}_{k_1 k_2 k_3}^{-1}} \otimes (\nabla \times \mathbf{T}_{k_1 k_2 k_3}). \quad (4.14)$$

Finally, taking the squared Frobenius norm $\|\mathbf{A}\|_{\mathbf{F}}^2 = A_{ij} A_{ij}$ on both sides gives us the desired curl measure

$$\begin{aligned} h_{\mathbf{T}} &\triangleq \left\| \sum_{k_1, k_2, k_3=1}^n \mathbf{T}_{k_1 k_2 k_3}^{-1} \otimes (\nabla \times \mathbf{T}_{k_1 k_2 k_3}) \right\|_{\mathbf{F}}^2 = \left\| \sum_{m=1}^n \frac{\mathbf{u}_m}{\lambda_m} \otimes (\nabla \times \mathbf{v}_m) \right\|_{\mathbf{F}}^2 \\ &= \sum_{m_1, m_2=1}^n \frac{u_{m_1}^{m_1}}{\lambda_{m_1}} (\nabla \times \mathbf{v}_{m_1})_j \frac{u_{m_2}^{m_2}}{\lambda_{m_2}} (\nabla \times \mathbf{v}_{m_2})_j \\ &= \sum_{m=1}^n \frac{\|\nabla \times \mathbf{v}_m\|^2}{\lambda_m^2} \triangleq h_{\mathbf{F}}. \end{aligned} \quad (4.15)$$

Note that the length λ_m in the denominator can be taken to any given power: if $\mathbf{M}$ is taken to the power $-p$, then $\mathbf{T}$ is to the power $1-p$ in $h_{\mathbf{T}}$, and λ_m is to the power $2-2p$. Choosing $p=2$ gives the desired expression $h_{\mathbf{F}}$, and $p=0$ gives the unnormalized squared curl from eq. (4.7). We distinguish $h_{\mathbf{F}}$ from $h_{\mathbf{T}}$ which extends to all symmetric tensors (not necessarily odeco).

Lie bracket of an odeco tensor field. In chapter 1 we have shown that a frame field can alternatively represent a parametrization's coordinate vectors instead of its gradients. In that case the integrability of the frame field is characterized by its *Lie bracket*. In (Couplet et al., 2024) we have shown, using the same machinery as for the curl, that the Lie bracket of an odeco tensor field $\mathbf{T}(\mathbf{x})$ can be expressed as

$$[\mathbf{v}_1, \mathbf{v}_2] = \epsilon_{j_2 k_2} \epsilon_{j_3 k_3} \epsilon_{j_4 k_4} (T_{k_2 k_3 k_4} \cdot \nabla T_{ij_2 j_3 j_4}), \quad (4.16)$$

where ϵ_{ij} is the two-dimensional Levi-Civita symbol. In the 3D case, we have tried to find an expression for the sum of normalized squared pairwise Lie brackets

$$\frac{[\mathbf{v}_1, \mathbf{v}_2]^2}{(\lambda_1 \lambda_2)^2} + \frac{[\mathbf{v}_2, \mathbf{v}_3]^2}{(\lambda_2 \lambda_3)^2} + \frac{[\mathbf{v}_3, \mathbf{v}_1]^2}{(\lambda_3 \lambda_1)^2}. \quad (4.17)$$

This quantity can be expressed as a fraction of two degree-6 polynomials in the tensor coefficients and their gradients. However, the numerator cannot be expressed in a factorized form as for the curl, which means that over 100000 scalar

coefficients are needed; this is impractical for computations. This complexity is the reason for which the gradient representation was chosen, as the integrability criterion can be expressed in a closed form and computed efficiently.

4.2 Curl optimization

We formulate the search for an integrable frame field as an optimization problem striving to minimize the integral of the integrability measure $h_{\mathbf{T}}$ over the domain Ω ,

$$H[\mathbf{T}] = \int_{\Omega} h_{\mathbf{T}}(\mathbf{x}) \, d\mathbf{x}, \quad (4.18)$$

which we call the *curl energy*. We start by explaining how this integral is computed in practice, then we present the optimization approach.

Discretization and integration. The computations are supported by a standard triangle mesh in 2D and tetrahedral mesh in 3D. At each mesh node we store the tensors as their spherical harmonics coefficients $\mathbf{q} \in \mathbb{R}^N$ ($N = 5$ in 2D, and $N = 15$ in 3D). These coefficients are interpolated throughout the domain by a continuous function $\mathbf{q}(\mathbf{x})$ that is piecewise linear on the mesh elements. To avoid redundancy in the sequel we restrict ourselves to the 3D case, but the 2D setup follows naturally. Consider a tetrahedron $\mathcal{T}$ with node positions $\mathbf{x}^1, \mathbf{x}^2, \mathbf{x}^3, \mathbf{x}^4$, nodal values $\mathbf{q}^1, \mathbf{q}^2, \mathbf{q}^3, \mathbf{q}^4$ and corresponding linear shape functions $\psi^1(\mathbf{x}), \psi^2(\mathbf{x}), \psi^3(\mathbf{x}), \psi^4(\mathbf{x})$ (such that $\psi^i(\mathbf{x}^j) = \delta_{ij}$). The interpolant over the tetrahedron is defined by

$$\mathbf{q}(\mathbf{x}) = \sum_{i=1}^4 \mathbf{q}^i \psi^i(\mathbf{x}); \quad (4.19)$$

note that even if the nodal tensors $\mathbf{q}^i$ were odeco, the interpolated tensor at $\mathbf{q}(\mathbf{x})$ would not be odeco in general. Computing the tensor curl $h_{\mathbf{T}}$ requires evaluating the spatial derivatives of $\mathbf{q}(\mathbf{x})$; this is done by taking the gradient on both sides:

$$\nabla \mathbf{q}(\mathbf{x}) = \sum_{i=1}^4 \mathbf{q}^i \nabla \psi^i(\mathbf{x}). \quad (4.20)$$

Note that this gradient is constant per element since the shape functions ψ^i are linear. In order to numerically integrate a function $f(\mathbf{q})$ over $\mathcal{T}$ (for example, the integrability measure $h_{\mathbf{T}}$), we apply a 4-point Gaussian quadrature rule

$$\int_{\mathcal{T}} f(\mathbf{q}(\mathbf{x})) \, d\mathbf{x} \approx \sum_{k=1}^4 w_k f(\mathbf{q}(\mathbf{x}(\xi_k))) J_{\mathcal{T}}, \quad (4.21)$$

where ξ_i are the quadrature points on a reference tetrahedron, w_i the corresponding quadrature weights, and $J_{\mathcal{T}}$ the determinant of the Jacobian of the

4.2. Curl optimization

transformation $\mathbf{x}(\xi)$ mapping the reference tetrahedron to $\mathcal{T}$. This 4-point quadrature rule has a degree of precision of 2, meaning that it is exact for quadratic polynomials. The integrability measure $h_{\mathbf{T}}$ is not a polynomial but we found that this degree of precision is accurate enough for our purposes. The integral of f over the whole domain is then obtained by summing the integral over the tets

$$\int_{\Omega} f(\mathbf{q}(\mathbf{x})) d\mathbf{x} = \sum_i \int_{\mathcal{T}_i} f(\mathbf{q}(\mathbf{x})) d\mathbf{x}. \quad (4.22)$$

This calculation is embarrassingly parallel, as the sum over the tets can be distributed over a large number of CPUs. This is the strategy we use to accelerate our code and compute frame fields with sufficient resolution in 3D.

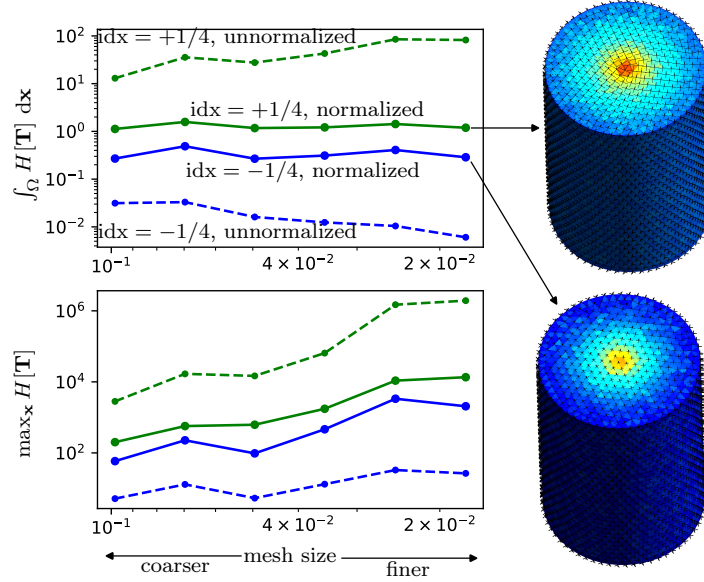


Figure 4.2: Mesh convergence of the curl energy $H[\mathbf{T}]$ around singularities of valence 3 and 5, in both its normalized and unnormalized versions.

Mesh independence. An important feature of an energy such as $H[\mathbf{T}]$ is that it is mesh-independent, i.e., asymptotically, its value remains the same when the mesh is refined. As we showed in the beginning of chapter 3, when approaching a singularity of index $k/4$, the frame field grows ($k > 0$) or shrinks ($k < 0$) at a rate of $r^{-k/4}$, where r is the distance to the singularity. This makes mesh independence in the vicinity of singularities challenging. We have empirically observed that the normalization we introduced in the integrability measure $h_{\mathbf{T}}$ achieves *mesh independence*. This is illustrated in fig. 4.2 where we compute the normalized and unnormalized curl energy of discretized integrable

frame fields. Even though the value of the integrand $h_{\mathbf{T}}$ blows up when refining the mesh, the normalized energy remains bounded and converges to a fixed value for both singularities; this value is higher for a valence 3 singularity than a valence 5 since the size is unbounded. This makes the curl energy mesh-independent, allowing the solver to introduce singularities even on fine support meshes.

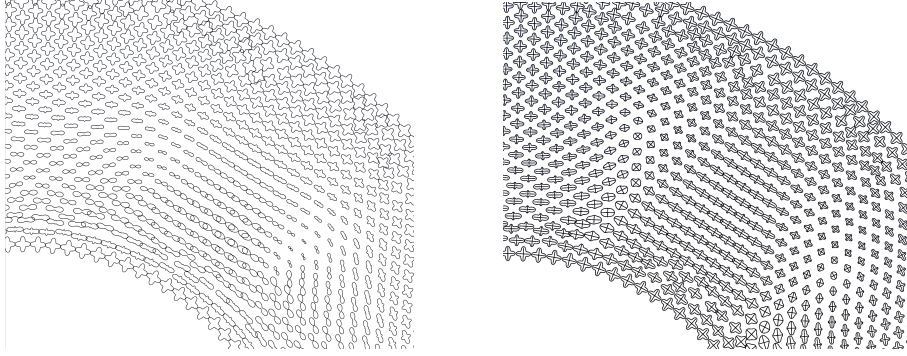


Figure 4.3: Curl normalization acts as a barrier preventing sizes from going negative. Left: a frame field optimized with the unnormalized curl produces frames with negative sizes, violating the local injectivity. Right: using the normalized curl forces sizes to remain positive (projected frames are also displayed).

Local injectivity. A second advantage of the curl normalization is that it provides a barrier that maintains the local injectivity of frames. Recall that a curl-free frame field integrates to a locally injective parametrization if its frame vectors are linearly independent: $\det(\mathbf{v}_1 \ \cdots \ \mathbf{v}_n) > 0$. Given that our frame vectors are mutually orthogonal by nature (as they are encoded by odeco tensors), local injectivity is verified if the frame vectors never become zero. This could happen if the eigenvalues of an odeco tensor become negative, which is not excluded by the representation. As illustrated in fig. 4.3, we found that applying the curl normalization acts as a barrier that keeps tensor eigenvalues positive. We can thus safely assume that the frame fields we compute are guaranteed to be locally injective.

Relaxation of the odeco constraints. At this point, it might be tempting to simply minimize $H[\mathbf{T}]$ under the constraint that the field is everywhere odeco. However, this would fail to introduce new singularities as it is often energetically cheaper to distribute the curl on the domain rather than to introduce the fixed discretization cost of singularities. To address this, we relax the odeconess constraint and allow the solver to introduce non-odeco terms that reduce the integrability cost in the neighborhood of singularities: we will see that these additional terms are isotropic in the plane of singularities. In

4.2. Curl optimization

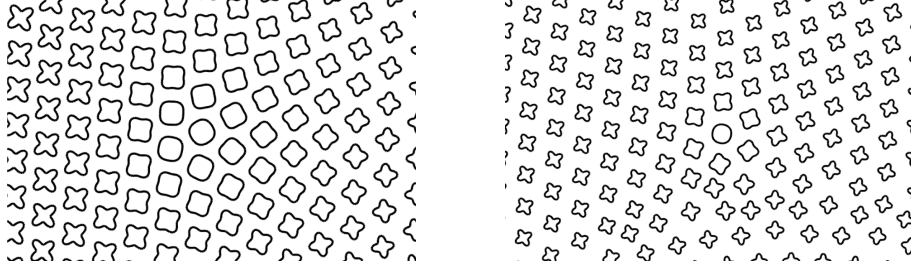


Figure 4.4: Solutions of eq. (4.24) producing a valence 5 singularity with different ϵ parameters. Left: a large ϵ produces a diffuse singularity. Right: a small ϵ locates the singularity more precisely. Setting an even smaller ϵ might destroy the singularity if it is not topologically necessary.

practice, we introduce an *odeco penalty*

$$C[\mathbf{T}] \triangleq \sum_i \int_{\Omega} c_i^2(\mathbf{T}(\mathbf{x})) \, d\mathbf{x}, \quad (4.23)$$

where $c_i(\cdot)$ are the quadrics defining the odeco variety; this functional is thus a 4th-order polynomial in the tensor coefficients. We assemble the curl energy and the odeco penalty in a Ginzburg-Landau-like functional

$$\min_{\mathbf{T}} E_{\epsilon}[\mathbf{T}] = H[\mathbf{T}] + \frac{1}{\epsilon^2} C[\mathbf{T}]. \quad (4.24)$$

Parameter ϵ has units of length and, like in the Ginzburg-Landau framework, controls the size of the singularity neighborhoods where the frames escape the odeco variety; we illustrate this in fig. 4.4. Choosing a large ϵ produces diffuse singularities with a large non-odeco region and, since the curl energy becomes invalid for non-odeco tensors, the resulting frame field obtained after projection of the tensors is not exactly integrable. On the other hand, choosing a too small ϵ prevents singularities from being created, as the error introduced by the discretization is too high. Appropriate values for ϵ depend on the mesh size and the size of geometrical features that the frame field must adapt to.

Dirichlet regularization. Integrable frame fields are not unique: one can take a set of arbitrary scalar potentials $(\phi_1, \dots, \phi_n)$, and add their gradients to obtain a new integrable frame field $(\mathbf{v}_1 + \nabla\phi_1, \dots, \mathbf{v}_n + \nabla\phi_n)$. Because of this, local minima of the objective function eq. (4.24) can be very flat (the gradient is near-zero) since there is a large region of neighboring integrable frame fields. This can cause the solver to converge very slowly to optimal solutions. To alleviate this problem we include a Dirichlet regularization term

$$E_D[\mathbf{T}] \triangleq \sum_j \frac{1}{2} \int_{\Omega} \|\nabla q_j\|^2 \, d\mathbf{x}, \quad (4.25)$$

	Boundary constraint	Used for	DOFs	#constraints
2D	Alignment with normal $\mathbf{n}$, all sizes	Corners	0	5
	Alignment with normal $\mathbf{n}$, size along $\mathbf{n}$	Boundaries	1	5
3D	Alignment with normals $\mathbf{n}_1, \mathbf{n}_2, \mathbf{n}_3$, all sizes	Corners	0	15
	Alignment with normals $\mathbf{n}_1, \mathbf{n}_2$, sizes along $\mathbf{n}_1, \mathbf{n}_2$	Creases	1	14
	Alignment with normal $\mathbf{n}$, size along $\mathbf{n}$	Boundary surface	5	10

Table 4.1: Types of boundary constraints on frames, along with the number of degrees of freedom (DOFs) in their coefficients $\mathbf{q}$, and the number of affine constraints in $\mathbf{A}\mathbf{q} + \mathbf{b} = \mathbf{0}$.

where q_j are the coefficients of tensor $\mathbf{T}$ in the basis of spherical harmonics. This energy is convex in the coefficients q_j and therefore improves the convergence of the solver. An additional benefit is that it steers the solver towards smooth solutions, which are desirable from a user perspective. We replace the curl energy $H[\mathbf{T}]$ with a convex combination with $E_D[\mathbf{T}]$ parametrized by a parameter $\kappa \in [0, 1]$:

$$\min_{\mathbf{T}} E_{\kappa}[\mathbf{T}] \triangleq (1 - \kappa) H[\mathbf{T}] + \kappa E_D[\mathbf{T}] + \frac{1}{\epsilon^2} C[\mathbf{T}], \quad (4.26)$$

with a schedule set on κ such that it starts at 1, providing a smooth (but not integrable) odeco frame field, and is gradually reduced to 0 to enforce integrability.

Boundary alignment. As seen in chapter 3, any alignment or sizing constraint on an odeco tensor can be written as an affine expression at each boundary node

$$\mathbf{A}\mathbf{q} + \mathbf{b} = \mathbf{0}, \quad (4.27)$$

with $\mathbf{A} \in \mathbb{R}^{M \times N}$ and $\mathbf{b} \in \mathbb{R}^M$, and M depends on the type of constraint imposed. As a reminder, we determine this linear subspace by randomly generating a batch of (odeco) tensors respecting the constraint we want to impose, which allows us to compute $\mathbf{A}$ and $\mathbf{b}$. Note that $\mathbf{A}$ and $\mathbf{b}$ are different at each boundary node, since they depend on the orientation of the alignment.

Table 4.1 summarizes the types of boundary constraints we can implement based on this systematic approach, as well as their number of degrees of freedom and affine constraints.

Isotropy constraint. In chapter 3, eq. (3.30) we have seen that the spherical harmonics $\mathbf{q}_2$ of band 2 encode the anisotropy of the corresponding frame, and

4.2. Curl optimization

that it is isotropic iff $\mathbf{q}_2 = \mathbf{0}$. This can be easily encoded as a set of linear constraints $q_2^m = 0$ for $m = -2, \dots, 2$. Alternatively, the corresponding degrees of freedom can be removed by substituting $\mathbf{q}_2 = \mathbf{0}$ in the rest of the problem.

Solver and gradient computation. The optimization problem we pose (minimizing eq. (4.26) subject to eq. (4.27)) is a nonlinear and nonconvex problem with linear constraints. We address it using a quasi-Newton method, namely the limited-memory BFGS algorithm, which we modify to impose the linear constraints at each update. Let $\mathbf{q}$ be a tensor at a node that respects its alignment constraint $\mathbf{A}\mathbf{q} + \mathbf{b} = \mathbf{0}$, and let $\mathbf{g} \in \mathbb{R}^N$ be the gradient of the objective function with respect to $\mathbf{q}$. To ensure that any L-BFGS update does not violate the constraint, we project $\mathbf{g}$ onto the space normal to the rows of $\mathbf{A}$, effectively ensuring $\mathbf{A}\mathbf{g} = \mathbf{0}$. Recalling that $\mathbf{A}$ was constructed to have orthonormal rows, this projection is simply done by iteratively subtracting from $\mathbf{g}$ its component along row $\mathbf{a}_i$:

$$\text{for } i = 1, \dots, M: \quad \mathbf{g} := \mathbf{g} - (\mathbf{g} \cdot \mathbf{a}_i) \mathbf{a}_i. \quad (4.28)$$

We use the L-BFGS implementation of ALGLIB (Bochkanov, n.d.). Computing the gradient of the objective function $E[\mathbf{q}]$ with respect to the tensor coefficients q_j at the nodes is done with a finite-difference scheme:

$$\frac{\partial E}{\partial q_j^\alpha} \approx \frac{E[q_j^\alpha + \delta q_j^\alpha] - E[q_j]}{\delta q_j^\alpha}, \quad (4.29)$$

where α refers to a node and j to one of the N tensor coefficients. Note that only the mesh elements incident to node α contribute to the derivative $\partial E / \partial q_j^\alpha$. This means that for each tetrahedron the functional $E[\mathbf{q}]$ must be evaluated $4N$ times, which makes the gradient computation the clear bottleneck of the optimization procedure. Future work will include an evaluation of the analytical gradient which should significantly speed up the optimization process.

Frame field recovery. The minimization of (4.26) provides a field of tensors that is odeco everywhere except in the vicinity of singularities. We recover a frame at every node of the mesh in two steps: first, the tensors are projected onto the odeco variety using the semidefinite projector described in the previous chapter, eq. (3.38). This operation results in a set of frames of which we only keep the direction vectors $\mathbf{u}_1, \dots, \mathbf{u}_n$. In a second step, the sizes of the direction vectors are computed by contracting the tensor 4 times, as if they were eigenvectors:

$$\lambda_m = \mathbf{T} \cdot \mathbf{u}_m^4. \quad (4.30)$$

We found that this approach for recovering sizes provided better results than using the sizes given by the projector; we argue that this is because the integrability is expressed in terms of the eigenvectors and eigenvalues of the odeco tensor field, and not the sizes of the projected tensor.

Summary. The optimization procedure to compute an integrable frame field is summarized in Algorithm 2.

Algorithm 2 Integrable frame field solver

Input: A triangle mesh on planar geometry, or a tetrahedral mesh, with mesh size and/or orientation constraints on boundaries

Output: A smooth integrable frame field

On boundaries, matrices $\mathbf{A}, \mathbf{b}$ constraining alignment.

In interior, assign zero frames. On boundary, assign frames with size 1 along normal, and size 0 along the rest.

Compute tensor $\mathbf{T}$ and spherical harmonics coefficients $\mathbf{q}$.

for κ in $[1, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 0]$ **do**

$\mathbf{q} \leftarrow \operatorname{argmin}_{\mathbf{q}} E_{\kappa}[\mathbf{q}], (4.26)$, subject to eq. (4.27) using L-BFGS.

end for

for each node **do**

Recover directions $\mathbf{u}_1, \dots, \mathbf{u}_n$ using semidefinite projection.

Recover frame sizes $\lambda_1, \dots, \lambda_m$ using (4.30).

end for

4.3 Computing 2D seamless parametrizations

Frame field guided parametrization and meshing We briefly review, in the 2D planar case, how a seamless parametrization is computed from the frame field $\mathbf{F}(\mathbf{x})$ obtained through the optimization of (4.26); for a more detailed treatment we refer to, e.g., (Bommes et al., 2009). We compute a field-guided seamless parametrization with the following steps:

Singularity detection. Singular triangles are identified by calculating the turning number (or *holonomy*) of the frames around a given triangle; this number is $+1/4$ or $-1/4$ for a singularity of valence 3 and 5, respectively.

Cut graph. The triangle mesh is "cut open" such that a closed curve cannot turn around a singularity without crossing the cut graph. The singular triangles also belong to the cut graph (the parametrization is not defined on them).

Frame vectors assignment. Thanks to the cut graph, two vectors $\mathbf{u}$ and $\mathbf{v}$ can be chosen from each frame to form two continuous vector fields $\mathbf{u}(\mathbf{x})$ and $\mathbf{v}(\mathbf{x})$.

4.4. 2D results and discussion

Integration. The vector fields are integrated to scalar potentials $u(\mathbf{x})$ and $v(\mathbf{x})$ by solving the least-squares problem

$$\min_{\substack{u(\mathbf{x}) \\ v(\mathbf{x})}} \int_{\Omega} \left(\|\nabla u - \mathbf{u}\|^2 + \|\nabla v - \mathbf{v}\|^2 \right) d\mathbf{x}. \quad (4.31)$$

During this optimization, the parametrization is constrained to be seamless, meaning that (i) on the boundary, one of the potentials u or v is constant, and (ii) along each cut, the matching potential gradients are equal (i.e., the conformity constraint presented in section 1.1). These seamlessness constraints can be written as linear constraint on the unknowns which are straightforward to impose.

We call the residual of the optimization objective (4.31) the *integration error*; it quantifies how integrable the frame field was, and in general how well the parametrization aligns to the prescribed frame field. Achieving a small integration error means we can adequately control the size and orientation constraints in the seamless parametrization process.

Different approaches, reviewed earlier in chapter 1, exist to generate a fully quadrilateral mesh from a seamless parametrization. For the quadrilateral meshes in this work we use the quasi-structured quadrilateral mesher of Reberol et al. (2021), which consists in a frontal point insertion coupled with cavity remeshing and mesh smoothing.

4.4 2D results and discussion

To demonstrate the validity of our method, we start by comparing smooth frame fields against integrable frame fields for a set of geometries where the frame orientations and sizes are fixed on the boundaries. For each frame field we compute a parametrization as described in the previous section, and measure the integration error. For the integrable frame field we also generate a quadrilateral mesh using the previously described frontal mesher.

Smooth vs integrable frames. The results of this comparison are illustrated in fig. 4.5. On average, the integrable frame field achieves an integration error 8.6 times smaller than the smooth frame field. The smooth frame fields do not have the correct singularity configuration to perform the required size transition. This causes the corresponding parametrizations to be very anisotropic, even though the frames are fully isotropic. On the other hand, the integrable frame fields have a set of singularities that ensure the size transitions, and this is reflected in the parametrizations where the isolines remain globally isotropic and have the correct sizes at the boundaries. We also see on testcase (d) that the solver is able to represent abrupt size transitions through a tight pair of singularities.

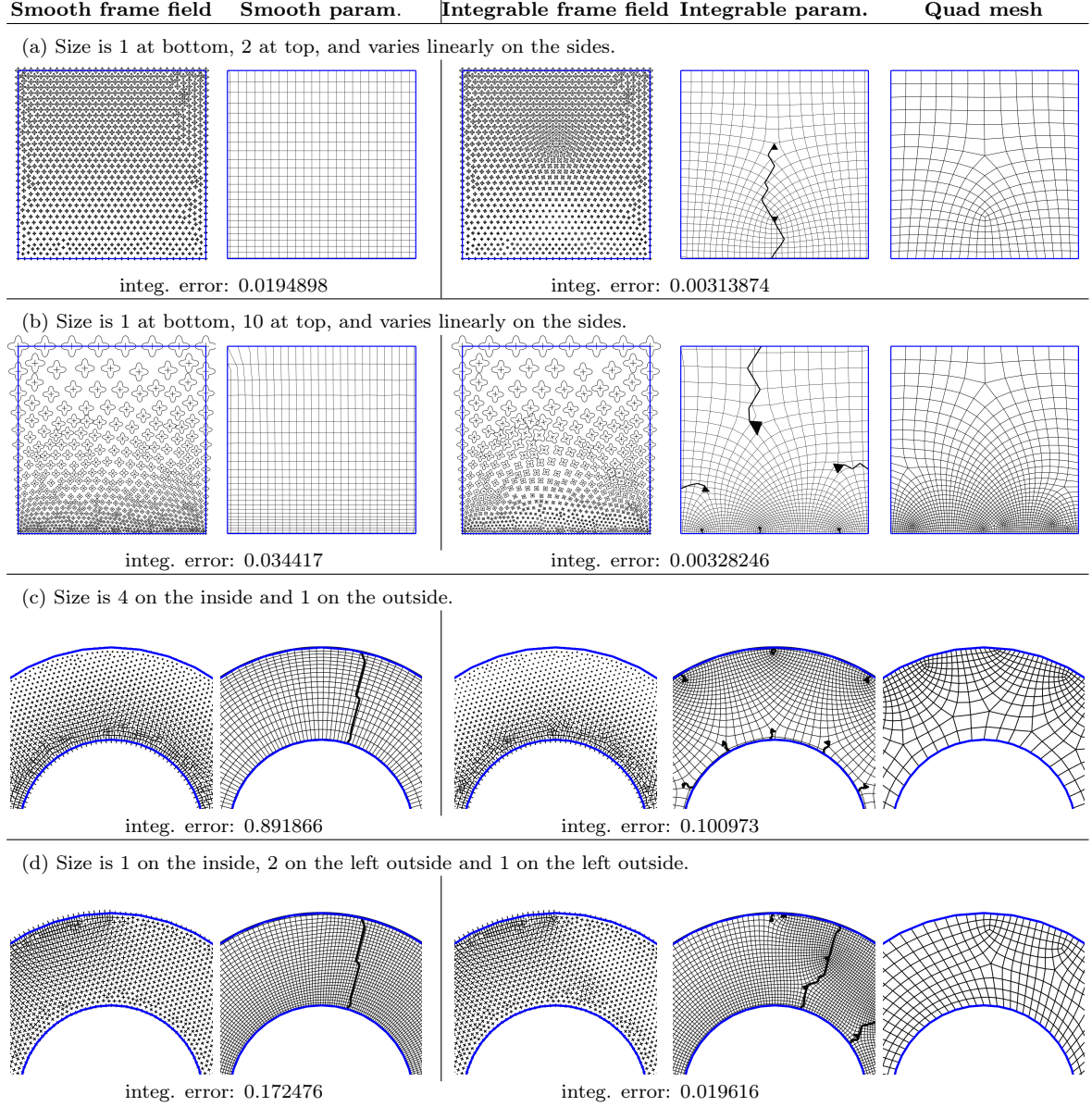


Figure 4.5: Comparison of smooth frame fields and integrable frame fields when sizes are imposed on boundaries. For each frame field the corresponding parametrization is illustrated, and a quad mesh is computed using guidance from the integrable frame field. All frame fields are constrained to be isotropic. For cases (a) and (b) we also illustrate the polynomials $p_T(\theta)$ corresponding to the frames.

4.4. 2D results and discussion

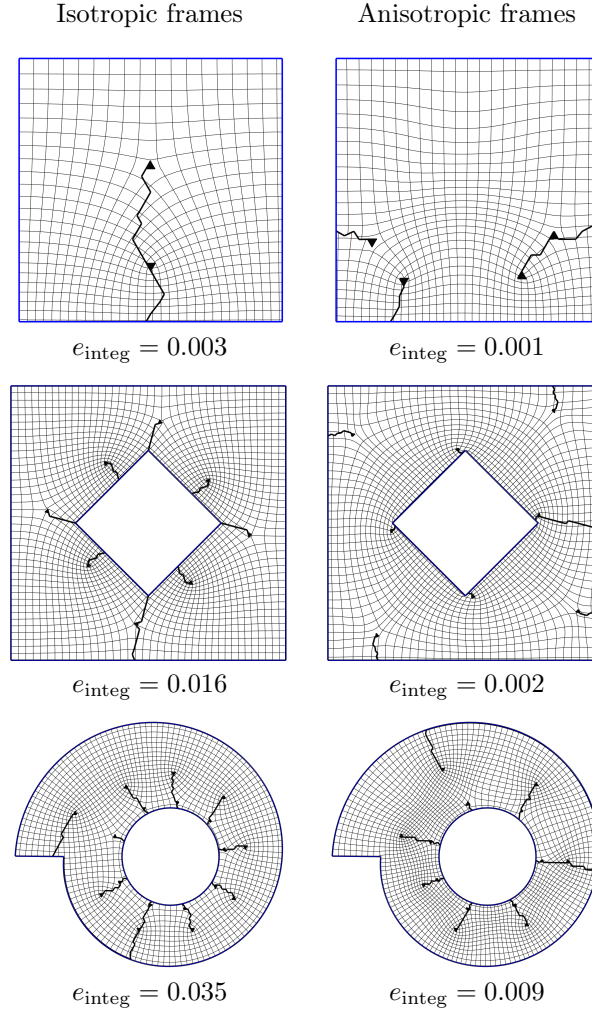


Figure 4.6: Seamless parametrizations computed for various models, including the *nautilus* (bottom) in the isotropic and anisotropic settings. e_{integ} denotes the integration error. Boundary size conditions are: (top) 1 at bottom and 2 at top, (middle) 1 on inner boundary and 2 on outer boundary, (bottom) 1 on inner and outer boundaries.

Isotropic vs anisotropic frames. Figure 4.6 compares the isotropic and anisotropic frame fields obtained with our solvers on additional examples. The anisotropic solver produces a very distinct singularity configuration compared to the isotropic solver, and achieves a much smaller integration error thanks to the additional degrees of freedom.

Limit cycles. Conventional frame field-driven quad meshing methods fail due to the presence of *limit cycles* in the frame field, which are the consequence of a non-meshable topology. A standard test case where a limit cycle appears is the *nautilus* model, for which we show our results on fig. 4.6, bottom. Our method successfully produces singularity configurations that are free of limit cycles, and a valid quad mesh can be extracted.

Existence of integrable frame fields. Given a set of boundary conditions, an exactly integrable frame field does not exist in general; in our case, by imposing sizes strongly on the boundaries, an integrable isotropic frame field very often does not exist. We refer to Jezdimirović et al. (2022) and the Abel-Jacobi framework for a complete treatment of these aspects. The non-existence of an integrable frame field can explain why a curl of zero is not achievable. Moreover, the smaller integration error achieved by allowing anisotropic frames (as illustrated in fig. 4.6) shows that the boundary conditions truly obstruct the integrability. This can be alleviated either by relaxing the boundary conditions or the isotropy constraint.

Additional examples. In figs. 4.7 to 4.9 we showcase additional, more realistic testcases, both in the isotropic and anisotropic settings. This illustrates the robustness of our method for computing seamless parametrizations that match the provided boundary constraints.

Convergence. Figure 4.10 shows how the different energies $H[\mathbf{T}]$, $C[\mathbf{T}]$ and $E_D[\mathbf{T}]$ converge with the L-BFGS iterations. We can clearly observe the jumps corresponding to the changes in the regularization parameter κ . As the smoothness requirement is relaxed, the field gets more integrable and odeco. The remaining *odeco* energy is concentrated in ϵ -neighborhoods around singularities. The remaining *curl* energy is partly concentrated at singularities, and partly distributed over the domain due to non-meshable boundary conditions, as mentioned above. In general, anisotropic frame fields require much more iterations than isotropic ones, probably because of the fact that local minima are flatter in the anisotropic case. As future work, we would like to explore second-order optimization methods, e.g., truncated Newton methods.

Limitations. These results also exhibit how the field of tensors behaves in the vicinity of singularities. As expected, the tensors become non-odeco as they narrow the singularities; this allows the solver to represent singularities

4.4. 2D results and discussion

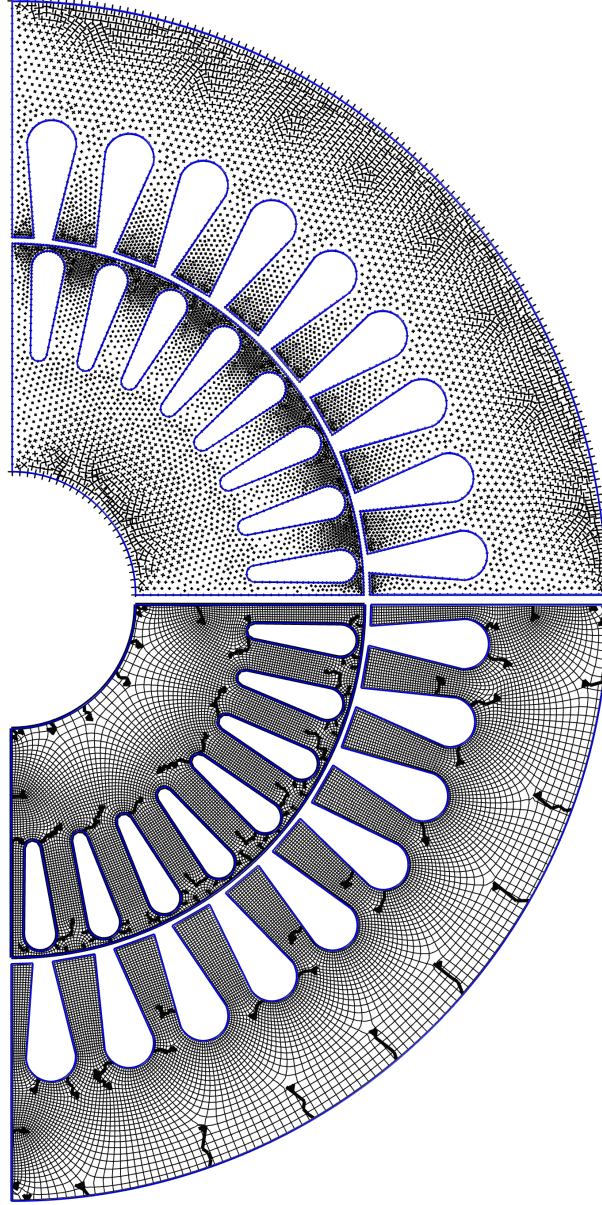


Figure 4.7: (Top) Isotropic integrable frame field. A size of 1 is imposed on the inner and outer circle arcs, and a size of 0.3 on the other boundaries. (Bottom) Corresponding seamless parametrization obtained by integration. The geometric model is a 3 kW asynchronous motor, from the Gmsh repository (Geuzaine et al., 2009).

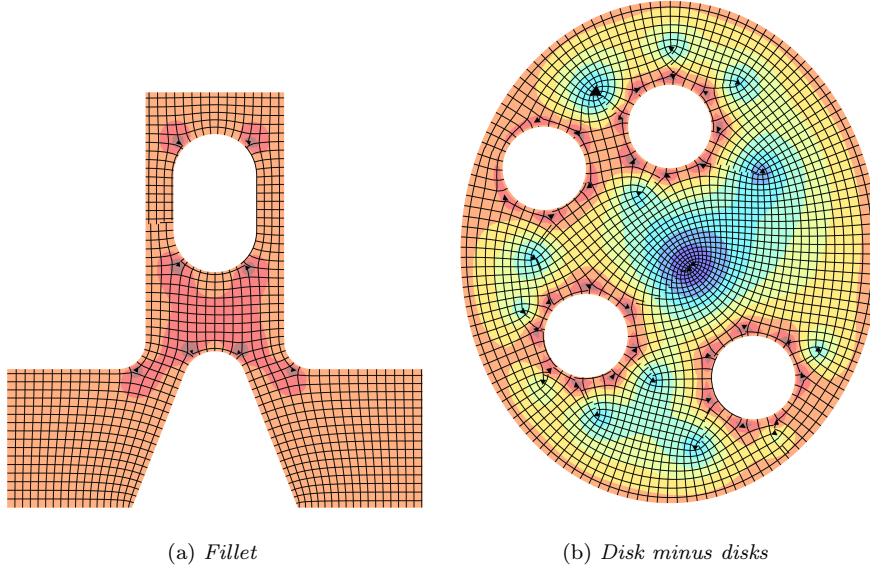


Figure 4.8: Seamless parametrizations computed from integrable isotropic frame fields. Colors indicate the local sizing of the *inverse* parametrization gradient (0 for valence 3, and $+\infty$ for valence 5 singularities). Unit frames are imposed on all boundaries.

in a way that is not too costly in terms of integrability energy. However it also brings a limitation: as the tensors are not odeco, the assumptions needed for the integrability energy fail, causing the frame field to not be exactly integrable close to singularities. In practice, one needs to tune parameter ϵ to achieve an appropriate trade-off between singularity cost and integrability.

The fixed cost in integrability and odeconess introduced by the singularities means that the solver could favor less singularities, that would otherwise improve the frame field integrability. In practice, this problem can be addressed by refining the triangle mesh and lower parameter ϵ , but this comes at a possibly significant computational cost.

A last important limitation is the *orthogonality* of the frames, which restricts the space of integrable solutions. This is particularly problematic when the geometry possesses corners that are not multiples of 90° , in which case the corner frame is ill-defined. This can be observed, to some extent, in fig. 4.8a.

4.4. 2D results and discussion

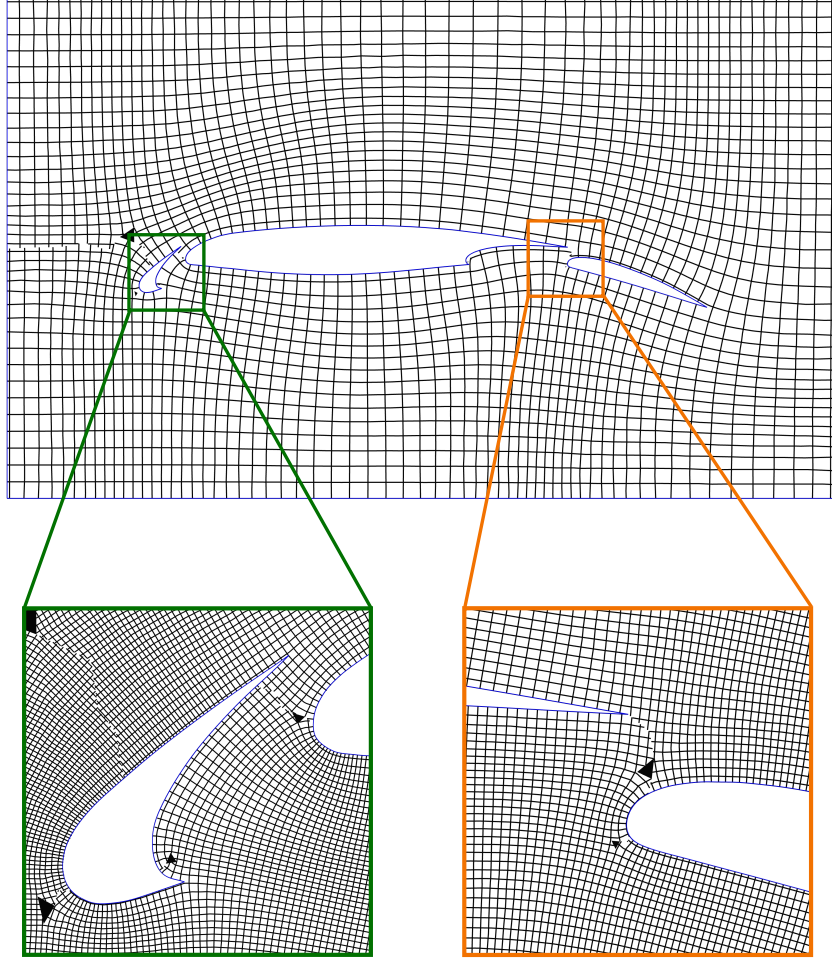


Figure 4.9: Cross-section of *three component wing* (with flap and slat). A seamless parametrization is computed from an integrable *anisotropic* frame field; the parametrization was sampled more finely (10x) in the zoomed parts. On all boundaries, a unit sizing is imposed in the normal direction, and the other direction has a free size.

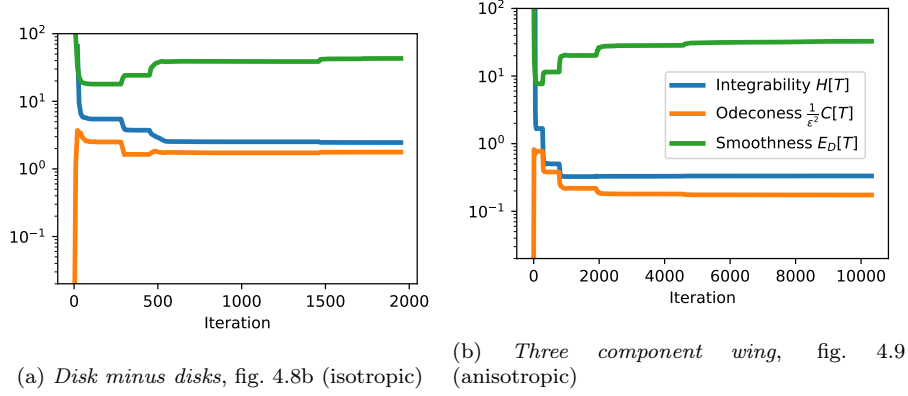
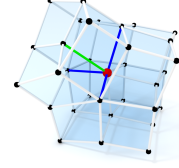


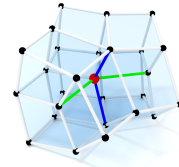
Figure 4.10: Convergence of the different energies $H[\mathbf{T}]$, $C[\mathbf{T}]$ and $E_D[\mathbf{T}]$ for two testcases shown before.

4.5 3D results and discussion

We applied the same algorithm 2 for computing integrable frame fields on 3D volumetric domains. In all testcases, the boundary constraint is to impose alignment with the surface normal, and a size of 1 in that direction. Like in 2D, we compare the classical smooth frame field (obtained by setting $\kappa = 1$ in the objective E_κ) against the integrable one. Figure 4.11 illustrates our results on the cube with a notch. As expected, the smooth frame field has a 3-5 singular node that makes it unmeshable. Upon convergence, our integrable frame field has automatically unzipped the 3-5 node all the way to the bottom of the cube, making it globally meshable; this frame field was fed to *AlgoHex* (Liu et al., 2023) as a guiding field and the corresponding all-hex mesh was correctly computed. We ran the same solver on a taller cube (extruded from the bottom) and see that it produced the alternative (meshable) singularity configuration, involving a 5-5-5-3 singular node (see inset), that does not require to unzip the node to the bottom, but rather to the sides of the cube (the additional valence 3 curve is independent and goes *above* the singular node). Feeding this to *AlgoHex* again produces a valid hex mesh with matching topology.



In fig. 4.12 we ran our solver on additional testcases. The first one is a sphere on top of a cube which possesses four 3-5 singular nodes in the smooth setting. For the integrable frame field it is not completely clear what the singular configuration is, but we interpret that the 3-5 nodes are broken by a valence 5 singularity running through it, producing a 5-5-3-3 singular node (see inset).



This configuration would be meshable, however, the mesher was not able to

4.5. 3D results and discussion

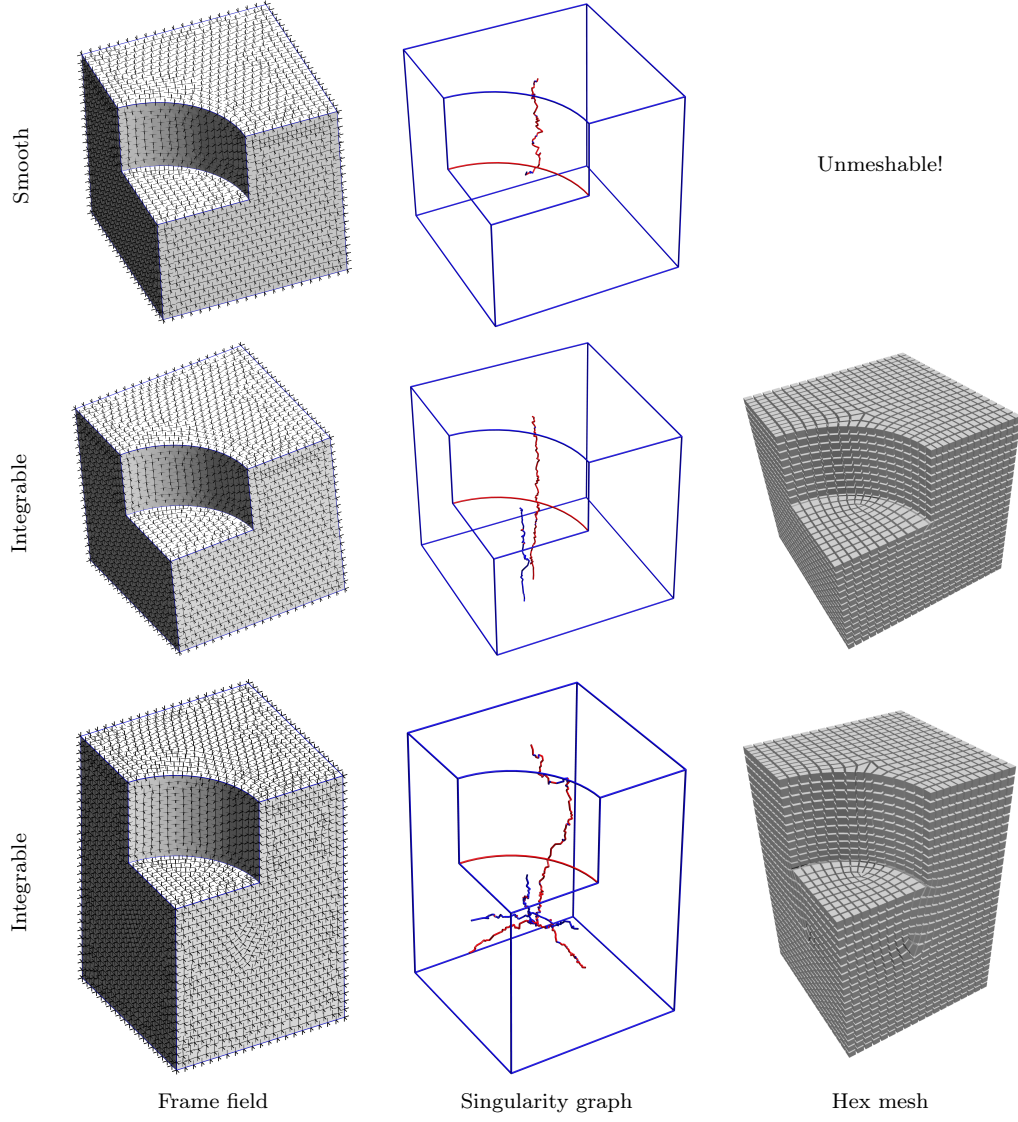


Figure 4.11: Results of our 3D integrable frame field solver on the famous *notch* geometry. Our integrable frame fields are globally meshable, and the full-hex meshes are generated by *AlgoHex* (Liu et al., 2023).

extract a hex mesh with matching topology. The second test case is the union of a large cylinder and a small cylinder. Again, we try to give an interpretation of the obtained singularity graph. The four valence 3 singular curves emanating from the small cylinder are meshable: two of them connect to the opposite side, and the other two turn to connect to the larger cylinder. The two remaining valence 3 singularities seem to go through the pair of valence 5 singularities, and we conjecture that they form, on each side, four singular nodes in 5-5-3-3 configuration. With this frame field as input, the mesher was not able to terminate in reasonable time.

These experiments demonstrate that our integrable frame field formulation brings the singularity graphs towards meshable topologies, but there is still room for improvement before reaching the same level of quality as in the 2D case. We discuss two issues that we believe hinder the full potential of our approach.

Performance. In all testcases, the supporting tetrahedral meshes possess about 150k tets and 25k vertices (hence $15 \times 25k = 375k$ degrees of freedom). The solver converges after about 15 000 L-BFGS iterations, which takes on the order of 8 hours on a compute node of 64 CPUs. The clear bottleneck is the finite difference gradient computation of the curl energy $H[\mathbf{T}]$, and this can be accelerated by deriving the analytical gradient. We are also planning on porting the code to GPU to speed up the computations. We believe that the noisy singular graphs we obtain is partly due to a poor resolution of the frame fields. Increasing the performance will give us access to finer frame fields and well-separated singular curves, which is crucial given the complexity of the graphs we aim to produce.

Odeco penalty normalization. While the curl energy $H[\mathbf{T}]$ has been normalized and should not blow up on integrable frame fields, it is not the case of the odeco penalty $C[\mathbf{T}]$ which is quadratic and blows up for valence 3 singular curves. We found that the ϵ parameter had to be set much higher than in the 2D case (i.e., a more permissive penalty); choosing lower values breaks the topology and re-zips the singular nodes to its initial invalid configuration. This has repercussions on the integrability of the resulting frame field as the curl energy becomes invalid when the solution is too far away from the odeco variety. In future work, we think it would be valuable to explore normalizations for the odeco penalty. The aim is to find an energy that depends at most on the length of the singular curve, and does not blow up when refining the mesh. This would enable us to better fit to the odeco variety, and produce frame fields that are more accurate in terms of integrability.

4.5. 3D results and discussion

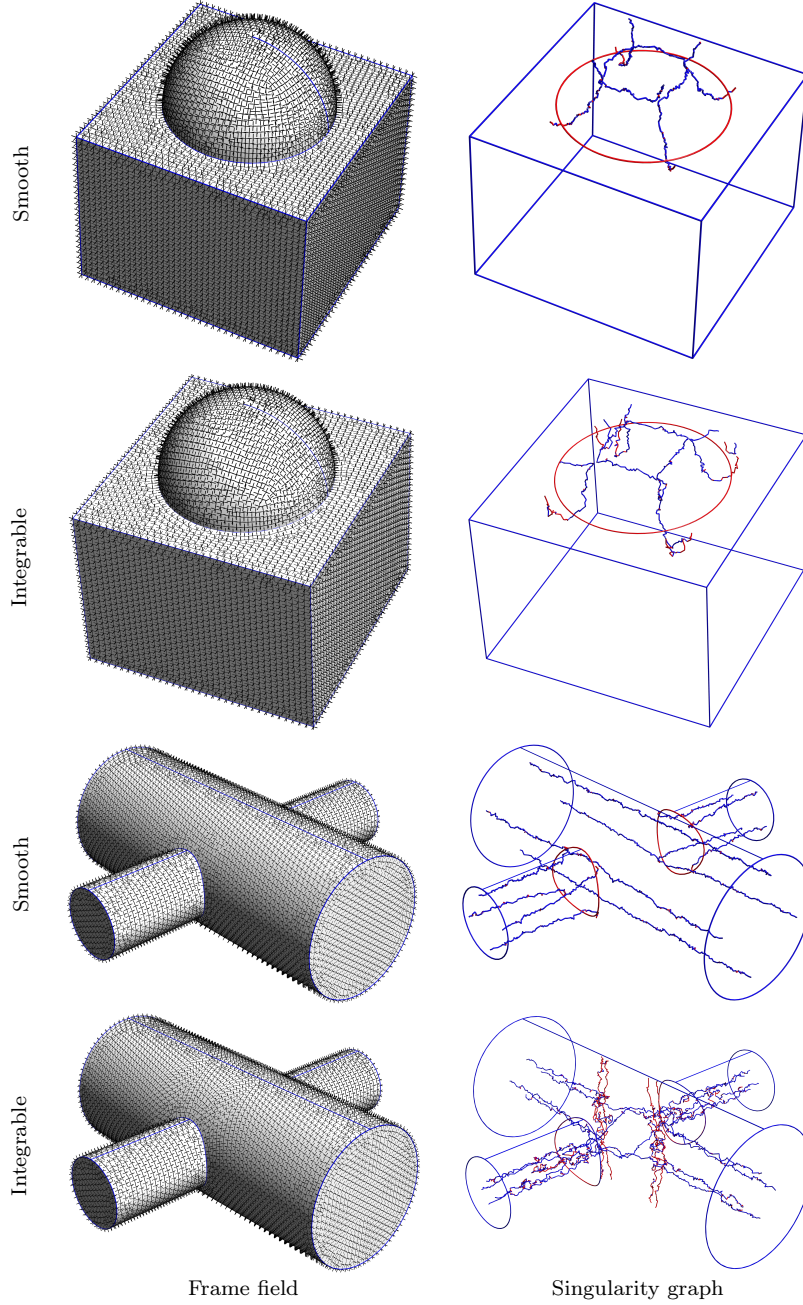


Figure 4.12: Results of our 3D integrable frame field solver on geometries from the HexMe dataset (Beaufort et al., 2022): `s05o_cube_sphere` and `s08o_cross_cyls_dr`. The hex mesher could not produce a hex mesh with matching singularity graph.

CHAPTER 5

Conclusion

This thesis has presented both practical and theoretical contributions to the structured meshing endeavor.

5.1 Summary of contributions

On the practical side, we propose an automatic method for generating a block-structured quadrilateral mesh (or a high-order coarse quad mesh) from a given quasi-structured mesh. Leveraging the robustness and quality of one of the best open-source industrial-grade quad meshers, we implement a quantization method that simplifies the topology and produces a mesh with 10-20 times less quadrilateral patches. This provides a robust end-to-end mesher that is available to computational engineers wishing to apply high-order numerical methods on complex CAD models.

On the theoretical side, we have constructed an integrability measure for both 2D and 3D frame fields based on odeco tensors, a versatile representation of orthogonal scaled frames that implicitly encodes the symmetries of seamless parametrizations. The integrability expression does not require any explicit knowledge on the frame vectors or their topology; it only depends on the tensor coefficients and their spatial gradients. This work is the first to express 2D integrability on a tensor (or, equivalently, polynomial) representation, and the first to express 3D integrability on any implicit frame field representation. A key ingredient of this success is the study of the sensitivity of a frame with respect to perturbations in the underlying tensor, which allows to get rid of

the spatial derivatives in the expression of the frame field curl.

Based on this result, we have proposed an optimization scheme that solves for integrable frame fields in the planar 2D case and the volumetric 3D case; this forms a challenging nonlinear and nonconvex optimization problem on the odeco variety. A key insight of our approach is to slightly relax the odeco constraint; this enables the solver to create singularities even though they introduce a cost due to the finite element discretization. Contrary to previous approaches based on smoothness, our frame fields often have a singularity configuration that is globally meshable, which guarantees that a seamless parametrization with the same topology can be easily computed.

5.2 What it means for meshing

Even if our integrable frame field solver has weaknesses which we discussed, we believe it is a proof of concept that offers exciting perspectives for the meshing communities. As we have seen, state-of-the-art meshers decompose the problem into a sequence of relaxed, simpler problems. The price to pay for this decomposition is that the whole process is dependent on the bad choices made early in the process, which are hard to fix *a posteriori*. In particular, this means that the user loses control over essential mesh features such as element sizing and anisotropy. Since integrable frame fields are essentially equivalent to seamless parametrizations, our approach removes a step of the pipeline and the loss of control that comes with it, due to smooth cross fields not being meshable. Thanks to our contribution, we think an exact and efficient integrable frame field solver is in reach, and will be part of an industrial-grade mesher that could finally abandon the unstructured meshing techniques.

In the 3D realm, generating all-hexahedral meshes even on simple geometries is still an open research problem. The reason is that smooth 3D frame fields are often not even locally meshable. This makes them practically useless, except as a guide for a hex-dominant mesher, or as a starting point for an algorithmic frame field repair. In this case, integrability is not a nice-to-have feature, but an essential property to guarantee global meshability. Like in the 2D case, making our ideas ready for practical use would yield a fully automatic all-hexahedral mesher. Given that a significant amount of human time is still devoted to manual or semi-automatic meshing, this research is well worth the effort to finally remove the meshing bottleneck from the numerical simulation process.

5.3 Research avenues

Before being ready to be integrated in a meshing pipeline, our integrable frame fields solver has room for improvement, which also raises a number of exciting research questions.

5.3. Research avenues

Staying on the manifold. As we explained before, escaping the odeco manifold (or variety) in the vicinity of singularities enables their creation. But it comes with a price: when tensors are non-odeco, it does not correspond exactly to a frame anymore, and the integrability measure loses its mathematical sense. The consequence is that the frame field obtained by projecting the tensors is not exactly integrable, and this can have repercussions on the fidelity of the computed seamless parametrizations. In the 3D case, the odeco constraint has to be strongly relaxed, which can cause the singularity graph to not be resolved enough to determine the field’s global topology.

A possible way to overcome this issue is to perform a second optimization step, once the global frame field topology is roughly determined, where odeconess becomes a hard constraint: this can be done with *manifold optimization*. Palmer et al. (2020) perform optimization on the odeco variety for computing smooth frame fields, using the *Manopt* optimization toolbox (Boumal et al., 2014). Our guess is that the hard odeco constraint would prevent the solver from destroying the topology, and guarantee that the optimized integrability is better reflected in the frame field.

Beyond odeco tensors. Frame fields (and seamless parametrizations) should not always be orthogonal everywhere. Industrial models often possess (sometimes very) acute or obtuse angles, and these models do not admit everywhere-orthogonal parametrizations and meshes. However, odeco tensors strongly constrain frame fields to be perfectly orthogonal. It would be interesting to consider an extension of odeco tensors to non-orthogonal frames. *Unitary* frame fields have been considered by Sageman-Furnas et al. (2019) (integrable 2D) and Desobry et al. (2021) (smooth 2D and 3D), but scaled non-orthogonal frames do not have an implicit representation yet. One might consider general, non-odeco tensors, but these typically possess more degrees of freedom than required. Just like odeco tensors have a one-to-one correspondence with orthogonal frames, a thrilling research question is to find a variety of tensors that have a one-to-one correspondence with non-orthogonal frames, and formulate an integrability criterion; we leave that for future work.

Optimal integrable frame fields. This work has not put much focus on the optimality of the generated frame fields, e.g., in terms of distortion of the subsequent seamless parametrization. *Optimality* is a fuzzy notion, though, since a parametrization’s distortion can be made arbitrarily small by introducing infinitely many singularities, which is not desirable from a meshing perspective as it would produce arbitrarily small quad elements. This suggests that optimality is application-dependent and, in the case of meshing, a minimal sizing constraint should be involved. In the current state, the only way we promote smoothness in the frame field (and thus lower distortion) is through the initialization, and the regularization term in the 2D case. Explicitly promoting smoothness with an additional term does not work as it would compete with

integrability. Instead, we should look for a way to impose integrability as a constraint and formulate smoothness as an objective. Integrability is a nonconvex quadratic constraint, and finding appropriate optimization methods remains an open research problem.

Bibliography

- Pierre-Alexandre Beaufort, Jonathan Lambrechts, François Henrotte, Christophe Geuzaine, and Jean-François Remacle (2017). Computing Cross Fields: A PDE Approach Based on the Ginzburg-Landau Theory. *Procedia Engineering* 203, pp. 219–231.
- Pierre-Alexandre Beaufort, Maxence Reberol, Denis Kalmykov, Heng Liu, Franck Ledoux, and David Bommes (2022). Hex Me If You Can. *Computer Graphics Forum* 41.5, pp. 125–134.
- Sergey Bochkhanov (n.d.). ALGLIB.
- David Bommes, Marcel Campen, Hans-Christian Ebke, Pierre Alliez, and Leif Kobbelt (2013a). Integer-Grid Maps for Reliable Quad Meshing. *ACM Transactions on Graphics* 32.4, pp. 1–12.
- David Bommes, Timm Lempfer, and Leif Kobbelt (2011). Global Structure Optimization of Quadrilateral Meshes. *Computer Graphics Forum* 30.2, pp. 375–384.
- David Bommes, Bruno Lévy, Nico Pietroni, Enrico Puppo, Claudio Silva, Marco Tarini, and Denis Zorin (2013b). Quad-Mesh Generation and Processing: A Survey. *Computer Graphics Forum* 32.6, pp. 51–76.
- David Bommes, Henrik Zimmer, and Leif Kobbelt (2009). Mixed-Integer Quadrangulation. *ACM Transactions on Graphics* 28.3, 77:1–77:10.
- Nicolas Boumal, Bamdev Mishra, Pierre-Antoine Absil, and Rodolphe Sepulchre (2014). Manopt, a Matlab Toolbox for Optimization on Manifolds. *Journal of Machine Learning Research* 15.42, pp. 1455–1459.
- O. Brodersen (2002). Drag Prediction of Engine-Airframe Interference Effects Using Unstructured Navier-Stokes Calculations. *Journal of Aircraft* 39.6, pp. 927–935.
- Marcel Campen (2017). Partitioning Surfaces Into Quadrilateral Patches: A Survey. *Computer Graphics Forum* 36.8, pp. 567–588.
- Alexandre Chemin, François Henrotte, Jean-François Remacle, and Jean Van Schaftingen (2019). Representing Three-Dimensional Cross Fields Using Fourth Order Tensors. *27th International Meshing Roundtable*. Ed. by Xevi Roca and Adrien Loseille. Vol. 127. Cham: Springer International Publishing, pp. 89–108.
- Etienne Corman and Keenan Crane (2019). Symmetric Moving Frames. *ACM Transactions on Graphics* 38.4, pp. 1–16.
- Mattéo Couplet, Alexandre Chemin, and Jean-François Remacle (2024). Integrable Frame Fields Using Odeco Tensors. *Proceedings of the 2024 International Meshing*

- Roundtable (IMR). International Meshing Roundtable (IMR).* Society for Industrial and Applied Mathematics.
- Mattéo Couplet, Maxence Reberol, and Jean-François Remacle (2021). Generation of High-Order Coarse Quad Meshes on CAD Models via Integer Linear Programming. *AIAA Aviation Forum*.
- Keenan Crane (2023). Discrete Differential Geometry: An Applied Introduction.
- David Desobry, Yoann Coudert-Osmont, Etienne Corman, Nicolas Ray, and Dmitry Sokolov (2021). Designing 2D and 3D Non-Orthogonal Frame Fields. *Computer-Aided Design* 139, p. 103081.
- Olga Diamanti, Amir Vaxman, Daniele Panozzo, and Olga Sorkine-Hornung (2014). Designing N -PolyVector Fields with Complex Polynomials. *Computer Graphics Forum* 33.5, pp. 1–11.
- (2015). Integrable PolyVector Fields. *ACM Transactions on Graphics* 34.4, pp. 1–12.
- Xianzhong Fang, Jin Huang, Yiyong Tong, and Hujun Bao (2021). Metric-Driven 3D Frame Field Generation. *IEEE Transactions on Visualization and Computer Graphics*, pp. 1–1.
- Christophe Geuzaine and Jean-François Remacle (2009). Gmsh: A 3-D Finite Element Mesh Generator with Built-in Pre- and Post-processing Facilities. *International Journal for Numerical Methods in Engineering* 79.11, pp. 1309–1331.
- Aaron Hertzmann and Denis Zorin (2000). Illustrating Smooth Surfaces. *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques - SIGGRAPH '00. The 27th Annual Conference*. Not Known: ACM Press, pp. 517–526.
- Christopher J. Hillar and Lek-Heng Lim (2013). Most Tensor Problems Are NP-Hard. *Journal of the ACM* 60.6, pp. 1–39.
- Jin Huang, Yiyong Tong, Hongyu Wei, and Hujun Bao (2011). Boundary Aligned Smooth 3D Cross-Frame Field. *ACM Transactions on Graphics* 30.6, pp. 1–8.
- T. J. R. Hughes, J. A. Cottrell, and Y. Bazilevs (2005). Isogeometric Analysis: CAD, Finite Elements, NURBS, Exact Geometry and Mesh Refinement. *Computer Methods in Applied Mechanics and Engineering* 194.39, pp. 4135–4195.
- Wenzel Jakob, Marco Tarini, Daniele Panozzo, and Olga Sorkine-Hornung (2015). Instant Field-Aligned Meshes. *ACM Transactions on Graphics* 34.6, pp. 1–15.
- Jovana Jezdimirović, Alexandre Chemin, and Jean François Remacle (2020). Multi-Block Decomposition and Meshing of 2D Domain Using Ginzburg-Landau PDE. *28th International Meshing Roundtable*. [object Object].
- Jovana Jezdimirović, Alexandre Chemin, and Jean-François Remacle (2022). Integrable Cross-Field Generation Based on Imposed Singularity Configuration – the 2D Manifold Case. *SIAM International Meshing Roundtable*. arXiv.
- Tengfei Jiang, Xianzhong Fang, Jin Huang, Hujun Bao, Yiyong Tong, and Mathieu Desbrun (2015). Frame Field Generation through Metric Customization. *ACM Transactions on Graphics* 34.4, 40:1–40:11.

Bibliography

- Tengfei Jiang, Jin Huang, Yuanzhen Wang, Yiyong Tong, and Hujun Bao (2014). Frame Field Singularity Correction for Automatic Hexahedralization. *IEEE Transactions on Visualization and Computer Graphics* 20.8, pp. 1189–1199.
- Felix Kälberer, Matthias Nieser, and Konrad Polthier (2007). QuadCover - Surface Parameterization Using Branched Coverings. *Computer Graphics Forum* 26.3, pp. 375–384.
- Felix Knöppel, Keenan Crane, Ulrich Pinkall, and Peter Schröder (2013). Globally Optimal Direction Fields. *ACM Transactions on Graphics* 32.4, pp. 1–10.
- P. M. Knupp (1999). Winslow Smoothing on Two-Dimensional Unstructured Meshes. *Engineering with Computers* 15.3, pp. 263–268.
- Eduard Kuznetsov and K. V. Kholshevnikov (1992). Estimation of the Number of Extrema of a Spherical Harmonic. *Soviet Astronomy* 36.69, pp. 439–442.
- Franck Ledoux (2019). MAMBO: Model dAtabase Mesh BLocking.
- Lek-Heng Lim (2005). Singular Values and Eigenvalues of Tensors: A Variational Approach. *1st IEEE International Workshop on Computational Advances in Multi-Sensor Adaptive Processing, 2005*. Puerto Vallarta, Mexico: IEEE, pp. 129–132.
- Heng Liu and David Bommes (2023). Locally Meshable Frame Fields. *ACM Transactions on Graphics* 42.4, pp. 1–20.
- Heng Liu, Paul Zhang, Edward Chien, Justin Solomon, and David Bommes (2018). Singularity-Constrained Octahedral Fields for Hexahedral Meshing. *ACM Transactions on Graphics* 37.4, pp. 1–17.
- M. Lyon, M. Campen, and L. Kobbelt (2021). Quad Layouts via Constrained T-Mesh Quantization. *Computer Graphics Forum* 40.2, pp. 305–314.
- M. Nieser, U. Reitebuch, and K. Polthier (2011). CubeCover - Parameterization of 3D Volumes. *Computer Graphics Forum* 30.5, pp. 1397–1406.
- Jonathan Palacios and Eugene Zhang (2007). Rotational Symmetry Field Design on Surfaces. *ACM Transactions on Graphics* 26.3, p. 55.
- David Palmer, David Bommes, and Justin Solomon (2020). Algebraic Representations for Volumetric Frame Fields. *ACM Transactions on Graphics* 39.2, pp. 1–17.
- Daniele Panozzo, Enrico Puppo, Marco Tarini, and Olga Sorkine-Hornung (2014). Frame Fields: Anisotropic and Non-Orthogonal Cross Fields. *ACM Transactions on Graphics* 33.4, pp. 1–11.
- Nico Pietroni, Marcel Campen, Alla Sheffer, Gianmarco Cherchi, David Bommes, Xifeng Gao, Riccardo Scateni, Franck Ledoux, Jean-Francois Remacle, and Marco Livesu (2022). Hex-Mesh Generation and Processing: A Survey. Version 1.
- Liqun Qi (2007). Eigenvalues and Invariants of Tensors. *Journal of Mathematical Analysis and Applications* 325.2, pp. 1363–1377.
- Nicolas Ray, Wan Chiu Li, Bruno Lévy, Alla Sheffer, and Pierre Alliez (2006). Periodic Global Parameterization. *ACM Transactions on Graphics* 25.4, pp. 1460–1485.
- Nicolas Ray, Dmitry Sokolov, and Bruno Lévy (2016). Practical 3D Frame Field Generation. *ACM Transactions on Graphics* 35.6, pp. 1–9.

- Nicolas Ray, Bruno Vallet, Laurent Alonso, and Bruno Levy (2009). Geometry-Aware Direction Field Processing. *ACM Transactions on Graphics* 29.1, pp. 1–11.
- Nicolas Ray, Bruno Vallet, Wan Chiu Li, and Bruno Lévy (2008). N-Symmetry Direction Field Design. *ACM Transactions on Graphics* 27.2, pp. 1–13.
- Faniry H. Razafindrazaka, Ulrich Reitebuch, and Konrad Polthier (2015). Perfect Matching Quad Layouts for Manifold Meshes. *Computer Graphics Forum* 34.5, pp. 219–228.
- Maxence Reberol, Alexandre Chemin, and Jean-Francois Remacle (2019). Multiple Approaches to Frame Field Correction for CAD Models. *28th International Meshing Roundtable*, p. 13.
- Maxence Reberol, Christos Georgiadis, and Jean-François Remacle (2021). Quasi-Structured Quadrilateral Meshing in Gmsh – a Robust Pipeline for Complex CAD Models. (Visited on 05/18/2021). Pre-published.
- Jean-François Remacle, Jonathan Lambrechts, Bruno Seny, Emilie Marchandise, Amaury Johnen, and Christophe Geuzaine (2012). Blossom-Quad: A Non-Uniform Quadrilateral Mesh Generator Using a Minimum-Cost Perfect-Matching Algorithm. *International Journal for Numerical Methods in Engineering* 89.9, pp. 1102–1119.
- Elina Robeva (2016). Orthogonal Decomposition of Symmetric Tensors. *SIAM Journal on Matrix Analysis and Applications* 37.1, pp. 86–102.
- Andrew O. Sageman-Furnas, Albert Chern, Mirela Ben-Chen, and Amir Vaxman (2019). Chebyshev Nets from Commuting PolyVector Fields. *ACM Transactions on Graphics* 38.6, pp. 1–16.
- Ruben Sevilla, Sonia Fernández-Méndez, and Antonio Huerta (2008). NURBS-enhanced Finite Element Method (NEFEM). *International Journal for Numerical Methods in Engineering* 76.1, pp. 56–83.
- Justin Solomon, Amir Vaxman, and David Bommes (2017). Boundary Element Octahedral Fields in Volumes. *ACM Transactions on Graphics* 36.3, pp. 1–16.
- Marco Tarini, Enrico Puppo, Daniele Panozzo, Nico Pietroni, and Paolo Cignoni (2011). Simple Quad Domains for Field Aligned Mesh Parametrization. *ACM Transactions on Graphics* 30.6, pp. 1–12.
- Ryan Viertel and Braxton Osting (2019). An Approach to Quad Meshing Based on Harmonic Cross-Valued Maps and the Ginzburg–Landau Theory. *SIAM Journal on Scientific Computing* 41.1, A452–A479.
- Alan M Winslow (1966). Numerical Solution of the Quasilinear Poisson Equation in a Nonuniform Triangle Mesh. *Journal of Computational Physics* 1.2, pp. 149–172.